LINE: Queueing Analysis Algorithms

Example Manual – R Edition

Last revision: November 28, 2026
QORE Lab, Imperial College London

Contents

1 About this manual **1**

2 Basic models **3**

2.1 Open queueing networks . . . 3

2.1.1 mam_transient_mapmap1 . . . 3

2.1.2 oqn_basic . . . 4

2.1.3 oqn_bmmapt . . . 4

2.1.4 oqn_cs_routing . . . 4

2.1.5 oqn_fourqueues . . . 4

2.1.6 oqn_mapt . . . 4

2.1.7 oqn_mapt_service . . . 6

2.1.8 oqn_mmapt . . . 6

2.1.9 oqn_multichain_cs . . . 6

2.1.10 oqn_nhpp . . . 6

2.1.11 oqn_nnpp . . . 6

2.1.12 oqn_oneline . . . 8

2.1.13 oqn_trace_driven . . . 8

2.1.14 oqn_vsinks . . . 8

2.2 Closed queueing networks . . . 8

2.2.1 cqn_bas_blocking . . . 10

2.2.2 cqn_bcmp_theorem . . . 10

2.2.3 cqn_lcfs_multiclass . . . 10

2.2.4 cqn_mmpp2_service . . . 10

2.2.5 cqn_multichain_cs . . . 10

2.2.6 cqn_multiserver . . . 12

2.2.7 cqn_multiserver_nc . . . 12

2.2.8 cqn_mvablo . . . 12

2.2.9 cqn_oneline . . . 12

2.2.10 cqn_repairmen . . . 14

2.2.11 cqn_repairmen_multi . . . 14

2.2.12 cqn_scheduling_dps . . . 14

2.2.13 cqn_threeclass_hyperl . . . 14

2.2.14 cqn_twoclass_erl . . . 14

2.2.15 cqn_twoclass_hyperl . . . 14

2.2.16 cqn_twoqueues_multi . . . 17

2.3 Mixed networks . . . 17

2.3.1 mqn_basic . . . 17

2.3.2 mqn_multichain_cs . . . 17

2.3.3 mqn_multiserver_fcfs . . . 19

2.3.4 mqn_multiserver_ps . . . 19
2.3.5 mqn_singleserver_fcfs . . . 19
2.3.6 mqn_singleserver_ps . . . 19
2.4 Class switching . . . 21
2.4.1 cs_implicit . . . 21
2.4.2 cs_multi_diamond . . . 21
2.4.3 cs_single_diamond . . . 21
2.4.4 cs_transient_class . . . 23
2.5 Priority models . . . 23
2.5.1 prio_hol_closed . . . 24
2.5.2 prio_hol_open . . . 24
2.5.3 prio_identical . . . 24
2.5.4 prio_psprio . . . 24
2.6 Fork-join networks . . . 26
2.6.1 fj_asymm . . . 26
2.6.2 fj_basic_closed . . . 26
2.6.3 fj_basic_nesting . . . 26
2.6.4 fj_basic_open . . . 28
2.6.5 fj_complex_serial . . . 28
2.6.6 fj_cs_multi_visits . . . 28
2.6.7 fj_cs_postfork . . . 28
2.6.8 fj_cs_prefork . . . 30
2.6.9 fj_deep_nesting . . . 30
2.6.10 fj_delays . . . 30
2.6.11 fj_mixed_openclosed . . . 32
2.6.12 fj_nojoin . . . 32
2.6.13 fj_route_overlap . . . 32
2.6.14 fj_serialfjs_closed . . . 34
2.6.15 fj_serialfjs_open . . . 34
2.6.16 fj_threebranches . . . 34
2.6.17 fj_tiny_closed . . . 36
2.6.18 fj_twoclasses_forked . . . 36
2.6.19 fj_variable_fanout . . . 36
2.7 Layered queueing networks . . . 38
2.7.1 lqn_basic . . . 38
2.7.2 lqn_bpmn . . . 38
2.7.3 lqn_bpmn_trace . . . 38
2.7.4 lqn_flatph . . . 41
2.7.5 lqn_fork_open_arrival . . . 41
2.7.6 lqn_init . . . 41
2.7.7 lqn_jsq . . . 43
2.7.8 lqn_moment3 . . . 43
2.7.9 lqn_multi_solvers . . . 43
2.7.10 lqn_ofbiz . . . 43
2.7.11 lqn_open_arrival . . . 46
2.7.12 lqn_paramident . . . 46
2.7.13 lqn_rrobin . . . 47
2.7.14 lqn_serial . . . 47
2.7.15 lqn_server_pools . . . 47
2.7.16 lqn_setup . . . 48

2.7.17 lqn_sockshop . . . 48
2.7.18 lqn_srvnph . . . 48
2.7.19 lqn_transient . . . 48
2.7.20 lqn_twotasks . . . 48
2.7.21 lqn_workflows . . . 51
2.8 Cache models . . . 51
2.8.1 cache_compare_replc . . . 52
2.8.2 cache_itemsize_costcap . . . 52
2.8.3 cache_mmap_rr_env . . . 52
2.8.4 cache_replc_climb . . . 52
2.8.5 cache_replc_fifo . . . 52
2.8.6 cache_replc_hlru . . . 52
2.8.7 cache_replc_lru . . . 52
2.8.8 cache_replc_qlru . . . 55
2.8.9 cache_replc_routing . . . 55
2.8.10 cache_replc_rr . . . 55
2.8.11 cache_rmf_transient . . . 56
2.8.12 cachenet_tandem . . . 56
2.8.13 retrieval_chain . . . 56
2.8.14 retrieval_default . . . 56
2.8.15 retrieval_ps . . . 56
2.8.16 retrieval_routing . . . 56
2.8.17 retrieval_simple . . . 60
2.9 Cluster models . . . 60
2.9.1 cl_basic . . . 60
2.9.2 cl_closed . . . 60
2.9.3 cl_compare . . . 62
2.9.4 cl_multiclass . . . 62
2.9.5 cl_scheduling . . . 62
2.9.6 cl_stations . . . 63
2.9.7 cl_sweep . . . 63
2.9.8 dispatch_closed . . . 63
2.9.9 dispatch_mixed . . . 63
2.9.10 dispatch_open . . . 63
2.10 Stochastic Petri nets . . . 65
2.10.1 spn_basic_closed . . . 65
2.10.2 spn_basic_open . . . 66
2.10.3 spn_closed_fourplaces . . . 66
2.10.4 spn_closed_twoplaces . . . 66
2.10.5 spn_colored_gspn . . . 66
2.10.6 spn_fluid_dae . . . 66
2.10.7 spn_fluid_onoff . . . 68
2.10.8 spn_fourmodes . . . 68
2.10.9 spn_inhibiting . . . 68
2.10.10 spn_lpbounds . . . 68
2.10.11 spn_open_sevenplaces . . . 70
2.10.12 spn_pareto_service . . . 70
2.10.13 spn_productform_nc . . . 70
2.10.14 spn_queueing_place . . . 71
2.10.15 spn_twomodes . . . 71

2.10.16 test_spn_nrm_open . . . 71
2.11 Workflow models . . . 71
2.11.1 wf_aph . . . 71
2.11.2 wf_branch . . . 73
2.11.3 wf_complex . . . 73
2.11.4 wf_erlang . . . 73
2.11.5 wf_loop . . . 73
2.11.6 wf_parallel . . . 73
2.11.7 wf_serial . . . 73
2.12 Workflow nets . . . 73
2.12.1 wfnet_and_xor . . . 74
2.12.2 wfnet_closed . . . 74
2.12.3 wfnet_nonsp_sound . . . 74
2.12.4 wfnet_pnml . . . 74
2.12.5 wfnet_resource . . . 74
2.12.6 wfnet_serial . . . 74
2.12.7 wfnet_unsound . . . 74

3 Advanced models 75
3.1 Load-dependent stations . . . 75
3.1.1 fes_aggregation . . . 75
3.1.2 fes_single_class . . . 75
3.1.3 ld_class_dependence . . . 76
3.1.4 ld_fes_multiclass . . . 76
3.1.5 ld_fes_singleclass . . . 76
3.1.6 ld_global_dependence . . . 76
3.1.7 ld_joint_dependence . . . 76
3.1.8 ld_multiserver_fcfs . . . 79
3.1.9 ld_multiserver_ps . . . 79
3.1.10 ld_multiserver_ps_twoclasses . . . 79
3.1.11 ld_whittle_bandwidth . . . 79
3.2 Finite capacity regions . . . 79
3.2.1 fcr_constraints . . . 81
3.2.2 fcr_lincon . . . 81
3.2.3 fcr_lossn . . . 81
3.2.4 fcr_mm1kdrop . . . 81
3.2.5 fcr_mm1waitq . . . 81
3.2.6 fcr_oqndrop . . . 83
3.2.7 fcr_oqnwaitq . . . 83
3.3 State probabilities . . . 83
3.3.1 statepr_aggr . . . 83
3.3.2 statepr_aggr_large . . . 83
3.3.3 statepr_allprobs_fcfs . . . 85
3.3.4 statepr_allprobs_ps . . . 85
3.3.5 statepr_sys_aggr . . . 85
3.3.6 statepr_sys_aggr_large . . . 85
3.3.7 statepr_sys_marg . . . 85
3.4 Initial state and warm start . . . 87
3.4.1 init_state_fcfs_exp . . . 87
3.4.2 init_state_fcfs_nonexp . . . 87

3.4.3 init_state_ps . . . 87
3.4.4 ldes_warmstart . . . 89
3.5 State-dependent routing . . . 89
3.5.1 sdroute_closed . . . 89
3.5.2 sdroute_jsq . . . 90
3.5.3 sdroute_krzesinski . . . 90
3.5.4 sdroute_multibranch . . . 90
3.5.5 sdroute_open . . . 91
3.5.6 sdroute_twoclasses_closed . . . 91
3.6 Cyclic polling . . . 91
3.6.1 polling_exhaustive_det . . . 91
3.6.2 polling_exhaustive_exp . . . 91
3.6.3 polling_gated . . . 93
3.6.4 polling_klimited . . . 93
3.7 Switchover times . . . 93
3.7.1 switchover_basic . . . 93
3.8 Response time distribution . . . 93
3.8.1 cdf_respt_closed . . . 95
3.8.2 cdf_respt_closed_threeclasses . . . 95
3.8.3 cdf_respt_distrib . . . 95
3.8.4 cdf_respt_open_twoclasses . . . 95
3.8.5 cdf_respt_populations . . . 95
3.9 Busy periods . . . 97
3.9.1 busyp_subnetwork . . . 97
3.10 Stochastic network calculus . . . 97
3.10.1 snc_delay_quantile . . . 98
3.10.2 snc_tandem_multiclass . . . 98
3.11 Random environments . . . 98
3.11.1 example_mapqn2renv . . . 98
3.11.2 renv_basic . . . 98
3.11.3 renv_container_terminal . . . 100
3.11.4 renv_fourstages_repairmen . . . 100
3.11.5 renv_genqn . . . 100
3.11.6 renv_lqn_attackgraph . . . 100
3.11.7 renv_lqn_twostages . . . 100
3.11.8 renv_map_fallback . . . 101
3.11.9 renv_node_breakdown . . . 101
3.11.10 renv_rotterdam_blending . . . 101
3.11.11 renv_threestages_repairmen . . . 101
3.11.12 renv_twostages_repairmen . . . 101
3.12 Layered cache-queueing models . . . 103
3.12.1 lcq_async_cache . . . 103
3.12.2 lcq_singlehost . . . 103
3.12.3 lcq_threehosts . . . 103
3.13 Pass and swap . . . 105
3.13.1 pas_closed_tandem_fig6 . . . 105
3.13.2 pas_compatibility_5class . . . 105
3.13.3 pas_cyclic_ctmc_vs_bruteforce . . . 105
3.13.4 pas_cyclic_normconst_bruteforce . . . 106
3.13.5 pas_mmk . . . 106

3.13.6 pas_nc_sampling . . . 106
3.13.7 pas_saturation . . . 106
3.13.8 pas_selfloop . . . 106
3.14 Reward models . . . 108
3.14.1 rewardModel_aggregation . . . 108
3.14.2 rewardModel_mm1k . . . 108
3.14.3 rewardModel_multiclass . . . 108
3.14.4 rewardModel_templates . . . 108
3.15 Agent models . . . 108
3.15.1 ag_closed_network . . . 110
3.15.2 ag_gnetwork . . . 110
3.15.3 ag_jackson_network . . . 110
3.15.4 ag_multiclass_closed . . . 110
3.15.5 ag_tandem_open . . . 110
3.15.6 ag_tandem_phasetype . . . 110
3.16 Large-scale models . . . 112
3.16.1 largescale_tbi . . . 112
3.17 Custom solvers . . . 112
3.17.1 solver_custom . . . 112
3.17.2 solver_custom_analyzer . . . 112

4 Model gallery **115**
4.1 Reading the names . . . 115
4.2 The catalogue . . . 115
4.2.1 gallery_aphm1 . . . 116
4.2.2 gallery_cache_lru . . . 116
4.2.3 gallery_cache_routing . . . 116
4.2.4 gallery_coxm1 . . . 118
4.2.5 gallery_cqn . . . 118
4.2.6 gallery_cqn_multiclass . . . 118
4.2.7 gallery_detm1 . . . 118
4.2.8 gallery_dm1 . . . 118
4.2.9 gallery_erldk . . . 118
4.2.10 gallery_erlerl1 . . . 118
4.2.11 gallery_erlerl1_reentrant . . . 121
4.2.12 gallery_erlm1 . . . 121
4.2.13 gallery_erlm1_ps . . . 121
4.2.14 gallery_erlm1_reentrant . . . 121
4.2.15 gallery_erlm1ps . . . 121
4.2.16 gallery_fcr . . . 121
4.2.17 gallery_fj_closed . . . 123
4.2.18 gallery_fj_open . . . 123
4.2.19 gallery_fj_quorum . . . 123
4.2.20 gallery_gamm1 . . . 123
4.2.21 gallery_hyperl1_feedback . . . 123
4.2.22 gallery_hyperl1_reentrant . . . 126
4.2.23 gallery_hyperlk . . . 126
4.2.24 gallery_hyphyp1_linear . . . 126
4.2.25 gallery_hyphyp1_reentrant . . . 126
4.2.26 gallery_hyphyp1_tandem . . . 126

