Name: ________________ **Entry Number:** ________________

Major Exam – Solution

COL334/672: Computer Networks

Sem I, 2025-26

There are 12 questions and 20 pages in this quiz booklet (including this page). There are **75 total points**, and you have **120 minutes** to answer the questions.

- **Feel free to think outside the box but write inside the box**
- **Write concise answers**
- **Do not start the exam until instructed to do so**

I Transport Layer

1. **[9 points]:** Answer the following questions:

A. TCP uses Go-Back-N for implementing reliability. True or False. Justify your answer.

B. Consider a connection-termination protocol in which each side must first send a "done" message and also wait to receive a "done" message from the other side before simultaneously closing the connection. Is this practical in real-world networks? Explain why or why not.

C. In TCP, if the instantaneous round trip time at any given time is *t* sec, the value of the retransmission timeout is always set to greater than or equal to t sec. True or False. Justify your answer.

D. Consider a TCP connection where the last byte ACKed is A, latest byte sent but not yet ACKed is S, current congestion window is C bytes, and the receiver advertised window is R bytes. What is the relationship among these variables? What role does the receiver-advertised window play?

Solution: A. False. TCP uses a sliding-window with selective retransmission (duplicate ACKs + fast retransmit/fast recovery) and buffers out-of-order segments, so it is not a pure Go-Back-N protocol.

Solution: **B.** No, this is not practical in real-world networks. This protocol suffers from the "two-army" problem: because the network is unreliable, a lost "done" can leave the peer unaware and both sides may wait indefinitely, producing deadlock/livelock.

C. The statement is FALSE because RTO is based on smoothed historical RTT values, not the current instantaneous RTT, and if the network suddenly becomes slower (RTT spikes upward), the current instantaneous RTT may temporarily exceed the computed RTO.

D. $S - A \leq \min(C, R)$.

The receiver-advertised window R is a flow-control mechanism that tells the sender how much buffer space the receiver currently has free. It prevents the sender from overrunning the receiver's buffer by limiting outstanding data to at most R bytes beyond the last ACKed byte.

2. [8 points]: Answer the following questions:

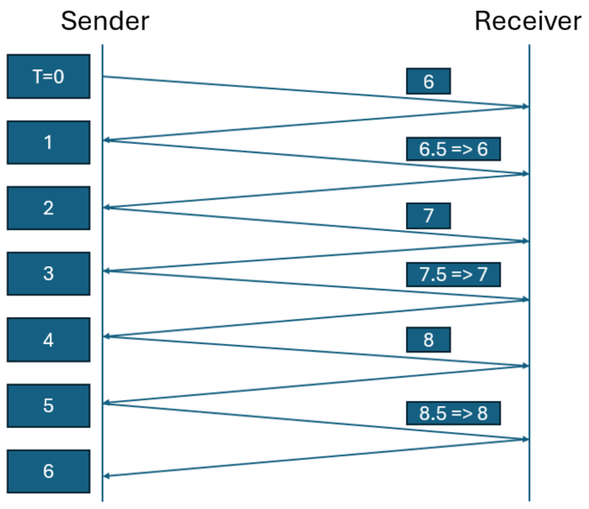
A. Suppose TCP uses AIMD congestion control with no slow start and an initial window of 6 MSS. *The receiver sends one ACK for every two packets, and the sender transmits only MSS-sized packets*. Assuming a constant RTT and no loss, what is the average throughput (in terms of MSS and RTT) over the first 6 RTTs?

B. Assuming TCP is using slow start with initial window = 1 MSS on a line with a 10-msec round-trip time and no congestion. The receiver-advertised window is 24 KB and the maximum segment size is 2 KB. How long does it take before the sender can transmit a full receiver window?

C. Suppose TCP starts using Multiplicative Increase, Multiplicative Decrease for congestion control. Would such an algorithm be TCP-fair? Justify your answer.

Solution:

A.
Since TCP increases the congestion window by 1 MSS every 2 RTTs, the increase rate is 0.5 MSS per RTT from an initial value of 6 MSS. Sender only sends MSS size packets. Therefore, with a window of x.5 MSS implies the sender sends x MSS packets.



$$\text{Average throughput} = \frac{6+6+7+7+8+8}{6} = 7 \text{ MSS/RTT}.$$

Solution:

B.
Receiver window $= \dfrac{24 \text{ KB}}{2 \text{ KB}} = 12$ MSS.
With slow start, the congestion window doubles every RTT:
starting from 1 MSS:
after 1 RTT: 2 MSS,
after 2 RTTs: 4 MSS,
after 3 RTTs: 8 MSS,
after 4 RTTs: 16 MSS ≥ 12.

