

# VU LEARNING HUB

[vulearninghub.xyz](https://vulearninghub.xyz)

**HIGHLIGHTED HANDOUT**

## CS721

Network Performance Evaluation

**CS721 by VU Learning HUB**

Prepared for VU students • Free study resources

<https://vulearninghub.xyz>

# CS721

## Network Performance Evaluation

---

### Study Handout

Complete lecture-wise notes for Virtual University students

## About This Handout

This handout covers CS721 Network Performance Evaluation — the mathematics and methods for predicting and measuring how networks behave under load. It unites probability, queueing theory, simulation, and measurement into one toolkit. The exam is numerical and conceptual in equal parts: master the formulas (M/M/1, Little, Erlang-C, P-K, Mathis) and the judgment of when each technique applies. After each lesson, close the handout and re-derive the key formulas from memory. 

## Table of Contents

|           |                                                 |
|-----------|-------------------------------------------------|
| Lesson 1  | Performance Evaluation: Concepts and Methods    |
| Lesson 2  | Probability Foundations for Networks            |
| Lesson 3  | Markov Chains                                   |
| Lesson 4  | The M/M/1 Queue                                 |
| Lesson 5  | Little's Law, M/M/c, and M/G/1                  |
| Lesson 6  | Networks of Queues: Jackson and Closed Networks |
| Lesson 7  | Discrete-Event Simulation                       |
| Lesson 8  | Measurement, Workloads, and Monitoring          |
| Lesson 9  | TCP, QoS, and Wireless Performance              |
| Lesson 10 | Birth-Death Processes and Erlang-B              |
| Lesson 11 | Operational Laws and Bottleneck Analysis        |
| Lesson 12 | Closed Networks and Mean Value Analysis         |
| Lesson 13 | Random Numbers and Variate Generation           |
| Lesson 14 | Simulation Output Analysis                      |
| Lesson 15 | Experimental Design and Benchmarking            |
| Lesson 16 | Wireless LAN Performance Analysis               |
| Lesson 17 | Data-Center and Cloud Performance               |
| Lesson 18 | Revision: Performance Evaluation Drill          |

# Lesson 1: Performance Evaluation: Concepts and Methods

## Learning Objectives

- Compare analytical, simulation, measurement techniques
- Define performance metrics
- Follow a systematic evaluation methodology
- Avoid common pitfalls (warm-up, replication)

Performance evaluation answers: how fast, how much, how reliable? The three techniques: analytical modeling (queueing theory — fast, approximate), simulation (flexible, expensive), and measurement (real, but costly and limited to existing systems). Evaluation metrics: throughput (jobs/sec), response time/delay, utilization, loss probability, and fairness. The workload model matters as much as the system model — garbage workload in, garbage predictions out.

The systematic approach: state goals → define metrics → characterize workload → choose technique → model/simulate/measure → validate → present with confidence intervals. Common pitfalls: ignoring warm-up (transient vs steady state), single-run simulation results without replication, and confusing the mean with the distribution (tail latency — the 99th percentile — is what users feel). Averages lie; distributions decide.

Why networks specifically: packets queue at buffers, links have finite capacity, and protocols (TCP congestion control) create feedback loops — making networks the richest application domain of queueing theory. This course builds the full toolkit: probability foundations, queueing models, simulation, and measurement.

## Key Terms & Definitions

- Analytical / simulation / measurement — The three evaluation techniques.
- Throughput / delay / utilization — Core performance metrics.
- Workload model — Characterization of offered traffic.
- Tail latency — High percentiles (p99); what users experience.

## Example / Illustration

- Goal: will a 1 Gbps link carry 800 Mbps of video traffic? Metric: 99th-percentile delay < 50 ms. Technique: M/M/1 queueing model first (fast), then simulation to validate, then measurement on a testbed.

## Lesson Summary

- Three techniques: model, simulate, measure.
- Metrics: throughput, delay, utilization, loss.
- Distributions beat averages; validate everything.

### Exam Pointers — Very Important

★ The three techniques with trade-offs, and 'averages lie' (tail latency), are standard opening questions.

## Lesson 2: Probability Foundations for Networks

### Learning Objectives

- Use PMF/PDF/CDF, expectation, variance
- Apply memoryless property
- Relate Poisson counts to Exponential times
- Use conditioning and PASTA

Random variables: discrete (counts — packets in a buffer) vs continuous (delay times). PMF/PDF, CDF, expectation  $E[X]$ , variance  $\text{Var}(X)$ . Key distributions: Bernoulli/Binomial (successes in  $n$  trials), Geometric (trials until first success — memoryless in discrete time), Poisson (counts in an interval), Exponential (interarrival/service times — memoryless in continuous time), and Normal (via CLT for aggregates).

Memoryless property:  $P(X > s+t \mid X > s) = P(X > t)$  — the exponential 'forgets' elapsed time, which is why it's the default for interarrival times (though real traffic is often burstier). Sums: Erlang- $k$  (sum of  $k$  exponentials — staged service), and the CLT says sums of many i.i.d. variables look Normal. Transform methods: moment-generating functions give moments; Laplace transforms solve queueing equations.

Conditioning and total expectation:  $E[X] = E[E[X|Y]]$  — decompose complex systems by conditioning on states. PASTA (Poisson Arrivals See Time Averages): with Poisson arrivals, an arriving packet sees the steady-state distribution — enormously simplifying analysis. These tools underpin every queueing result that follows.