4.2.27 gallery_hypm1 . . . 126
4.2.28 gallery_hypm1_reentrant . . . 128
4.2.29 gallery_lqn_basic . . . 128
4.2.30 gallery_lqn_random . . . 128
4.2.31 gallery_lqn_workflows . . . 128
4.2.32 gallery_lukumar_reentrant . . . 128
4.2.33 gallery_mapm1 . . . 128
4.2.34 gallery_mapmk . . . 128
4.2.35 gallery_mdk . . . 132
4.2.36 gallery_merl1 . . . 132
4.2.37 gallery_merl1_linear . . . 132
4.2.38 gallery_merl1_reentrant . . . 132
4.2.39 gallery_merl1_tandem . . . 132
4.2.40 gallery_merlk . . . 132
4.2.41 gallery_mhyp1 . . . 132
4.2.42 gallery_mhyp1_linear . . . 135
4.2.43 gallery_mhyp1_reentrant . . . 135
4.2.44 gallery_mhyp1_tandem . . . 135
4.2.45 gallery_mhypk . . . 135
4.2.46 gallery_mm1 . . . 135
4.2.47 gallery_mm1_feedback . . . 135
4.2.48 gallery_mm1_linear . . . 137
4.2.49 gallery_mm1_multiclass . . . 137
4.2.50 gallery_mm1_prio . . . 137
4.2.51 gallery_mm1_ps . . . 137
4.2.52 gallery_mm1_ps_feedback . . . 137
4.2.53 gallery_mm1_ps_multiclass . . . 137
4.2.54 gallery_mm1_ps_reentrant . . . 139
4.2.55 gallery_mm1_reentrant . . . 139
4.2.56 gallery_mm1_tandem . . . 139
4.2.57 gallery_mm1_tandem_multiclass . . . 139
4.2.58 gallery_mm1k . . . 139
4.2.59 gallery_mmap1 . . . 139
4.2.60 gallery_mmap1_multiclass . . . 142
4.2.61 gallery_mmapk . . . 142
4.2.62 gallery_mmk . . . 142
4.2.63 gallery_mpar1 . . . 142
4.2.64 gallery_multitier . . . 142
4.2.65 gallery_multitier_storage . . . 142
4.2.66 gallery_parm1 . . . 142
4.2.67 gallery_qn_random . . . 145
4.2.68 gallery_renv_breakdown . . . 145
4.2.69 gallery_repairmen . . . 145
4.2.70 gallery_replayerm1 . . . 145
4.2.71 gallery_um1 . . . 145

5 Discrete-time models **147**
5.1 The slotted models . . . 147
5.1.1 dt_bernoulli_loaddep . . . 147
5.1.2 dt_cycle . . . 147
5.1.3 dt_cycle_loaddep . . . 148
5.1.4 dt_cycle_multiclass . . . 148
5.1.5 dt_geogeo1 . . . 148
5.1.6 dt_geogeo1_loss . . . 148

6 Solver-specific examples **150**
6.0.1 ctmc_spn_mm1 . . . 150
6.0.2 ctmc_spn_vs_nc_closed_qn . . . 150
6.0.3 ctmc_tandem_mm1 . . . 150

7 Model interchange **153**
7.1 The fixtures . . . 153
7.1.1 json_balking_retrial . . . 153
7.1.2 json_classcap_droprule . . . 153
7.1.3 json_fcr_details . . . 153
7.1.4 json_hetero_servers . . . 155
7.1.5 json_join_deadline . . . 155
7.1.6 json_signal_classes . . . 155

8 Inference and estimation **157**
8.1 The estimators . . . 157
8.1.1 est_ekf_changepoint . . . 157
8.1.2 est_ekf_closed . . . 158
8.1.3 est_ekf_open . . . 158
8.1.4 est_erps_closed . . . 158
8.1.5 est_erps_open . . . 158
8.1.6 est_mcmc_closed . . . 158
8.1.7 est_mle_open . . . 158
8.1.8 est_qmle_closed . . . 160
8.1.9 est_trace_closed . . . 160
8.1.10 est_ubo_changepoint . . . 160
8.1.11 est_ubo_closed . . . 160
8.1.12 est_ubo_open . . . 160
8.1.13 est_ubo_variants_closed . . . 160
8.1.14 est_ubr_changepoint . . . 160
8.1.15 est_ubr_closed . . . 163
8.1.16 est_ubr_open . . . 163
8.1.17 est_vi_closed . . . 163

9 Optimization **165**
9.1 Sizing an M/M/c queue . . . 165
9.2 Exact sizing by bisection . . . 165
9.3 Continuous service rate tuning . . . 166
9.4 Load balancing across heterogeneous servers . . . 166
9.5 Population sizing in a closed network . . . 166
9.6 Robust sizing under workload scenarios . . . 166

9.7 Cost-performance tradeoff frontier . . . 167
9.8 Decomposing a multivariable problem . . . 167
9.9 The models behind the tutorials . . . 167
9.9.1 opt_bisection_sizing . . . 167
9.9.2 opt_decomposition . . . 168
9.9.3 opt_load_balancing . . . 168
9.9.4 opt_lqn_host_demand . . . 168
9.9.5 opt_pareto_frontier . . . 168
9.9.6 opt_population_sizing . . . 168
9.9.7 opt_robust_sizing . . . 168
9.9.8 opt_sensitivity_report . . . 171
9.9.9 opt_server_sizing . . . 171
9.9.10 opt_service_rate . . . 171

Chapter 1

About this manual

Read this first if you came to the R edition. R is a *binding* over the C++ engine rather than a fifth implementation, and its example inventory is correspondingly different from the other four.

- **Eight of the suite's examples have an R twin** under `r/linesolver/inst/examples/`. They are chosen to cover the R *assembler* – the part R actually owns – and not the solvers, which four other suites already measure: one FCFS open model, the same under processor sharing, the multiclass form of both, a two-stage tandem, and the first closed model needing an explicit routing matrix.
- **Every other example in this manual is still reachable from R**, because R does not need to declare a model in order to solve it: `load_model()` reads any `Network` document in the `line-model` format the other four editions write, and `capi_run()` reaches the layered and environment documents and the `.lqnx`, `.jsimg` and `.pnml` readers as well. R also declares caches, fork-joins, Petri nets, finite capacity regions, impatience and uncertain (`Prior`) service, so an unported example is one nobody has translated yet rather than one R cannot express. What R cannot *declare* is a layered network or a random environment; it can still solve both.
- **The count is a ratchet, not a target.** `r/linesolver/inst/goldens/ported.txt` lists the twins and `ratchet.txt` records the number still unported; the R test suite fails if that number rises. So this chapter's arithmetic is checked by a test rather than by a reader.

Where a chapter below names a family R does not declare, take the model from the corresponding document and solve it as the tutorials show.

This manual documents the worked-example inventory of LINE, one edition per codebase. An entry is runnable only when this edition identifies a source implementation and does not mark the required operation as unsupported: `r/linesolver/inst/examples/`, and each chapter follows the organization of that tree. Entries are titled by the reference (MATLAB) script name, and most editions use the same name, so an example can be read side by side between editions. Where an edition spells it differently, its listing shows the name it actually uses; where an edition has no implementation of an entry, the entry says so rather than leaving the reader to infer it.

The reference material on model classes, solvers and options is in the LINE user manual, and the guided tour and the introductory tutorials are in the LINE primer. Neither is repeated here: this manual starts where the primer stops, at the point where one already knows how to build a model and wants to see the feature one needs.

Every model in the suite is written in the same four blocks, and it is worth learning to read them:

1. the nodes: stations, sources, sinks and the special nodes;
2. the classes and their arrival and service distributions;

3. the topology, as a routing matrix or a serial routing shorthand;
4. the solution: one or more solvers, and the metrics read from them.

The chapters are:

- *Basic models* (Chapter 2): the elementary model families, one per node or class feature.
- *Advanced models* (Chapter 3): features that leave the product-form setting.
- *Model gallery* (Chapter 4): a catalogue of small named models, in Kendall notation where one applies.
- *Discrete-time models* (Chapter 5): models on a slotted time scale.
- *Solver-specific examples* (Chapter 6): scripts written around one solver rather than one model.
- *Model interchange* (Chapter 7): the JSON model format and its round-trip fixtures.
- *Inference and estimation* (Chapter 8): fitting model parameters to measurements.
- *Optimization* (Chapter 9): design problems on queueing networks.

Each chapter opens with a worked example and then gives one section per script in the reference inventory. A complete entry contains a topology figure, a feature table, source code, and captured output. These components are generated from the harvested model, the implementation in the named language tree, and an actual run.

Chapter 2

Basic models

The basic examples cover one model family each: a node type, a class type or a scheduling feature, in the smallest model that shows it. They live under `r/linesolver/inst/examples/`, one file per ported model. A model keeps its script name in every codebase that ports it, so it can be compared line by line between editions; an entry this edition does not port says so in place of its listing.

2.1 Open queueing networks

An open network has an unbounded population: jobs enter at a `Source`, visit stations according to the routing matrix and leave at a `Sink`. Stability requires the utilization of every station to stay below one, which is a property of the arrival rates and the demands, not something the model declares.

The archetype is `oqn_basic`: a Poisson stream feeds a Delay station with hyperexponential think time and an FCFS queue with exponential service. Four blocks build it – nodes, classes and their distributions, the routing matrix, and the solve – and every model in this manual follows the same four blocks.

The scripts of this family are:

2.1.1 `mam_transient_mapmap1`

Transient analysis of a MAP/MAP/1 queue started empty, through the Laplace-domain transient QBD method that `getTranAvg` auto-selects when both the arrival and the service process are correlated.

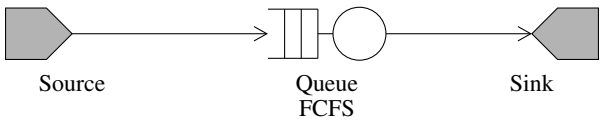


Figure 2.1: Topology of `mam_transient_mapmap1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: MAP
Service	Queue – Class1: MAP

Table 2.1: Features of `mam_transient_mapmap1`.

2.1.2 oqn_basic

Source, Delay and an FCFS queue with one open class and a hyperexponential think time, solved with every solver that admits the model.

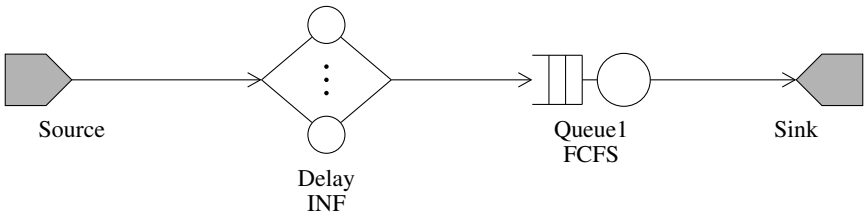


Figure 2.2: Topology of `oqn_basic`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	1: Class1 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(0.1)
Service	Delay – Class1: HyperExp(0.5,0.5;3,10) Queue1 – Class1: Exp(1)
Solvers	CTMC, FLD, MVA, MAM, NC, JMT, SSA, LDES

Table 2.2: Features of `oqn_basic`.

2.1.3 oqn_bmmapt

A BATCH, MARKED, time-inhomogeneous MAP arrival stream: a block is indexed by segment, mark and batch size, so the schedule moves the class mix and the batch size together while the epoch rate stays put. The job rate therefore differs from the epoch rate, and per class within a segment.

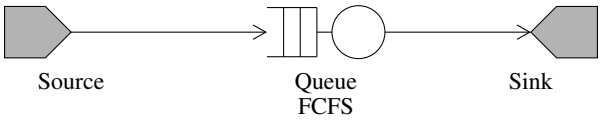


Figure 2.3: Topology of `oqn_bmmapt`: an open network over 1 queueing station.

2.1.4 oqn_cs_routing

Three open classes over two PS queues, two of which switch class at a `ClassSwitch` node; the topology is imported from a JSIMgraph file.

2.1.5 oqn_fourqueues

A larger multiclass network over four stations mixing PS and FCFS scheduling.

2.1.6 oqn_mapt

A time-inhomogeneous MAP arrival stream and the Ko-Pender fluid and diffusion limits returned by the `kp` fluid method.

Nodes	3: Source, Queue, Sink
Classes	2: Premium (open), Economy (open)
Scheduling	Queue: FCFS
Arrivals	Source – Premium: BMMAPt(True); Economy: BMMAPt(True)
Service	Queue – Premium: Exp(16); Economy: Exp(16)
Solvers	LDES

Table 2.3: Features of `oqn_bmmapt`.

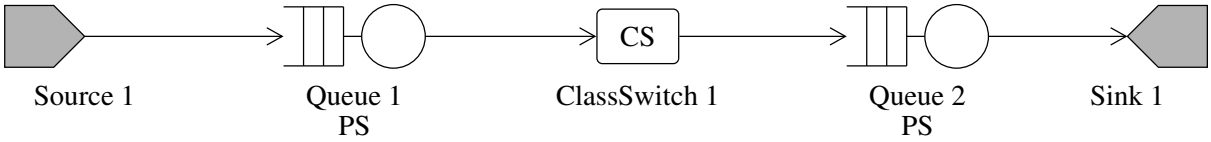


Figure 2.4: Topology of `oqn_cs_routing`: an open network over 2 queueing stations with a class-switch node.

Nodes	5: Source, Queue × 2, ClassSwitch, Sink
Classes	3: Class A (open), Class B (open), Class C (open)
Scheduling	Queue 1: PS; Queue 2: PS
Arrivals	Source 1 – Class A: Exp(2); Class B: Exp(1); Class C: Disabled
Service	Queue 1 – Class A: Exp(5); Class B: Exp(3.333); Class C: Exp(3) Queue 2 – Class A: Exp(1); Class B: Exp(1); Class C: Exp(6.667)
Solvers	CTMC, FLD, MVA, MAM, NC, JMT, SSA, LDES

Table 2.4: Features of `oqn_cs_routing`.

