CS 4251/6250 — Quiz 2 Cheat Sheet

Lectures 9–18 · HW2 · cumulative

Quiz 2: covers L9–L18 (PCC → QUIC → TE → MPLS/SDN → Meta → Google → Microsoft → Verification). **BGP / Unit 3 is OUT. 8 questions**, ≈ 90%+ new material; the **only** Part 1 content is part of the PCC question (§4, PCC vs Quiz 1 table). Mostly select-all MCQs written like **confident chatbot statements** (find the wrong word); **1–2 numericals**, variations on HW2. **It is long**, with less time to flip than Quiz 1: use the index, don't browse. Select-all scoring on Quiz 1 was **+1 / –1 per mark, floor 0**: unsure ⇒ don't mark. ★ **grey rows** = must-know; ▸ **use:** = which question it answers. **Start any question at the question-shape index (top of §2)**; equation questions → §1.2; essay/short answers → §7.2. **Bring the Exam 1 sheet too** for L1–8 (§9 here points into it).

1 All formulas and rules 1
1.1 Units 3
1.2 Read the equation (what it SAYS) 3
2 Recipes (do the numericals step by step) . 3
2.1 Bandwidth functions (HW Q4.1 / lecture C = 4, 8, 12) 4
2.2 B4 waterfill (lecture example) 5
2.3 HPRR (HW Q5) 5
2.4 SWAN congestion-free plan 5
2.5 TeaVaR 5
2.6 Verification: reachability layers 5
2.7 QUIC scheduling (HW Q2.2 method) . . . 6
2.8 Quick methods for every other question type 6
3 Concepts by lecture 6
3.1 PCC (L9) 6
3.2 QUIC (L10) 7
3.3 TE foundations (L11) 7
3.4 MPLS & SDN (L12) 7
3.5 Meta EBB (L12–14) 8
3.6 Google (L14–16) 8
3.7 Microsoft (L16–17) 8
3.8 Verification (L18) 8
4 Comparisons 9
5 HW2 answers (condensed) 9
6 Lecture problem bank (numbers to recognize) 10
7 Traps, numbers, answer patterns, glossary 10
7.1 Numbers to know 10
7.2 Answer patterns + one-liners 11
7.3 Glossary (acronyms) 11
8 True / false bank (every MCQ statement from the problems) 11
9 Part 1 bridges (bring the Exam 1 sheet for L1–8) 14

FLAGGED BY THE PROFESSOR / HIGH-YIELD

- **Meta's heuristic (HPRR)**: *"going to be part of the covered material"* (like Vegas) → §1 Meta panel, §2.3, §5 Q5
- **HW2 problem types** = the numerical template → §5, §2, index at top of §2
- *"Low-level details"*: stack depth 3, Binding SID, version bit, sFlow, NIB pub-sub, SWAN max(old,new), TeaVaR loss → §1, §1.2, §3
- **Bandwidth functions + B4 waterfill** (HW Q4) → §1 Google panel, §2.1–2.2
- **PCC vs model-based** (the only CC after Quiz 1) → §1 PCC panel, §1.2, §3.1
- **Verification** (L18, last lecture, in scope) → §1 panel, §2.6, §3.8
- TA's spoken TeaVaR numbers were wrong → **compute from probabilities** (§2.5)

1 All formulas and rules

PCC / performance-oriented CC (L9)

★ **Allegro** u	$u(x) = T \cdot \mathrm{Sigmoid}(L - 0.05) - xL, \quad T = x(1-L)$ sigmoid keeps throughput only while loss < ≈5% · without it, $x(1-L)$ has a high-loss Nash equilibrium ▸ ***use:*** *evaluate Allegro utility at given rate and loss*
Trials (RCT)	send $r(1-\varepsilon)$ and $r(1+\varepsilon)$, each ≥ 1 RTT, wait ≥ 1 RTT for feedback; **2 pairs, random order** both pairs agree → move · disagree → **inconclusive**: stay at r, **raise ε**
States	STARTING (×2 while U rises) → DECISION (compare) → ADJUSTING (move) · U falls while adjusting → revert 1 step, back to DECISION
★ **Vivace** u	$u(x) = x^t - bx\dfrac{d\ \mathrm{RTT}}{dt} - cxL\ 0 < t < 1; b, c > 0$; slide: $x^{\alpha_1} - \alpha_2 x \cdot d\ \mathrm{RTT}/dt - \alpha_3 xL$ ▸ ***use:*** *RTT **gradient**, not magnitude: negative slope (queue draining) is rewarded*
★ **Gradient step**	$g \approx \dfrac{U_+ - U_-}{r_+ - r_-}, \ \Delta r = \theta g$ θ = confidence amplifier (grows if direction repeats); Δr bounded relative to r ▸ ***use:*** *big |g| ⇒ far from peak ⇒ big step; small |g| ⇒ small step*
Fluid RTT	$\dfrac{d\ \mathrm{RTT}}{dt} = \dfrac{x - C}{C}$ slope measures how far above C, independent of existing queue lecture: C = 100, x = 120 → slope 0.20; x = 110 → 0.10 (110 better, though avg RTT higher)
★ **PCC-β (HW)**	$U = g - \beta q$ g goodput (Mb/s), q queueing delay (ms); β = Mb/s you trade per ms indifferent at $\beta^* = \dfrac{\Delta g}{\Delta q}$ · larger β does **not** always pick lower delay ▸ ***use:*** *HW Q1.2: compute $U_{\mathrm{lo}}, U_{\mathrm{hi}}$ for each β*
Rate ↔ window	rate-based can cap inflight ≈ $2 \cdot \mathrm{rate} \cdot \mathrm{RTT}$; window-based paces ≈ $2 \cdot \mathrm{cwnd} \ / \ \mathrm{RTT}$

BBR after a route change (HW Q1.1)

★ **Spiral**	min-RTT filter (10 s) keeps $\mathrm{RTT}_{\mathrm{old}}$ ⇒ $\mathrm{cwnd} \approx 2 \cdot \mathrm{BW} \cdot \mathrm{RTT}_{\mathrm{old}}$ too small for $\mathrm{RTT}_{\mathrm{new}}$ ⇒ rate = $\mathrm{cwnd} \ / \mathrm{RTT}_{\mathrm{new}}$ falls ⇒ 10-round max-BW filter loses high samples ⇒ BW ↓ ⇒ cwnd ↓ … until min-RTT entry expires (≈ 10 s)
Fix (HW)	several RTT samples consistently > stored min ⇒ temporarily **drain** inflight, re-sample. Stays high at low load ⇒ propagation changed: reset min-RTT, recompute BDP, re-probe. Falls back ⇒ it was queueing: keep min. Imperfect: other flows' standing queue looks the same.

QUIC (L10, HW Q2)

★ **RTTs (fresh)**	**request sent** at: TCP+TLS1.3 = 2 RTT · QUIC = 1 RTT · 0-RTT = 0 **response at**: 3 · 2 · 1 RTT (0-RTT needs remembered state + server accepts early data; handshake still completes)
★ **Identities**	**stream ID + offset** = which bytes (same on resend) · **packet number** = which transmission (new, never reused) · TCP seq = byte position in the one stream (ambiguous ACK on resend)
Loss detect	packet threshold (higher pkt number acked; RFC 9002: 3) · time threshold (older pkt past ≈ 9/8·RTT) · PTO: send a **probe** when ACKs stall; **PTO alone doesn't declare loss**
Scope	per **connection**: congestion control, connection flow-control credit, crypto · per **stream**: ordering & delivery
Migration	connection = CID, not 4-tuple · new path: other issued CID (unlinkable) + PATH_CHALLENGE/RESPONSE · **re-learn** cwnd/RTT for new path
HOL timing	display time = arrival of the **last in-order byte the app needs** · TCP: everything after a gap waits · QUIC: only that stream waits

TE foundations (L11, HW Q3)

★ **min-MLU LP**	$\min \rho \quad \text{s.t.} \ \sum_p x_{kp} = d_k, \quad \sum_{k,p \ni e} x_{kp} \le \rho c_e, \quad x \ge 0$
★ **Parallel paths**	disjoint paths, one demand D: split $\propto$ **capacity**, $x_p = Dc_p / \sum c, \quad \mathrm{MLU}^* = D / \sum c$ 2 paths: $a/c_1 = (D-a)/c_2$ · 12 on (10, 5) → 8 / 4, MLU 0.8 ▸ ***use:*** *any "min MLU on parallel links" question*
ECMP	splits **flows** equally by hash (src/dst IP, ports, QoS) → ignores capacity · 12 on (10, 5) → 6/6 = 60% / **120%**

Cut limit	TE can't beat the cut: demand > cut capacity ⇒ admission / CC must cut demand (12 on 15 OK; 16 on 15 impossible)
Latency	mean latency $= \sum_p x_p \mathrm{lat}_p / D$
Staged swap	two flows swap paths, headroom h: move ≤ h of one, then ≤ h of the other, confirm each · A8/B8 on two 10s, h = 2 → 4 rounds

MPLS / SDN (L12)

★ Shim (32 b)	label **20** · Exp/TC **3** · S (bottom of stack) **1** · TTL **8** (slide prints 5 — it's 8)
LSP	ingress **push** (classify by FEC) → transit **swap** → (penultimate) **pop** → egress · labels **local** per router · stackable
Units	packets/flows → FEC → **traffic trunk** (FEC + bw, burst, priority, preemption, resilience, policing) → LSP
★ CSPF	① trunk request (A→F, 6 G, exclude RED) ② **prune** links w/o bandwidth or with excluded color ③ shortest path on rest → install · trunks one-by-one are **order-dependent**
★ Resilience	survive f failures of k parallel links: reserve $D/(k-f)$ per link (100 on 3×400, f = 1 → 50 each) · then **oversubscribe** (450 on 400)
Signaling	OSPF-TE / IS-IS-TE flood bw + reserved bw; ingress uses **RSVP-TE** to set up LSP (or central PCE/SDN)
Flow table	match (L2–L4, wildcards) · action (fwd / drop / modify / to controller) · priority · counters
SDN failure	switch → OpenFlow **port-status** → controller updates state → routing app callback → recompute → flow tables → install · 100s ms–s ⇒ hybrids

Meta EBB (L12–14, HW Q5)

MCF	commodity k = (src DC, dst DC, class) · $\sum_p x_{kp} = d_k$ · $\sum_{k,p \ni e} x_{kp}(+\text{burst margin}) \le U c_e$ · min U or min Σ cost·x
★ HPRR rule	reroute **one LSP at a time**; accept iff max util on **new route after** < max util on **current route before** $w_e = \exp[\alpha(u'_e/u^* - 1)]$, route cost = $\sum_e w_e$ (Dijkstra; current route is a candidate) HW: α = 66.4, u* = 0.855 ▸ ***use:*** *HW Q5.3 compare* W_{direct} *vs* $W_{\text{via}} = 2w$
HPRR limits	① LSP granularity (8 G as 4×2 G on 10/6 → 6/2, MLU 0.6 vs 0.5) ② stuck: only a **swap** helps ③ sum-of-costs penalizes hops (deliberate)
★ Stack depth	EBB hardware: **3 labels** → longer path needs a **Binding SID** (label = programmed continuation at a mid router) · dynamic SID per (site pair, class) → next-hop group = path bundle
★ Version bit	install v1 rules everywhere → wait for **all ACKs** → flip source to tag v1 → in-flight v0 pkts finish on v0 · 1 bit alternates 0/1
★ Hose	$\sum_j D_{ij} \le E_i, \quad \sum_i D_{ij} \le I_j$ (2n numbers vs pipe n(n–1)) · plan: TE feasible for every matrix in H and every failure in S
sFlow	ASIC samples 1-in-N headers + counters → collector → bytes per (DC pair, class) · estimate ≈ samples × N × avg size

Google (L14–16, HW Q4)

★ Band. function	$B_i(s)$: x = fair share s (common), y = bandwidth · **slope = weight/priority** · **flat = demand cap** allocate: raise s until $\sum_i B_i(s) = C$ on the bottleneck ▸ ***use:*** *HW Q4.1: for each C find s, read each* $B_i(s)$
Freeze	group stops when any link on its path is full or it hits its cap; others keep growing
★ Aggregate	parent $B_P(s) = \sum B_{\text{child}}(s)$ (**add heights at the same s**) · decompose: allocation → s via parent, then children at that s
★ B4 waterfill	each FG uses **cheapest path first**; when a link fills, the rate there stays and further growth **spills** to next path; freeze at cap / no path · then **quantize** splits (K units), fix overload greedily
BwE	host (HTB) ← job ← cluster ← global (MPFA): usage ↑, budgets ↓ · host limit caps TCP (if cwnd/RTT < limit, CC binds)
★ Stale budget	waste $= (C - \sum \text{actual sent rates}) \times \text{interval}$ HW: (12 – 2 – 6) G × 0.2 s = 0.8 Gb = 100 MB
dSDN data	41 largest B4 outages: 42% control-only + 10% control+data = **52%** control-related (32% data-only, 16% other)