### Key Terms & Definitions

- Memoryless — Exponential/Geometric forget elapsed time.
- Poisson process — Counts  $\sim$  Poisson; interarrivals  $\sim$  Exponential.
- PASTA — Poisson arrivals see time averages.
- Erlang- $k$  — Sum of  $k$  exponentials; staged service.

### Example / Illustration

- Packet interarrivals  $\sim \text{Exp}(\text{rate } 100/\text{s})$ .  $P(\text{interarrival} > 20 \text{ ms}) = e^{(-100 \times 0.02)} = e^{-2} \approx 0.135$ . Memoryless: still 0.135 even after waiting 10 ms already.

### Lesson Summary

- Exponential  $\Leftrightarrow$  memoryless  $\Leftrightarrow$  Poisson arrivals.
- PASTA simplifies arriving-packet analysis.
- Conditioning decomposes complex models.

### Exam Pointers — Very Important

- ★ Memoryless property statements and Poisson/Exponential relationships are heavily tested.

## Lesson 3: Markov Chains

### Learning Objectives

- Define DTMC and CTMC
- Write transition matrices and rate diagrams
- Solve for steady-state distributions
- Model channels/protocols as Markov chains

A stochastic process  $\{X(t)\}$  evolves randomly in time. Markov property: the future depends only on the present state, not the history —  $P(X_{n+1}|X_n, \dots, X_0) = P(X_{n+1}|X_n)$ . Discrete-time Markov chains (DTMC): transition matrix  $P$  with  $P_{ij} = P(X_{n+1}=j|X_n=i)$ ;  $n$ -step transitions via  $P^n$ . Continuous-time (CTMC): transition rates  $q_{ij}$ ; sojourn in state  $i \sim \text{Exp}(\text{total outgoing rate})$ .

Steady state:  $\pi = \pi P$  (DTMC) or  $\pi Q = 0$  (CTMC) with  $\sum \pi = 1$  — the long-run fraction of time in each state, when the chain is ergodic (irreducible, positive recurrent, aperiodic for DTMC). Balance equations: rate in = rate out per state (global balance); for birth-death processes, detailed balance between neighbors suffices. Classification: transient vs recurrent states; absorbing chains (fundamental matrix  $N = (I - Q)^{-1}$  gives expected visits).

Applications: Gilbert-Elliott channel model (good/bad states for bursty errors), TCP state machines, protocol verification, and buffer-occupancy modeling. Solving: write balance equations, solve linear system — small chains by hand, large ones numerically.

### Key Terms & Definitions

- Markov property — Future depends only on present.
- Balance equations —  $\pi = \pi P$  or  $\pi Q = 0$ ; rate in = rate out.
- Ergodic — Long-run steady state exists and is unique.
- Gilbert-Elliott — Two-state bursty-error channel model.

### Example / Illustration

- Link: Good  $\rightarrow$  Bad rate 0.1/s, Bad  $\rightarrow$  Good 0.9/s. Balance:  $0.1\pi_G = 0.9\pi_B$ ,  $\pi_G + \pi_B = 1 \rightarrow \pi_G = 0.9$ ,  $\pi_B = 0.1$ . Channel available 90% of time.

### Lesson Summary

- Markov: memoryless state evolution.
- Steady state from balance equations.
- Gilbert-Elliott models bursty errors.

### Exam Pointers — Very Important

- ★ Writing and solving balance equations for a 2-3 state chain is a standard numerical.

## Lesson 4: The M/M/1 Queue

### Learning Objectives

- Read Kendall notation
- Derive M/M/1 steady-state distribution
- Compute  $L$ ,  $W$ ,  $L_q$ ,  $W_q$
- Analyze finite-buffer M/M/1/K

Kendall notation  $A/B/c/K/m/Z$ : arrival process / service process / servers / capacity / population / discipline.  $M$  = Markovian (Poisson arrivals/Exponential service),  $D$  = deterministic,  $G$  = general.  $M/M/1$ : Poisson arrivals rate  $\lambda$ , one exponential server rate  $\mu$ , infinite buffer, FCFS. Stability:  $\rho = \lambda/\mu < 1$  — else the queue grows forever. This is the atom of queueing theory.

$M/M/1$  results:  $P(n \text{ in system}) = (1-\rho)\rho^n$  (geometric!); mean number  $L = \rho/(1-\rho)$ ; mean time in system  $W = 1/(\mu-\lambda)$ ; mean waiting in queue  $W_q = \rho/(\mu-\lambda)$ ; mean queue length  $L_q = \rho^2/(1-\rho)$ . The nonlinear blow-up as  $\rho \rightarrow 1$  is the key insight: at 90% utilization,  $L = 9$ ; at 50%,  $L = 1$ . Dimensioning rule: keep  $\rho \leq 0.7-0.8$  for acceptable delay.

Derivation sketch: birth-death with birth rate  $\lambda$ , death rate  $\mu$ ; detailed balance  $\lambda\pi_{n-1} = \mu\pi_n$  gives the geometric distribution. PASTA lets arrivals see this same distribution.  $M/M/1/K$  (finite buffer): same balance with reflecting boundary; loss probability =  $\pi_K$  — the Erlang-B-like blocking result for buffers.

### Key Terms & Definitions

- Kendall notation —  $A/B/c/K/m/Z$  queue descriptor.
- $\rho = \lambda/\mu$  — Utilization; stability needs  $\rho < 1$ .
- $M/M/1$  — Poisson arrivals, exponential service, 1 server.
- Blocking — Finite-buffer loss:  $\pi_K$ .