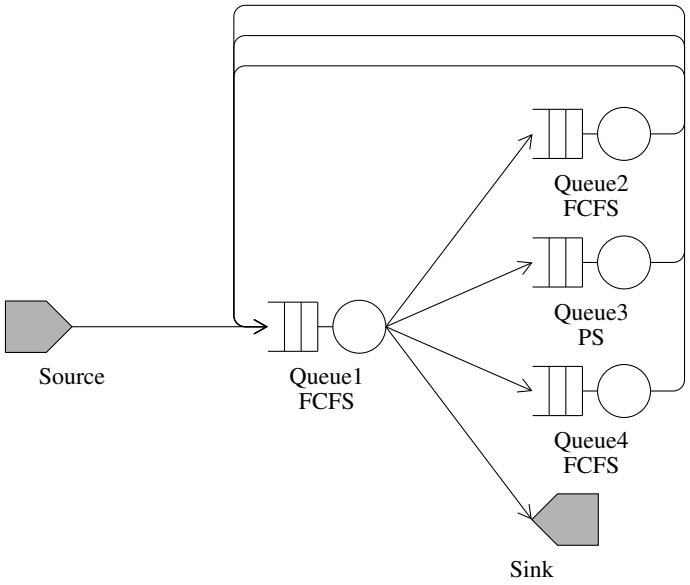


Figure 2.5: Topology of `oqn_fourqueues`: an open network over 4 queueing stations.

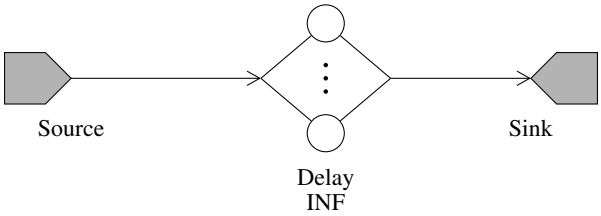


Figure 2.6: Topology of `oqn_mapt`: an open network over 1 queueing station.

Nodes	6: Source, Queue × 4, Sink
Classes	3: Class1 (open), Class2 (open), Class3 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: PS; Queue4: FCFS
Arrivals	Source – Class1: Exp(0.2); Class2: Exp(0.125); Class3: Exp(0.1429)
Service	Queue1 – Class1: Exp(3.333); Class2: Exp(2); Class3: Exp(1.667) Queue2 – Class1: Exp(0.9091); Class2: Exp(0.7692); Class3: Exp(0.6667) Queue3 – Class1: Exp(0.5); Class2: Exp(0.4762); Class3: Exp(0.5263) Queue4 – Class1: Exp(0.6667); Class2: Exp(1.111); Class3: Exp(0.4348)
Solvers	CTMC, MVA, MAM, JMT, LDES

Table 2.5: Features of `oqn_fourqueues`.

Nodes	3: Source, Delay, Sink
Classes	1: OpenClass (open)
Scheduling	Delay: INF
Arrivals	Source – OpenClass: MAPt
Service	Delay – OpenClass: Exp(2)
Solvers	FLD

Table 2.6: Features of `oqn_mapt`.

2.1.7 `oqn_mapt_service`

A time-inhomogeneous MAP as the SERVICE process of a finite-buffer queue: the phase walk is taken from the service start instant and the modulating phase carries across successive services, so the stream is correlated as well as non-stationary.

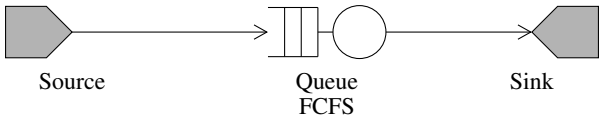


Figure 2.7: Topology of `oqn_mapt_service`: an open network over 1 queueing station.

2.1.8 `oqn_mmapt`

A MARKED time-inhomogeneous MAP arrival stream: one modulating chain drives every marked class, the mark selects the class of the arriving job, and the schedule moves the class MIX while leaving the aggregate rate flat.

2.1.9 `oqn_multichain_cs`

Five classes grouped into three chains by class switching, the smallest model in which chains and classes differ.

2.1.10 `oqn_nhpp`

A cyclic nonhomogeneous Poisson arrival stream, whose rate is a periodic function of time.

2.1.11 `oqn_nnpp`

A cyclic nonstationary non-Poisson arrival stream: a piecewise-linear rate with dispersion ratio 0.5, generated by SCIATA and simulated by LDES.

Nodes	3: Source, Queue, Sink
Classes	1: OpenClass (open)
Scheduling	Queue: FCFS
Arrivals	Source – OpenClass: Exp(10)
Service	Queue – OpenClass: MAPt
Features	finite buffer 5 at Queue
Solvers	LDES

Table 2.7: Features of `oqn_mapt_service`.

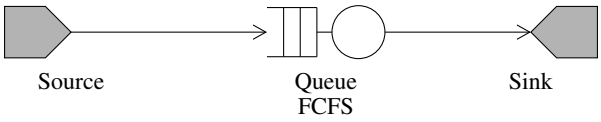


Figure 2.8: Topology of `oqn_mmapt`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Morning (open), Evening (open)
Scheduling	Queue: FCFS
Arrivals	Source – Morning: MMAPt(True); Evening: MMAPt(True)
Service	Queue – Morning: Exp(8); Evening: Exp(8)
Solvers	LDES

Table 2.8: Features of `oqn_mmapt`.

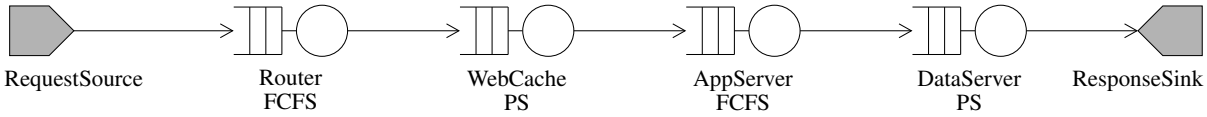


Figure 2.9: Topology of `oqn_multichain_cs`: an open network over 4 queueing stations.

Nodes	6: Source, Queue $\times$ 4, Sink
Classes	5: InteractiveA (open), InteractiveB (open), BatchA (open), BatchB (open), RealTime (open)
Scheduling	Router: FCFS; WebCache: PS; AppServer: FCFS; DataServer: PS
Arrivals	RequestSource – InteractiveA: Exp(0.3); InteractiveB: Exp(0.1); BatchA: Exp(0.15); BatchB: Exp(0.12); RealTime: Exp(0.2)
Service	Router – InteractiveA: Exp(6.667); InteractiveB: Exp(5.556); BatchA: Exp(5); BatchB: Exp(4.545); RealTime: Exp(10) WebCache – InteractiveA: Exp(3.333); InteractiveB: Exp(2.857); BatchA: Exp(1); BatchB: Exp(0.8333); RealTime: Exp(5) AppServer – InteractiveA: Exp(2); InteractiveB: Exp(1.667); BatchA: Exp(0.6667); BatchB: Exp(0.5556); RealTime: Exp(3.333) DataServer – InteractiveA: Exp(2.5); InteractiveB: Exp(2.222); BatchA: Exp(1.25); BatchB: Exp(1); RealTime: Exp(4)
Solvers	MVA, FLD, SSA

Table 2.9: Features of `oqn_multichain_cs`.

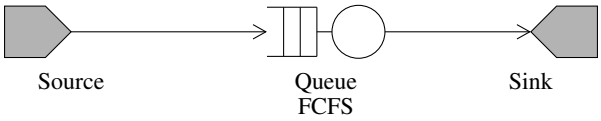


Figure 2.10: Topology of `oqn_nhpp`: an open network over 1 queueing station.

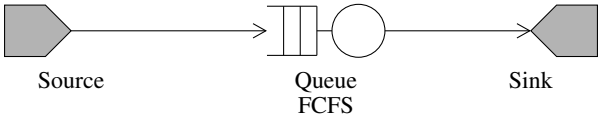


Figure 2.11: Topology of `oqn_nnpp`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: OpenClass (open)
Scheduling	Queue: FCFS
Arrivals	Source – OpenClass: NHPP
Service	Queue – OpenClass: Exp(10)
Solvers	LDES

Table 2.10: Features of `oqn_nhpp`.

Nodes	3: Source, Queue, Sink
Classes	1: OpenClass (open)
Scheduling	Queue: FCFS
Arrivals	Source – OpenClass: NNPP(200,True)
Service	Queue – OpenClass: Exp(38.75)
Solvers	LDES

Table 2.11: Features of `oqn_nnpp`.

2.1.12 `oqn_oneline`

The same tandem written in one line from the arrival, demand and think time matrices, through `Network.tandemPsInf`.

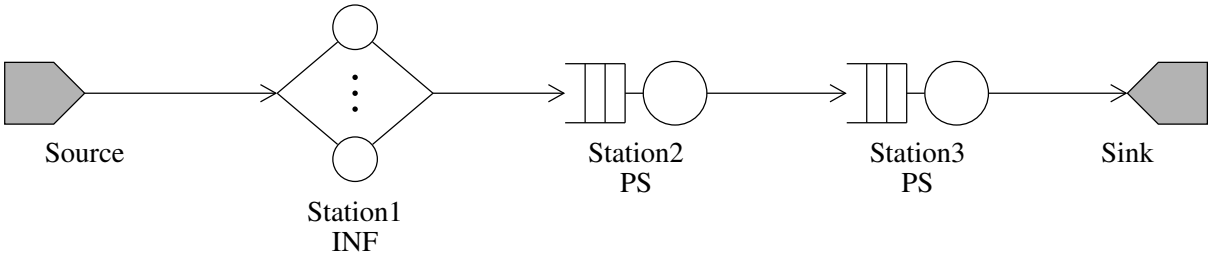


Figure 2.12: Topology of `oqn_oneline`: an open network over 3 queueing stations.

2.1.13 `oqn_trace_driven`

Arrivals and service times replayed from a measurement trace with `Replayer`.

2.1.14 `oqn_vsinks`

Several sinks in one model, the virtual-sink idiom for measuring departure streams separately.

2.2 Closed queueing networks

A closed network holds a fixed population per class. Each closed class names a reference station, at which its visit ratio is one and through which its throughput is measured; the population is the third argument of the class constructor. A Delay station with think time Z and a queue with demand D is the interactive-system archetype, and the repairman model is its oldest form.

Because the population is finite, every metric is bounded and the network is stable by construction. What varies across the scripts below is the scheduling discipline, the service distribution and the number of servers, that is, exactly the assumptions under which a product-form solution does or does not hold.

Nodes	5: Source, Queue × 2, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Station1: INF; Station2: PS; Station3: PS
Arrivals	Source – Class1: Exp(0.02); Class2: Exp(0.04)
Service	Station1 – Class1: Exp(0.01099); Class2: Exp(0.01087) Station2 – Class1: Exp(0.1); Class2: Exp(0.2) Station3 – Class1: Exp(0.2); Class2: Exp(0.1111)
Solvers	MVA

Table 2.12: Features of `oqn_oneline`.

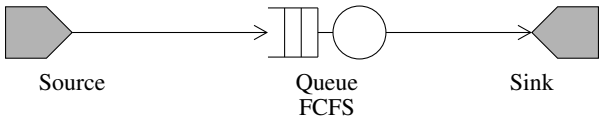


Figure 2.13: Topology of `oqn_trace_driven`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: OpenClass (open)
Scheduling	Queue: FCFS
Arrivals	Source – OpenClass: Exp(1)
Service	Queue – OpenClass: Replayer(trace)
Solvers	JMT, LDES

Table 2.13: Features of `oqn_trace_driven`.

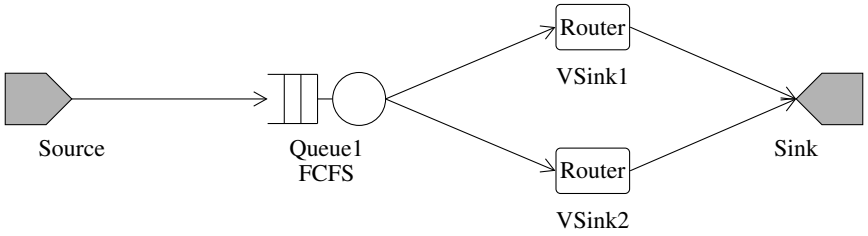


Figure 2.14: Topology of `oqn_vsinks`: an open network over 1 queueing station with a router.

Nodes	5: Source, Queue, Router × 2, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue1: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Exp(1)
Service	Queue1 – Class1: Exp(100); Class2: Exp(100)
Solvers	MVA, MAM, NC

Table 2.14: Features of `oqn_vsinks`.

2.2.1 cqn_bas_blocking

Blocking-after-service with finite buffers, where a completing job holds its server until the destination frees a slot.

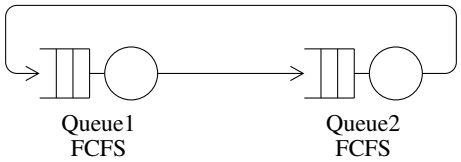


Figure 2.15: Topology of cqn_bas_blocking: a closed network over 2 queueing stations.

Nodes	2: Queue $\times$ 2
Classes	1: Class1 (closed, $N = 2$)
Scheduling	Queue1: FCFS; Queue2: FCFS
Service	Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(0.8)
Features	drop rule {'Class1': 'blockingAfterService'} at Queue1; finite buffer 1 at Queue2
Solvers	MVA

Table 2.15: Features of cqn_bas_blocking.

2.2.2 cqn_bcmp_theorem

The product-form scheduling policies of the BCMP theorem (FCFS with class-independent exponential service, PS, LCFS-PR, INF) compared on one network.

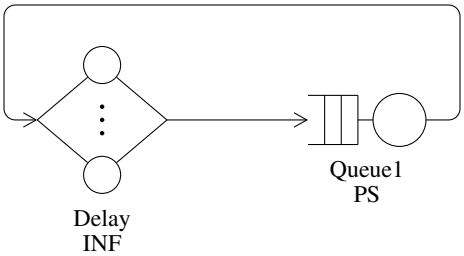


Figure 2.16: Topology of cqn_bcmp_theorem: a closed network over 2 queueing stations.

2.2.3 cqn_lcfs_multiclass

Multiclass LCFS and LCFS-PR scheduling, and the difference preemption makes.

2.2.4 cqn_mmpp2_service

MMPP(2) service times, a correlated service process that no product-form solver can represent exactly.

2.2.5 cqn_multichain_cs

Five classes in three chains, the closed counterpart of oqn_multichain_cs.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Erlang(3,2); Class2: HyperExp(0.5,0.5;3,10) Queue1 – Class1: Exp(1); Class2: Exp(1)
Solvers	CTMC

Table 2.16: Features of `cqn_bcmp_theorem`.