Microsoft (L16–17)

★ SWAN update	for every link e and step i → i+1: $\sum_f \max\left(x_{f,e}^{(i)}, x_{f,e}^{(i+1)}\right) \le c_e$ · flow conservation each stage · maximize headroom; infeasible ⇒ more stages
★ Scratch s	at most $\lceil 1/s \rceil - 1$ steps (s = 10% → 9, 20% → 4) · lecture 20%: U 64/16 → 44/36 → 24/56 → 16/64
Rule slots	spare rule entry: add new tunnel **before** removing old · {1,2}→{3,4}: +3, shift→{2,3}, –1, +4, shift→{3,4}, –2
★ TeaVaR	$P(q) = \prod_\ell P(\ell \text{ state in } q)$ (independence; 2^n scenarios) flow loss in q $= 1 -$ $\text{deliverable on surviving tunnels} \;/\; \text{demand}$ (no rerouting) $L(x,q) = \max_{\text{flows}} \text{loss} \cdot V(x) = \text{VaR}_\beta$ = smallest v with $P(L \le v) \ge \beta$ $x^*, V^* = \arg\min V(x)$ · **guarantee** = allocation × $(1 - V^*)$, β of the time
k-failure	n disjoint links of c: guarantee $(n-k)c$ under any k failures (3×10: 30 / 20 / 10)
OneWAN	agg routers: full BGP (10^5 DC + 10^6 Internet), pick egress · backbone: labels only · steering: drop high-delay entries → max-flow weights · prune trunks < 500 Mb/s (2.5% traffic → 5× rows, 7× vars) · CLP/GLOP 5×

Verification (L18)

★ One hop	$N(u,v,h,e) \Leftrightarrow \text{Sel}(u,v,h,e) \wedge \text{Up}_{uv}(e) \wedge$ all three $\text{Permit}(u,v,h)$
ACL	$\text{hit}_i(h) \Leftrightarrow \text{match}_i(h) \wedge \bigwedge_{j<i} \neg\text{match}_j(h)$ · $\text{Permit}(h) \Leftrightarrow \bigvee_i (\text{hit}_i(h) \wedge \text{allow}_i)$
Path	$\text{Path}(\pi,h,e) \Leftrightarrow \bigwedge_{i=0}^{m-1} N(v_i, v_{i+1}, h, e)$
★ Reach layers	$R_0(v) \Leftrightarrow v = D$, $R_{k+1}(v) \Leftrightarrow R_k(v) \vee \bigvee_u (N(v,u,h,e) \wedge R_k(u))$ · $R(S,D) \Leftrightarrow R_{\text{diam}}(S)$
Requirements	reach: ∃ delivered path · isolation: ∀h ∈ H no delivery · waypoint: every delivered path visits F (+reach if must arrive) · loop-free: no prefix repeats a router
★ ≤ 1 failure	$f_\ell(e) \in \{0,1\}$, $K(e) = \sum_\ell f_\ell(e) \le 1$ · Boolean SAT: $\bigwedge_{\ell<m} (\text{Up}_\ell \vee \text{Up}_m) = \binom{n}{2}$ clauses
★ Query	$\exists e,h,x : A(e,h) \wedge M(C,e,x,h) \wedge \neg P(e,x,h)$ · **SAT** = counterexample · **UNSAT** = holds **in the model only**

1.1 Units

Bits/bytes	1 B = 8 b · 1 MB = 10^6 B (decimal) · Gb/s × s = Gb; ÷ 8 = GB · 12 Gb/s × 200 ms = 2.4 Gb = 300 MB
Delay	queueing delay = queued bits / link rate · one-way = RTT/2 · BDP = rate × RTT

1.2 Read the equation (what it SAYS)

The professor's format (Quiz 1): *"I give you an equation, you say what it describes."* For each: what the terms mean, which way they push, the limit cases.

$$u = x(1-L)\cdot S(L) - xL$$

Allegro utility. Reward = delivered throughput, but only while the sigmoid S is ≈ 1 (loss well below 5%); $-xL$ charges for lost packets. Limits: $L \to 0 \Rightarrow u \approx x$ (more rate always better); $L > 5\% \Rightarrow S \to 0$, score collapses. No delay term ⇒ fills buffers. Without S: high-loss Nash equilibrium.

$$u = x^t - bx\frac{d\ \mathrm{RTT}}{dt} - cxL$$

Vivace utility. x^t, $0 < t < 1$: concave reward (diminishing returns ⇒ competing flows converge to fair). Middle term: penalty for **growing** the queue (RTT rising), **bonus** when draining ($d\ \mathrm{RTT}/dt < 0$); slope, not RTT value, so an inherited queue isn't blamed on you. cxL: loss penalty.

$$\frac{d\ \mathrm{RTT}}{dt} = \frac{x - C}{C}$$

RTT gradient in a fluid queue. Queue grows at $x - C$ bits/s; divide by drain rate C ⇒ delay grows per second. Measures how far above capacity you are **now**. $x = C \Rightarrow 0$; $x < C \Rightarrow$ negative (draining).

$$g = \frac{U_+ - U_-}{r_+ - r_-}, \quad \Delta r = \theta g$$

Vivace step. g = score gained per extra Mb/s near r. Sign = direction, size = distance from the peak (flat near optimum ⇒ tiny step). θ = confidence amplifier (grows while direction repeats, resets on flip); Δr capped relative to r (noise). Allegro instead: direction from trials, size from a fixed schedule.

$$U = g - \beta q$$

PCC delay preference. β = Mb/s of goodput you'd give up for 1 ms less queueing. Picks high rate iff $\Delta g > \beta \Delta q$; break-even $\beta^* = \Delta g/\Delta q$. Larger β does **not** always pick the lower-delay rate.

$$\mathrm{cwnd} \approx 2 \cdot \mathrm{BtlBw} \cdot \mathrm{RTT}_{\mathrm{old}} < \mathrm{BDP}_{\mathrm{new}}$$

BBR after a route change. Min-RTT filter (10 s) remembers the old short RTT ⇒ cwnd too small for the new path ⇒ delivery falls ⇒ max-BW filter (10 rounds) forgets high samples ⇒ cwnd shrinks again. Ends when min-RTT entry expires.

$$\min \rho \quad \text{s.t.} \quad \sum_p x_{kp} = d_k, \quad \sum_{k,p \ni e} x_{kp} \le \rho c_e$$

Min-MLU TE (MCF). ρ = utilization of the busiest link, pushed down. Line 1: carry every demand. Line 2: everything crossing link e fits in ρ × capacity — **couples the demands**. Parallel paths ⇒ equal utilizations ⇒ split ∝ capacity, $\rho^* = D/\sum c$. Swap objective to $\min \sum \mathrm{cost} \cdot x$ ⇒ min latency (Meta gold).

$$w_e = \exp[\alpha(u'_e/u^* - 1)]$$

HPRR link cost. u'_e = utilization if the LSP is placed on e. $u' < u^* \Rightarrow$ cost ≈ 0 (free); $u' = u^* \Rightarrow 1$; above ⇒ explodes (steer away from hot links). Route cost = **sum** ⇒ multi-hop routes pay several times (hop bias, deliberate). Accept only if new max util < current max util.

$$\sum_j D_{ij} \le E_i, \quad \sum_i D_{ij} \le I_j$$

Hose model. Each site's total send (E) and receive (I) budget, not per-pair forecasts (2n numbers vs n(n−1)). Any matrix within the budgets must be routable under every failure in S. Max $i \to j$ = min(E_i, I_j).

$$\sum_i B_i(s) = C$$

Bandwidth-function allocation. s = common fair share raised for everyone; each group gets $B_i(s)$. Slope = weight/priority, flat = demand cap; a group also freezes when a link on its path fills. Slope that changes ⇒ policy no fixed weight ratio can express.

$$B_P(s) = \sum_{\text{children}} B_c(s)$$

Aggregation. Add heights at the same s (never add s). Each bend = a child capping. Decompose: parent allocation → s → read each child.

$$\sum_f \max\left(x_{f,e}^{(i)}, x_{f,e}^{(i+1)}\right) \le c_e$$

SWAN congestion-free step. Switches update asynchronously, so each flow may be at its old or new rate during the step; the worst case is the larger of each. Holds ∀ links, ∀ steps ⇒ no transient overload. Scratch s free per link ⇒ $\le \lceil 1/s \rceil - 1$ steps (s = 20% ⇒ 4); bigger s ⇒ fewer steps, less usable capacity.

$$P(q) = \prod_\ell P(\ell \text{ in its state in } q), \quad L(x,q) = \max_{\text{flows}} \text{loss}$$

TeaVaR. Independent failures ⇒ product. Scenario loss = the **worst flow's** unmet fraction (no rerouting inside q). $V(x) = \mathrm{VaR}_\beta$ = smallest loss v with $P(L \le v) \ge \beta$. Minimize V; promise allocation × $(1 - V^*)$, holds β of the time.

$$N(u,v) \Leftrightarrow \mathrm{Sel} \wedge \mathrm{Up} \wedge \mathrm{Permit}$$

One usable hop (verification): routing picks v, link up, ACL permits — all three. ACL: $\mathrm{hit}_i = \mathrm{match}_i \wedge$ no earlier match (first match wins).

$$R_{k+1}(v) \Leftrightarrow R_k(v) \vee \bigvee_u (N(v,u) \wedge R_k(u))$$

Reachability layers. Grow backward from D one hop at a time; S reachable iff it ever joins. k up to the longest loop-free path.

$$\exists e, h, x : A(e,h) \wedge M \wedge \neg P$$

Verification query. Allowed failure ∧ network behaviour ∧ requirement broken. SAT = concrete counterexample (failure, packet, forwarding). UNSAT = holds **only within the model**. "≤ 1 down" in pure SAT = every pair $(\mathrm{Up}_a \vee \mathrm{Up}_b)$.

$$c \Leftrightarrow a \wedge \mathrm{Up} \wedge \mathrm{Accept}$$

Minesweeper import. S has a usable route from B iff B advertised it, the link is up, and S's import policy accepts it. Stable state: no router would change its choice.

2 Recipes (do the numericals step by step)

Question-shape index. *Match what the question **gives** and **asks**, then go to the method. Numbers in brackets = practice problem in the notes.*

If the question gives… and asks…	Go to
loss/RTT observation → "what should the sender do?"	§3.1 four scenarios; trap: same signal, four causes [1.1]
Allegro trial utilities (pairs) → move?	§1 PCC panel, Trials row (agree → move; disagree → stay, ε↑) [1.2]
rate + loss → Allegro utility	§1.2 Allegro; worked 96.2 / 66.2 / 19.3 [1.3]
RTT trace over time → better rate	§1.2 RTT gradient: compare **slopes** [1.4]
two trial scores + θ → new rate	§1.2 Vivace step [1.5]
(goodput, delay) pairs + β → choice	§1.2 PCC-β; break-even $\Delta g/\Delta q$ [1.6]
route change, old/new RTT → BBR behaviour	§1 BBR panel, §1.2 [1.7]
trial length / learned policy MCQ	§3.1 [1.8, 1.9]
RTT → when does the response arrive?	§1 QUIC panel: 3 / 2 / 1 RTT; request one RTT earlier [2.1]
lost packet + objects → display times (TCP vs QUIC)	§2.7 HOL line, §2.8 QUIC delivery [2.2, 2.3]
buffered streams + cwnd → scheduler table	§2.7 [2.4]
acked packet numbers → which are lost?	§2.8 QUIC loss [2.6]
paths + capacities + demand → split, MLU	§1 TE panel (parallel paths) [3.1, 3.2]
link counter → can you infer demand?	§3.3 five problems [3.3]
two aggregates to swap + headroom → plan	§2.8 Staged swap [3.4]
hourly loads + batch job → schedule	§2.8 Defer batch [3.5]
classes (RPC/bulk) + ECMP → policy violation, MLU	§2.8 Classes vs ECMP; §5 Q3 [3.6, M.3]
MPLS fields / labels / overhead	§1 MPLS panel, §2.8 [4.1, 4.2]
graph with (cost, bw, color) + trunks → path	§2.8 CSPF [4.3]