### Example / Illustration

- $\lambda = 800$  pkt/s,  $\mu = 1000$  pkt/s:  $\rho = 0.8$ ,  $L = 4$  packets,  $W = 1/200 = 5$  ms,  $W_q = 4$  ms. At  $\lambda = 950$ :  $\rho = 0.95$ ,  $W = 20$  ms — 4× delay for 19% more load!

### Lesson Summary

- M/M/1: geometric occupancy,  $L = \rho/(1-\rho)$ ,  $W = 1/(\mu-\lambda)$ .
- Delay explodes as  $\rho \rightarrow 1$ ; dimension below  $\sim 0.8$ .
- Finite buffers lose packets:  $\pi_K$ .

### Exam Pointers — Very Important

- ★ M/M/1 formulas ( $L$ ,  $W$ ,  $W_q$ ) and the  $\rho < 1$  stability condition are the most-tested formulas in the course.

## Lesson 5: Little's Law, M/M/c, and M/G/1

### Learning Objectives

- Apply Little's law to systems and subsystems
- Dimension M/M/c with Erlang-C
- Use P-K for M/G/1
- Analyze priority queues

Little's law:  $L = \lambda W$  — mean number in system = arrival rate  $\times$  mean time in system. Astonishingly general: holds for any system in steady state, any arrival/service distributions, any discipline — even non-queueing systems (a highway, a hard disk). Proof intuition: each arrival pays \$1/sec while in system; total payment rate =  $L = \lambda W$ . Apply to subsystems too:  $L_q = \lambda W_q$  for the queue alone.

M/M/c (Erlang-C):  $c$  identical exponential servers. Erlang-C formula  $C(c, \rho) = \text{probability an arrival waits (all servers busy)}$ ;  $W_q = C(c, \rho)/(c\mu - \lambda)$  where  $\rho = \lambda/\mu$ . Used for call centers, multi-core packet processors, server farms. Dimensioning: choose  $c$  so that  $P(\text{wait})$  or  $W_q$  meets SLA — the classic capacity-planning calculation.

M/G/1: general service times — Pollaczek-Khinchine:  $L_q = \lambda^2 E[S^2]/(2(1-\rho))$ . Insight: variance hurts! For fixed mean service, higher variance  $\rightarrow$  longer queues. M/D/1 (deterministic service):  $L_q = \rho^2/(2(1-\rho))$  — exactly half the M/M/1 queue. Priority queues: non-preemptive priority PK formula per class; preemptive-resume for strict priority. Takeaway: variability is the enemy of delay.

### Key Terms & Definitions

- Little's law —  $L = \lambda W$ ; universal steady-state relation.
- Erlang-C —  $P(\text{wait})$  and  $W_q$  for M/M/c.
- Pollaczek-Khinchine — M/G/1:  $L_q = \lambda^2 E[S^2]/(2(1-\rho))$ .
- Variance penalty — Higher service variance  $\rightarrow$  longer queues.

### Example / Illustration

- Server farm:  $\lambda = 90/s$ ,  $\mu = 20/s$  per core. Need  $c$ : try  $c = 5$  ( $\rho = 0.9$ ):  $C(5, 4.5) \approx 0.76$ ,  $Wq = 0.76/(100-90) = 76$  ms.  $c = 6$  ( $\rho=0.75$ ):  $Wq \approx 9$  ms — one more core cuts delay 8×.

### Lesson Summary

- $L = \lambda W$  universally.
- Erlang-C dimensions multi-server systems.
- P-K: variance inflates queues; deterministic halves them.

### Exam Pointers — Very Important

- ★ Little's law applications and the P-K 'variance hurts' insight are favourite conceptual questions.

## Lesson 6: Networks of Queues: Jackson and Closed Networks

### Learning Objectives

- Write traffic equations
- Apply product-form results
- Distinguish open vs closed networks
- Find bottlenecks

Real networks are networks of queues: packets traverse ingress → switching fabric → egress queues. Jackson networks: open networks of M/M/1 (or M/M/c) queues with probabilistic routing — the product-form theorem says the joint distribution factors:  $P(n_1, \dots, n_k) = \prod P(n_i)$  as if each queue were independent M/M/1 with effective arrival rate  $\lambda_i$ . Traffic equations:  $\lambda_i = \gamma_i + \sum_j \lambda_j \cdot p_{ji}$  (external + routed). Each queue needs  $\rho_i = \lambda_i / \mu_i < 1$ .

BCMP networks extend to multiple classes, general service (with restrictions), and disciplines — still product form. Mean-value analysis (MVA) computes closed-network metrics iteratively without the normalization constant. Closed networks model fixed populations: window-based flow control (TCP windows!), multiprogramming systems — throughput via MVA or Buzen's algorithm.

Limitations: product form needs exponential assumptions and no blocking/dependencies; real networks have correlated arrivals (burstiness), finite buffers, and feedback (TCP) — so use Jackson for insight and dimensioning, simulation for accuracy. Bottleneck analysis: the queue with max  $\rho_i$  dominates — find it, fix it.

### Key Terms & Definitions

- Jackson network — Open M/M/1 network; product-form solution.
- Traffic equations —  $\lambda_i = \text{external} + \text{routed arrivals}$ .
- Product form — Joint distribution factors per queue.
- Bottleneck — Max- $\rho_i$  queue dominates delay.

### Example / Illustration

- Two routers in tandem: external  $\lambda = 400/\text{s}$  to R1; all traffic R1→R2.  $\mu_1 = 500$ ,  $\mu_2 = 1000$ .  $\lambda_1 = \lambda_2 = 400$ .  $\rho_1 = 0.8$ ,  $\rho_2 = 0.4$ . R1 is the bottleneck:  $W_{\text{total}} = 1/(500-400) + 1/(1000-400) = 10 + 1.67 = 11.67 \text{ ms}$ .