Thus a full 12-MSS receiver window can be sent after 4 RTTs, i.e., 4×10 ms $= 40$ ms.

C.
Consider two flows with congestion windows w_1 and w_2 where $w_1 > w_2$.
Under MIMD, on each increase step:

$w_1' = (1+\alpha)w_1$, $w_2' = (1+\alpha)w_2$, $\alpha > 0$.

Then $\dfrac{w_1'}{w_2'} = \dfrac{(1+\alpha)w_1}{(1+\alpha)w_2} = \dfrac{w_1}{w_2}$,

so the **ratio stays the same** and the **larger flow always gains more in absolute terms**.

On a loss,
$w_1'' = \beta w_1$, $w_2'' = \beta w_2$, $0 < \beta < 1$,
and $\dfrac{w_1''}{w_2''} = \dfrac{\beta w_1}{\beta w_2} = \dfrac{w_1}{w_2}$,
again preserving the ratio.

Thus MIMD never drives the flows toward equal share: a flow that starts larger stays larger, so it is not TCP-fair.

If instead $w_1 = w_2$ initially, both increase and decrease by the same factor and remain equal in throughput.

3. [6 points]: Recall the macroscopic description of TCP throughput. In the period of time from when the connection's rate varies from $\frac{W}{2\times RTT}$ to $\frac{W}{RTT}$, only one packet is lost (at the very end of the period).

A. Show that the loss rate (fraction of packets lost) is equal to $\frac{1}{\frac{3}{8}W^2+\frac{3}{4}W}$

B. Use the result above to show that if a connection has a loss rate p, then its average throughput is approximately given by $\frac{1.22\times MSS}{RTT\sqrt{p}}$

Solution:

A. Loss rate: $L = \frac{1}{N_{\text{sent}}}$.
Packets sent:

$$\sum_{n=0}^{W/2}\left(\frac{W}{2}+n\right) = \left(\frac{W}{2}+1\right)\frac{W}{2} + \frac{\frac{W}{2}\left(\frac{W}{2}+1\right)}{2}$$

Hence:

$$N_{\text{sent}} = \frac{3}{8}W^2 + \frac{3}{4}W$$

So:

$$L = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

B. For large W: $p \approx \frac{8}{3W^2}$, so $W \approx \sqrt{\frac{8}{3p}}$.
Throughput:

$$\text{throughput} \approx \frac{3}{4}\cdot\frac{W}{RTT}\cdot MSS \approx \frac{1.22\,MSS}{RTT\sqrt{p}}$$

4. [4 points]: Consider a single bottleneck link shared by two long-lived flows. Flow 1 uses TCP Vegas, a delay-based congestion control algorithm, while Flow 2 uses TCP Reno, a loss-based congestion control algorithm. Assume both flows have the same RTT and traverse the same bottleneck router. The router can be configured in three ways:

A. as a simple drop-tail FIFO queue, or

B. using a per-flow round-robin scheduling policy, or

C. using Random Early Detection (RED), which drops packets probabilistically before the buffer is full.

Rank the three cases in decreasing order of the fairness achieved between the delay-based and loss-based flows. Justify your answer.

Solution:

Fairness Ranking: $B > C > A$

- **B (Per-flow Round-Robin)** — Most fair, since each flow receives an equal share of bandwidth regardless of congestion control behavior.
- **C (RED)** — Moderately fair, since probabilistic early dropping moderates Reno's aggressive behavior, allowing Vegas to retain some bandwidth.
- **A (Drop-tail FIFO)** — Least fair, because Vegas backs off early due to rising delay, allowing Reno to dominate the available bandwidth.

II Application

5. [6 points]: Consider a DASH-based video streaming client. The video is divided into chunks of duration L seconds each. Let:

- B_k = buffer occupancy (in seconds) immediately after chunk k has finished downloading and has been added to the buffer
- B_{max} = buffer threshold (in seconds) beyond which the client pauses new chunk requests
- R_{k+1} = bitrate selected for $k+1^{th}$ chunk (bits/sec)
- C_{k+1} = observed bandwidth during the $k+1$ download (bits/sec)

Assume that playback continues during downloading as long as the buffer is non-empty. A chunk is available only after it has been fully downloaded.

A. How long does the client wait (if at all) before sending the request for chunk k+1?

B. Derive an expression for the buffer occupancy B_{k+1} (in terms of variables above) at the end of downloading chunk $k+1$.