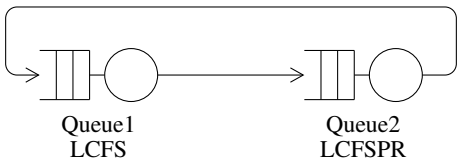


Figure 2.17: Topology of `cqn_lcfs_multiclass`: a closed network over 2 queueing stations.

Nodes	2: Queue $\times$ 2
Classes	3: Class1 (closed, $N = 1$), Class2 (closed, $N = 1$), Class3 (closed, $N = 1$)
Scheduling	Queue1: LCFS; Queue2: LCFSPR
Service	Queue1 – Class1: Exp(1); Class2: Exp(0.3333); Class3: Exp(0.2) Queue2 – Class1: Exp(0.5); Class2: Exp(0.25); Class3: Exp(0.1667)
Solvers	MVA, CTMC

Table 2.17: Features of `cqn_lcfs_multiclass`.

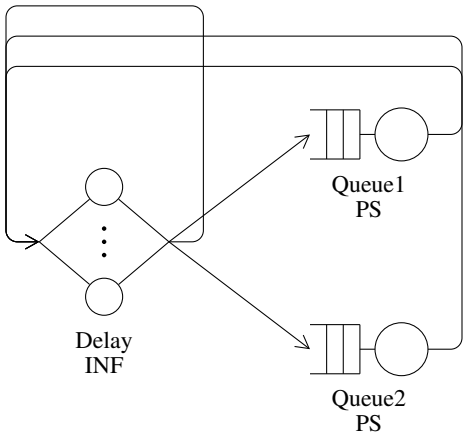


Figure 2.18: Topology of `cqn_mmpp2_service`: a closed network over 3 queueing stations.

Nodes	3: Queue $\times$ 2, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – Class1: Erlang(3,2); Class2: HyperExp(0.5,0.5;3,10) Queue1 – Class1: HyperExp(0.1,0.9;1,10); Class2: MMPP2 Queue2 – Class1: HyperExp(0.1,0.9;1,10); Class2: Erlang(1,2)
Features	class switching on 3 links
Solvers	JMT, LDES

Table 2.18: Features of `cqn_mmpp2_service`.

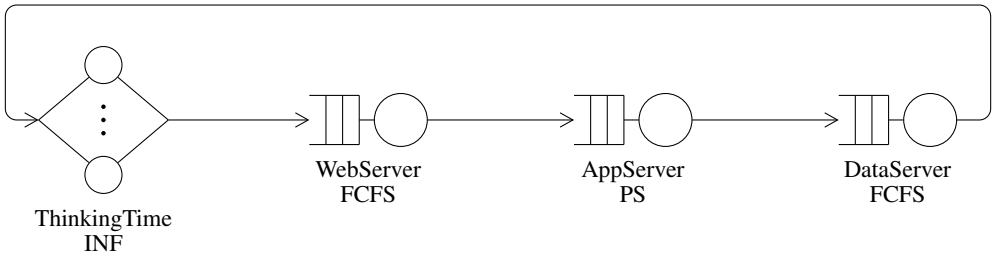


Figure 2.19: Topology of `cqn_multichain_cs`: a closed network over 4 queueing stations.

Nodes	4: Queue × 3, Delay
Classes	5: HighPriorityA (closed, $N = 2$), HighPriorityB (closed, $N = 1$), RegularA (closed, $N = 3$), RegularB (closed, $N = 2$), Background (closed, $N = 2$)
Scheduling	ThinkingTime: INF; WebServer: FCFS; AppServer: PS; DataServer: FCFS
Service	ThinkingTime – HighPriorityA: Exp(0.6667); HighPriorityB: Exp(0.625); RegularA: Exp(0.5); RegularB: Exp(0.4762); Background: Exp(0.3333) WebServer – HighPriorityA: Exp(2.5); HighPriorityB: Exp(2.381); RegularA: Exp(2); RegularB: Exp(1.923); Background: Exp(1.25) AppServer – HighPriorityA: Exp(1.429); HighPriorityB: Exp(1.333); RegularA: Exp(1.111); RegularB: Exp(1.053); Background: Exp(0.8333) DataServer – HighPriorityA: Exp(2); HighPriorityB: Exp(1.923); RegularA: Exp(1.667); RegularB: Exp(1.613); Background: Exp(1)
Solvers	MVA, SSA

Table 2.19: Features of `cqn_multichain_cs`.

2.2.6 `cqn_multiserver`

A closed network with multiserver FCFS and PS stations, Erlang service and a class disabled at one station.

Nodes	3: Queue × 2, Delay
Classes	4: Class1 (closed, $N = 2$), Class2 (closed, $N = 2$), Class3 (closed, $N = 2$), Class4 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: FCFS ($c = 3$); Queue2: FCFS ($c = 3$)
Service	Delay – Class1: Exp(1); Class2: Exp(1); Class3: Exp(10); Class4: Exp(1) Queue1 – Class1: Exp(1); Class2: Erlang(1,2); Class3: Exp(10); Class4: Exp(1) Queue2 – Class1: Disabled; Class2: Disabled; Class3: Erlang(1,2); Class4: Exp(1)
Solvers	MVA

Table 2.20: Features of `cqn_multiserver`.

2.2.7 `cqn_multiserver_nc`

Delay plus a multiserver FCFS queue, solved by the normalizing constant method.

2.2.8 `cqn_mvablo`

A closed cyclic network with Blocking-After-Service finite buffers, solved by MVABLO and by the exact CTMC.

2.2.9 `cqn_oneline`

The closed counterpart of the one-line construction, from the demand and think time matrices.

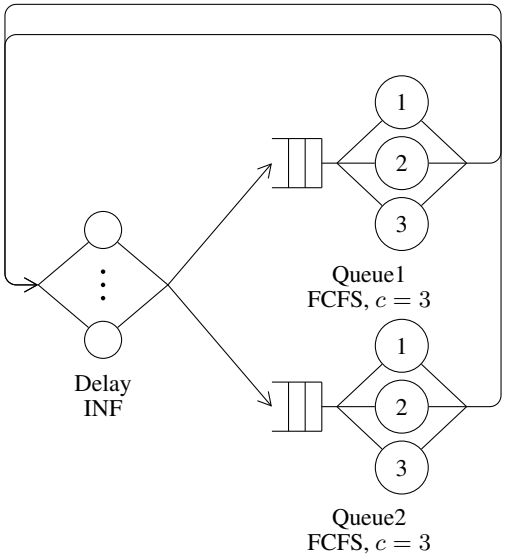


Figure 2.20: Topology of `cqn_multiserver`: a closed network over 3 queueing stations.

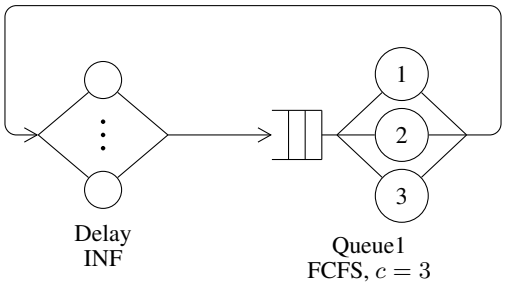


Figure 2.21: Topology of `cqn_multiserver_nc`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: FCFS ($c = 3$)
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(1.25)
Solvers	CTMC, MVA, NC

Table 2.21: Features of `cqn_multiserver_nc`.

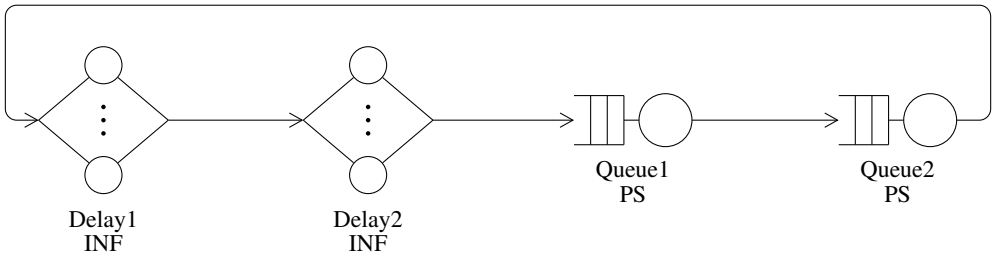


Figure 2.22: Topology of `cqn_oneline`: a closed network over 4 queueing stations.

Nodes	4: Queue × 2, Delay × 2
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 2$)
Scheduling	Delay1: INF; Delay2: INF; Queue1: PS; Queue2: PS
Service	Delay1 – Class1: Exp(0.01099); Class2: Exp(0.01087) Delay2 – Class1: Exp(0.01075); Class2: Exp(0.01064) Queue1 – Class1: Exp(0.1); Class2: Exp(0.2) Queue2 – Class1: Exp(0.2); Class2: Exp(0.1111)
Solvers	MVA

Table 2.22: Features of `cqn_oneline`.

2.2.10 `cqn_repairmen`

The machine repairman model: a Delay station of thinking machines and one FCFS repair queue.

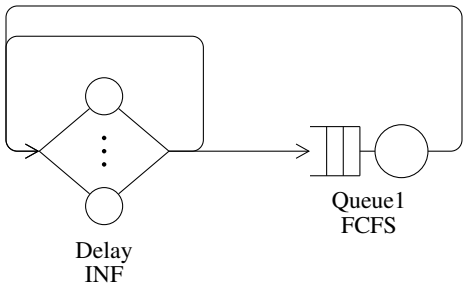


Figure 2.23: Topology of `cqn_repairmen`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 10$)
Scheduling	Delay: INF; Queue1: FCFS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(0.6667)
Solvers	CTMC, JMT, SSA, FLD, MVA, NC, MAM, LDES

Table 2.23: Features of `cqn_repairmen`.

2.2.11 `cqn_repairmen_multi`

The repairman model with several repair stations.

2.2.12 `cqn_scheduling_dps`

Discriminatory processor sharing, with the class weights that PS ignores.

2.2.13 `cqn_threeclass_hyperl`

The three-class version, with multiserver stations.

2.2.14 `cqn_twoclass_erl`

Two classes with Erlang service over PS and INF stations, exercising the RAND, round-robin and weighted round-robin routing policies.

2.2.15 `cqn_twoclass_hyperl`

Two classes with hyperexponential and Erlang service on PS stations.

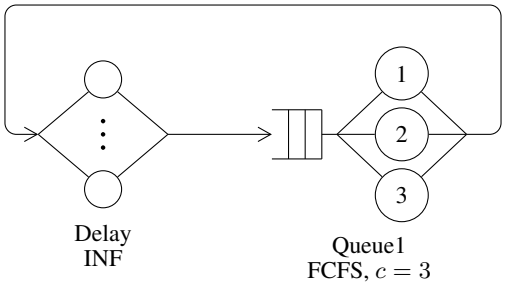


Figure 2.24: Topology of `cqn_repairmen_multi`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 4$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: FCFS ($c = 3$)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(1); Class2: Exp(10)
Solvers	CTMC, LQNS, MVA, NC

Table 2.24: Features of `cqn_repairmen_multi`.

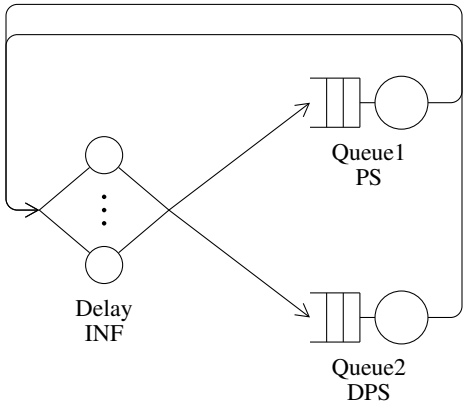


Figure 2.25: Topology of `cqn_scheduling_dps`: a closed network over 3 queueing stations.

Nodes	3: Queue $\times$ 2, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS; Queue2: DPS
Service	Delay – Class1: Exp(3); Class2: Exp(0.5) Queue1 – Class1: Exp(0.1); Class2: Exp(1) Queue2 – Class1: Exp(0.1); Class2: Exp(1)
Solvers	CTMC, JMT, FLD, MVA, LDES

Table 2.25: Features of `cqn_scheduling_dps`.

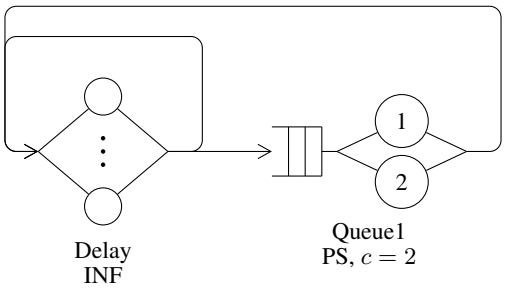


Figure 2.26: Topology of `cqn_threeclass_hyperl`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	3: Class1 (closed, $N = 2$), Class2 (closed, $N = 0$), Class3 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS ($c = 2$)
Service	Delay – Class1: Erlang(3,2); Class2: HyperExp(0.5,0.5;3,10); Class3: Exp(1) Queue1 – Class1: HyperExp(0.1,0.9;1,10); Class2: Exp(2); Class3: Exp(3)
Solvers	CTMC, JMT, SSA, FLD, MVA, NC, MAM, LDES

Table 2.26: Features of `cqn_threeclass_hyperl`.

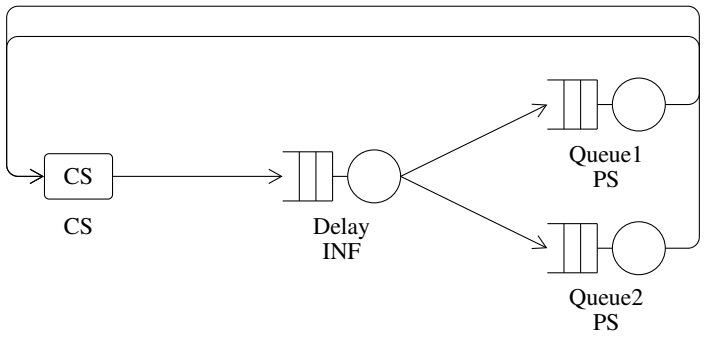


Figure 2.27: Topology of `cqn_twoclass_erl`: a closed network over 3 queueing stations with a class-switch node.