### Lesson Summary

- Jackson: solve traffic equations, treat queues as independent M/M/1.
- Closed networks: MVA/Buzen; model windows.
- Bottleneck = highest  $\rho_i$ .

### Exam Pointers — Very Important

- ★ Traffic equations + per-queue  $\rho_i < 1$  stability is the standard Jackson-network question.

## Lesson 7: Discrete-Event Simulation

### Learning Objectives

- Design discrete-event simulations
- Generate random variates
- Handle warm-up and replications
- Verify and validate models

When models get intractable (general distributions, complex protocols, feedback), simulate. Discrete-event simulation: state changes only at events (arrival, departure); the event list (priority queue by time) drives a clock jumping event-to-event — vastly more efficient than time-stepped simulation. Components: system state, event routines, random-variate generation, statistics collection, initialization, and the main loop.

Random-variate generation: inverse transform ( $X = F^{-1}(U)$ ) — exponential:  $-\ln(U)/\lambda$ , acceptance-rejection, and normal via Box-Muller. Random numbers: LCGs and Mersenne Twister; seeds and streams for replications. Verification (code correct?) vs validation (model matches reality?) — both mandatory. Warm-up: discard transient (Welch's method); run length: replications for confidence intervals, batch means for single long runs.

Tools: ns-2/ns-3 (packet-level network simulation — the standard), OMNeT++, MATLAB, or Python (SimPy). Simulation study steps: formulate → model → verify/validate → design

experiments → run → analyze (confidence intervals!) → document. Never report a single run's mean as 'the answer'.

### Key Terms & Definitions

- Discrete-event — State changes only at events; event-list clock.
- Inverse transform —  $X = F^{-1}(U)$  for variate generation.
- Warm-up — Discard transient before collecting stats.
- ns-3 — Standard packet-level network simulator.

### Example / Illustration

- Simulate M/M/1: events arrival/departure; generate interarrivals  $-\ln(U)/\lambda$ ; on arrival: if server free, schedule departure at  $\text{now} + 1/\mu$ ; collect waiting times; discard first 10,000 events (warm-up); 30 replications →  $W_q \pm \text{CI}$ .

### Lesson Summary

- Event list drives the clock; generate via inverse transform.
- Warm-up + replications + CIs = credible results.
- ns-3 for networks; verify AND validate.

### Exam Pointers — Very Important

- ★ Event-list mechanics and inverse-transform generation are the core simulation exam topics.

## Lesson 8: Measurement, Workloads, and Monitoring

### Learning Objectives

- Compare active vs passive measurement
- Use SNMP, NetFlow, packet capture
- Characterize workloads (heavy tails)
- Plan capacity from measurements

Measurement grounds models in reality. Active measurement: inject probes (ping, traceroute, iperf, OWAMP) — controlled but adds load and may not reflect real traffic. Passive: observe without injecting — SNMP counters (interface octets/errors), NetFlow/IPFIX (flow records: 5-tuple, bytes, times), packet capture (tcpdump/Wireshark — full fidelity, huge volume), sFlow sampling. Each trades fidelity, overhead, and coverage.

Workload characterization: fit distributions to measured interarrivals/sizes (beware: real traffic is heavy-tailed — Pareto file sizes, self-similar/bursty aggregates — not Poisson!). The 1990s Bellcore discovery of self-similarity shattered Poisson assumptions for LAN/WAN traffic; long-range dependence means bursts at all timescales. Model with heavy-tailed (Pareto) sizes and FARIMA/MAP processes where it matters.

Monitoring architecture: collectors → time-series DB → dashboards/alerts (the observability pipeline). Benchmarking: RFC 2544 (throughput/latency/loss for network devices), realistic traffic mixes, and honest reporting (state the workload!). Capacity planning: measure → model growth → headroom analysis → upgrade before p hits the knee.

### Key Terms & Definitions

- SNMP / NetFlow — Counter polling / flow-record export.
- Self-similarity — Bursty at all timescales; Bellcore finding.
- Heavy-tailed — Pareto sizes; infinite variance effects.
- RFC 2544 — Device benchmarking methodology.

### Example / Illustration

- SNMP polls show link at 70% avg utilization but NetFlow reveals 5-min bursts at 98% — the average hid the congestion; upgrade or QoS-shape the bursty class.

### Lesson Summary

- Probe actively, observe passively; combine both.
- Real traffic is bursty/self-similar, not Poisson.
- Measure → characterize → plan headroom.

### Exam Pointers — Very Important

- ★ Active vs passive measurement and the self-similarity discovery are key measurement questions.

## Lesson 9: TCP, QoS, and Wireless Performance

### Learning Objectives

- Model TCP throughput (Mathis)
- Explain AIMD and fairness
- Compare IntServ/DiffServ/MPLS
- Analyze wireless contention

TCP performance: congestion control (slow start, AIMD — additive increase/multiplicative decrease) creates the sawtooth; throughput  $\approx c \cdot \text{MSS} / (\text{RTT} \cdot \sqrt{p})$  (Mathis formula — loss probability  $p$  dominates!). Bufferbloat: oversized buffers inflate delay without helping throughput — fq\_codel/SQM fixes. Fairness: Jain's index  $(\sum x)^2 / (n \sum x^2)$  quantifies it; AIMD converges to fairness. Wireless: 802.11 DCF contention (CSMA/CA backoff), hidden terminals (RTS/CTS), rate adaptation — model as coupled queues with collision probabilities.