C. Give an expression for the stall (rebuffering) duration, if any, during the download of chunk k+1.

D. Suppose you are designing a bitrate adaptation algorithm that optimizes the following function:

$$QoE = f(x) - \alpha \times g(y) - \beta \times h(z)$$

Here x, y, and z are the **application-level** performance metrics and f, g, and h are non-decreasing functions. Briefly state what these metrics could represent in the video streaming context?

Solution:

(1) Waiting time before requesting chunk $k+1$:

$$w = \max\{0,\ B_k - B_{\text{max}}\}.$$

(If $B_k \le B_{\text{max}}$ then $w = 0$ and the request is sent immediately.)

(2) Buffer occupancy B_{k+1} after downloading chunk $k+1$:
Chunk download time:

$$D = \frac{R_{k+1}\,L}{C_{k+1}}.$$

Buffer at download start:

$$B_{\text{start}} = \min\{B_k,\ B_{\text{max}}\}.$$

Two cases:

$$B_{k+1} = \begin{cases} B_{\text{start}} - D + L, & \text{if } D \le B_{\text{start}}, \\ L, & \text{if } D > B_{\text{start}}. \end{cases}$$

Solution:

(3) Stall (rebuffering) duration during download of chunk $k+1$:

$$S = \max\{0,\ D - B_{\text{start}}\} = \max\left\{0,\ \frac{R_{k+1}L}{C_{k+1}} - \min\{B_k, B_{\max}\}\right\}.$$

(4) Choice of application-level QoE metrics (compact):

- x — **video quality (higher is better)**: e.g., average bitrate or average PSNR/SSIM. $f(x)$ non-decreasing.
- y — **rebuffering / stall penalty (negatively correlated with x)**: e.g., total rebuffering time, number of stalls, or stall duration. $g(y)$ non-decreasing.
- z — **quality-smoothness penalty (negatively correlated with x)**: e.g., number of bitrate switches or number of quality changes (or sum of $|R_{i+1} - R_i|$). $h(z)$ non-decreasing.

Example objective:

$$QoE = \text{avg bitrate} \ - \ \alpha \cdot (\text{total stall time}) \ - \ \beta \cdot (\text{number of bitrate switches}).$$

6. [8 points]: Answer the following questions:

A. A Web page consists of multiple objects fetched over a persistent HTTP connection. When these objects are requested sequentially, the total page load time is 3 seconds, of which 150 ms is spent establishing the connection. When HTTP pipelining is used for the same page, the total load time decreases to 200 ms, assuming all object requests are issued at once and each object has the same response time. How many objects (HTTP requests) are required to load this page?

B. The `If-Modified-Since` header can be used to check whether a cached page is still valid. Requests can be made for pages containing images as well as only HTML text. Do you think the effectiveness of this technique is better or worse for JPEG images as compared to HTML? Think carefully about what "effectiveness" means and explain your answer.

C. Why does HTTP/2 still suffer from head-of-line (HOL) blocking, even though it supports multiplexing over a single TCP connection? Briefly describe a solution that avoids this limitation.

Solution:

A. $RTT = 200\text{ms} - 150\text{ms} = 50\text{ms}$
3000 ms = Connection establishment time $+ N \times$ RTT
$N = \frac{3000-150}{50} = 57$

B. Better for JPEG.
Efficiency: saving time to load the page, bandwidth saved, saved network time, etc.
JPEG files are larger in size than HTML text, hence better to save as they save more bandwidth.
JPEG files are static, hence more chances of hitting "Not Modified" than HTML files.

C. HTTP/2 suffers HOL blocking because TCP guarantees in-order delivery.
If any packet is lost, all streams are paused until the lost packet is recovered, causing HOL blocking.
Solution: Protocol with independent delivery of streams.
QUIC: loss in one stream does not block others.

7. [5 points]: Briefly answer the following:

A. In BitTorrent, suppose a peer downloads file chunks strictly in order (i.e., chunk 1 first, then chunk 2, and so on). Describe one advantage of this policy. Describe one limitation of this policy and propose an alternative chunk selection policy that addresses the limitation.

B. Recall that DNS MX records are used for email delivery. Suppose DNS were redesigned to eliminate MX records and rely solely on CNAME or A records to identify the mail server for a domain. What problems would arise with such a design?

Solution: A.

- In-order Chunk Download in BitTorrent. Advantage: Downloading chunks strictly in order removes the need for reordering or maintaining a large buffer at the receiver. The file can be reconstructed immediately as chunks arrive, which also supports streaming.
- Limitation: If the initial chunks (e.g., chunk 1) are rare or only available from slow peers, the entire download is delayed even if later chunks are widely available. This reduces parallelism and results in poor download performance. Also impacts the performance of the overall swarm as now
- Alternative Chunk Selection Policy: The rarest-first chunk selection policy addresses this limitation by prioritizing the download of the least available chunks in the swarm. This improves parallelism, prevents bottlenecks, and ensures uniform chunk distribution among peers.