k parallel links, survive f failures → reservation	§1 MPLS Resilience: $D/(k-f)$ [4.4]
weighted graph → least-cost path, force a path	§2.8 Link weights [4.5]
flow table with priorities → action	§2.8 Flow table [4.6]
two paths with latency → min-latency vs min-MLU	§6 Meta short/long; §1.2 MCF [5.1, 5.2]
LSPs on links → does HPRR move? optimal?	§2.3 HPRR [5.3, M.5]
background loads + α, u* → HPRR route cost	§2.3 HW Q5.3, §1.2 HPRR cost [5.4]
update steps order → black hole?	§1 Meta Version bit [5.5]
samples, N, packet size → demand	§2.8 sFlow [5.6]
site egress/ingress budgets → max pair demand, cut	§2.8 Hose bounds [5.7]
hop count + stack depth → Binding SIDs	§2.8 Stack depth [5.8]
statement → intent / policy / config?	§4 Level table, §3.6 [6.1]
demands on one link + policy → rates	§2.8 Sharing 1 link [6.2]
bandwidth functions + C → allocation	§2.1 [6.3, 6.4, M.6]
children functions → aggregate, decompose	§2.8 Aggregate, §1.2 [6.5]
network + functions, a link fills → freeze	§6 Freeze row [6.6]
network + flow groups + path costs → waterfill	§2.2 [6.7]
continuous splits + K units → overload?	§2.2 Quantization [6.8]
budgets, new demands, update delay → waste	§2.8 BwE waste; §5 Q4.3 [6.9]
old/new per-link rates → congestion-free plan	§2.4, §1.2 SWAN [7.1, 7.2, M.7]
tunnel sets + rule slots → steps, LPs	§2.8 Tunnel swap [7.3]
n links, k failures → guarantee	§2.8 k-failure [7.4]
failure probabilities + allocation → VaR, guarantee	§2.5 [7.5, 7.6]
entry routers + capacities → steering weights	§2.8 Steering [7.7]
routing + ACLs + failures → SAT/UNSAT, layers	§2.6 [8.1, 8.2, M.8]
ordered ACL → hit/permit formula	§1 Verification ACL row [8.3]
write isolation / waypoint / loop / ≤ 1 failure	§2.8 Requirements, Counting [8.4, 8.5]
NUM, Copa, Vegas, bufferbloat, BBR, CoDel, fluid, ECN	Exam 1 sheet; §9 bridges; parking lot in §2.8 Part 1 row [9.1–9.10]

2.1 Bandwidth functions (HW Q4.1 / lecture C = 4, 8, 12)

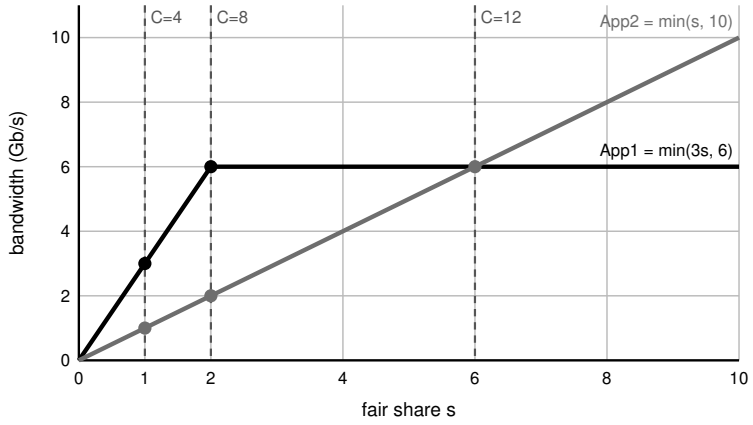
Find the allocation on one bottleneck L15 · HW Q4

Recognize: given $B_i(s)$ and C
Use: $\sum B_i(s) = C$, solve piece by piece

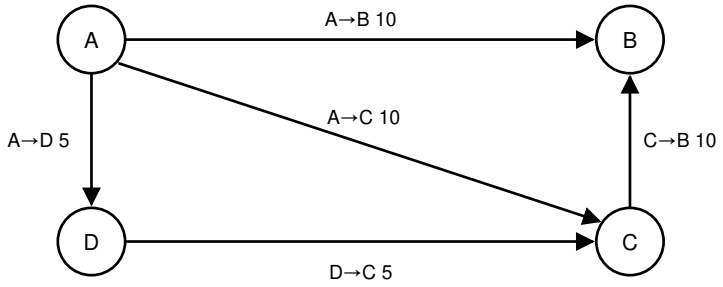
① Write each B_i piecewise. ② Guess the segment for s (try increasing s; at each breakpoint compute $\sum B_i$). ③ On the right segment solve the linear equation. ④ If $C \geq$ sum of all caps, everyone is capped; rest unused.

C	s	App1 = min(3s, 6), App2 = s	why
4	1	3, 1	3s + s = 4
8	2	6, 2	App1 caps at s = 2
12	6	6, 6	6 + s = 12

HW: $B_A = \min(2s, 8)$, $B_B = \min(s, 8)$ → C = 9: s = 3 → (6, 3) · C = 12: s = 4 → (8, 4) · C = 15: s = 7 → (8, 7). Non-work-conserving 2:1 reservations at 15: (8, 5), 2 unused.



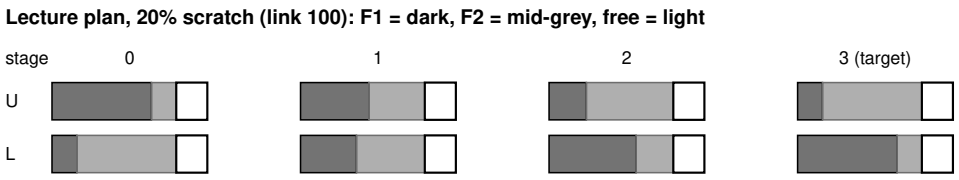
B(s) for App1 = min(3s, 6) and App2 = s; vertical lines at the s chosen for C = 4, 8, 12



FG1: A→B, cap 20
paths AB(1) · ACB(2) · ADCB(12)

FG2: A→C, slope ½, cap 10
paths AC(1) · ADC(11)

Directed links; cost 1 each except A–D (10). FG1 (A→B) = 11s up to s = 1.5 (16.5), then slope 1 to cap 20 at s = 5. FG2 (A→C) = s/2, cap 10.



Links U and L (100 each), 20% scratch. Dark = F1, mid-grey = F2, light = free. U: 64/16 → 44/36 → 24/56 → 16/64; L mirrors.

2.2 B4 waterfill (lecture example)

event → consequence	s	FG1	FG2
A–B fills → FG1 spills to ACB	10/11	10 (AB)	0.45 (AC)
A–C fills → both move to D routes	10/3	18.33 = 10 AB + 8.33 ACB	1.67 (AC)
FG1 hits cap → frozen	5	20 (+1.67 ADCB)	2.5 (+0.83 ADC)
A–D, D–C fill → FG2 frozen by network (cap 10)	10	20	5 = 1.67 AC + 3.33 ADC

At "A–C fills": solve (FG1(s) – 10) + FG2(s) = 10. Method: raise s; find which link fills first; keep rates on filled paths, spill growth to the next-cheapest path with room; freeze groups at cap or with no path; repeat. (Slide check: at s = 2, A–C carries 7 + 1 = 8; FG1 = 17.)

Quantization: splits must be multiples of 1/K. Rounding each FG alone can overload a shared link: FG1 (½, 5/12, 1/12) → (½, ½, 0), FG2 (⅓, ⅔) → (½, ½) ⇒ A–C = 20·½ + 5·½ = **12.5 > 10**. Fix greedily. Final: FG1 = 10 AB + 10 ACB, FG2 = 5 on ADC.

2.3 HPRR (HW Q5)

One HPRR step — L13 · HW Q5

Recognize: LSPs on links, "does HPRR reroute?"
Use: accept iff new-route max util (after) < current-route max util (before)

① Current MLU of the LSP's route (before). ② For each candidate route (incl. current), compute utilization **after** moving the LSP there. ③ Costs $w_e = \exp[\alpha(u'_e/u^* - 1)]$, route cost = sum (more hops ⇒ more cost). ④ Pick min-cost route; accept only if its max util < current max util. ⑤ Repeat for each LSP.

HW Q5.2	L: 4 fixed + A (6); R: B (4). Current max((4+6)/10, 4/10) = 1.0. Move A → R: R = 10 → 1.0 (not <). Move B → L: L = 14 → 1.4. **Stuck.** Swap (A→R, B→L): L = 8, R = 6 → 0.8 optimal ⇒ consider pairs.
HW Q5.3	direct: u = 0.9, $w = \exp[66.4(0.9/0.855 - 1)] = 32.9$ · via M: two links at 0.895, $W = 2\exp[66.4(0.895/0.855 - 1)] = 44.7$ ⇒ HPRR picks direct (MLU 0.9) though via M gives 0.895.
HW Q5.1	8 G on L (10) / R (6): continuous 5/3 → MLU 0.5 · 4×2 G LSPs: 6/2 → 0.6 · 8×1 G: 5/3 → 0.5 (more state)

2.4 SWAN congestion-free plan

Check a plan — L16–17

Recognize: stages of per-link rates
Use: $\sum_f \max(\text{old}, \text{new}) \le c$ per link per step

For every link and consecutive pair of stages, add the **larger** of each flow's two values. 0→1 on U: max(64,44) + max(16,36) = 100 ✓ · 1→2: 44 + 56 = 100 ✓ · 2→3: 24 + 64 = 88 ✓. Direct 80/20 → 20/80 swap: F2 first ⇒ U = 80 + 80 = **160 ✗**. Whiteboard LP: vars $a_u^i, a_l^i, b_u^i, b_l^i$; $a_u + a_l = 80, b_u + b_l = 80; \max(a_u^i, a_u^{i+1}) + \max(b_u^i, b_u^{i+1}) + h \le 100;$ max h, h ≥ 0.

2.5 TeaVaR

Loss distribution and VaR — L17

Recognize: link failure probs + allocation, "guarantee β of the time"
Use: independence, max over flows, β-percentile

① List scenarios q (2^n), $P(q) = \prod$. ② Per scenario, per flow: loss = traffic on failed links / demand. ③ $L(x, q)$ = max over flows. ④ Sort losses, accumulate P until ≥ β ⇒ VaR. ⑤ Guarantee = demand × (1 – VaR).

event (slide probs)	P	meaning
none fail	$0.999^2 \cdot 0.9 \approx 89.8\%$	30 available
exactly one	≈ 10.2%	20 left (any 2 survivors)
≥ 2 fail	$\approx 2 \cdot 10^{-3} \cdot 10^{-1}$ = **0.02%**	≤ 10 left
top or bottom fails	$1 - 0.999^2 \approx$ **0.2%**	10+10 on top/bottom fails

⇒ 20 Gb/s spread over 3 links: **99.98%**. 10+10 on top/bottom only: **99.8%** (TA said 99.98% — wrong).

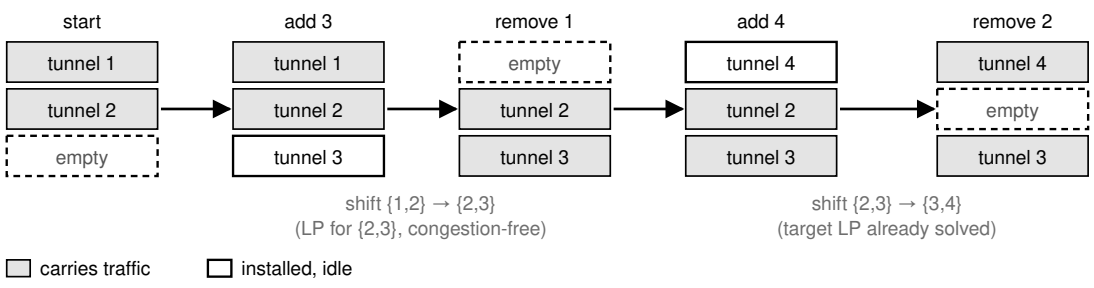
2.6 Verification: reachability layers

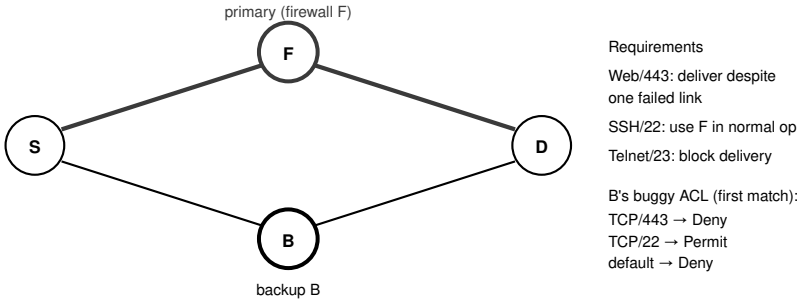
Does web survive one failure? — L18

Recognize: SAT/UNSAT question on the running example
Use: R_k layers backward from D

S–F down, buggy ACL at B: $R_0 = \{D\}$, $R_1 = \{F, D\}$ (B→D denied), $R_2 = \{F, D\}$ ⇒ S never enters ⇒ **SAT** (counterexample: S–F down, web pkt; $\text{Up}_{\text{SF}} = 0$, others 1, Selected(S,B) = 1, N(B,D) = 0).

failed link	web before fix	after B permits web
none	delivered	delivered
S–F	**dropped**	delivered
F–D	**dropped**	delivered
S–B	delivered	delivered
B–D	delivered	delivered





Running example: primary path via firewall F, backup via B. B's ACL is first-match: TCP/443 deny, TCP/22 permit, default deny.