QoS: IntServ (per-flow RSVP reservations — doesn't scale), DiffServ (per-class PHBs: EF for voice, AF for assured, BE for rest — scales, deployed), MPLS-TE (explicit paths, traffic engineering). Scheduling: WFQ/GPS (fair queueing approximates bit-by-bit round robin), strict priority (starves low classes), RED/WRED (early drop signals TCP before overflow).

Case pattern for exams: identify bottleneck → model as queue → compute delay/loss → propose fix (more capacity, shaping, scheduling, TCP tuning). Data-center specifics: incast (many-to-one bursts), ECN marking, DCTCP — modern variants of the same queueing story.

### Key Terms & Definitions

- Mathis formula — TCP throughput  $\approx \text{MSS}/(\text{RTT}\sqrt{p})$ .
- AIMD — Additive increase, multiplicative decrease.
- DiffServ — Scalable per-class QoS (EF/AF/BE).
- Bufferbloat — Oversized buffers → delay; fix with fq\_codel.

### Example / Illustration

- RTT 100 ms, MSS 1460 B,  $p = 1\%$ : throughput  $\approx 1.22 \times 1460 / (0.1 \times 0.1) \approx 178$  kB/s — tiny! Halve  $p$  → double throughput. Loss, not bandwidth, is the lever.

### Lesson Summary

- TCP throughput  $\propto 1/(\text{RTT}\sqrt{p})$ ; cut loss first.
- DiffServ scales; IntServ doesn't.
- Same queueing story: find the bottleneck.

### Exam Pointers — Very Important

- ★ Mathis formula and DiffServ vs IntServ are the favourite applied-protocol questions.

## Lesson 10: Birth-Death Processes and Erlang-B

### Learning Objectives

- Solve birth-death processes
- Apply detailed balance
- Use Erlang-B for loss systems
- Model finite populations

Many queues are birth-death processes: from state  $n$ , only transitions to  $n+1$  (birth, rate  $\text{Ln}$ ) or  $n-1$  (death, rate  $\text{Mn}$ ) are possible. Detailed balance:  $\text{Ln} \cdot \text{Pn} = \text{M}_{n+1} \cdot \text{P}_{n+1}$  — flow across each cut balances. Solving recursively:  $\text{Pn} = \text{P0} \cdot \text{product of } (\text{Li}/\text{M}_{i+1})$ , with  $\text{P0}$  from normalization.  $\text{M}/\text{M}/1$  is the special case  $\text{Ln} = \text{L}$ ,  $\text{Mn} = \text{M}$ ;  $\text{M}/\text{M}/c$  has  $\text{Mn} = \min(n, c) \cdot \text{M}$ ;  $\text{M}/\text{M}/\text{infinity}$  (infinite servers, e.g. think-time modeling) has  $\text{Mn} = n \cdot \text{M}$  giving Poisson occupancy!

State-dependent rates model reality: discouragement ( $\lambda_n$  decreasing -- users balk at long queues), and machine-repair models (finite population  $m$ :  $\lambda_n = (m-n)*\lambda$ ). The machine-repairman  $M/M/c/m/m$  is the classic finite-population model for a technician pool serving  $m$  machines.

Erlang-B ( $M/M/c/c$  loss system): blocking probability  $B(c,a) = (a^c/c!)/\sum(a^k/k!)$  -- the telephone trunking formula, also used for optical networks and cloud admission control. Erlang-B assumes blocked calls are lost (cleared); Erlang-C assumes they wait. Know which fits: lost goes to B, waiting goes to C.

### Key Terms & Definitions

- Birth-death — Only  $n$  to  $n+1$  or  $n-1$  transitions.
- Detailed balance —  $\lambda_n * P_n = \mu_{n+1} * P_{n+1}$ .
- Erlang-B — Blocking probability for loss systems  $M/M/c/c$ .
- Machine-repair — Finite-population  $M/M/c/m/m$  model.

### Example / Illustration

- $M/M/\infty$  with  $L = 10/s$ ,  $M = 2/s$ : occupancy is Poisson with mean  $a = 5$ .  $P(8 \text{ busy}) = e^{-5} * 5^8/8! = 0.065$ . Models 'how many users are simultaneously idle/thinking'.

### Lesson Summary

- Detailed balance solves all birth-death chains.
- Erlang-B: blocked = lost; Erlang-C: blocked = waits.
- State-dependent rates model balking and finite users.

### Exam Pointers — Very Important

- ★ Deriving  $P_n$  via detailed balance for a state-dependent case is a strong exam numerical.

## Lesson 11: Operational Laws and Bottleneck Analysis

### Learning Objectives

- Apply operational laws
- Infer service demands from measurements
- Compute bottleneck bounds
- Use the interactive response-time law

Operational laws relate measurable quantities without stochastic assumptions -- they hold for any system in flow balance. Utilization law:  $U_i = X_i * S_i$  (utilization = throughput x mean service time). Forced flow law:  $X_i = V_i * X_0$  (device throughput = visits x system throughput). Service demand law:  $D_i = V_i * S_i$ . Bottleneck law: the device with max  $D_i$  saturates first;  $X_{\max} = 1/D_{\max}$ .

Little's law (operational form):  $Q_i = X_i * R_i$ . Interactive response-time law:  $R = M/X_0 - Z$  (M users, think time Z) -- response time from throughput alone! Asymptotic bounds: with N jobs,  $X(N) \leq \min(1/D_{\max}, N/(D+Z))$  -- achievable by bottleneck analysis without solving the network.

