5단원. 전송 계층: TCP와 UDP (최빈출 단원)

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

Q1. TCP와 UDP의 차이를 설명하세요. ★★★

답안 **TCP**는 연결 지향으로, 3-way 핸드셰이크로 연결을 수립하고 시퀀스 번호·ACK·재전송으로 **신뢰성(순서 보장, 유실 복구)**을 제공하며 흐름·혼잡 제어를 수행합니다. 대신 헤더가 크고(기본 20바이트) 지연이 있습니다. **UDP**는 비연결형으로 포트 구분과 체크섬만 제공하는 최소한의 프로토콜입니다(헤더 8바이트). 빠르고 오버헤드가 적지만 유실·순서·중복을 애플리케이션이 처리해야 합니다. TCP는 웹, 메일, 파일 전송 등 정확성이 중요한 곳에, UDP는 DNS, 실시간 스트리밍·게임, VoIP 등 속도·실시간성이 중요한 곳에 쓰입니다. "왜 DNS는 UDP인가"(작은 질의-응답에 연결 수립이 낭비, 단 큰 응답과 존 전송은 TCP) 같은 후속 질문에 대비하세요.

관련 개념 신뢰성 메커니즘, 헤더 비교, DNS의 UDP/TCP 혼용, QUIC(UDP 기반 신뢰 전송)

Q2. TCP 3-way 핸드셰이크 과정을 설명하세요. ★★★

답안 ① 클라이언트가 **SYN**(시퀀스 번호 x)을 전송 → ② 서버가 **SYN+ACK**(자신의 시퀀스 y, 응답 x+1)로 응답 → ③ 클라이언트가 **ACK**(y+1)를 보내면 연결 수립(ESTABLISHED). 3번의 교환이 필요한 이유는 **양쪽 모두 자신의 초기 시퀀스 번호(ISN)를 상대에게 알리고 확인받아야** 하기 때문입니다. 2-way로는 서버의 ISN을 클라이언트가 확인해줬는지 서버가 알 수 없습니다. ISN을 난수로 정하는 이유는 이전 연결의 지연 패킷과의 혼동 방지와 시퀀스 추측 공격 방어입니다. 서버는 SYN을 받으면 SYN_RECV 상태로 백로그 큐에 쌓아두는데, 이를 노린 것이 SYN 플러딩 공격입니다.

관련 개념 ISN, SYN_RECV/백로그 큐, SYN 플러딩과 SYN 쿠키, TFO(TCP Fast Open)

Q3. TCP 연결 종료(4-way handshake)와 TIME_WAIT 상태를 설명하세요. ★★★

답안 종료는 ① FIN → ② ACK → ③ (반대 방향) FIN → ④ ACK의 4단계입니다. 4단계인 이유는 TCP가 전이중이라 **양방향을 각각 닫아야** 하고, FIN을 받은 쪽이 남은 데이터를 마저 보낼 수 있어야 하기 때문입니다(하프 클로즈). 먼저 닫은 쪽은 마지막 ACK 후 **TIME_WAIT** 상태로 2MSL(보통 60초) 대기합니다. 이유는 ① 마지막 ACK가 유실되면 상대의 FIN 재전송에 다시 응답하기 위해, ② 이전 연결의 지연 패킷이 같은 포트 조합의 새 연결에 섞이는 것을 방지하기 위해서입니다. 짧은 연결을 대량으로 맺는 클라이언트(프록시, LB)에서는 TIME_WAIT 소켓이 수만 개 쌓여 포트 고갈이 날 수 있으며, 커넥션 풀·keep-alive 사용과 tcp_tw_reuse 튜닝으로 대응합니다.

관련 개념 하프 클로즈, 2MSL, TIME_WAIT 포트 고갈, tcp_tw_reuse, CLOSE_WAIT 누적(앱이 close 안 함)

Q4. TCP 헤더의 주요 필드와 플래그를 설명하세요. ★★

답안 주요 필드는 출발/목적지 **포트**(16비트씩), **시퀀스 번호**(보낸 바이트 위치), **확인 응답 번호**(다음에 받기를 기대하는 바이트), **윈도우 크기**(수신 가능 버퍼), 체크섬, 옵션(MSS, 윈도우 스케일, SACK, 타임스탬프)입니다. 플래그는 **SYN**(연결 요청), **ACK**(확인), **FIN**(정상 종료), **RST**(강제 리셋: 닫힌 포트 접근, 비정상 상태), **PSH**(버퍼링 없이 즉시 전달), **URG**(긴급)입니다. 실무에서 RST의 의미(연결 거부 vs 방화벽 리셋 vs 애플리케이션 크래시)를 패킷 캡처로 구분하는 질문이 나올 수 있습니다.