After fix: $M_{C'} \wedge K(e) \leq 1 \wedge \neg R_2(S)$ is **UNSAT**. 5 cases (none + 4 single) = one symbolic query.

2.7 QUIC scheduling (HW Q2.2 method)

Display time under a scheduler L10 · HW Q2

Recognize: cwnd limited, all data buffered
Use: table: t | ACKs | cwnd | sent; arrival = t + one-way

t (ms)	ACKs	cwnd	B-first sends
0	—	1	B1
50	B1	2	B2, A1
100	B2, A1	3	A2, A3

RTT 50, one-way 25: B done at **75 ms** (B-first) vs **125 ms** (round robin: A1 | B1,A2 | B2,A3). A = 125 either way. Priority can **starve** A if B keeps getting data. HOL (Q2.1): TCP delivers B only when lost A1 is resent (t = 60) · QUIC delivers B at 23 ms.

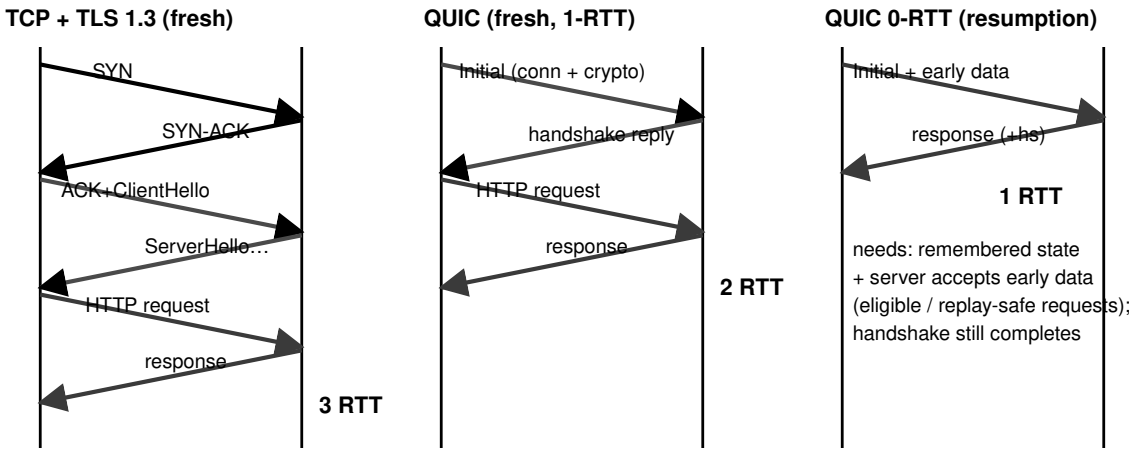
2.8 Quick methods for every other question type

Question	Method (problem in brackets)
MPLS overhead	shim = 4 B per label; k labels = 4k B; % = 4k / packet size (3 labels on 1500 B = 12 B = 0.8%) · label space 2^{20} = 1,048,576 per router (local) [4.1]
Label trace	follow each router's table: push at ingress, swap per hop (in → out label), pop at penultimate/egress; labels local, so equal numbers at different routers are unrelated [4.2]
CSPF	① delete links with available bw < trunk or with excluded color ② shortest path (sum of costs) on the rest ③ reserve bw (subtract) before the next trunk; try both orders if asked [4.3]
Link weights	list every u→z path with its cost sum; to force a path, raise weights on competitors' links and re-check **all** paths; side effect: every flow using those links moves; can't split one destination or treat classes differently [4.5]
Flow table	among rules whose match fits the packet, the **highest priority** wins; no match → default/table-miss [4.6]
Staged swap	move ≤ headroom per step, alternate flows; moves per flow = ⌈amount / headroom⌉; check each intermediate state ≤ capacity [3.4]
Defer batch	compute utilization each hour; to balance, set load₁ + b₁ = load₂ + b₂ with b₁ + b₂ = batch volume [3.5]
Classes vs ECMP	ECMP: each class splits like the aggregate (∝ path split) → check policy violations; policy routing: put each class on allowed paths, recompute MLU; needs class-based forwarding, not a split ratio [3.6, M.3]
Hose bounds	max i→j = min(E_i, I_j) · out of a set across a cut ≤ min(Σ egress inside, Σ ingress outside) · pipe needs n(n–1) numbers, hose 2n [5.7]
Stack depth	depth d, Binding SID uses 1 slot: each non-final stack covers d – 1 hops, final covers d → BSIDs = ⌈(hops – d)/(d – 1)⌉ (7 hops, d = 3 → 2) [5.8]
sFlow	rate ≈ samples × N × avg packet size × 8 / interval (5000 × 1000 × 1500 B × 8 / 1 s = 60 Gb/s) [5.6]
Sharing 1 link	equal hard: C/n each, cap by demand, rest wasted · max-min: water-fill (satisfy smallest, re-split rest) · weighted: rates ∝ weights then water-fill · strict: highest priority takes all it wants first [6.2]
Aggregate	parent(s) = Σ children(s); slopes add, each cap removes that child's slope; decompose: parent allocation → s → read each child [6.5]
BwE waste	sent = Σ min(demand, current budget) × interval; possible = min(C, Σ demand) × interval; waste = difference; convert Gb → MB (× 1000 / 8) [6.9]
Tunnel swap	with k spare entries: add k new, shift (congestion-free), remove k old, repeat; each intermediate tunnel set needs an allocation LP, each shift a plan LP; final allocation already known [7.3]
k-failure	n disjoint links of c: guarantee (n – k)·c, spread equally; with probabilities: P(≥ k+1 fail) = how often the guarantee breaks [7.4]
Steering	① drop entry routers with excessive onward delay ② per entry: min(link to it, its onward capacity) ③ weights ∝ those values; flows split by weight (B 30, C 25 → 6/11, 5/11) [7.7]
QUIC loss	packet threshold: lost if a packet ≥ 3 higher is acked; else wait for time threshold (≈ 9/8 RTT); no ACKs at all → PTO probe (declares nothing) [2.6]
QUIC delivery	per stream: deliver bytes up to the first gap; other streams unaffected; resend = new packet number, same stream ID + offsets [2.3]
Counting	≤ k failures of n links: $\Sigma_{i=0}^{k}$ C(n, i) scenarios (4 links, ≤ 1: 5; ≤ 2: 11) · "≤ 1 down" in SAT: C(n, 2) pair clauses (4 links → 6) · all failure sets 2^n [8.4, 8.5]
Requirements	reach: ∃ delivered path · isolation: ∀h ∈ H: ¬R · waypoint: ∀ delivered paths visit F (vacuous if nothing delivered) · loop: no prefix repeats a node · query = allowed ∧ M ∧ ¬P [8.4]
Part 1 types	NUM, Copa, Vegas, bufferbloat, BBR, CoDel, fluid models, ECN/INT → Exam 1 sheet; parking lot: max-min (½, ½, ½)·C, proportional (⅓, ⅔, ⅔)·C [9.x]

3 Concepts by lecture

3.1 PCC (L9)

- **Event → action tables** embed an interpretation. Same "100 Mb/s, 1% loss": our overload → **decrease a lot** · shallow buffer → **decrease slightly** · other flow dominates → **maintain** · random loss → **increase**.
- **Model-based** (Reno…BBR, Copa) asks "what is the path doing?", infers hidden state; fails when the model misreads feedback. BBR v1→v2→v3 keeps adding filters/phases/bounds. **Performance-oriented** (PCC) asks "which action worked?", treats network as black box; fails on utility/trial-length/exploration design.
- **Allegro problems:** no latency term (bufferbloat) · tolerates 5% loss ⇒ **starves CUBIC** · fixed steps (slow far / overshoot near) · short flows end before trials · more state. Whiteboard: *"doesn't take delay into account; doesn't adapt exploration to target rate"*.
- **Vivace vs Copa:** Copa reacts to delay **magnitude** (≈ 1/δ pkts queued, a model); Vivace only says "lower RTT slope is better" — outcome, not model.



Response times with RTT = 40 ms: 120 / 80 / 40 ms. Request goes out one RTT earlier.

- Fair convergence among PCC flows proven only for **some** utilities, assuming comparable signals. PCC oscillates around the peak; doesn't synchronize (Reno does).
- **Learning:** rate x online (Allegro/Vivace) · rule table offline (**Remy**) · NN θ offline (**Aurora**). Fixed policy ⇒ still changing rate. Kernel avoids floats ⇒ ML in user space ⇒ reacts per RTT, not per ACK.
- **Evidence slide** (Vivace paper, capacity/RTT/loss change every 5 s): Vivace reacts quickly to a falling edge; Allegro tracks capacity most closely only because it accepts 2–3× more loss. One trace has limited scope.
- **Exploration:** trial ≥ 1 RTT (else misattribution); longer = less noise but stale/mixed if network changes. *"Long enough to be reliable, short enough to stay relevant."*
- **BBR evolution:** v1 = windowed max BW (10 rounds) + windowed min RTT (10 s), probes +25%, ignores loss → unfair to CUBIC · v2 adds loss/ECN limits, phases CRUISE / REFILL / UP / DOWN · v3 better probe stopping, tells a bandwidth plateau from app self-limitation. Why not binary-search the rate: big swings hurt others and make feedback noisy.
- **Vivace utility terms:** x^t (0 < t < 1) = reward with diminishing returns (concave → fair sharing) · $-bx\,d\ \mathrm{RTT}/dt$ = penalty for **growing** the queue, a bonus when it drains · $-cxL$ = loss penalty. dRTT/dt = RTT change per unit time in the interval (40 → 60 ms over 100 ms = 0.2). Gradient g = slope of score vs rate; sign = direction, size = step. Allegro numbers to recognize: u(100 Mb/s) = 96.2 / 66.2 / 19.3 at 1 / 4 / 6% loss.

3.2 QUIC (L10)

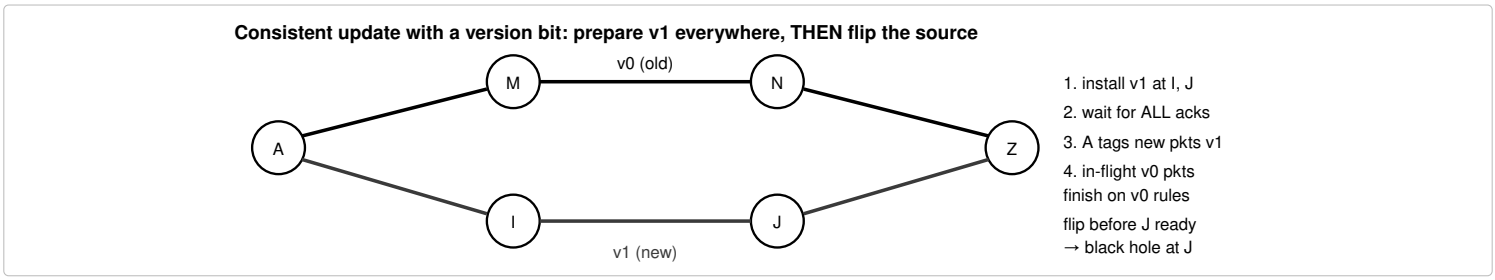
- App needs → mechanisms: secure start → TLS 1.3 in transport handshake · concurrent objects → streams · recover loss → pkt-number ACKs + loss inference + offsets · receiver capacity → stream + connection flow control · Wi-Fi→cell → CIDs + path validation · **CC governs all**.
- **Why not modify TCP?** Must reach endpoint stacks (kernels), the peer, **and the path**: middleboxes strip/reject options. Ossification cycle: narrow behaviors seen → depended on → new values fail → fallback keeps old. New protocol number = dropped even more.
- QUIC = UDP (passes middleboxes, every OS) + **user space** (app update = transport update) + **encrypt headers** (path can't depend on them). **Not "reliable UDP"**: full secure connection-oriented transport. Side effect: many implementations, interop issues.
- HTTP/1.1: connection per object (handshakes, state) · HTTP/2: streams over **one TCP byte order** ⇒ TCP HOL · QUIC: per-stream order.
- **UDP provides:** datagram delivery, ports (demultiplexing), broad acceptance by paths and OSes. **QUIC adds:** connections + packet protection (encryption), streams, flow control, loss recovery, congestion control. Encryption = the path can't interpret or rewrite internals (anti-ossification).
- **Independent ordering ≠ independent performance:** in-stream gaps still block; shared CC + connection credit; app dependencies.
- ACK frames list packet ranges (SACK-like) ⇒ more header bytes, slightly lower goodput than TCP.
- **0-RTT** needs remembered resumption state + server acceptance; only replay-safe (idempotent) requests, because the first flight can be **replayed**. **Nesting:** UDP datagram ⊃ QUIC packet (packet number) ⊃ STREAM frames (stream ID, offset). TCP retransmits reuse sequence numbers → ambiguous ACK/RTT sample; QUIC never reuses packet numbers. Migration: switch to another issued CID (unlinkable), PATH_CHALLENGE 8 random bytes ↔ PATH_RESPONSE, then re-learn cwnd/RTT.