Use: from measured utilizations and throughputs, infer service demands ( $D_i = U_i/X_i$ ) -- then predict: what if load doubles? This 'operational analysis' (Denning/Buzen) is the practitioner's shortcut around full queueing solutions -- and a favourite exam topic because it needs only arithmetic.

### Key Terms & Definitions

- Utilization law —  $U_i = X_i * S_i$ .
- Bottleneck law —  $X_{\max} = 1/D_{\max}$ ; max- $D_i$  device saturates first.
- Service demand —  $D_i = V_i * S_i$ ; total service per job.
- Asymptotic bounds —  $X(N) \leq \min(1/D_{\max}, N/(D+Z))$ .

### Example / Illustration

- Devices: CPU  $D = 0.02$  s, disk  $D = 0.05$  s. Bottleneck = disk;  $X_{\max} = 1/0.05 = 20$  jobs/s. At  $X = 15$ :  $U_{\text{disk}} = 0.75$ ,  $U_{\text{cpu}} = 0.3$ . Double load -- disk saturates at 20/s; CPU never the issue.

### Lesson Summary

- Operational laws need no distributions.
- Bottleneck = max service demand; bounds throughput.
- Infer demands from  $U_i$ ,  $X_i$  -- then predict.

### Exam Pointers — Very Important

- ★ Bottleneck bounds  $X \leq 1/D_{\max}$  with a numerical example is a classic question.

## Lesson 12: Closed Networks and Mean Value Analysis

### Learning Objectives

- Define closed queueing networks
- Execute MVA iterations
- Model window flow control
- Size systems with MVA

Closed networks have a fixed population N circulating (TCP windows, terminal systems, multiprogramming). No external arrivals -- throughput is determined by the network itself. Mean Value Analysis (MVA) computes exact mean metrics iteratively: start with  $N=0$  (empty queues), then for  $n = 1..N$  apply:  $R_i(n) = D_i * (1 + Q_i(n-1))$  (an arrival sees n-1 jobs),  $X_0(n) = n / (Z + \sum R_i(n))$ ,  $Q_i(n) = X_0(n) * R_i(n)$ . Exact for product-form

networks, no normalization constant needed!

Buzen's convolution algorithm is the alternative exact method; MVA is preferred for intuition and extensions (approximate MVA for non-product-form). Multiclass MVA iterates over class populations.

Applications: sizing server pools (how many cores for N concurrent sessions?), TCP window tuning (window = population!), database connection pools. The window-flow-control insight: a TCP connection with window W over a path is a closed network with  $N = W$  packets -- throughput =  $W/RTT$  when uncongested.

### Key Terms & Definitions

- Closed network — Fixed population N; no external arrivals.
- MVA — Iterative exact means; arrival sees n-1 jobs.
- Window = population — TCP window as closed-network N.
- Buzen's algorithm — Convolution method for closed networks.

### Example / Illustration

- CPU  $D=0.02$ , disk  $D=0.05$ ,  $Z=0$ ,  $N=1$ :  $R_{cpu}=0.02$ ,  $R_{disk}=0.05$ ,  $X=1/0.07=14.3/s$ .  $N=2$ : iterate with Q from  $N=1...$  X saturates near 20/s (bottleneck).

### Lesson Summary

- Closed nets: fixed N; MVA iterates  $n=1..N$ .
- $R_i(n) = D_i(1+Q(n-1))$ ;  $X = n/(Z+\sum R)$ .
- TCP windows are closed populations.

### Exam Pointers — Very Important

- ★ One MVA iteration by hand for a 2-device network is a standard exam numerical.

## Lesson 13: Random Numbers and Variate Generation

### Learning Objectives

- Compare RNG algorithms
- Generate exponential/normal/Pareto/discrete variates
- Test RNG quality
- Use streams and seeds properly

Simulation needs  $U(0,1)$  randomness: linear congruential generators (LCG:  $x_{n+1} = (a \cdot x_n + c) \bmod m$  -- fast but short period), Mersenne Twister (period  $2^{19937}-1$ , the Python/numpy default), and modern counter-based generators (Philox) for parallel streams. Requirements: long period, uniformity, independence, reproducibility (seeds!),

and multiple streams (common random numbers for comparing configurations -- variance reduction!).

Inverse transform:  $X = F^{-1}(U)$ . Exponential:  $-\ln(1-U)/L$ . Weibull, Pareto: closed forms exist. Normal: Box-Muller (two uniforms to two normals) or ziggurat (fast). Discrete: table lookup on CDF. Acceptance-rejection: for densities without closed-form inverses -- propose from an envelope, accept with probability ratio. Composition and convolution build complex variates from simple ones (Erlang = sum of exponentials).

Testing RNGs: chi-square uniformity, runs tests, autocorrelation -- never trust an untested generator. Seeds: set explicitly for reproducibility; use separate streams per randomness source (arrivals vs services) so model changes don't scramble all randomness.

### Key Terms & Definitions

- LCG / Mersenne Twister — Classical / modern uniform RNGs.
- Inverse transform —  $X = F^{-1}(U)$ .
- Acceptance-rejection — Sample via envelope; accept probabilistically.
- Common random numbers — Shared streams sharpen config comparisons.

### Example / Illustration

- Pareto(alpha, xm):  $F(x) = 1 - (x_m/x)^\alpha$ , so  $X = x_m \cdot (1-U)^{-1/\alpha}$ . File sizes:  $x_m = 1$  KB,  $\alpha = 1.2$  -- heavy tail:  $P(X > 1 \text{ MB}) = (1/1000)^{1.2} = 0.00025$ .