Nodes	4: Queue $\times$ 3, ClassSwitch
Classes	2: Class1 (closed, $N = 15$), Class2 (closed, $N = 5$)
Scheduling	Queue1: PS; Queue2: PS; Delay: INF
Service	Queue1 – Class1: Exp(0.6667); Class2: Erlang(1.333,2) Queue2 – Class1: Erlang(1.333,2); Class2: Exp(0.6667) Delay – Class1: Exp(1); Class2: Exp(1)
Features	class switching on 3 links
Solvers	JMT, LDES

Table 2.27: Features of `cqn_twoclass_erl`.

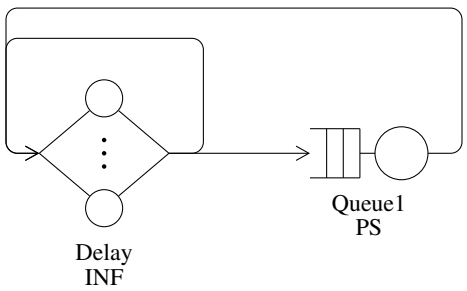


Figure 2.28: Topology of `cqn_twoclass_hyperl`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Erlang(3,2); Class2: HyperExp(0.5,0.5;3,10) Queue1 – Class1: HyperExp(0.1,0.9;1,10); Class2: Exp(1)
Solvers	CTMC, JMT, SSA, FLD, MVA, NC, MAM, LDES

Table 2.28: Features of `cqn_twoclass_hyperl`.

2.2.16 `cqn_twoqueues_multi`

Two FCFS queues and a Delay with several closed classes.

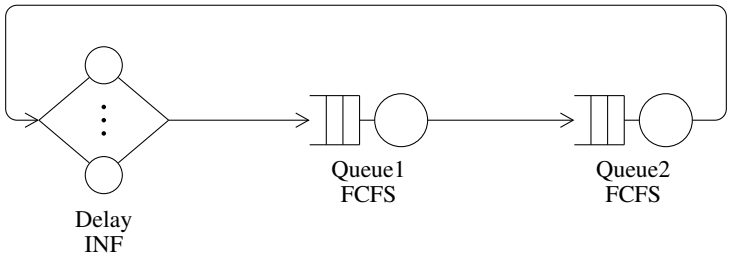


Figure 2.29: Topology of `cqn_twoqueues_multi`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 10$), Class2 (closed, $N = 10$)
Scheduling	Delay: INF; Queue1: FCFS; Queue2: FCFS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.6667) Queue2 – Class1: Exp(0.3333); Class2: Exp(0.3333)
Solvers	MVA

Table 2.29: Features of `cqn_twoqueues_multi`.

2.3 Mixed networks

A mixed network carries open and closed classes at the same stations. The open classes contribute a fixed arrival rate, the closed ones a fixed population, and the two compete for the same servers, so neither can be solved in isolation.

The scripts pair a single-server and a multiserver station under FCFS and under PS. They are deliberately awkward instances: the load-dependent rates they induce are sparse, and they are the models on which the approximate solvers disagree first.

2.3.1 `mqn_basic`

Open and closed classes at the same stations, the smallest mixed model.

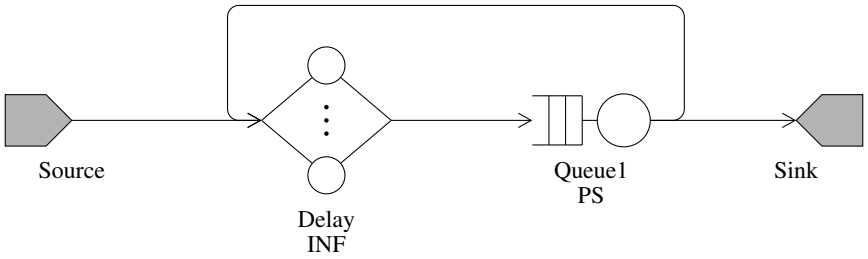


Figure 2.30: Topology of `mqn_basic`: an open network over 2 queueing stations.

2.3.2 `mqn_multichain_cs`

A mixed network whose classes are grouped into several chains by class switching.

Nodes	4: Source, Queue, Delay, Sink
Classes	2: ClosedClass (closed, $N = 2$), OpenClass (open)
Scheduling	Delay: INF; Queue1: PS
Arrivals	Source – ClosedClass: Disabled; OpenClass: Exp(0.1)
Service	Delay – ClosedClass: Erlang(3,2); OpenClass: HyperExp(0.5,0.5;3,10) Queue1 – ClosedClass: HyperExp(0.1,0.9;1,10); OpenClass: Exp(1)
Features	class switching on 1 link
Solvers	CTMC, JMT, SSA, MVA, LDES

Table 2.30: Features of `mqn_basic`.

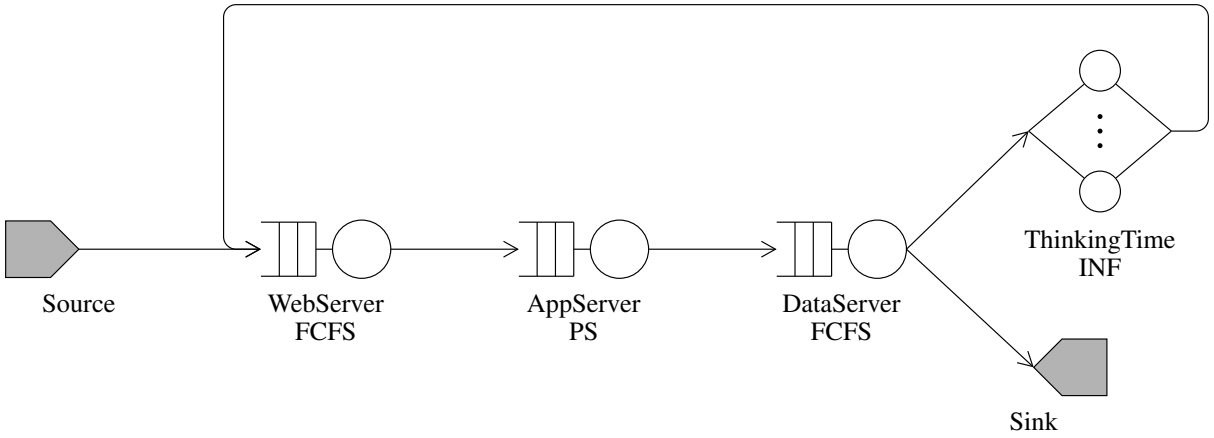


Figure 2.31: Topology of `mqn_multichain_cs`: an open network over 4 queueing stations.

Nodes	6: Source, Queue $\times$ 3, Delay, Sink
Classes	5: InteractiveA (closed, $N = 3$), InteractiveB (closed, $N = 2$), BatchA (open), BatchB (open), ExternalLoad (open)
Scheduling	ThinkingTime: INF; WebServer: FCFS; AppServer: PS; DataServer: FCFS
Arrivals	Source – InteractiveA: Disabled; InteractiveB: Disabled; BatchA: Exp(0.1); BatchB: Exp(0.05); ExternalLoad: Exp(0.05)
Service	ThinkingTime – InteractiveA: Exp(0.6667); InteractiveB: Exp(0.5); BatchA: Disabled; BatchB: Disabled; ExternalLoad: Disabled WebServer – InteractiveA: Exp(2.5); InteractiveB: Exp(2); BatchA: Exp(3.333); BatchB: Exp(2.857); ExternalLoad: Exp(5) AppServer – InteractiveA: Exp(1.25); InteractiveB: Exp(1); BatchA: Exp(1.667); BatchB: Exp(1.429); ExternalLoad: Exp(2) DataServer – InteractiveA: Exp(2); InteractiveB: Exp(1.667); BatchA: Exp(0.8333); BatchB: Exp(0.6667); ExternalLoad: Exp(1.25)
Features	drop rule {'BatchA': 'waitingQueue', 'BatchB': 'waitingQueue', 'ExternalLoad': 'waitingQueue'} at ThinkingTime; class switching on 1 link
Solvers	MVA, SSA

Table 2.31: Features of `mqn_multichain_cs`.

2.3.3 mqn_multiserver_fcfs

The multiserver FCFS version of the same model.

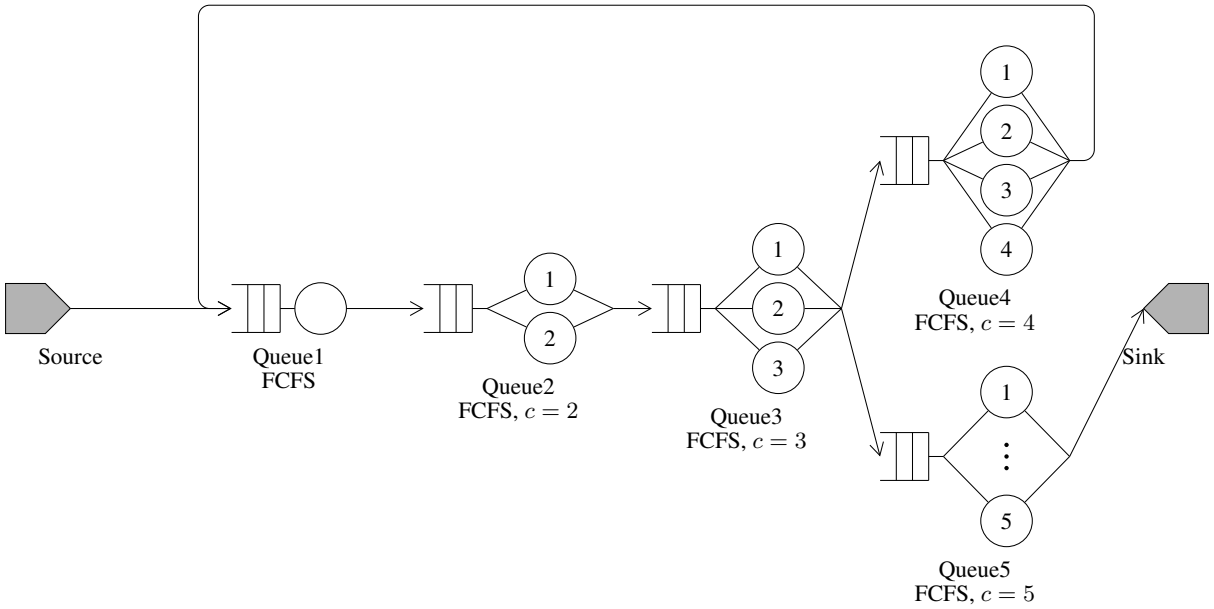


Figure 2.32: Topology of mqn_multiserver_fcfs: an open network over 5 queueing stations.

Nodes	7: Source, Queue × 5, Sink
Classes	2: ClosedClass (closed, $N = 3$), OpenClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS ($c = 2$); Queue3: FCFS ($c = 3$); Queue4: FCFS ($c = 4$); Queue5: FCFS ($c = 5$)
Arrivals	Source – ClosedClass: Disabled; OpenClass: Exp(0.3)
Service	Queue1 – ClosedClass: Exp(1); OpenClass: Exp(1) Queue2 – ClosedClass: Exp(2); OpenClass: Exp(1.414) Queue3 – ClosedClass: Exp(3); OpenClass: Exp(1.732) Queue4 – ClosedClass: Exp(4); OpenClass: Exp(2) Queue5 – ClosedClass: Exp(5); OpenClass: Exp(2.236)
Features	class switching on 3 links
Solvers	CTMC, JMT, SSA, MVA, LDES

Table 2.32: Features of mqn_multiserver_fcfs.

2.3.4 mqn_multiserver_ps

The multiserver PS version.

2.3.5 mqn_singleserver_fcfs

A single-server FCFS station under a mixed workload, a sparse load-dependent multiclass instance that stresses the solvers.

2.3.6 mqn_singleserver_ps

The processor sharing counterpart.

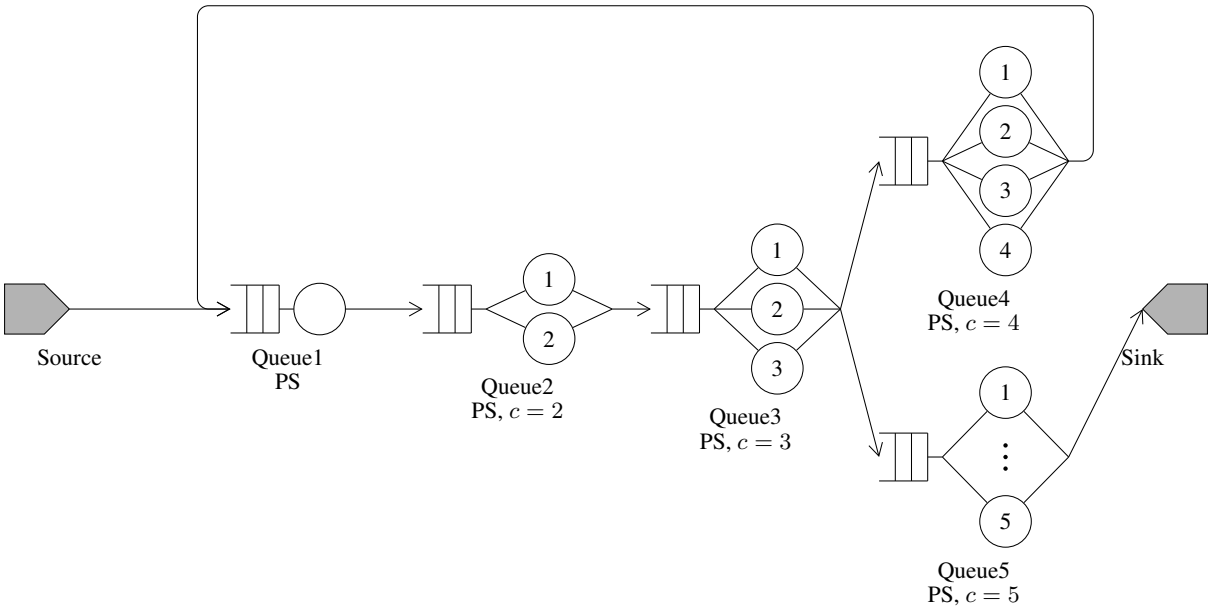


Figure 2.33: Topology of `mqn_multiserver_ps`: an open network over 5 queueing stations.