3.3 TE foundations (L11)

- Private WAN: one operator owns DCs + backbone (+ often hosts) ⇒ can choose routes **and** rates. Inputs = aggregates (traffic matrix).
- **TE vs CC:** CC changes demand (rates, per RTT, distributed); TE redistributes (routes, seconds–minutes, centralized). Timescale separation breaks the feedback loop. TE fixes **inefficient mapping**, not **insufficient resources**.
- **Why min-MLU:** demand changes faster than TE reacts ⇒ maximize headroom everywhere (not just "for failures").
- RFC 9522: **measurement, characterization, modeling, control**. Objectives: traffic performance (loss, delay, throughput, commitments) vs resource use (no hotspots, alternate capacity, headroom, cost). Control surfaces: traffic mgmt (limit/reject), routing, resource constraints.
- **Five problems:** ① demand isn't given (counter 10 = 5+5? 8+2?; delivered ≠ load ≠ demand) ② allocation needs **policy** (A 10 + B 8 on 15) + time (defer batch: 17/15 → 12/15 then 13/15) ③ allocation ≠ router config (ECMP, LPM too coarse) ④ feasible ≠ safe transition (stage moves) ⑤ verify & adapt (U's stale discard route black-holes B).
- Scale trade-offs: aggregate (fewer rules / coarser) · plan less (cheaper / staler) · sample (less data / less visibility) · batch updates (fewer ops / bigger steps). Axes: historical vs live · offline vs online · central vs distributed vs hybrid · local vs global · open vs closed loop (flap risk).

3.4 MPLS & SDN (L12)

- MPLS + SDN are **not separate**: controller (PCE) programs an MPLS data plane. MPLS table separate from IP FIB ⇒ route same destination differently (src, port, class) · exact-match fast · **fast reroute** with precomputed backups.
- Traditional routing: only **link weights** (moving u→z moves everyone; can't split one destination; can't separate classes).
- SDN: generalized flow-based forwarding · control/data separation · control external to switches · programmable apps. Controller layers: app interface (network graph, **RESTful** API, intent API = northbound) · network-wide state (distributed DB) · communication (OpenFlow, SNMP = southbound). Logically central ≠ one machine (replicated, leader election).
- **Mainframe → PC analogy** (McKeown): vertically integrated, closed, proprietary, slow innovation (specialized hardware + OS + apps from one vendor) → horizontal layers with open interfaces (microprocessor, open OS, many apps), rapid innovation. SDN does the same for networking: commodity switches + controller + apps. Traditional per-router control plane: each router runs its own routing algorithm components that interact (monolithic, proprietary).



3.5 Meta EBB (L12–14)

- CBB = DC→users via PoPs/ISPs (reach, reliable, externally compatible, costly) · EBB = DC↔DC (internal, cheap, custom). IPv6-native. Same split: Google B2/B4, Microsoft CORE/SWAN.
- **4 parallel physical planes** (8 newer), each own routers + controller: roll out to ¼, A/B tests, failure isolation. Plane ≠ control plane ≠ data plane ≠ traffic class. Two layers of BGP.
- Architecture: **BGP route injectors** (reachability in/out) · **sFlow collector** (demand; heavy-tailed traffic ⇒ sampling OK; switches keep no per-flow state at Tb/s) · **Open/R** on every router (Spark neighbors, LinkMonitor RTT, KvStore **eventually consistent**, Decision, FIB) · controller: snapshot → path allocation → driver (every 1–3 min) → **LSP agents**.
- Failure: **local** backup (100s of ms) then **global** reoptimization next cycle; reachability ≠ capacity.
- Classes ICP > Gold > Silver > Bronze (keep 4–8). Gold: shortest RTT + burst margin. Silver/Bronze: K-shortest paths, then min MLU (K-shortest restricts choices, doesn't split). Allocation ≠ enforcement: bronze 5 → 90 burst hurts all ⇒ **strict priority + buffer carving**; entitlements + host rate limits.
- Segment can be a **node** SID (shortest path to it) or **adjacency** SID (that link): A pushes [16104, 203].
- Sampling works because traffic is **heavy-tailed** (Pareto): a few heavy hitters carry most bytes. Sample at ingress edge: each DC-to-DC byte counted once. Version update: install v1 everywhere → wait for **all ACKs** → flip source → remove v0 only after in-flight v0 drains. Binding SID example: SRC [→A, →B→C, 19999]; C replaces 19999 with [→D, →DST].
- Capacity planning: forecast from compute purchases; hose (segmented) vs pipe; check simple cuts; 1 M scenarios < 3 min; full plan < 6 h (300+ nodes, 1000+ links).
- Planning output = a plan for **capacity turnup and fiber procurement** (basic TE: given capacities choose rates; planning: choose capacities so TE stays feasible). A geographic map becomes a graph: DCs and transit midpoints as nodes, links with capacities.

3.6 Google (L14–16)

- **Intent** (what, not how) > **policy** (preference, allocation undecided) > **configuration** (exact state). Non-examples: interface IP (config), threshold alert (rule), "deny P1 unless P2" (access policy). Unwrap: scope, required service, maintenance scope, failure assumptions, preference.
- Lifecycle: interpret → check & plan → fulfill → assure → respond. Infeasible (8 G over 6 G cut) / conflicting (two mandatory 8 G on 10 G). Assurance at config / network behavior / service outcome; offered 2 G ⇒ can't prove 8 G.
- **Orion** (per domain, BGP between; microservices): NIB **pub-sub** (not callbacks), Config / Topology / Flow managers, OpenFlow front end. **Reconciliation**: desired ↓ observed ↑. Route request: request → resolve → realize → report. TE app (RouteAdverts), Path Exporter (model for TE + BwE), Raven (BGP/IS-IS).
- **B4** = Google's private inter-DC WAN (B2 = user-facing): sites become vertices, inter-site capacity becomes edges. Orion managers: **Config Manager** validates and applies config as a unit; **Topology Manager** tracks ports/links/devices at runtime (configured ≠ usable capacity: a link can exist in config yet be down); **Flow Manager** compares desired vs switch-reported rules; **OpenFlow Front End** talks to switch agents. **Intent examples:** steer traffic from geography X away from Y unless the destination is in Y · avoid vendor V's equipment for customer C even at reduced service · smart-city signal traffic on dedicated, redundant, non-fate-sharing slices.
- Sharing: equal hard limits (non-work-conserving) · max-min · weighted · strict priority. Real policy = **hierarchy** (strict classes + minimum guarantees). Bandwidth functions express any of it; slope change = policies no fixed weight can (HW: $B_A = \min(2s, s+2, 8)$: 2:1 at C=6, 3:2 at C=10).
- BwE pros: predictability, protects network, "why am I limited?". Cons: enforcer on every host (bug = fleet), complex loops, BwE-vs-CC debugging; stale budgets waste capacity → event-driven reports (more msgs).
- Intent is **declarative** (what, like SQL). **Unwrap** an intent: traffic scope, required service, maintenance scope, failure assumptions, preference. Intent depends on the layer: service contract → global TE choice → domain routing → switch entries. What changed since SDN began: router vendors opened up, so operators run their own code on router CPUs (Open/R, dSDN).
- **SDN pros:** programmability, abstraction, global optimum, cheap boxes. **Cons:** logical SPOF (blast radius), external dependencies, scaling, complexity, still run distributed routing (two control planes). **dSDN:** keep global TE + programmability; run TE on every router, in-band (fate-sharing), flood link/capacity/demand/prefixes/sequence numbers, ingress pushes label stack. SDN ≠ centralized.
- **dSDN on-router components:** LocalState + NodeStateExchange (observe & flood) → StateDB → Pathing + TE solver (replaceable) → Programmer (install). Failure: central = notify controller → recompute → program transit → activate ingress; dSDN = notify each headend → compute → update own ingress.

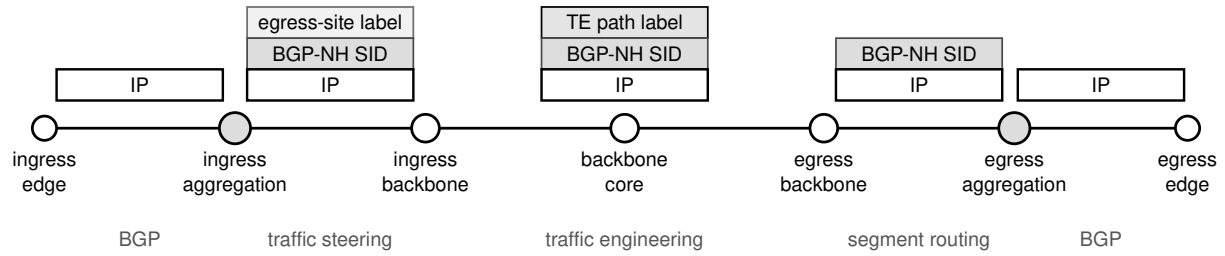
3.7 Microsoft (L16–17)

- SWAN: hosts → **service brokers** → controller; network agents (topology). Interactive first, then background (iterative max-min LPs). Configures **edge only**; core has static labels; tunnels + weights (50/30/20 %). 5 min.
- SWAN max(old, new): switches apply updates at different times, so mid-step each flow may be at either rate on a link; summing the larger of the two is the worst case. Reaching a new tunnel set = a **series of LPs** (allocations for intermediate sets + congestion-free shift plans).
- TeaVaR: EBB/B4/SWAN recover **reactively** (no availability guarantee). Deterministic k-failure wastes capacity; probabilistic β-guarantee uses failure probabilities.
- OneWAN: split WAN = isolation, different tech, independent growth; became costly (fragmented capacity, duplicate connectivity, two stacks, RSVP-TE scale, global blast radius). Challenges: table scale, fast repair, TE scale, Internet demand estimation, seamless migration. Solver chain: candidate paths → reserve committed share → P0 max-min → min cost; P1 (scavenger) max-min → min MLU → risk-diverse paths.

3.8 Verification (L18)

- Goal: guarantee **configs** (thousands, generated from intents) are error-free. Simulation/digital twin can't cover all packets × failures; **symbolic** search does.
- Model = configuration + packet class (symbolic headers) + environment (links up/failed). Requirements: reachability, isolation, waypointing, loop freedom, + failure tolerance scope.
- SAT: Boolean (links, next hops, forwarding, reach); NP-complete but practical. **SMT** adds bit-vectors (addresses), integers (metrics), linear arithmetic (capacity $x_1 + x_2 < 5$, latency).
- **Minesweeper** (SIGCOMM'17): config compiler → SMT; import/export/selection policies ($c \Leftrightarrow a \wedge \mathrm{Up}_{SB} \wedge \mathrm{Accept}_B(P)$), preferences, protocol interactions; checks **stable** routing states.
- Caveats: SAT = real counterexample; UNSAT only within the model (missing ACLs, external controllers, humans). Expensive (NP-hard).

OneWAN: what the packet carries on each segment (L17)



only the purple aggregation routers hold full IP tables

- **Unit 2 summary** (last slide): private WANs control routes, divide capacity among tenants, rate-limit at sources for predictability, but still need CC for RTT-scale reaction · central control = easier interface + global optimum, but must coexist with distributed control and is a logical SPOF · modeling failure to hedge against it is critical.