B. CNAME or A records are for HTTP content whereas MX records are for email. Having a single type for both would imply that both these content types would now need to be hosted on the same server. This is an issue in terms of modular design.

8. [5 points]: A Distributed Hash Table (DHT) stores key-value pairs across multiple nodes arranged in a circular identifier space $[0, 2^m - 1]$. Each node is responsible for the keys in some portion of this space, and maintains pointers to its successor and predecessor nodes.

You are given the following basic primitives:

- `lookup(x)`: returns the node responsible for key x (returns the successor node if x is not present)
- `store(k, v)`: stores key–value pair (k, v) at the local node
- `get_keys(x)`: returns all key–value pairs currently stored at node x
- `del(k)`: deletes key k and its value at the local node
- `update_successor(x, v)`: node x sets node v as its successor
- `update_predecessor(x, v)`: node x sets node v as its predecessor

A new node u with identifier `u_id` wants to join the DHT. It knows one existing node c in the system and can send point-to-point messages. Describe the join protocol for node u, i.e., how does u update the DHT structure and take responsibility for its portion of the key space using the primitives listed above.

Solution:

Step 1. Node u contacts node c and computes s using lookup of `u_id` to find its successor.

Step 2. Node u requests all keys from node s using get keys(s). For each key, that is less than `u_id`, node u stores it using store(k, v) and node s deletes it using del(k).

Step 3. Node u computes its predecessor p using lookup of minimum key value minus one modulo 2^m.

Step 4. Node u sets its successor using update successor(u, s) and sets its predecessor using update predecessor(u, p).

Step 5. Node p updates its successor using update successor(p, u) and node s updates its predecessor using update predecessor(s, u).

III Network Security

9. [6 points]: Briefly answer the following questions:

A. Suppose N people want to communicate with each of N- 1 other people using symmetric key encryption. All communication between any two people, i and j, is visible to all other people in this group of N, and no other person in this group should be able to decode their communication. How many keys are required in the system as a whole? Now suppose that public key encryption is used. How many keys are required in this case?

B. Suppose a TLS session employs a block cipher with cipher block chaining (CBC). True or false: The server sends to the client the IV in the clear

C. Consider a variation of the MAC algorithm where the sender sends *(m, H(m)+s)*, where *H(m)+s* is the concatenation of *H(m)* and *s*. Is this variation flawed? Why or why not?

Solution:

A. For symmetric key encryption, every unordered pair of users (i, j) requires a unique shared secret key. Thus the total number of keys needed is

$$\binom{N}{2} = \frac{N(N-1)}{2}.$$

Packets sent: For public key encryption, each user has a public key and a private key. Therefore the total number of keys required is

$$2N.$$

B. **True.** In CBC mode, the IV does not need to be kept secret; it only needs to be unpredictable for security. TLS therefore sends the IV in the clear at the start of the record so the receiver can properly decrypt the first block.

Solution:

C. This is flawed because the secret s is transmitted directly as part of the tag; an attacker who observes $H(m) + s$ immediately learns s as he can compute $H(m)$, and can then forge valid MACs for arbitrary messages.
Thus the construction provides <u>no</u> message integrity.

10. [6 points]: Answer the following questions:

A. The secure email protocol we discussed in class did not use *nonces*. Is this a problem? Explain.

B. What is a *truncation* attack, and how does TLS mitigate this attack?

C. Recall that in IPSec ESP mode, the original datagram is followed by padding, a pad length field, and a next-header field. Explain the purpose of each of these fields.

Solution:

A. **No**—the absence of nonces is **not a problem** for secure email. The email is **not an interactive protocol**. It is a *store-and-forward*, one-way message system, and there is no notion of a *session or liveness* to prove. Nonces are mainly needed to prevent **replay attacks** in authentication protocols. But in secure email:

  - Each message is signed by the sender's private key, and the receiver can simply **ignore duplicate messages**, so replaying the same email has **no security impact**.
  - A replayed email does *not* cause the recipient to perform unintended actions automatically—email is passive content.

  Therefore, absence of nonces does not weaken confidentiality, integrity, or authentication in the context of secure email. OR We have also awarded marks for alternate answers that state email is now susceptible to replay attacks and mention the issue of repeated emails.