Nodes	7: Source, Queue $\times$ 5, Sink
Classes	2: ClosedClass (closed, $N = 3$), OpenClass (open)
Scheduling	Queue1: PS; Queue2: PS ($c = 2$); Queue3: PS ($c = 3$); Queue4: PS ($c = 4$); Queue5: PS ($c = 5$)
Arrivals	Source – ClosedClass: Disabled; OpenClass: Exp(0.3)
Service	Queue1 – ClosedClass: Exp(1); OpenClass: Exp(1) Queue2 – ClosedClass: Exp(2); OpenClass: Exp(1.414) Queue3 – ClosedClass: Exp(3); OpenClass: Exp(1.732) Queue4 – ClosedClass: Exp(4); OpenClass: Exp(2) Queue5 – ClosedClass: Exp(5); OpenClass: Exp(2.236)
Features	class switching on 3 links
Solvers	CTMC, JMT, MVA, LDES

Table 2.33: Features of `mqn_multiserver_ps`.

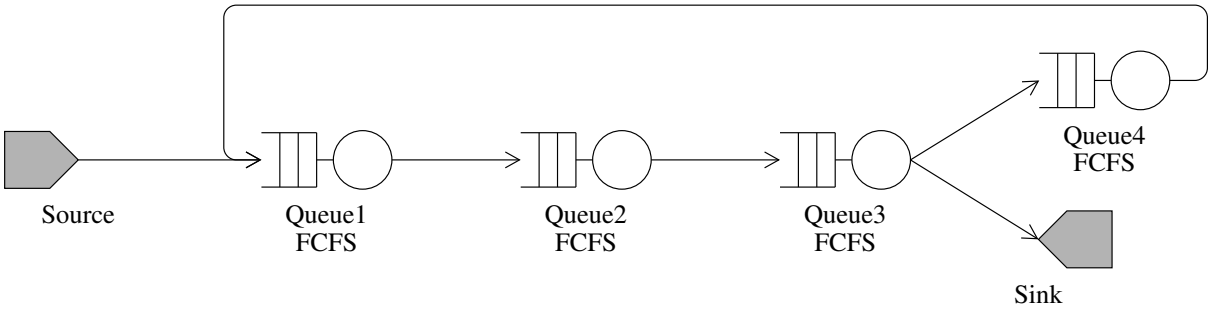


Figure 2.34: Topology of `mqn_singleserver_fcfs`: an open network over 4 queueing stations.

Nodes	6: Source, Queue $\times$ 4, Sink
Classes	2: ClosedClass (closed, $N = 100$), OpenClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS; Queue4: FCFS
Arrivals	Source – ClosedClass: Disabled; OpenClass: APH
Service	Queue1 – ClosedClass: Exp(1); OpenClass: Exp(1) Queue2 – ClosedClass: Exp(2); OpenClass: Exp(1.414) Queue3 – ClosedClass: Exp(3); OpenClass: Exp(1.732) Queue4 – ClosedClass: Exp(4); OpenClass: Exp(2)
Features	class switching on 2 links
Solvers	JMT, MVA, NC, LDES

Table 2.34: Features of `mqn_singleserver_fcfs`.

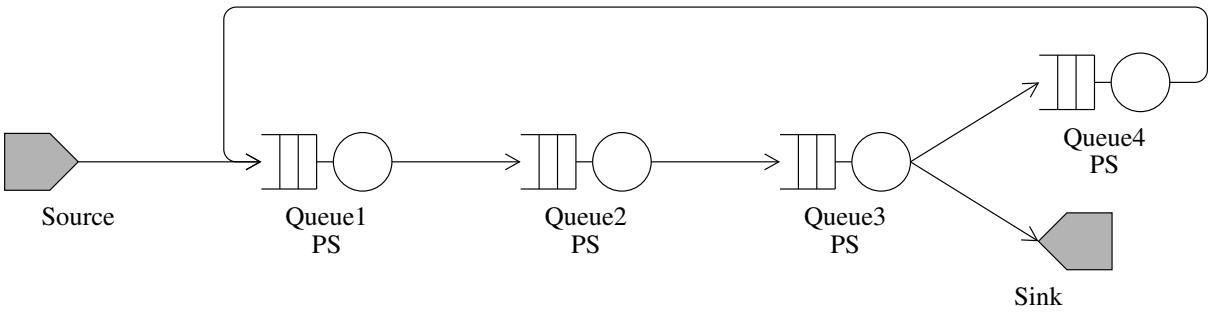


Figure 2.35: Topology of `mqn_singleserver_ps`: an open network over 4 queueing stations.

Nodes	6: Source, Queue × 4, Sink
Classes	2: ClosedClass (closed, $N = 100$), OpenClass (open)
Scheduling	Queue1: PS; Queue2: PS; Queue3: PS; Queue4: PS
Arrivals	Source – ClosedClass: Disabled; OpenClass: Exp(0.3)
Service	Queue1 – ClosedClass: Exp(1); OpenClass: Exp(1) Queue2 – ClosedClass: Exp(2); OpenClass: Exp(1.414) Queue3 – ClosedClass: Exp(3); OpenClass: Exp(1.732) Queue4 – ClosedClass: Exp(4); OpenClass: Exp(2)
Features	class switching on 2 links
Solvers	JMT, SSA, FLD, MVA, NC, LDES

Table 2.35: Features of `mqn_singleserver_ps`.

2.4 Class switching

A job may change class as it moves. Switching is expressed either by off-diagonal blocks of the routing matrix, which give the probability of leaving node i as class r and arriving at node j as class s, or by a dedicated `ClassSwitch` node that performs the change in place.

Switching is what makes classes and chains differ: the classes that can reach one another form a chain, and it is the chain, not the class, that carries a conserved population. The diamond examples build a switching chain that is reducible on purpose, so that transient classes appear.

2.4.1 `cs_implicit`

A ClassSwitch node declared explicitly, against the implicit switching that a routing matrix with off-diagonal class blocks already expresses.



Figure 2.36: Topology of `cs_implicit`: an open network over 1 queueing station with a class-switch node.

2.4.2 `cs_multi_diamond`

The same pattern with several branches.

2.4.3 `cs_single_diamond`

Class switching driven by a reducible Markov chain over the classes, the diamond pattern with one branch.

Nodes	4: Source, Queue, ClassSwitch, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue 1: FCFS
Arrivals	Source 1 – Class1: Exp(0.1); Class2: Exp(0.5)
Service	Queue 1 – Class1: Exp(1); Class2: Exp(1)
Solvers	MVA

Table 2.36: Features of `cs_implicit`.

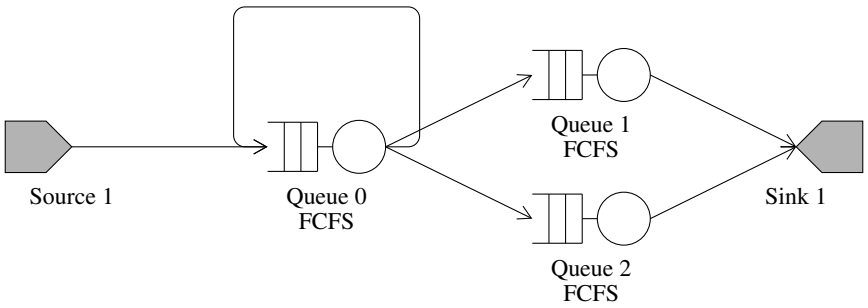


Figure 2.37: Topology of `cs_multi_diamond`: an open network over 3 queueing stations.

Nodes	5: Source, Queue × 3, Sink
Classes	3: Class1 (open), Class2 (open), Class3 (open)
Scheduling	Queue 0: FCFS; Queue 1: FCFS; Queue 2: FCFS
Arrivals	Source 1 – Class1: Exp(1); Class2: Disabled; Class3: Disabled
Service	Queue 0 – Class1: Exp(10); Class2: Disabled; Class3: Disabled Queue 1 – Class1: Disabled; Class2: Exp(20); Class3: Disabled Queue 2 – Class1: Disabled; Class2: Disabled; Class3: Exp(30)
Features	drop rule {'Class2': 'waitingQueue', 'Class3': 'waitingQueue'} at Queue 0; drop rule {'Class1': 'waitingQueue', 'Class3': 'waitingQueue'} at Queue 1; drop rule {'Class1': 'waitingQueue', 'Class2': 'waitingQueue'} at Queue 2; class switching on 1 link
Solvers	MVA

Table 2.37: Features of `cs_multi_diamond`.

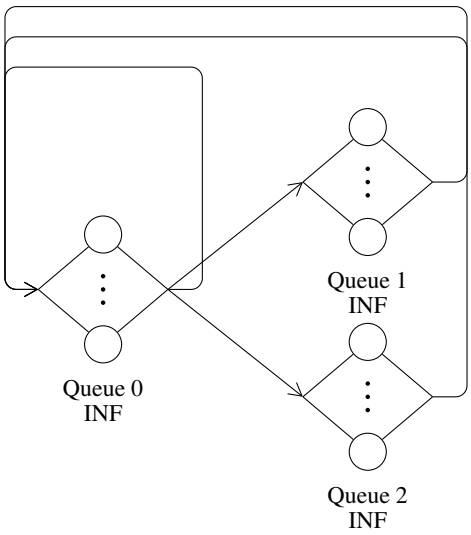


Figure 2.38: Topology of `cs_single_diamond`: a closed network over 3 queueing stations.

Nodes	3: Delay × 3
Classes	3: Class1 (closed, $N = 1$), Class2 (closed, $N = 0$), Class3 (closed, $N = 0$)
Scheduling	Queue 0: INF; Queue 1: INF; Queue 2: INF
Service	Queue 0 – Class1: Exp(1); Class2: Disabled; Class3: Disabled Queue 1 – Class1: Disabled; Class2: Exp(0.5); Class3: Disabled Queue 2 – Class1: Disabled; Class2: Disabled; Class3: Exp(0.3333)
Features	drop rule {'Class2': 'waitingQueue', 'Class3': 'waitingQueue'} at Queue 0; drop rule {'Class1': 'waitingQueue', 'Class3': 'waitingQueue'} at Queue 1; drop rule {'Class1': 'waitingQueue', 'Class2': 'waitingQueue'} at Queue 2
Solvers	MVA

Table 2.38: Features of `cs_single_diamond`.

2.4.4 `cs_transient_class`

A class that is transient in the switching chain, so it is visited but never returned to.

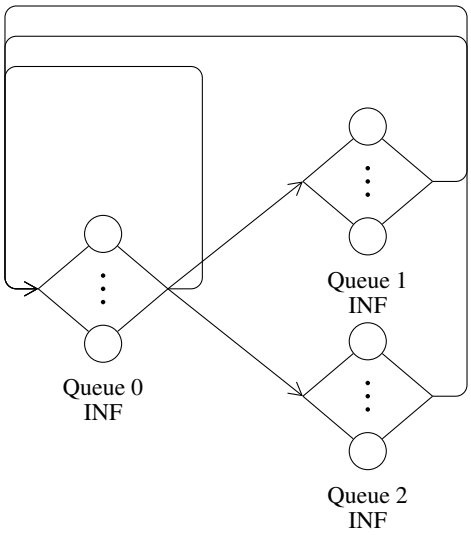


Figure 2.39: Topology of `cs_transient_class`: a closed network over 3 queueing stations.

Nodes	3: Delay × 3
Classes	3: Class1 (closed, $N = 1$), Class2 (closed, $N = 0$), Class3 (closed, $N = 0$)
Scheduling	Queue 0: INF; Queue 1: INF; Queue 2: INF
Service	Queue 0 – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1) Queue 1 – Class1: Disabled; Class2: Exp(1); Class3: Disabled Queue 2 – Class1: Disabled; Class2: Disabled; Class3: Exp(1)
Features	drop rule {'Class1': 'waitingQueue', 'Class3': 'waitingQueue'} at Queue 1; drop rule {'Class1': 'waitingQueue', 'Class2': 'waitingQueue'} at Queue 2
Solvers	MVA

Table 2.39: Features of `cs_transient_class`.

2.5 Priority models

Priorities are declared on the class and enforced by the discipline of the station. Head-of-line (HOL) service is non-preemptive: a low-priority job in service completes, and only the waiting order changes. The preemptive variants interrupt it instead, and the priority processor sharing family divides the capacity in proportion to the priority weights.

Priority breaks product form, so these models are the province of the exact CTMC solver, of simulation, and of the approximations that the MVA solver offers for HOL.

2.5.1 `prio_hol_closed`

The closed counterpart.

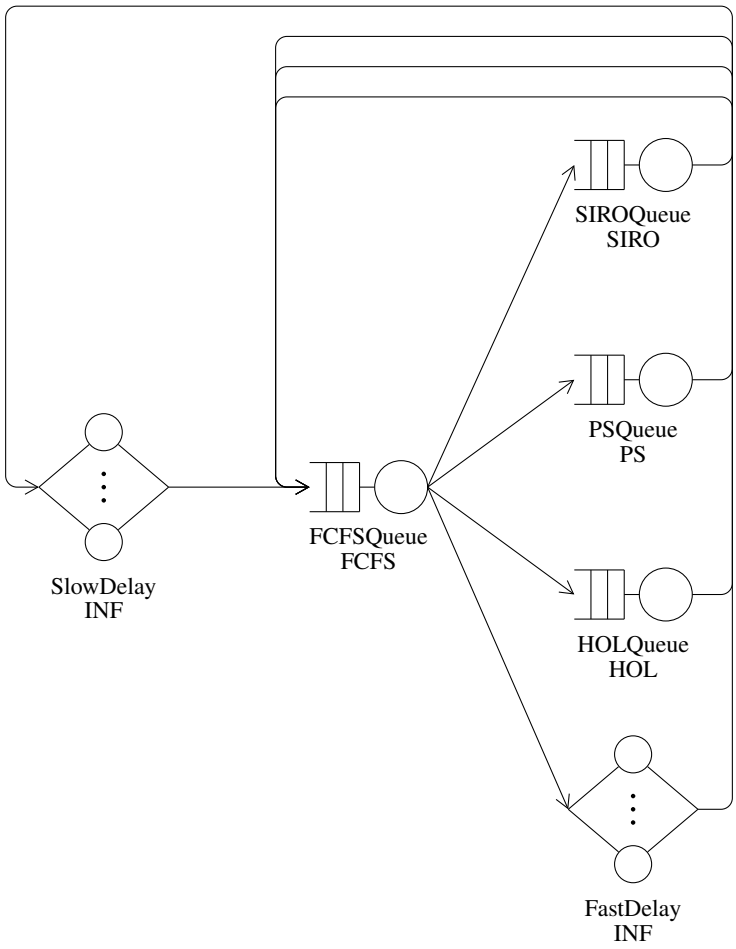


Figure 2.40: Topology of `prio_hol_closed`: a closed network over 6 queueing stations.