4 Comparisons

	Meta (EBB)	Google (B4)	Microsoft (SWAN)
DNA	nitty-gritty hybrid details	express intent / fairness (bandwidth functions)	failure handling; solve LPs exactly
Demand	sFlow sampling at routers	hosts report (BwE)	hosts → service brokers
Solve	MCF + HPRR heuristic	waterfill heuristic + quantize	iterative LPs (max-min)
Program	segment routing, Binding SID, version bit	Orion per domain, reconciliation	edge tunnels + weights, scratch capacity
Enforce	priority queues, buffer carving, entitlements	BwE hierarchy (HTB on hosts)	brokers rate-limit hosts
Failure	local backup → reoptimize	Orion/Raven fallback; dSDN proposal	reactive → TeaVaR β-guarantee

PCC vs Quiz 1 algorithms — L19: the only Part 1 content is part of the PCC question.

	signal	random loss	next to CUBIC
Reno	loss: +1/RTT, halve	backs off (wrong)	RTT-unfair, fills buffer
CUBIC	loss; cubic in time since loss	backs off	fills buffer
Vegas	queue (Exp – Act)·BaseRTT in α..β	halves too	starves
Copa	delay **magnitude**, target $1/(\delta d_q)$	ignores	competitive mode
BBR v1	max BW, min RTT, paces	ignores (robust)	unfair to CUBIC
Allegro	utility: throughput, loss (≤ 5%)	tolerates (robust)	grabs bandwidth
Vivace	utility: rate, RTT **slope**, loss	penalized in utility	low queue

NUM link: every CCA implies a utility (Reno ≈ $-1/(\mathrm{RTT}^2 x)$); Copa turns its utility into a fixed rule, PCC searches it online; concavity ⇒ fair convergence. Model-based = assumes what a signal means; PCC never names the cause.

	Allegro	Vivace	Copa / BBR
type	performance-oriented	performance-oriented	model-based
signal	throughput, loss	rate, RTT **gradient**, loss	Copa: delay magnitude · BBR: max BW, min RTT
step	direction from RCTs, fixed schedule	gradient: direction + size	designed rules
weakness	no delay term, starves CUBIC, overshoot	utility/trial design	model can misread feedback

	Link-state (IS-IS)	Central TE	dSDN
decides with	topology + link costs	global topology + demand	global TE at every router
path set by	each router's next hop	controller installs paths	ingress label stack
after failure	local recompute (no TE)	notify → recompute → program transit → ingress	flood → each headend computes → own ingress

Requirement	Meaning	Constrains
Reachability	∃ delivered path	delivered paths
Isolation	no delivered path for forbidden traffic	delivered paths
Waypointing	every delivered path visits F (doesn't require delivery)	delivered paths
Loop freedom	no feasible prefix repeats a router (can end before delivery)	all forwarding prefixes
Failure tolerance	property holds for every allowed failure (e.g. ≤ 1)	scope

Level	Example	Still undecided
Intent	keep ≥ 8 Gb/s for S while e is drained	paths, splits, rules, order
Policy	prefer P1 when it has capacity	allocation under current conditions
Configuration	install these next hops/ weights	nothing — realize exactly

5 HW2 answers (condensed)

- **Q1.1 BBR route change:** old (smaller) min RTT kept up to 10 s ⇒ BDP/ cwnd underestimated ⇒ delivery rate ↓ ⇒ 10-round BW filter loses high samples ⇒ BW ↓ ⇒ pacing & cwnd ↓. Recovery after min-RTT filter expires. **Fix:** early reassessment if RTTs consistently > min; drain inflight; resample; reset min-RTT only if high at low load. Imperfect (persistent queues).
- **Q1.2 PCC β:** (g, q) = lo (80, 4), hi (90, 24): β = 0.25 → 79 vs **84** (hi) · β = 1 → **76** vs 66 (lo). Counterexample lo (80, 5), hi (100, 10): β = 0.25 → 78.75 / **97.5**; β = 1 → 75 / **90** ⇒ both pick hi.

- **Q1.3:** longer experiments when noise is short-term and conditions stable; ≥ 1 RTT; faster updates ⇒ misattribution; too long ⇒ mixed/stale.
- **Q2.1 HOL:** TCP: B rendered at 60 ms (waits for resent A1) · QUIC: B at 23 ms.
- **Q2.2:** B-first: B at 75 ms; round robin: 125 ms; A 125 both. Window limits sends; scheduler picks who uses it; starvation risk.
- **Q3.1:** 6+6 = 12 on two 10s: MLU 0.6; ECMP puts 3 G RPC on slow link. **Q3.2:** RPC on fast, replication on slow — needs **class-based** forwarding, not a split ratio. **Q3.3:** 9+6: min-MLU 7.5/7.5 = 0.75 (ECMP 4.5 G RPC on slow) vs policy 9 / 6 ⇒ 0.9.
- **Q4.1:** C = 9, 12, 15 → s = 3, 4, 7 → A (6, 8, 8), B (3, 4, 7). **Q4.2:** non-work-conserving at 15: (8, 5) + 2 unused; work-conserving (8, 7). Policy no fixed ratio can do: $B_A = \min(2s, s+2, 8)$, $B_B = \min(s, 8)$. **Q4.3:** A sends 2 G × 0.2 s = 50 MB, B 6 G × 0.2 = 150 MB; link could 300 MB ⇒ **100 MB wasted**; fix: proactive demand reports (more messages/compute).
- **Q5:** see §2.3 (granularity 0.6 vs 0.5; stuck at 1.0 vs swap 0.8; 32.9 vs 44.7 ⇒ direct, though via-M MLU 0.895). Fix for Q5.3: use post-move MLU as route cost (still no joint optimization).

6 Lecture problem bank (numbers to recognize)

Allegro RCT	r ±ε: $U_- = 94$, $U_+ = 90$ → decrease. Pairs (98→102: 94, 90 lower wins) and (102→98: 96, 93 higher wins) disagree → stay, raise ε.
Vivace RTT	C = 100, queue nonempty: x = 120 slope 0.20, x = 110 slope 0.10 → 110 better (avg RTT misleads).
12 Gb/s TE	paths 10 & 5: shortest 12/10 ✗ · ECMP 6/6 → 120% ✗ · min-MLU 8/4 → 0.8 · 16 Gb/s > 15 cut: impossible.
Meta short/long	12 G, short 10 G 5 ms, long 5 G 20 ms: 12/0 ✗, 6/6 ✗, **10/2** (MLU 100%, 7.5 ms) = min-latency, **8/4** (80%, 10 ms) = min-MLU.
Swap headroom	A8, B8 on two 10 G paths, headroom 2: move 2 A, then 2 B, ×4; whole move = 16/10.
Defer batch	paths 15: urgent 12 + batch 5 = 17 ✗ → defer: 12 now, 8 + 5 = 13 next hour.
Resilience	100 G trunk, 3×400 G, survive 1: 50 each; oversubscribe 450/400.
Freeze	A–B 8 shared; App1 A→B→C min(3s,6) via B–C 2: frozen at 2 → App2 = 6 · with B–D–C 4 route: s = 2 → (6, 2).
Aggregate	parent gets 11 → s = 5 → children (6, 5) (add heights at same s).
Sharing 8 G link	equal 4/4 · weighted+capped 6/2 · priority to app2 0/8 · whiteboard 10 G, demands 8 & 2: hard 5/5 → 5/2 (3 wasted), max-min 8/2.
TeaVaR q4	L1 down: F1 (2,3,5) loses 0.2, F2 (4,3,3) loses 0.4 ⇒ L = 0.4; P(q4) = $10^{-3}{\cdot}0.9{\cdot}0.999 \approx 0.0009$.
k-failure	3 × 10 G: 30 / 20 / 10 for 0 / 1 / 2 failures.
Intent conflict	8 G over remaining 6 G cut = infeasible; two mandatory 8 G on 10 G = conflicting.
dSDN	52% of 41 largest B4 outages control-related (42 + 10).

7 Traps, numbers, answer patterns, glossary

⚠ **Request vs response RTTs:** slide labels mark when the **request** goes out (2 / 1 / 0 RTT); responses at 3 / 2 / 1.
⚠ **Independent stream ordering ≠ independent performance** (in-stream gaps, shared CC/credit, app dependencies).
⚠ **PTO expiry doesn't declare loss** — it sends a probe. Resend = new packet number, same offsets.
⚠ **TE can't create capacity:** above the cut you need admission/CC. ECMP splits flows, not bytes, and ignores capacity.
⚠ **Min-MLU reason = timescales/headroom**, not only failures.
⚠ **MPLS TTL is 8 bits** (slide said 5). Labels are **local**. MPLS and SDN are **not** alternatives.
⚠ **Logically centralized ≠ one machine.** SPOF = a wrong centralized decision (blast radius), not a crash.
⚠ **K-shortest restricts choices; doesn't set the split.** Allocation ≠ enforcement (priority alone doesn't stop buffer filling).
⚠ **HPRR accepts only strictly lower max util**; sums link costs (penalizes hops on purpose); can get stuck (needs swaps).
⚠ **Bandwidth functions: add heights at the same s**, never add s values. Frozen ≠ capped.
⚠ **Fulfillment ≠ assurance;** low offered load can't prove capacity.
⚠ **SWAN uses max(old, new)** per flow because switches update asynchronously; scratch s ⇒ ≤ ⌈1/s⌉ – 1 steps.
⚠ **TeaVaR loss = max over flows**; no rerouting inside a scenario; guarantee = allocation × (1 – V*).
⚠ **UNSAT ≠ network is correct** — only the model. Waypointing doesn't imply delivery. Loop witness can end before delivery.
⚠ **dSDN still computes global TE**; SDN = programmability, not centralization.

Not on the quiz: WAN hardware/topology design · DC/WAN commonalities · how Meta enumerates cuts · "regulation" · BGP (Unit 3).

7.1 Numbers to know

Topic	Numbers
PCC	Allegro 2015, Vivace 2018 · sigmoid flips at L = 5% (α = 100 in problems) · u(100 Mb/s) = 96.2 / 66.2 / 19.3 at 1 / 4 / 6% · slide RCT r = 100, ε = 2%: (98, 102) → 94, 90; (102, 98) → 96, 93 ⇒ disagree · Vivace: C = 100, x = 120 slope 0.20, x = 110 slope 0.10 · Allegro 2–3× Vivace's loss · Remy 2013 · HW: (80, 4) vs (90, 24), β = 0.25 → hi, β = 1 → lo
BBR	min-RTT filter 10 s · max-BW filter 10 rounds · cwnd ≈ 2·BDP · probe +25%
QUIC	fresh: TCP+TLS 1.3 response at 3 RTT, QUIC 2, 0-RTT 1 (requests sent at 2 / 1 / 0) · slide RTT 40 ms · packet threshold 3 · time threshold 9/8 RTT · PATH_CHALLENGE 8 random bytes · Wi-Fi 20 ms / 100 Mb/s vs cellular 80 ms / 10 Mb/s · RFC 9000–9002 · HW2 Q2: RTT 50, B-first 75 ms vs RR 125 ms; HOL TCP 60 ms vs QUIC 23 ms
TE	12 on (10, 5): shortest 120%, ECMP 6/6 → 120%, min-MLU 8/4 → 0.8; 16 > 15 cut · A10 + B8 on 15 → (10, 5) or (7.5, 7.5) · batch: 17/15 → 12/15 + 13/15 · swap A8/B8, headroom 2 → 4 rounds · RFC 9522 (2024) · HW Q3: 6+6 → 0.6; 9+6 → 0.75 vs policy 0.9
MPLS	label 20 · Exp 3 · S 1 · TTL 8 = 32 b = 4 B · 2^{20} labels · 100 G trunk on 3 × 400, survive 1 → 50 each · oversubscribe 450/400
Meta	4 planes (8 newer) · controller every 1–3 min · local backup 100s of ms · classes ICP > Gold > Silver > Bronze, keep 4–8 · stack depth 3 · BSID 19999 · node SID 16104, adj 203 · short/long 12 G: 10/2 (MLU 100%, 7.5 ms) vs 8/4 (80%, 10 ms) · burst: gold 20+20, silver 10+5, bronze 5 → 90 on 100 G · buffer carve 2 / 1 / 0.5 MB · planning: 1 M scenarios < 3 min, < 6 h for 300+ nodes / 1000+ links · HW Q5: α 66.4, u* 0.855 → 32.9 vs 44.7; 8 G on 10/6 → 0.5 split, 0.6 with 4 LSPs; stuck at 1.0 vs swap 0.8
Google	B4 example: s = 10/11, 10/3, 5, 10 → FG1 20, FG2 5; FG1 = 11s to 16.5 at s = 1.5, slope 1 to 20 at s = 5; FG2 = s/2 · quantization K = 2 → A–C 12.5 > 10 · BWF lecture: C = 4/8/12 → (3,1), (6,2), (6,6) · freeze: (2, 6) vs (6, 2) · aggregate: 11 → s = 5 → (6, 5) · intent: 8 G across 6 G cut infeasible; two 8 G on 10 G conflicting · 41 largest outages: 42% control + 10% both = 52% (32% data, 16% other) · RFC 9315 (2022) · dSDN 2024 · HW Q4: C = 9/12/15 → s = 3/4/7; 100 MB wasted
Microsoft	SWAN 2013, recompute ≈ 5 min, weights e.g. 50/30/20 · transient 80+80 = 160 on 100 · 20% scratch: U 64/16 → 44/36 → 24/56 → 16/64 · s = 10% → 9 steps, 20% → 4 · TeaVaR 2019, β = 99.98%, slide p(fail) 10^{-3} / 10^{-1} / 10^{-3} → ≥ 2 fail 0.02%, top/bottom 0.2% · q4 loss 0.4 · k-failure 3 × 10: 30/20/10 · OneWAN: 10^5 DC vs 10^6 Internet routes; prune < 500 Mb/s = 2.5% traffic → 5× rows, 7× vars; CLP/GLOP 5×; steering ⅓ via B (10), ⅔ via C (20)