### Lesson Summary

- Mersenne Twister default; seed everything.
- Inverse transform first; acceptance-rejection as fallback.
- Common random numbers = free variance reduction.

### Exam Pointers — Very Important

- ★ Inverse-transform derivations (exponential, Pareto) are standard; state the common-random-numbers idea.

## Lesson 14: Simulation Output Analysis

### Learning Objectives

- Build confidence intervals
- Choose replications vs batch means
- Remove warm-up bias
- Compare configurations statistically

A simulation produces random output -- report it as such. Replications: run  $n$  independent runs (different seeds) giving sample mean  $\pm t_{n-1} * s/\sqrt{n}$  confidence interval. More replications narrow the CI by  $\sqrt{n}$ . Batch means: one long run split into  $k$  batches (after warm-up); treat batch means as quasi-independent; tune batch size until autocorrelation dies. Never report point estimates without CIs.

Warm-up: Welch's method (average transient across replications, find where it flattens). Initial bias inflates early stats -- always discard. Comparing configurations: paired-t on common-random-number differences, or Bonferroni for multiple comparisons. Variance reduction: antithetic variates, control variates, importance sampling (for rare events like  $10^{-9}$  loss probabilities -- naive simulation would need  $10^{11}$  packets!).

Rare-event simulation matters in networking: backbone loss rates, optical burst blocking. Importance sampling biases the simulation toward the rare event, then un-biases the estimator -- the standard technique for probabilities below  $\sim 10^{-6}$ .

### Key Terms & Definitions

- Replications — Independent runs; CI via t-distribution.
- Batch means — One long run,  $k$  batches, autocorrelation-aware.
- Warm-up — Discard transient (Welch's method).
- Importance sampling — Bias toward rare events; un-bias estimator.

### Example / Illustration

- 10 replications of  $W_q$ : mean 4.2 ms,  $s = 0.5$ . 95% CI:  $4.2 \pm 2.262 * 0.5/\sqrt{10} = 4.2 \pm 0.36$  ms. Want  $\pm 0.1$ ? Need about  $(0.36/0.1)^2 * 10 = 130$  replications.

### Lesson Summary

- Report mean  $\pm$  CI, always.
- Warm-up discarded; replications or batch means.
- Importance sampling for rare events.

### Exam Pointers — Very Important

- ★ Constructing a CI from replication data is a very common numerical.

## Lesson 15: Experimental Design and Benchmarking

### Learning Objectives

- Design factorial experiments
- Avoid one-factor-at-a-time
- Specify honest benchmarks
- Present results with integrity

Comparing systems fairly needs experimental design. Full factorial: test all combinations of  $k$  factors at 2 levels ( $2^k$  runs) -- estimates each factor's effect and interactions via ANOVA; fractional factorials screen many factors cheaply. One-factor-at-a-time misses interactions and wastes runs -- the classic amateur mistake. Randomize run order; block nuisance factors.

Benchmarking discipline: define the workload (traffic mix!), the metrics, the environment, and report all three -- 'System A is 2x faster' is meaningless without them. Benchmarking (vendor benchmarks tuned to flatter) is the enemy: use independent, realistic workloads. RFC 2544/2889 for devices; application benchmarks for systems.

Presenting results: plot with error bars/CIs, use appropriate scales (log for wide ranges), never truncate axes deceptively, and state assumptions. The evaluation chapter of any thesis/paper follows: goals, metrics, workload, method, results with CIs, threats to validity. Learn this template -- it structures exam essays too. 

### Key Terms & Definitions

- Factorial design — All factor combinations; finds interactions.
- ANOVA — Attributes variance to factors.
- Benchmarking — Vendor-tuned misleading benchmarks.
- Threats to validity — Honest limits of your conclusions.

### Example / Illustration

- Factors: buffer size (2), scheduling (2), load (2):  $2^3 = 8$  runs. ANOVA shows scheduling x load interaction: WFQ helps only at high load -- one-at-a-time would miss this.

### Lesson Summary

- Factorial designs find interactions; OFAT misses them.
- Benchmarks need workload + metrics + environment.
- Error bars, honest scales, stated assumptions.

### Exam Pointers — Very Important

- ★ 'Why is one-factor-at-a-time bad?' and benchmark-reporting requirements are favourite theory questions.

## Lesson 16: Wireless LAN Performance Analysis

### Learning Objectives

- Explain DCF/BEB operation
- Use Bianchi's model qualitatively
- Analyze hidden terminals and RTS/CTS
- Describe Wi-Fi 6 improvements

802.11 DCF: CSMA/CA -- sense the channel; if idle for DIFS, transmit; if busy, backoff: random in  $[0, CW]$ , doubling CW on collision (binary exponential backoff, BEB), resetting on success. Bianchi's model: each station transmits with probability  $\tau$  in a slot; coupled fixed-point equations for  $\tau$  and collision probability  $p$  give throughput as a function of  $n$  stations. Key result: throughput degrades with  $n$  (collisions waste airtime).

Hidden terminals: A and C both hear the AP but not each other, causing collisions at the AP; RTS/CTS reserves the channel (at overhead cost). Rate adaptation (match modulation to SNR) interacts with performance -- slow stations hog airtime (the 'performance anomaly': one 1 Mbps station drags the cell). Per-station throughput is about fair airtime share, not fair bit share.