2.5.2 `prio_hol_open`

Head-of-line non-preemptive priority in an open network, with PS, SIRO and FCFS stations alongside.

2.5.3 `prio_identical`

Priority classes with identical parameters, the sanity check that priority alone does not move the aggregate metrics.

2.5.4 `prio_psprio`

Priority processor sharing, where the share of a class is set by its priority.

Nodes	6: Queue × 4, Delay × 2
Classes	3: Class1 (closed, $N = 18$), Class2 (closed, $N = 18$), Class3 (closed, $N = 18$)
Scheduling	SlowDelay: INF; FCFSQueue: FCFS; SIROQueue: SIRO; PSQueue: PS; HOLQueue: HOL; FastDelay: INF
Service	SlowDelay – Class1: Exp(0.1); Class2: Exp(0.1); Class3: Exp(0.1) FCFSQueue – Class1: Exp(3.333); Class2: Exp(2); Class3: Exp(1.667) SIROQueue – Class1: Exp(0.9091); Class2: Exp(0.7692); Class3: Exp(0.6667) PSQueue – Class1: Exp(1); Class2: Exp(0.9091); Class3: Exp(0.5263) HOLQueue – Class1: Exp(0.4); Class2: Exp(0.5263); Class3: Exp(0.2326) FastDelay – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1)
Solvers	MVA, JMT, SSA, LDES

Table 2.40: Features of `prio_hol_closed`.

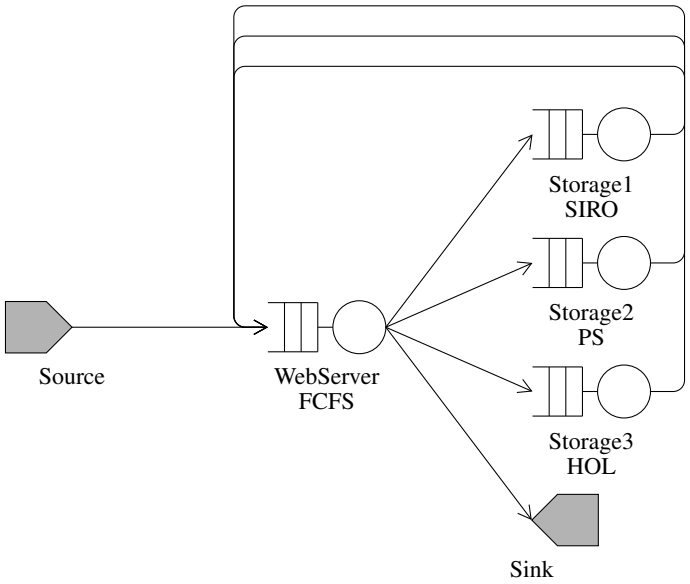


Figure 2.41: Topology of `prio_hol_open`: an open network over 4 queueing stations.

Nodes	6: Source, Queue × 4, Sink
Classes	3: Class1 (open), Class2 (open), Class3 (open)
Scheduling	WebServer: FCFS; Storage1: SIRO; Storage2: PS; Storage3: HOL
Arrivals	Source – Class1: Exp(0.1); Class2: Exp(0.1); Class3: Exp(0.1)
Service	WebServer – Class1: Exp(3.333); Class2: Exp(2); Class3: Exp(1.667) Storage1 – Class1: Exp(0.9091); Class2: Exp(0.7692); Class3: Exp(0.6667) Storage2 – Class1: Exp(0.5); Class2: Exp(0.4762); Class3: Exp(0.5263) Storage3 – Class1: Exp(0.4); Class2: Exp(0.5263); Class3: Exp(0.2326)
Solvers	CTMC, MVA, JMT, SSA

Table 2.41: Features of `prio_hol_open`.

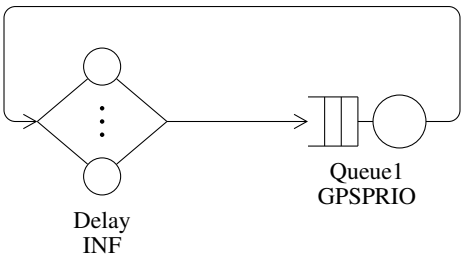


Figure 2.42: Topology of `prio_identical`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	4: Class1 (closed, $N = 6$), Class2 (closed, $N = 4$), Class3 (closed, $N = 4$), Class4 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: GPSPRIO
Service	Delay – Class1: Erlang(3,2); Class2: Exp(1); Class3: Exp(1); Class4: Exp(2) Queue1 – Class1: Exp(30); Class2: Exp(2); Class3: Exp(12); Class4: Exp(1)
Solvers	CTMC, JMT

Table 2.42: Features of `prio_identical`.

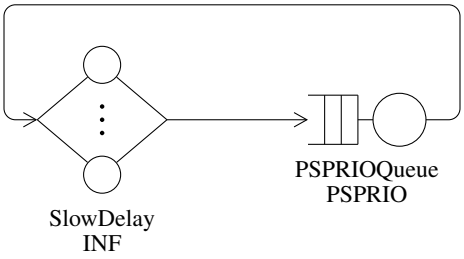


Figure 2.43: Topology of `prio_psprio`: a closed network over 2 queueing stations.

2.6 Fork-join networks

A `Fork` node splits an arriving job into sibling tasks, one per outgoing branch, and a `Join` node recombines them. The response time of the parent job is the maximum over its siblings, which is why fork-join has no product-form solution and why the join is where the modeling subtleties live: how many siblings are required (the quorum), whether the join is present at all, and what happens when branches share a station.

The family is the largest in the basic set because each of those questions has its own script. Nesting a fork inside a branch, three or more branches, asymmetric branch demands and class switching on the siblings are all covered separately, so a failure can be localized.

2.6.1 `fj_asymm`

Asymmetric branches, whose response time is set by the slowest one.

2.6.2 `fj_basic_closed`

The closed counterpart, on PS branches.

2.6.3 `fj_basic_nesting`

A fork inside a branch of another fork, with open and closed classes.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 2$)
Scheduling	SlowDelay: INF; PSPRIOQueue: PSPRIO
Service	SlowDelay – Class1: Erlang(3,2); Class2: HyperExp(0.5,0.5;3,10) PSPRIOQueue – Class1: HyperExp(0.1,0.9;1,10); Class2: Exp(1)
Solvers	CTMC, JMT, SSA

Table 2.43: Features of `prio_psprio`.

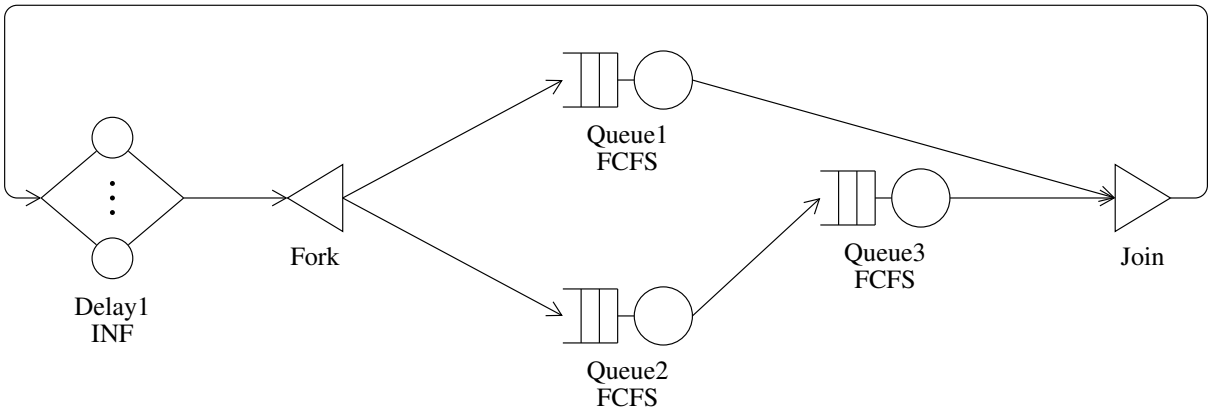


Figure 2.44: Topology of `fj_asymm`: a closed network over 4 queueing stations with a fork-join block.

Nodes	6: Queue × 3, Delay, Fork, Join
Classes	1: class1 (closed, $N = 10$)
Scheduling	Delay1: INF; Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Delay1 – class1: Exp(0.5) Queue1 – class1: Exp(1) Queue2 – class1: Exp(2) Queue3 – class1: Exp(1)
Solvers	JMT, MVA

Table 2.44: Features of `fj_asymm`.

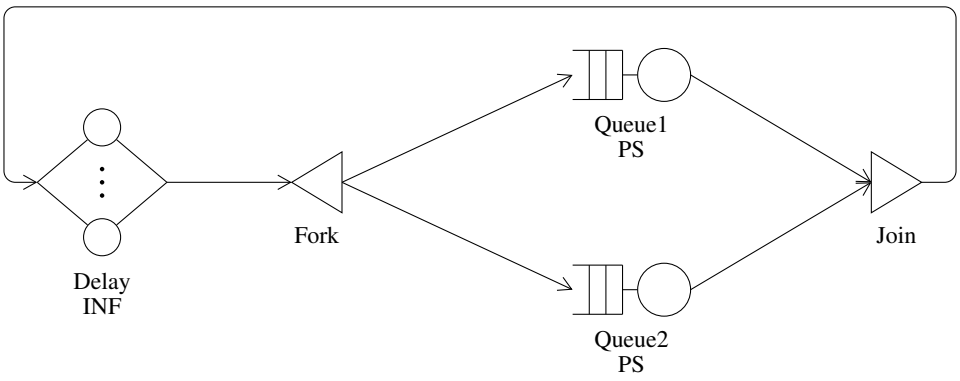


Figure 2.45: Topology of `fj_basic_closed`: a closed network over 3 queueing stations with a fork-join block.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	1: class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – class1: Exp(1) Queue1 – class1: Exp(1) Queue2 – class1: Exp(1)
Solvers	JMT, MVA

Table 2.45: Features of `fj_basic_closed`.

Nodes	7: Queue × 2, Delay, Fork × 2, Join × 2
Classes	2: class1 (closed, $N = 5$), class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – class1: Exp(0.25); class2: Exp(0.25) Queue1 – class1: Exp(1); class2: Exp(2) Queue2 – class1: Exp(0.75); class2: Exp(2)
Features	class switching on 2 links
Solvers	JMT, MVA

Table 2.46: Features of `fj_basic_nesting`.

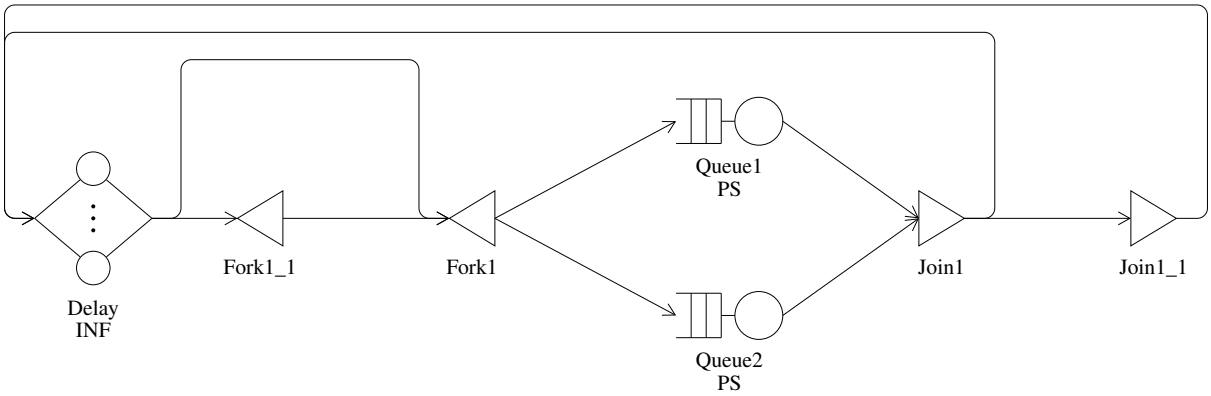


Figure 2.46: Topology of `fj_basic_nesting`: a closed network over 3 queueing stations with a fork-join block.

2.6.4 `fj_basic_open`

One fork, two parallel FCFS branches and a join, in an open network.

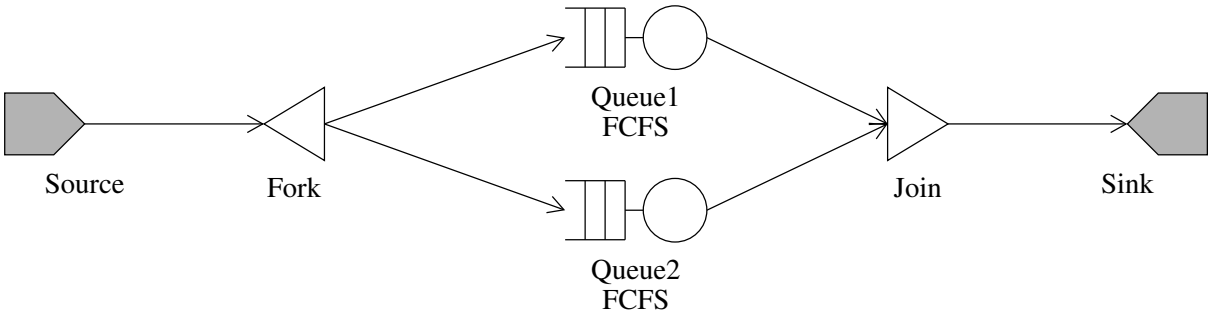


Figure 2.47: Topology of `fj_basic_open`: an open network over 2 queueing stations with a fork-join block.

Nodes	6: Source, Queue × 2, Fork, Join, Sink
Classes	1: class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – class1: Exp(0.05)
Service	Queue1 – class1: Exp(1) Queue2 – class1: Exp(2)
Solvers	JMT, MVA, LDES, MAM

Table 2.47: Features of `fj_basic_open`.