Verification	Minesweeper SIGCOMM 2017 · 4 links, ≤ 1 failure → 5 cases, C(4, 2) = 6 pair clauses · web dropped for S–F or F–D down before fix · diameter 2 → layers R_0–R_2

7.2 Answer patterns + one-liners

Question	Answer structure
"What does this equation describe?"	name it + whose · every symbol with units · which way each term pushes · limits / edge cases · who computes it (sender, controller, router) → §1.2
Propose a modification + trade-off (HW Q1.1b, Q4.3, Q5)	① the fix in one sentence ② what improves (number if possible) ③ its cost: more state / computation / messages / complexity ④ what it still doesn't solve (e.g. pairs of moves still not globally optimal)
Did SDN simplify or complicate WAN ops? [6.11]	pros: programmability, abstraction, global optimum, cheaper boxes · cons: logical SPOF (blast radius), external deps + control network, scaling, two control planes · what changed: vendors opened up routers · dSDN: keep global TE + programmability, move control onto routers in-band (fate-sharing) · SDN ≠ centralized
Model-based vs performance-oriented	"what is the path doing?" (infer state, designed actions, grows by patches, fails when the model misreads) vs "which action worked?" (black box, utility compares nearby rates, fails on utility/trial design)
Why not just modify TCP?	must reach endpoint stacks, the peer, **and** the path; middleboxes strip/reject (ossification cycle) → QUIC: UDP + user space + encrypted headers
TE vs congestion control	knob (routes vs rates) · timescale (minutes vs RTT) · who (one operator, centralized vs independent endpoints) · TE fixes inefficient mapping, not insufficient resources
Why min-MLU?	demand changes faster than TE re-runs ⇒ maximize headroom everywhere (not just for failures)
Compare Meta / Google / Microsoft	§4 table rows: DNA · demand estimation · solver · route programming · enforcement · failure handling
Why does HPRR / B4 / OneWAN use a heuristic?	exact LP with millions of variables every minute is too slow ⇒ approximate; cost: suboptimality (stuck states, quantization, pruning) vs speed
Is the property guaranteed?	SAT ⇒ concrete bug; UNSAT ⇒ only within the model (missing ACLs, external controllers, humans); NP-hard, expensive

Memorize

- **Same loss, four meanings**: overload ↓↓, shallow buffer ↓, other flow ⇒ hold, random loss ↑.
- **PCC never names the cause; it compares nearby rates.**
- **Vivace judges the queue's slope, not its size.**
- **The gradient sets the step size; θ sets the confidence.**
- **TE recovers stranded capacity; it cannot create capacity.**
- **Constraints say what is possible; the objective chooses what is preferred.**
- **K-shortest restricts choices; it doesn't set the split.**
- **Allocation ≠ enforcement** (priority queues + buffer carving + host limits).
- **Labels are local; MPLS and SDN are partners, not rivals.**
- **Prepare everywhere, then flip the source** (version bit).
- **Bandwidth functions: add heights at the same s.**
- **Routes decide where growth stops** (freezing).
- **Fulfillment ≠ assurance; low offered load proves nothing.**
- **Switches don't update together ⇒ sum max(old, new).**
- **TeaVaR promises the worst flow, β of the time.**
- **Only aggregation routers know IP prefixes** (OneWAN).
- **A counterexample is real; its absence is only as good as the model.**
- **SDN = programmability, not centralization** (dSDN).

7.3 Glossary (acronyms)

ACL access-control list (first match wins) · **BSID** Binding SID (label naming a programmed continuation) · **BwE** Bandwidth Enforcer · **CBB** / **EBB** Meta's classic (user-facing) / express (DC↔DC) backbone · **CID** QUIC connection ID · **CSPF** constraint-based shortest path first · **ECMP** equal-cost multi-path (hash flows) · **FEC** forwarding equivalence class · **FIB** forwarding information base · **HOL** head-of-line blocking · **HPRR** heuristic path rerouting · **HTB** hierarchical token bucket (Linux rate limiter) · **ICP** Meta's network-control class · **LPM** longest-prefix match · **LSP** / **LSR** label-switched path / router · **MCF** multicommodity flow · **MI** monitor interval · **MLU** maximum link utilization · **MPFA** multi-path fair allocation · **NIB** network information base (Orion, pub-sub) · **PCE** path computation element · **PHP** penultimate-hop pop · **PTO** probe timeout · **RCT** randomized controlled trial · **RSVP-TE** reservation signalling for LSPs · **SAT** / **SMT** satisfiability / modulo theories · **SID** segment identifier (node or adjacency) · **sFlow** 1-in-N packet sampling · **SPOF** single point of failure · **SRLG** / **shared risk** links that fail together · **VaR** value at risk (β-percentile loss) · **WRR** weighted round robin · **0-RTT** resumption early data.

8 True / false bank (every MCQ statement from the problems)

✔ = true, ✘ = false, with the reason. Generated from the notes' problem bank; problem id in brackets. Quiz 1 scoring was +1/–1 per mark: if a statement isn't here and you're unsure, leave it blank.

8.1 PCC (L9)

- ✔ If the loss is random, non-congestive loss with spare capacity, the useful response is to increase the rate. — Random loss isn't a congestion signal, so there's room to grow. [1.1]
- ✘ If another, more aggressive flow is occupying the shared buffer, the useful response is to back off substantially. — You're the victim and haven't reached your fair share. The useful response is to **maintain**. [1.1]
- ✔ Reno's rule (halve on loss) embeds one interpretation of loss — that this flow overloaded the link — and applies it in every scenario. — Event→action tables bake an interpretation of the event into the action. That's what "model-based" means. [1.1]
- ✘ PCC resolves the ambiguity by first identifying which of the four hidden scenarios is happening and then applying the matching response. — PCC deliberately **doesn't name the cause**. It treats the network as a black box and compares the utility of nearby rates. [1.1]
- ✔ If a shallow buffer overflowed from a small burst while the link is far from capacity, a slight decrease or better pacing is the useful response. — Lecture table: shallow buffer → decrease slightly. [1.1]
- ✘ Remy updates its table of control rules during live operation on the Internet. — Remy generates its rule table **offline**. [1.8]
- ✔ A fixed learned policy can still produce a constantly changing sending rate. — The policy maps observation history → action. Observations change, so the rate changes. [1.8]
- ✘ Moving a neural congestion controller to user space lets it react to every ACK at microsecond timescales. — User-space models react every RTT or every few RTTs; they can't react per ACK. [1.8]
- ✔ Linux kernel code generally avoids floating point, one reason heavy learned controllers run in user space behind an API. — Integer-only kernel math, plus the cost of kernel development. [1.8]
- ✔ PCC Vivace learns its sending rate through live experiments. — Allegro and Vivace learn the rate x online. [1.8]
- ✘ Because PCC is rate-based, it cannot limit the number of packets in flight. — Rate and window are linked through the RTT. A rate-based sender can cap inflight at about 2·rate·RTT, just as BBR (window-based) paces at about 2·cwnd/RTT. PCC's analysis assumes pure rate control, so the math would need revisiting. [1.8]
- ✘ A good way to find the best rate is a binary search that alternates very low and very high rates. — The professor's aside: big rate swings hurt other flows and make the feedback noisy. PCC compares nearby rates $r(1\pm\varepsilon)$ instead. [1.8]
- ✔ A trial shorter than one RTT risks attributing performance to the wrong rate. — Feedback lags by one RTT, so you'd score the previous rate. [1.9]
- ✘ Longer trials always produce better decisions. — Not if the network changes during the trial. [1.9]
- ✔ Longer trials average out measurement noise, but if conditions change during the trial the result mixes regimes and may be stale. — The core trade-off: reliable vs relevant. [1.9]

✔ Short flows may finish before PCC has run enough trials to converge. — Mentioned in lecture as a PCC weakness. [1.9]

✘ Exploration is free when the network is idle, so trials should be as long as possible. — Exploration costs throughput or queueing, and long trials go stale. [1.9]

8.2 QUIC (L10)

✔ A TCP connection is identified by its 4-tuple, so the client IP change breaks it. — src IP, dst IP, src port, dst port. [2.5]

✔ QUIC packets carry a connection ID, letting the server associate packets from the new address with the existing connection state. — Connection IDs locate state; packet protection authenticates the traffic. [2.5]

✘ Because the QUIC connection survives, it should keep its Wi-Fi congestion window on the cellular path. — "A surviving connection doesn't imply a surviving path model." Re-establish what the new path supports (e.g., 20 ms/100 Mb/s → 80 ms/10 Mb/s). [2.5]

✔ Path validation (PATH_CHALLENGE / PATH_RESPONSE with random data) checks that the new path works. — Explicit challenge/response before relying on the path. [2.5]

✘ The client must keep using exactly the same connection ID on the new path so the server can recognize it. — It switches to **another issued ID** (cid-X → cid-Y). The server maps both to the same connection. [2.5]

✔ A QUIC stream can still be blocked by its own missing bytes. — Ordering is per stream: missing prefix → later bytes of that stream wait. [2.7]

✘ Because streams are independent, a congested bottleneck slows down only the stream whose packet was dropped. — Congestion control and connection flow-control credit are **shared** by all streams. [2.7]

✔ QUIC runs over UDP partly because UDP is broadly allowed by Internet paths and supported by endpoint operating systems. — And a brand-new transport protocol number would be dropped even more. [2.7]

✔ Encrypting most transport header fields makes it harder for middleboxes to depend on or rewrite them, reducing future ossification. — Middleboxes see the IP and UDP headers and very little of QUIC's. [2.7]

✘ Deploying new QUIC behavior typically requires a kernel upgrade on both endpoints. — QUIC lives in user space, so an **application** update changes the transport. [2.7]

✔ Different QUIC implementations can have interoperability problems even though they follow the same RFCs. — Each makes its own assumptions (e.g., about path challenges and stream handling). IETF interop matrices exist because of this. [2.7]

8.3 TE foundations (L11)

✔ TE typically acts on aggregates every few seconds to minutes; congestion control acts per flow every RTT. — This timescale separation is what breaks the NUM↔TE feedback loop. [3.7]

✘ Minimizing MLU is attractive mainly because it minimizes the average path latency. — It's about **headroom**. Min-MLU often raises mean latency (Meta example: 10 ms vs 7.5 ms). [3.7]

✘ If total demand exceeds the capacity of the minimum cut, a better TE split can still carry all of it. — TE fixes inefficient mapping, not insufficient resources. [3.7]

✔ ECMP balances flows, which isn't the same as balancing bytes. — Hash-based per-flow splitting ignores flow sizes and path capacities. [3.7]

✘ If both the current and the target configurations are feasible, any order of router updates is safe. — Transient states can overload (swap example) or black-hole and loop. [3.7]

✔ Besides routing, TE's control surfaces include traffic management such as rate limiting or admission. — Traffic management, routing, and resource constraints. [3.7]

8.4 MPLS / SDN (L12)

✘ MPLS and SDN are alternatives: a network uses one or the other. — The professor stressed they are **not** separate. Hybrids are the norm. [4.7]

✔ A centralized controller (PCE) can compute paths and program an MPLS data plane. — The SDN + MPLS TE architecture slide. [4.7]

✔ Under purely centralized control, link-failure recovery takes at least a report to the controller, recomputation, and reprogramming, often hundreds of ms or more. — Hence hybrid designs with local fast reaction. [4.7]

✔ The southbound API (e.g., OpenFlow) defines what in a switch is controllable. — Slide: "defines what is controllable, what is not." [4.7]

✘ A logically centralized SDN controller must physically run on a single machine. — It is replicated, with leader election. The single point of failure is logical (bugs, bad config). [4.7]

✘ MPLS labels have network-wide meaning, so every router must use the same label for an LSP. — Labels are local; LSRs swap them hop by hop. [4.7]

✔ In distributed MPLS TE, the ingress router uses RSVP-TE to set up and reserve the LSP at downstream routers, using bandwidth information flooded by modified OSPF/IS-IS. — The MPLS signaling slide: OSPF-TE/IS-IS-TE flood link and reserved bandwidth, and RSVP-TE installs the LSP. [4.7]

✘ When an SDN switch sees a link fail, it recomputes its own routes locally and then informs the controller. — Reversed. The switch sends an OpenFlow port-status message; the controller updates link state, calls the routing app, recomputes, and installs new flow tables via OpenFlow. [4.7]