관련 개념 시퀀스/ACK 번호 계산, 윈도우 스케일 옵션, SACK, RST의 발생 원인들

Q5. TCP 흐름 제어(슬라이딩 윈도우)를 설명하세요. ★★★

답안 흐름 제어는 **수신자의 처리 능력**을 초과하지 않도록 송신 속도를 조절하는 것입니다. 수신자는 ACK에 자신의 남은 버퍼 크기(**수신 윈도우, rwnd**)를 실어 보내고, 송신자는 ACK 없이 보낼 수 있는 데이터량을 그 범위로 제한합니다. 윈도우 안의 여러 세그먼트를 한꺼번에 보내고 ACK가 오면 윈도우를 미끄러뜨리는(slide) 방식이라 슬라이딩 윈도우입니다. 윈도우가 0이 되면 송신을 멈추고, 제로 윈도우 프로브로 재개 시점을 확인합니다. 원래 16비트(64KB) 한계는 **윈도우 스케일 옵션**으로 확장하며, 고대역폭·고지연(BDP가 큰) 구간에서는 윈도우가 작으면 대역폭을 못 채우므로 버퍼 튜닝이 성능 포인트입니다.

관련 개념 rwnd, 제로 윈도우, 윈도우 스케일, BDP와 버퍼 사이징, 실리 윈도우 신드롬

Q6. TCP 혼잡 제어(슬로 스타트, 혼잡 회피, 빠른 재전송/회복)를 설명하세요. ★★★

답안 혼잡 제어는 **네트워크의 수용 능력**을 초과하지 않도록 조절하는 것으로, 송신자가 혼잡 윈도우(**cwnd**)를 유지하며 실제 전송량은 min(cwnd, rwnd)입니다. ① **슬로 스타트**: cwnd를 1 MSS부터 ACK마다 2배씩(RTT당) 지수 증가시켜 가용 대역을 빠르게 탐색합니다. ② 임계값(ssthresh) 도달 후 **혼잡 회피**: RTT당 1 MSS씩 선형 증가(AIMD). ③ **빠른 재전송**: 중복 ACK 3개를 받으면 타임아웃 전에 즉시 재전송. ④ **빠른 회복**: 이때 cwnd를 절반으로 줄이고 혼잡 회피로 진행(타임아웃 시에는 cwnd=1로 리셋). 손실을 혼잡 신호로 보는 전통 방식(Reno/CUBIC)과 달리, 구글의 **BBR**은 대역폭·RTT를 직접 측정해 무선처럼 손실이 혼잡이 아닌 환경에서 성능이 좋습니다. 리눅스 기본은 CUBIC입니다.

관련 개념 cwnd/ssthresh, AIMD, 3 중복 ACK, CUBIC vs BBR, 버퍼블로트

Q7. TCP 재전송은 어떤 조건에서 발생하나요? ★★

답안 ① **RTO(재전송 타임아웃)**: 보낸 세그먼트의 ACK가 RTO 내에 안 오면 재전송합니다. RTO는 측정한 RTT의 평균과 편차로 동적 계산되며(Jacobson 알고리즘), 재전송할 때마다 2배씩 늘어납니다(지수 백오프). ② **빠른 재전송**: 수신자가 결번을 발견하면 같은 ACK를 반복 전송하고, 송신자가 **중복 ACK 3개**를 받으면 타임아웃을 기다리지 않고 재전송합니다. **SACK** 옵션은 어느 블록이 도착했는지 정확히 알려줘 불필요한 재전송을 줄입니다. 운영 관점에서 재전송률(retrans)은 네트워크 품질의 핵심 지표로, ss -i나 netstat -s, 모니터링 대시보드로 추적합니다.

관련 개념 RTO 계산, 지수 백오프, SACK, 재전송률 모니터링(ss -i)

Q8. UDP 기반인 QUIC(HTTP/3)이 등장한 이유는 무엇인가요? ★★★

답안 TCP의 구조적 한계 때문입니다. ① **HOL(Head-of-Line) 블로킹**: HTTP/2가 한 TCP 연결에 스트림을 다중화해도, TCP는 바이트 순서를 보장하므로 패킷 하나가 유실되면 모든 스트림이 함께 멈춥니다. QUIC은 스트림별 독립 전달로 이를 해결합니다. ② **핸드셰이크 지연**: TCP+TLS는 2~3 RTT가 필요하지만 QUIC은 전송과 암호화 핸드셰이크를 통합해 1 RTT, 재연결 시 0 RTT입니다. ③ **연결 마이그레이션**: TCP는 4-튜플이 바뀌면(와이파이→LTE) 연결이 끊기지만 QUIC은 연결 ID로 유지합니다. ④ TCP는 커널·중간 장비에 박혀 개선이 어려운 반면(ossification), QUIC은 사용자 공간 구현이라 빠르게 진화합니다. TLS 1.3이 내장되어 항상 암호화됩니다.

