

Model Structure			
<b>MATLAB</b> model = Network('name'); source = Source(model,'Src'); queue = Queue(model,'Q', SchedStrategy.FCFS); sink = Sink(model,'Snk'); oclass = OpenClass(model,'C1'); source.setArrival(oclass,Exp(1)); queue.setService(oclass,Exp(2)); model.link(Network.serialRouting( source,queue,sink)); MVA(model).avgTable()	<b>Java</b> model = new Network("name"); source = new Source(model,"Src"); queue = new Queue(model,"Q", SchedStrategy.FCFS); sink = new Sink(model,"Snk"); oclass = new OpenClass(model,"C1"); source.setArrival(oclass,new Exp(1)); queue.setService(oclass,new Exp(2)); model.link(Network.serialRouting( source,queue,sink)); new SolverMVA(model).getAvgTable()	<b>Kotlin</b> val model = Network("name") val source = Source(model,"Src") val queue = Queue(model,"Q", SchedStrategy.FCFS) val sink = Sink(model,"Snk") val oclass = OpenClass(model,"C1") source.setArrival(oclass,Exp(1.0)) queue.setService(oclass,Exp(2.0)) model.link(Network.serialRouting( source,queue,sink)) SolverMVA(model).avgTable()	<b>Python</b> model = Network("name") source = Source(model,"Src") queue = Queue(model,"Q", SchedStrategy.FCFS) sink = Sink(model,"Snk") oclass = OpenClass(model,"C1") source.setArrival(oclass,Exp(1)) queue.setService(oclass,Exp(2)) model.link(Network.serialRouting( source,queue,sink)) SolverMVA(model).getAvgTable()

Node Types

Cache	Caching station
ClassSwitch	Class switching
Delay	Infinite server
Fork	Forks jobs
Join	Joins jobs
Place	Petri net place
Queue	Queueing station
Router	Routing node
Sink	Job departures (open)
Source	Job arrivals (open)
Transition	Petri net transition

Job Classes

ClosedClass(model, 'name', N, refStat)  
OpenClass(model, 'name')  
SelfLoopingClass(model, 'name', N, ref)

Tip: The reference station (refStat/ref) is where response time and throughput are measured for closed classes. For SelfLoopingClass, jobs remain at the reference station.

Distributions

APH( $\alpha$ , T)	Acyclic PH
Coxian( $\mu$ , $\phi$ )	Coxian
Det( $v$ )	Deterministic
Disabled	None
Erlang( $\lambda$ , k)	Erlang-k
Exp( $\lambda$ )	Exponential
Gamma( $\alpha$ , $\beta$ )	Gamma
HyperExp( $p$ , $\lambda_1$ , $\lambda_2$ )	Hyper-exp
Lognormal( $\mu$ , $\sigma$ )	Log-normal
MAP(D0, D1)	MAP
MMPP2( $\lambda_1$ , $\lambda_2$ , $\sigma_1$ , $\sigma_2$ )	MMPP(2)
Pareto( $\alpha$ , k)	Pareto
PH( $\alpha$ , T)	Phase-type
Replayer(trace)	Trace replay
Uniform(a, b)	Uniform
Weibull( $\alpha$ , $\beta$ )	Weibull

Tip: Fit distributions to moments: Exp.fitMean(m), Erlang.fitMeanAndSCV(m,scv), APH.fit(mean,scv,skew).

Scheduling Strategies

DPS	Discriminatory PS
DPSPRIO	DPS with Priority
FB	Feedback (LAS)
FCFS	First-Come First-Served
FCFSPRIO	FCFS with Priority
GPS	Generalized PS
GPSPRIO	GPS with Priority
HOL	Head-of-Line Priority
INF	Infinite Servers
LCFS	Last-Come First-Served
LCFSPRIO	LCFS with Priority
LCFSPR	LCFS Preemptive
LEPT	Longest Expected PT
LJF	Longest Job First
LRPT	Longest Remaining PT
PS	Processor Sharing
PSJF	Preemptive SJF
PSPRIO	PS with Priority
SEPT	Shortest Expected PT
SETF	Shortest Elapsed Time First
SIRO	Random (SIRO)
SJF	Shortest Job First
SRPT	Shortest Remaining PT

Tip: Set DPS/GPS weights with queue.setService(class, dist, weight). Set priorities via ClosedClass(..., priority) or class.setPriority(p). Only \*PRIO policies are sensitive to priorities.

Solvers

AUTO	Auto-select best
CTMC	CTMC (exact, small)
DES	Discrete Event Sim
ENV	Random environments
FLD	Fluid/mean-field ODEs
JMT	JMT simulator
LN	Layered networks
LQNS	LQNS interface
MAM	Matrix Analytic Methods
MVA	Mean Value Analysis
NC	Normalizing Constant Methods
QNS	LQNS product-form solver
SSA	Stochastic simulation

Tip: Use listMethods() to see available methods for each solver. Use MVA for product-form networks (fastest), CTMC for exact small models, JMT/SSA for general simulation.

Solver Options

MVA(model, 'method', 'exact')  
JMT(model, 'seed', 123,  
'samples', 10000)  
SSA(model, 'samples', 5000,  
'method', 'para')

Output Metrics

ArvR	Arrival rate
QLen	Mean queue length
ResidT	Residence time (total across visits)
RespT	Response time (per visit)
Tput	Throughput
Util	Server utilization

Analysis Methods

solver.avgTable() % alias: aT  
solver.avgQLen(), avgUtil()  
solver.avgRespT(), avgTput()  
solver.avgSysRespT()  
solver.avgSysTput()  
solver.avgChainQLen()

Method Aliases

aCT	getAvgChainTable
aNT	getAvgNodeTable
aST	getAvgSysTable
aT	getAvgTable

Routing

model.link(Network.serialRouting(  
n1,n2,n3));  
n1.setProbRouting(class, n2, 0.7);  
n1.setRouting(class,  
RoutingStrategy.RROBIN);

Multi-Server & Finite Capacity

queue.setNumServers(k);  
queue.setCapacity(bufferSize);

Class Switching

cs = ClassSwitch(model, 'CS');  
P = [0.3 0.7; 0.5 0.5];  
cs.setClassSwitching(P);

Fork-Join Networks

fork = Fork(model, 'Fork');  
join = Join(model, 'Join', fork);

Layered Networks

lqn = LayeredNetwork('name');  
P = Processor(lqn,'P',1,  
SchedStrategy.PS);  
T = Task(lqn,'T',1,  
SchedStrategy.REF).on(P);  
E = Entry(lqn,'E').on(T);  
A = Activity(lqn,'A').on(T)  
.boundTo(E).setHostDemand(Exp(1));

Caching

cache = Cache(model,'Cache',  
nItems, cap, ReplacementStrategy.LRU);  
cache.setPopularity(Zipf(1.4));

Model Import/Export

model.view() % Open in JMT  
model.save('file.jsimg')  
Network.load('file.jsimg')

Tip: Check model properties: model.hasProductFormSolution() (exact MVA possible), model.getNumStations(), model.getNumClasses(), model.getStruct() (internal representation).

Routing Strategies

JSQ	Join shortest queue
KCHOICES	Power of k choices
PROB	Probabilistic routing
RAND	Random selection
RROBIN	Round-robin