8.5 Meta (L12–14)

✔ Order 1, 3, 2, 4 can black-hole packets at J. — v1 packets reach J before J has a v1 rule, so they are dropped. [5.5]

✔ Order 2, 1, 3, 4 is safe, provided step 3 waits for acknowledgments from I and J. — All v1 rules are ready before any v1 packet exists. The order among the preparation steps doesn't matter. [5.5]

✘ Step 4 may run immediately after step 3 is issued, because new packets are tagged v1. — v0 packets already in flight still need M and N's v0 rules. Remove them only after the old packets drain. [5.5]

✔ A single bit suffices because only the old and the new version ever coexist; the bit alternates 0/1 across updates. — Version alternation. [5.5]

✔ With this scheme, packets of the same flow can briefly take two different paths, which the sender may see as an RTT change unrelated to congestion. — The lecture made this connection to congestion control. [5.5]

✔ EBB's planes are parallel physical networks with their own controllers, so new controller software can be rolled out to a fraction of capacity and A/B tested. — Phased releases and separate failure domains. [5.9]

✘ An EBB "plane" is the same thing as a traffic class such as gold or bronze. — Physical plane ≠ control plane ≠ data plane ≠ traffic class. The slide calls this out. [5.9]

✔ When a primary path fails, an LSP agent activates a preinstalled backup within hundreds of milliseconds; the central controller re-optimizes in a later cycle. — Two-stage failure response. [5.9]

✘ Restoring reachability after a failure guarantees that the rerouted demands have enough capacity. — Several rerouted demands can compete for the backup's links. [5.9]

✘ Open/R's KvStore replicates link state with strong consistency, so all routers see an update at the same instant. — It is **eventually** consistent. [5.9]

✔ Gold traffic is allocated before silver and bronze, on shortest paths measured by RTT. — Priority order; gold uses shortest paths with RTT from LinkMonitor. [5.9]

8.6 Google (L14–16)

✔ In BwE, usage reports flow up the hierarchy and bandwidth budgets flow down to host enforcers. — Usage ↑, budgets ↓. [6.10]

✔ Host enforcers cap tasks' rates (e.g., with Linux HTB); TCP congestion control still operates underneath that cap. — If cwnd/RTT < limit, CC binds; otherwise BwE's limit binds. [6.10]

✘ A single Orion controller instance manages all of B4; domains don't need BGP between them. — Orion is **per domain**; domains act like ASes and exchange BGP (Raven). [6.10]

✔ Orion's NIB is a pub-sub system: components subscribe to state changes rather than the NIB invoking every component's callback. — This avoids making the NIB a bottleneck that knows every reaction. [6.10]

✘ dSDN gives up programmability to gain reliability. — It **keeps** operator-defined control and global TE. SDN = programmability, not centralization. [6.10]

✔ In dSDN, every router learns the TE inputs by flooding and computes the global TE solution; the ingress encodes its chosen path as an MPLS label stack. — Source routing, like Meta's. [6.10]

✔ 52% of the 41 largest B4 outages studied had a control-related root cause. — 42% control-only + 10% control-and-data. [6.10]

8.7 Microsoft (L16–17)

- ✔ SWAN's controller configures only the ingress (edge) switches; core routers forward on statically configured label paths. — Only tunnel selection and weights at the edge. [7.8]
- ✘ SWAN allocates the background class first and the interactive class with the remaining capacity. — Interactive (high priority) first. [7.8]
- ✔ Congestion-free updates are needed because switches apply updates at different times, so links can see a mix of old and new rates. — The 160 Gb/s transient example. [7.8]
- ✔ TeaVaR assumes independent link failures, so a scenario's probability is a product of per-link probabilities. — Multiply per-link up/down probabilities. [7.8]
- ✘ Guaranteeing service under any 2 simultaneous failures always uses capacity more efficiently than a probabilistic β-guarantee. — It's the opposite: it sizes for rare events and wastes capacity. [7.8]
- ✔ A split WAN can strand capacity: one WAN overloaded while the other has spare. — Fragmented capacity: inter-DC traffic grew to dominate. [7.8]
- ✔ OneWAN removes trunks below 500 Mb/s from the optimization, cutting variables about 7× while affecting about 2.5% of traffic. — And 5× fewer constraint rows. [7.8]

8.8 Verification (L18)

- ✘ If the solver returns UNSAT for (allowed failures ∧ model ∧ ¬property), the property is guaranteed in the real network under all conditions. — Only within the **modeled scope**. Missing behaviors (external controllers, humans, unmodeled ACLs) aren't covered. [8.6]
- ✔ A SAT result yields a concrete counterexample: a failure condition, a packet, and forwarding state that violate the property. — The witness explains the problem. [8.6]
- ✔ Pure Boolean SAT can't directly express "the sum of competing traffic on a link ≤ its capacity" over integer rates; SMT can. — SMT adds integers, bit-vectors, linear arithmetic. [8.6]
- ✔ Minesweeper derives routing and forwarding constraints from router configurations, including import policies and route preferences, and checks stable routing states. — A compiler for configs; stable states after failures. [8.6]
- ✘ Waypointing alone guarantees that packets are delivered. — It constrains only delivered paths; add reachability. [8.6]
- ✔ Loop freedom is checked on forwarding prefixes, because a looping packet may never be delivered. — A loop witness can end before delivery. [8.6]

8.9 Mixed (mock quiz)

- ✔ Allegro's default utility counts throughput only while the loss rate stays below about 5%. — Sigmoid(L − 0.05). [M.1]
- ✘ Vivace penalizes the magnitude of the RTT, as Copa does with queueing delay. — It penalizes the **gradient** dRTT/dt. [M.1]
- ✔ In Vivace, a larger magnitude of the estimated utility gradient produces a larger rate change, subject to a bound. — $\Delta r = \theta g$, bounded. [M.1]
- ✘ When Allegro's two trial pairs disagree, it moves in the direction of the pair with the larger utility difference. — It stays at r and widens ε. [M.1]
- ✔ Under random non-congestive loss, PCC can keep increasing its rate because higher rates still yield higher utility, without identifying the cause of the loss. — No cause identification; just utility comparison. [M.1]
- ✔ With HTTP/2 over one TCP connection, a lost packet of one object can delay delivery of other objects whose bytes already arrived. — TCP-level head-of-line blocking. [M.2]
- ✘ QUIC 0-RTT can be used on the first-ever connection to a server. — It needs remembered resumption state. [M.2]
- ✘ A QUIC retransmission reuses the original packet number so the receiver can match it to the lost packet. — A **new** packet number; stream offsets identify the bytes. [M.2]
- ✔ All QUIC streams in a connection share one congestion controller. — CC is per connection. [M.2]
- ✔ An MPLS label is 20 bits, and the S bit marks the bottom of the label stack. — label 20 | Exp 3 | S 1 | TTL 8. [M.4]
- ✔ CSPF prunes links that can't meet a trunk's bandwidth or color constraints before running shortest path. — Prune → shortest path → install. [M.4]
- ✘ With Meta's version bit, the source may start tagging packets v1 as soon as it has sent the update messages to the new path's routers. — It must wait for **acknowledgments** that every v1 rule is installed. [M.4]
- ✘ sFlow lets switches keep exact per-flow byte counts at line rate. — It samples 1-in-N packets; switches keep no per-flow state. [M.4]
- ✔ Under the hose model, a site's total egress is bounded, but how it splits across destinations may change. — That is the point of the hose. [M.4]
- ✔ A Binding SID lets an intermediate router supply the rest of a label stack, working around a stack-depth limit. — Continuation at the BSID node. [M.4]
- ✘ If service S offers 2 Gb/s and receives 2 Gb/s, an intent of "≥ 8 Gb/s for S" is confirmed. — Offered = delivered = 2 says nothing about 8 Gb/s availability. [M.6]
- ✔ Orion's reconciliation repairs drift such as rules missing after a component restart. — Desired vs observed state, at every layer. [M.6]
- ✔ TE gives BwE paths and split weights; BwE enforces budgets at hosts and reports prioritized demand to TE. — The TE ↔ BwE loop. [M.6]
- ✔ A non-work-conserving 2:1 policy can leave capacity unused while one group still has unmet demand. — E.g., (8, 5.33) with 2.67 unused while Y wanted 6. [M.6]

8.10 Part 1 + bridges

- ✔ Experiment 1 meets the persistent-delay condition after about 100 ms, even though its maximum is lower than experiment 2′s. — The **minimum** sojourn stays above target for a whole interval. [9.6]
- ✘ Experiment 2 enters dropping as soon as a 30 ms sojourn is observed. — Within every interval the min is 3 ms < 5, so it's good queueing (a burst). [9.6]
- ✔ If experiment 1 persists, successive drop gaps shrink as 100/√count ms (100, 70.7, 57.7, 50 …). — Control law. [9.6]
- ✘ CoDel's target must be re-tuned whenever the link rate changes because it is measured in bytes. — Sojourn time is in time units, independent of link rate. [9.6]
- ✔ ECN marks let the sender respond to congestion before buffer overflow causes loss, without retransmitting the marked packets. — Signal before loss. [9.8]
- ✘ With Reno's standard ECN response, if 20% of a window's packets are marked the window shrinks by exactly 20%. — It reacts like one loss: halve, once per RTT. [9.8]
- ✔ INT records can show which hop had the largest queue; a single ECN mark can't. — Per-hop telemetry. [9.8]
- ✘ By the end-to-end argument, reliable link-level retransmission makes an endpoint file checksum unnecessary. — Only the endpoints can verify the complete transfer; link checks just help performance. [9.8]
- ✘ BBRv1 reduces its rate in response to ECN marks. — v1 ignores loss and ECN; v2 adds them. [9.8]
- ✔ dSDN's fate-sharing applies Clark's principle to control: control should depend only on the routers and links it manages, so it fails only when they do. — Clark: state is lost only when the entity that needs it is lost. dSDN: control runs on the routers, in-band, so no separate control network or server can fail independently. [9.9]
- ✘ Host rate limits (BwE) work in Google's WAN for the same reason ECN never deployed on the open Internet. — Opposite reasons. BwE works because one authority controls both the senders and the network. ECN (and ICMP source quench) failed because authority on the open Internet is fragmented and networks don't trust each other. [9.9]
- ✘ Min-MLU traffic engineering and NUM congestion control turn the same knob: both adjust source sending rates. — Different knobs: CC changes **rates** with routes fixed; TE changes **routes** with demand fixed (Exam 1 ref §3.1). [9.9]
- ✔ QUIC's fast rollout fits the Part 1 "de-ossification" story: one company controlling servers, clients and much of the traffic can change transport behavior quickly. — Exam 1 ref §7.2: deployability went up because one company can own the server, much of the network and the client. QUIC also runs in user space, so updating the app updates the transport. [9.9]
- ✔ In B4′s waterfill, if every flow group has the same bandwidth function B(s) = s and no cap, the result is max-min fair. — Equal slopes mean equal priority, so raising the common fair share s is exactly water-filling: max-min fairness. [9.9]
- ✘ SDN's single-point-of-failure risk is mainly the controller machine crashing, which replication solves. — Controllers are replicated. The risk is

a **logically** centralized wrong decision (a bug or bad config) applied network-wide: the blast radius. [9.9]

9 Part 1 bridges (bring the Exam 1 sheet for L1–8)

Part 2	Part 1 idea	Ex1 ref
TE vs CC	two knobs: rates (CC) vs routes (TE); both only in hyperscaler WANs	§3.1, §4.6
min-MLU / MCF	routing matrix A, $Ax \le c \to \sum x_{kp} \le \rho c_e$	§3.2, §3.5
B4 / bandwidth fn	max-min by water-filling; weighted utilities; parking lot: max-min (½,½,½) vs prop. (⅓,⅔,⅔)	§3.3–3.4
PCC	utilities, concavity; Copa $\ln x - \delta \ln d$ (1/δ queued); BBR BtlBw & RTprop	§3.4, §6, §10
rate vs window	pacing hybrid: cap inflight at cwnd, pace 2·cwnd/RTT	§5.3
QUIC	ossification then vs now (one company owns server, net, client); end-to-end argument	§7.2, §2.5
SDN / dSDN	SDN benefits (cheap HW, global opt) vs blast radius; **fate-sharing** (Clark)	§2.1, §2.3
BwE / entitlements	deployment authority: who controls bottleneck **and** sender (why ECN/ source quench failed)	§10.2, §9.3

Exam 1 formulas you may still need: BDP = rate × RTT · Copa $x = 1/(\delta d_q)$ · Vegas Diff (α=1, β=3) · CoDel 5 ms / 100 ms, 100/√count · DCTCP $\mathrm{cwnd}(1 - \alpha/2)$ · BBR cwnd ≈ 2·BDP, PROBE_BW 1.25/0.75/1×6.