관련 개념 HOL 블로킹, 0-RTT, 연결 ID, 프로토콜 경직화, HTTP/3

Q9. 포트 번호의 범위와 역할, 소켓의 식별 방법을 설명하세요. ★★★

답안 포트는 16비트(0~65535)로 호스트 내 **프로세스(서비스)를 구분**합니다. 0~1023은 well-known(HTTP 80, HTTPS 443, SSH 22, DNS 53 등, 바인드에 루트 권한 필요), 1024~49151은 등록 포트, 49152~65535는 임시(ephemeral) 포트로 클라이언트가 사용합니다. TCP 연결은 **(출발 IP, 출발 포트, 목적 IP, 목적 포트) 4-튜플**로 유일하게 식별되므로, 서버는 80 포트 하나로도 수만 클라이언트와 동시 연결이 가능합니다. "서버 포트는 하나인데 어떻게 여러 연결을 받나"는 이 4-튜플 개념을 확인하는 최빈출 질문입니다.

관련 개념 well-known 포트, 임시 포트 범위(ip_local_port_range), 4-튜플, SO_REUSEPORT

Q10. 백로그 큐와 accept의 관계, 그리고 대량 접속 시 생기는 문제를 설명하세요. ★★

답안 서버 커널은 두 큐를 유지합니다. **SYN 큐**(핸드셰이크 진행 중)와 **accept 큐**(핸드셰이크 완료, 앱의 accept() 대기). listen(backlog)의 backlog와 somaxconn이 accept 큐 크기를 정합니다. 앱이 accept를 빨리 못 하면(스레드 부족, GC 멈춤) accept 큐가 넘쳐 새 연결이 드롭되거나 SYN+ACK 재전송이 발생하고, 클라이언트는 간헐적 연결 타임아웃을 겪습니다. SYN 큐를 겨냥한 SYN 플러딩은 SYN 쿠키로 방어합니다. ss -lnt의 Recv-Q/Send-Q로 큐 상태를 확인하는 것이 실무 진단 포인트입니다.

관련 개념 SYN 큐/accept 큐, somaxconn, SYN 쿠키, ss -lnt Recv-Q

Q11. keep-alive(TCP와 HTTP)는 각각 무엇을 하나요? ★★

답안 TCP keepalive는 유휴 연결이 살아 있는지 커널이 주기적 프로브로 확인하는 기능입니다(기본 2시간 후 시작, 튜닝 가능). 상대가 죽었거나 중간 장비(NAT/방화벽)가 세션을 지웠으면 연결을 정리합니다. NAT 테이블은 유휴 세션을 수 분 만에 지우므로, 장수 연결(DB, gRPC)은 keepalive 간격을 NAT 타임아웃보다 짧게 설정해야 "한참 놀다가 첫 쿼리만 실패"하는 장애를 예방합니다. **HTTP keep-alive**는 별개 개념으로, 요청마다 TCP 연결을 새로 맺지 않고 **하나의 연결을 재사용**하는 것입니다(HTTP/1.1 기본). 핸드셰이크·슬로 스타트 비용을 절약합니다.

관련 개념 NAT 세션 타임아웃, tcp_keepalive_time, 커넥션 풀, HTTP/1.1 persistent connection

---

Q12. 대역폭이 충분한데 단일 TCP 연결의 전송 속도가 안 나오는 이유는? ★★

답안 후보는 ① **BDP 대비 작은 윈도우**: 처리량 ≤ 윈도우/RTT이므로, RTT 100ms에 윈도우 64KB면 최대 5Mbps뿐입니다. 윈도우 스케일과 소켓 버퍼(tcp_rmem/wmem) 확인. ② **RTT 자체가 큼**: 슬로 스타트·혼잡 회피의 증가 속도가 RTT에 비례해 장거리 구간은 느리게 가속합니다. ③ **패킷 손실**: 1%의 손실도 손실 기반 혼잡 제어에서는 치명적으로 속도를 깎습니다(BBR로 개선 가능). ④ 중간 장비의 대역 제한·셰이핑. ⑤ 애플리케이션이 데이터를 공급 못 함(디스크 병목). 진단은 iperf3로 순수 네트워크 성능 분리 → ss -i로 cwnd·재전송 확인 → 손실 구간을 mtr로 추적하는 순서가 정석입니다.