Solution:

B. A **truncation attack** occurs when an attacker forges or injects a **TCP connection-close** (FIN) segment, causing one or both endpoints to believe that the other side has finished sending data earlier than it really has. This can make a party think the session ended normally even though some data was never delivered.
TLS mitigates this by **authenticating the connection-close message**. TLS wraps the *CLOSE-NOTIFY* alert inside a TLS record that includes a **MAC over the message and the TLS sequence number**. Therefore, an attacker cannot forge or modify the close alert without detection, preventing premature or false termination of the session.

C. The purpose of each of these fields is as follows:

  - ∗ **Padding:** Padding is added so that the payload aligns with the block size of the underlying block cipher (e.g., AES). Block ciphers require the payload to be a multiple of the block length; padding ensures proper alignment and may also help obscure the true length of the data.
  - ∗ **Pad Length:** This 1-byte field specifies **how many padding bytes** were added. Since padding can vary, the receiver needs this value to correctly remove the padding after decryption.
  - ∗ **Next-Header:** The Next-Header field tells the receiver what protocol follows after the ESP payload is decrypted. It identifies the type of the encapsulated payload, e.g., IPv4 (protocol 4), or IPv6 (protocol 41). Since ESP encrypts the original payload (including the inner IP header in tunnel mode), the Next-Header field is needed so the receiver knows how to interpret the decrypted data.

11. [4 points]: Consider the following network policy: drop any incoming TCP packet unless it belongs to a flow for which a valid outgoing SYN packet has already been observed. Explain how this policy can be implemented in an SDN network consisting of OpenFlow switches and a controller.

Solution: To enact this network policy, we essentially create a stateful firewall implemented by the controller, using OpenFlow switches as a stateless match–action devices. The important points that need to be addressed while building this firewall are as follows:

1. **Installation of default/starting switch rules:**

   $*$ **Outgoing SYN (internal → external):** match `in_port=internal, eth_type=IP, ip_proto=TCP, TCP flags=SYN`; action: forward to external and send copy to controller.

   $*$ **Incoming TCP (external → internal):** low-priority match on `in_port=external, ip_proto=TCP`; action: drop.

   → By default, all incoming TCP is dropped unless an explicit allow rule is added.

2. **Controller state update on outgoing SYN:** On receiving `PACKET_IN` for an outgoing SYN, the controller extracts the 4-tuple `(internalIP, internalPort, externalIP, externalPort)` and adds an entry marking this flow as **allowed / SYN_SENT** in its connection table.

3. **Controller actions after updating state:** The controller installs two high-priority rules:

   $*$ **Internal → External:** match on src/dst IP and ports; action: forward to external.

   $*$ **External → Internal:** reverse match; action: forward to internal.

   → These override the default drop rule, permitting only flows whose outgoing SYN was observed.

4. **Cleanup handling:** On seeing FIN/RST (via `PACKET_IN`), the controller removes the connection from its table and deletes the associated allow rules. Optionally, flow entries use `idle_timeout` to auto-expire inactive connections.

12. [8 points]: Recall that BGP is a path-vector protocol used for inter-domain routing. Each AS advertises its own prefixes to its neighbors, as well as forwards the announcements it has received from others after appending its own AS information in the AS-PATH. A malicious AS can launch at least two kinds of attacks:

A. It can falsely claim to originate a prefix (e.g., pretend to be the origin of IP prefix P).

B. It can modify the BGP route advertisements it has received from its peers (e.g., by adding or removing ASes in the AS-PATH).

Design an extension to BGP, called Secure BGP (S-BGP), that mitigates both of the above attacks. In your design, describe the cryptographic mechanisms and trust infrastructure used. Also, explain how your design prevents each of the two attacks mentioned above. Finally, mention **two** practical challenges of deploying your design at Internet scale.

Solution: Secure BGP relies on a digital certificate signed by a trusted Certificate Authority (CA). This certificate binds the IP prefix and the AS number to the AS's public key.
S-BGP uses public–private key cryptography: each AS holds a private key, while its public key is made available through the PKI and can be distributed on request.
Every AS route advertisement first appends its own AS number to the AS-PATH and then signs the updated path using its private key. Thus, the advertisement carries a sequence of signatures, one contributed by each AS in the path. The receiving AS verifies the signature using the sender's public key. This ensures that the route advertisement indeed came from the claimed AS. ASes can then sequentially verify every AS in the AS-PATH using the corresponding AS's public keys. If any AS is missing, added, or altered, the signature verification fails and the intrusion is detected.
Challenges:

- A global PKI is required, including a hierarchy of trusted authorities.
- A globally trusted Certificate Authority must exist to assign certificates securely.
- Routers may require hardware upgrades to support the new secure infrastructure.
- There is significant computational overhead, as routers must verify multiple signatures for each BGP update.