Modern: 802.11n/ac/ax (Wi-Fi 6) -- aggregation (A-MPDU), MIMO, OFDMA (ax splits channels among stations -- scheduled, cellular-like), MU-MIMO. Each generation attacks overhead and contention -- the performance story is 'amortize the fixed costs of contention over bigger transmissions'.

### Key Terms & Definitions

- CSMA/CA — Carrier sense, collision avoidance via backoff.
- BEB — Binary exponential backoff on collision.
- Hidden terminal — Mutually deaf senders collide at receiver.
- OFDMA — 802.11ax splits channel among stations.

### Example / Illustration

- 10 stations, Bianchi model:  $\tau = 0.05$ ,  $p = 0.4$  -- 40% of attempts collide; throughput ~60% of raw rate. RTS/CTS helps if packets are large; hurts if small (overhead).

### Lesson Summary

- BEB stabilizes contention; throughput falls with  $n$ .
- Hidden terminals need RTS/CTS (overhead trade-off).
- Aggregation + OFDMA amortize contention costs.

### Exam Pointers — Very Important

- ★ BEB mechanics and the hidden-terminal problem are the core 802.11 questions.

## Lesson 17: Data-Center and Cloud Performance

### Learning Objectives

- Explain incast and mitigations
- Fight tail latency at scale
- Compare load-balancing schemes
- Evaluate cloud cost-performance

Data-center networks: incast -- many servers replying to one (e.g. MapReduce shuffle) synchronize bursts that overflow shallow buffers, causing throughput collapse; solutions: ECN + DCTCP (mark early, react proportionally), pacing, randomized retransmission. Tail latency at scale: a job touching 1,000 servers waits for the slowest -- p99 of each must be excellent; techniques: hedged requests (send backup after 95th percentile), careful GC/scheduling.

Load balancing: ECMP hashing (flow-level, can collide -- 'hash polarization'), consistent hashing (minimal remap on changes), power-of-two-choices (sample 2 servers, pick less loaded -- exponentially better than random). Modeling: join-the-shortest-queue, supermarket models -- d choices give double-exponential tail improvement.

Cloud performance evaluation: noisy neighbors (measure with interference!), autoscaling dynamics (model as birth-death with setup times -- M/M/c/Setup), and cost-performance trade-offs (the real metric: dollars/request at p99 SLA). Serverless: cold starts as setup-time queues. The unifying theme: classic queueing models, re-applied at warehouse scale.

### Key Terms & Definitions

- Incast — Many-to-one synchronized bursts; buffer overflow.
- Hedged requests — Backup request after a delay; cuts tail latency.
- Power of two choices — Sample 2, pick better; huge tail gains.
- Noisy neighbor — Cloud interference; measure under contention.

### Example / Illustration

- 1,000-way fan-out, each server p99 = 10 ms, p50 = 2 ms: job latency is about the max of 1,000 draws -- far into the tail (~50 ms+). Hedged requests at 8 ms cut job p99 dramatically for 5% extra load.

### Lesson Summary

- Scale turns p99 into the mean experience.
- Hedging and 2-choices tame tails.
- Classic queues, warehouse scale.

### Exam Pointers — Very Important

★ Incast and hedged requests are the modern 'scale' questions examiners love.

## Lesson 18: Revision: Performance Evaluation Drill

### Learning Objectives

- Run a timed mixed-topic drill
- Apply the numerical method
- Recall all formulas from memory
- Self-assess readiness

Drill (two per lesson): (1) choose technique for a scenario; (2) tail-latency vs mean; (3) memoryless computation; (4) PASTA application; (5) 2-state chain steady state; (6) Gilbert-Elliott availability; (7) M/M/1 L/W/Wq; (8) rho sensitivity (0.5 to 0.95); (9) M/M/1/K blocking; (10) detailed-balance derivation; (11) Erlang-B blocking; (12) operational-laws bottleneck; (13) Little's law on subsystems; (14) Erlang-C dimensioning; (15) P-K variance comparison; (16) Jackson traffic equations; (17) MVA one iteration; (18) event-list trace; (19) inverse-transform variate; (20) CI from replications; (21) factorial design; (22) BEB/hidden terminal; (23) Mathis throughput; (24) incast mitigation. Formula sheet:  $L = \lambda W$ ; M/M/1 (L, W, Wq, Lq); Erlang-B/C; P-K; operational laws; MVA; Mathis; Jains index; balance equations.

Method for numericals: write given ( $\lambda$ ,  $\mu$ , c), check stability, pick formula, compute, sanity-check ( $W > 0$ ,  $\rho < 1$ , delay reasonable). For design questions: metrics, workload, technique, model, validate, recommend with numbers.

Final checklist: three techniques; memoryless; PASTA; Markov balance; Kendall; M/M/1; Little; Erlang-B/C; P-K; Jackson; MVA; bottleneck; event simulation; ns-3; SNMP/NetFlow; self-similarity; Mathis; DiffServ; incast. State each formula cold -- this paper rewards fluency.

### Key Terms & Definitions

- Formula sheet —  $L = \lambda W$ ; M/M/1; Erlang-B/C; P-K; MVA; Mathis -- one page.
- Numerical method — Given, stability, formula, compute, sanity-check.
- Checklist — The eighteen must-know topics above.
- Fluency — State each formula cold.

### Lesson Summary

- Twenty-four drills across all lessons.
- Given-stability-formula-compute-check wins marks.
- Formula sheet + checklist = final prep.

**Exam Pointers — Very Important** 

★ Time yourself: 24 drills, 5 minutes each, formula sheet from memory.