관련 개념 처리량=윈도우/RTT, BDP, 손실과 CUBIC 성능 곡선, iperf3/ss -i/mtr

Q13. netstat/ss에서 CLOSE_WAIT가 계속 쌓이면 무엇이 문제인가요? ★★

답안 CLOSE_WAIT는 **상대가 FIN을 보냈는데 우리 쪽 애플리케이션이 아직 close()를 호출하지 않은** 상태입니다. 이 상태가 쌓인다는 것은 네트워크 문제가 아니라 **애플리케이션 버그**입니다. 커넥션·응답 객체를 닫지 않는 리소스 누수, 예외 경로에서 close 누락, 커넥션 풀 반환 실패 등이 원인입니다. 방치하면 파일 디스크립터 고갈("Too many open files")로 서비스가 멈춥니다. 대응은 코드에서 try-with-resources/defer 등으로 닫음을 보장하고, fd 사용량 모니터링을 거는 것입니다. TIME_WAIT(정상적 종료 흔적)와 성격이 완전히 다르다는 점을 구분해 설명하는 것이 포인트입니다.

관련 개념 FIN 수신 후 상태, fd 누수, ulimit -n, TIME_WAIT와의 구분

Q14. Nagle 알고리즘과 TCP_NODELAY는 무엇인가요? ★

답안 Nagle 알고리즘은 작은 데이터를 즉시 보내지 않고, 이전에 보낸 데이터의 ACK가 올 때까지 모아서 보내 작은 패킷 범람(예: 1바이트 페이로드에 40바이트 헤더)을 막는 기법입니다. 처리량에는 유리하지만 지연에 민감한 통신에는 해롭고, 특히 **지연 ACK와 결합하면** 상호 대기로 수십~수백 ms 지연이 생깁니다. 그래서 실시간성이 중요한 애플리케이션(게임, 금융, RPC)은 **TCP_NODELAY** 소켓 옵션으로 Nagle을 끕니다. 현대 프레임워크(gRPC, Redis 클라이언트 등) 상당수가 기본으로 끕니다.

관련 개념 작은 패킷 오버헤드, 지연 ACK와의 상호작용, TCP_NODELAY, writev/버퍼링 대안

Q15. 소켓 프로그래밍 관점에서 blocking, non-blocking, I/O 멀티플렉싱을 설명하세요. ★★

답안 블로킹 소켓은 read/accept가 데이터·연결이 올 때까지 스레드를 멈추므로, 동시 처리를 하려면 연결당 스레드가 필요해 수천 연결에서 한계가 옵니다. **논블로킹 소켓**은 즉시 반환(EAGAIN)하지만 폴링 낭비가 생깁니다. **I/O 멀티플렉싱**은 여러 소켓을 한꺼번에 감시해 준비된 것만 처리하는 방식으로, select(fd 1024 제한, 매번 전체 스캔) → poll(개수 제한 해소) → **epoll**(리눅스, 준비된 fd만 반환, O(1)) 순으로 발전했습니다. nginx, Redis, Node.js가 epoll 기반 이벤트 루프로 C10K 문제를 해결한 대표 사례입니다. 최신 io_uring은 완료 기반 비동기로 시스템 콜 자체를 줄입니다.

관련 개념 C10K, select/poll/epoll 차이, 에지/레벨 트리거, io_uring, 이벤트 루프

Q16. TCP 연결 상태 전이도에서 주요 상태를 순서대로 설명하세요. ★★

답안 클라이언트: CLOSED → SYN_SENT(SYN 전송) → ESTABLISHED(SYN+ACK 수신, ACK 전송) → FIN_WAIT_1(close, FIN 전송) → FIN_WAIT_2(ACK 수신) → TIME_WAIT(상대 FIN에 ACK) → 2MSL 후 CLOSED. **서버**: CLOSED → LISTEN → SYN_RECV(SYN 수신) → ESTABLISHED(ACK 수신) → CLOSE_WAIT(FIN 수신) → LAST_ACK(FIN 전송) → CLOSED(ACK 수신). 각 상태가 무엇을 기다리는 상태인지(FIN_WAIT_2=상대 FIN 대기, LAST_ACK=내 FIN의 ACK 대기)로 이해하면 암기가 아니라 논리로 답할 수 있고, ss로 본 상태 분포에서 장애를 읽어내는 실무 능력으로 연결됩니다.

관련 개념 상태 전이도, 동시 열기/닫기, ss -tan state 필터, 상태별 장애 시그니처