2.6.5 `fj_complex_serial`

Several fork-join blocks in series inside a larger network.

2.6.6 `fj_cs_multi_visits`

Class switching with several visits per branch.

2.6.7 `fj_cs_postfork`

Class switching after the fork, on the sibling tasks.

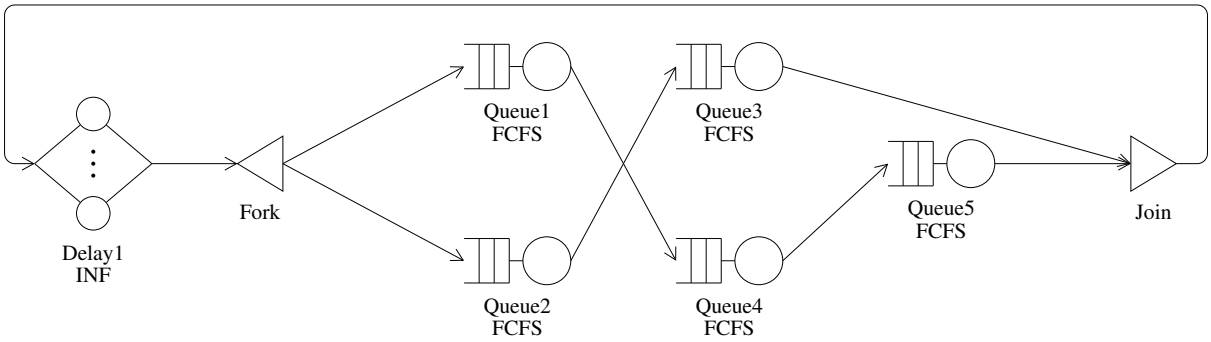


Figure 2.48: Topology of `fj_complex_serial`: a closed network over 6 queueing stations with a fork-join block.

Nodes	8: Queue × 5, Delay, Fork, Join
Classes	1: class1 (closed, $N = 10$)
Scheduling	Delay1: INF; Queue1: FCFS; Queue2: FCFS; Queue3: FCFS; Queue4: FCFS; Queue5: FCFS
Service	Delay1 – class1: Exp(0.5) Queue1 – class1: Exp(1) Queue2 – class1: Exp(2) Queue3 – class1: Exp(1) Queue4 – class1: Exp(3) Queue5 – class1: Exp(0.8)
Solvers	JMT, MVA

Table 2.48: Features of `fj_complex_serial`.

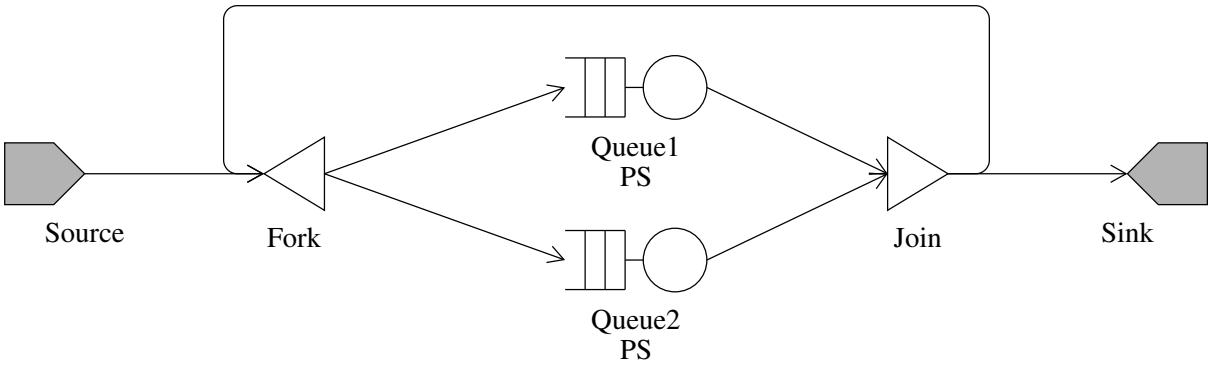


Figure 2.49: Topology of `fj_cs_multi_visits`: an open network over 2 queueing stations with a fork-join block.

Nodes	6: Source, Queue × 2, Fork, Join, Sink
Classes	2: class1 (open), class2 (open)
Scheduling	Queue1: PS; Queue2: PS
Arrivals	Source – class1: Exp(0.1); class2: Disabled
Service	Queue1 – class1: Exp(1); class2: Exp(1) Queue2 – class1: Exp(1); class2: Exp(1)
Features	class switching on 1 link
Solvers	MVA, JMT

Table 2.49: Features of `fj_cs_multi_visits`.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	2: class1 (closed, $N = 1$), class2 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – class1: Exp(0.25); class2: Exp(0.25) Queue1 – class1: Exp(2); class2: Exp(2) Queue2 – class1: Exp(2); class2: Exp(2)
Features	class switching on 3 links
Solvers	MVA, JMT

Table 2.50: Features of `fj_cs_postfork`.

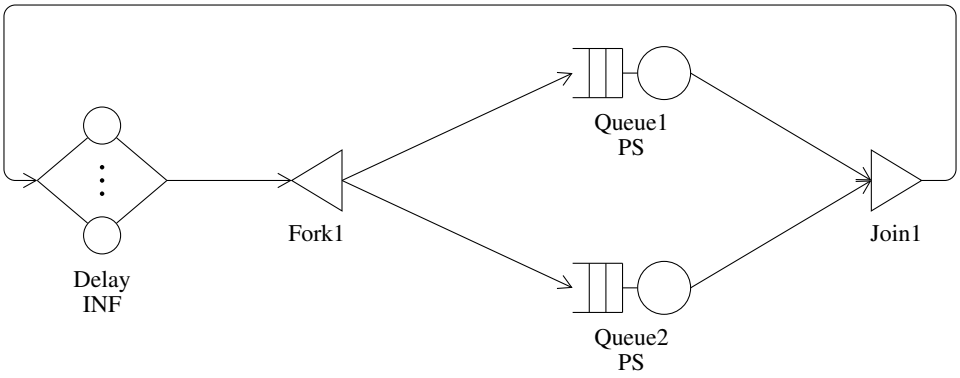


Figure 2.50: Topology of `fj_cs_postfork`: a closed network over 3 queueing stations with a fork-join block.

2.6.8 `fj_cs_prefork`

Class switching before the fork.

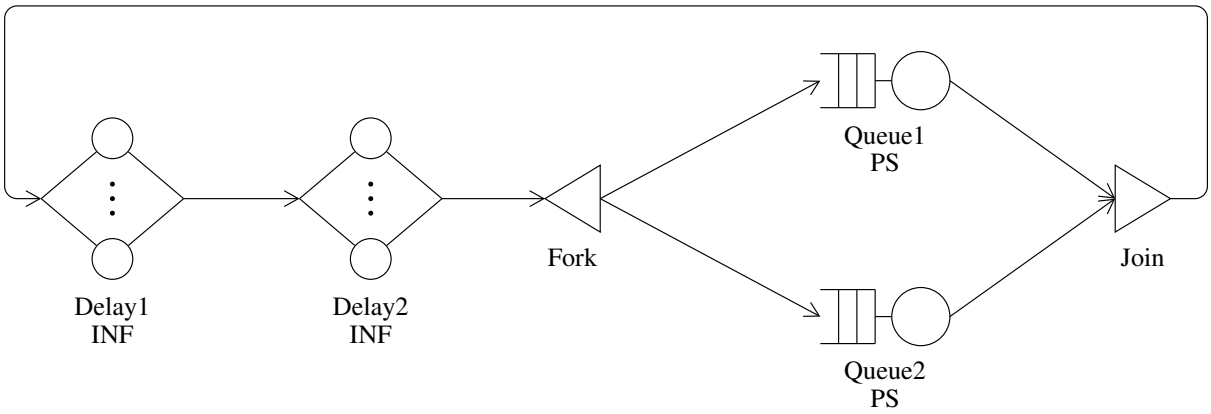


Figure 2.51: Topology of `fj_cs_prefork`: a closed network over 4 queueing stations with a fork-join block.

Nodes	6: Queue × 2, Delay × 2, Fork, Join
Classes	2: class1 (closed, $N = 10$), class2 (closed, $N = 10$)
Scheduling	Delay1: INF; Delay2: INF; Queue1: PS; Queue2: PS
Service	Delay1 – class1: Exp(0.5); class2: Exp(0.5) Delay2 – class1: Disabled; class2: Exp(2) Queue1 – class1: Exp(1); class2: Disabled Queue2 – class1: Exp(1); class2: Disabled
Features	drop rule {'class1': 'waitingQueue'} at Delay2; drop rule {'class2': 'waitingQueue'} at Queue1; drop rule {'class2': 'waitingQueue'} at Queue2; class switching on 5 links
Solvers	JMT, MVA

Table 2.51: Features of `fj_cs_prefork`.

2.6.9 `fj_deep_nesting`

Several nesting levels, the stress case for the join bookkeeping.

2.6.10 `fj_delays`

Delay stations inside the branches.

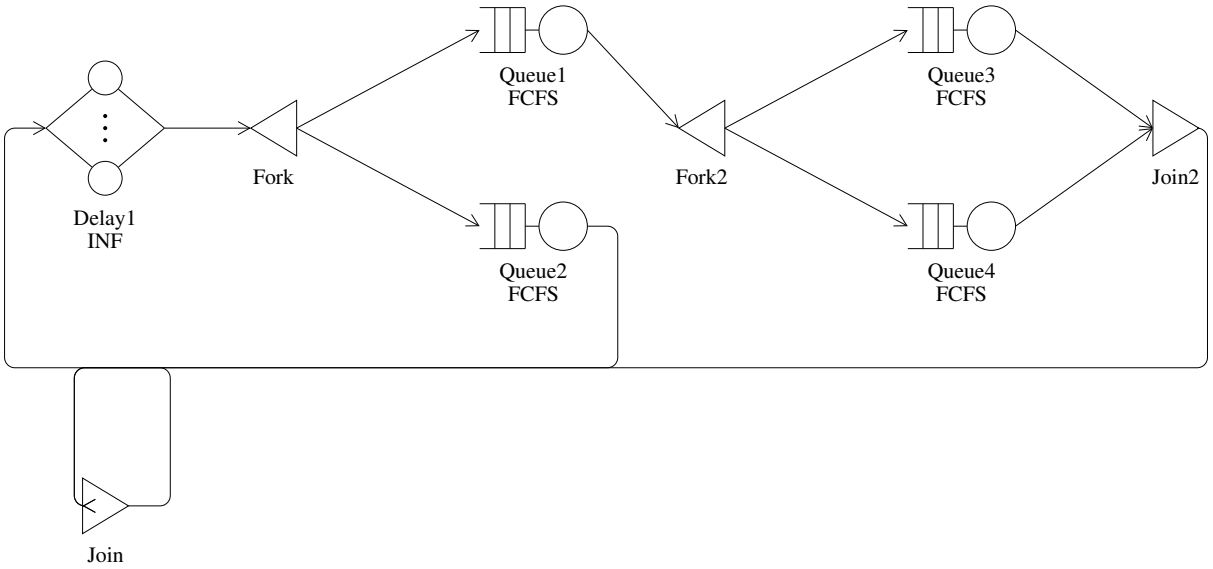


Figure 2.52: Topology of `fj_deep_nesting`: a closed network over 5 queueing stations with a fork-join block.

Nodes	9: Queue × 4, Delay, Fork × 2, Join × 2
Classes	1: class1 (closed, $N = 1$)
Scheduling	Delay1: INF; Queue1: FCFS; Queue2: FCFS; Queue3: FCFS; Queue4: FCFS
Service	Delay1 – class1: Exp(0.5) Queue1 – class1: Exp(1) Queue2 – class1: Exp(1) Queue3 – class1: Exp(2) Queue4 – class1: Exp(2)
Solvers	JMT, MVA

Table 2.52: Features of `fj_deep_nesting`.

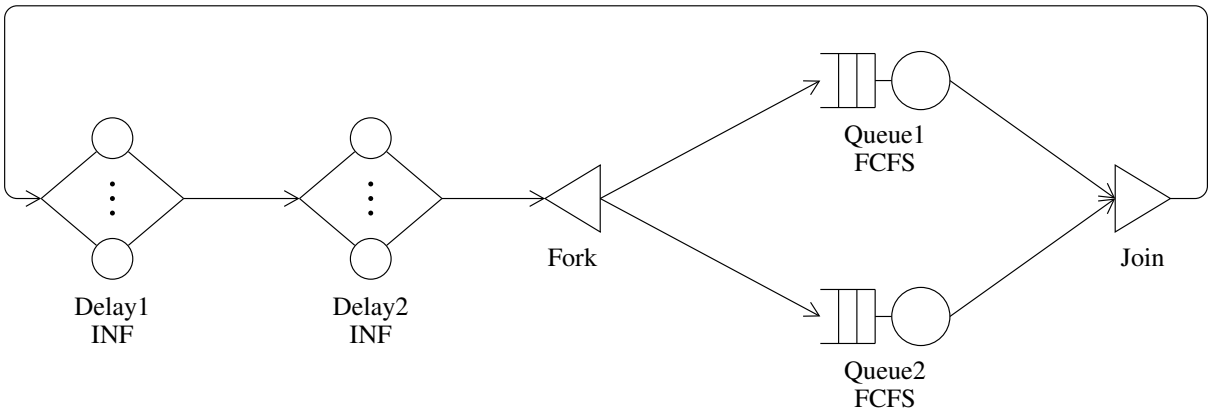


Figure 2.53: Topology of `fj_delays`: a closed network over 4 queueing stations with a fork-join block.

Nodes	6: Queue × 2, Delay × 2, Fork, Join
Classes	1: class1 (closed, $N = 10$)
Scheduling	Delay1: INF; Delay2: INF; Queue1: FCFS; Queue2: FCFS
Service	Delay1 – class1: Exp(0.5) Delay2 – class1: Exp(2) Queue1 – class1: Exp(1) Queue2 – class1: Exp(1)
Solvers	JMT, MVA

Table 2.53: Features of `fj_delays`.

2.6.11 `fj_mixed_openclosed`

Open and closed classes through one fork-join block, with a class disabled on a branch.

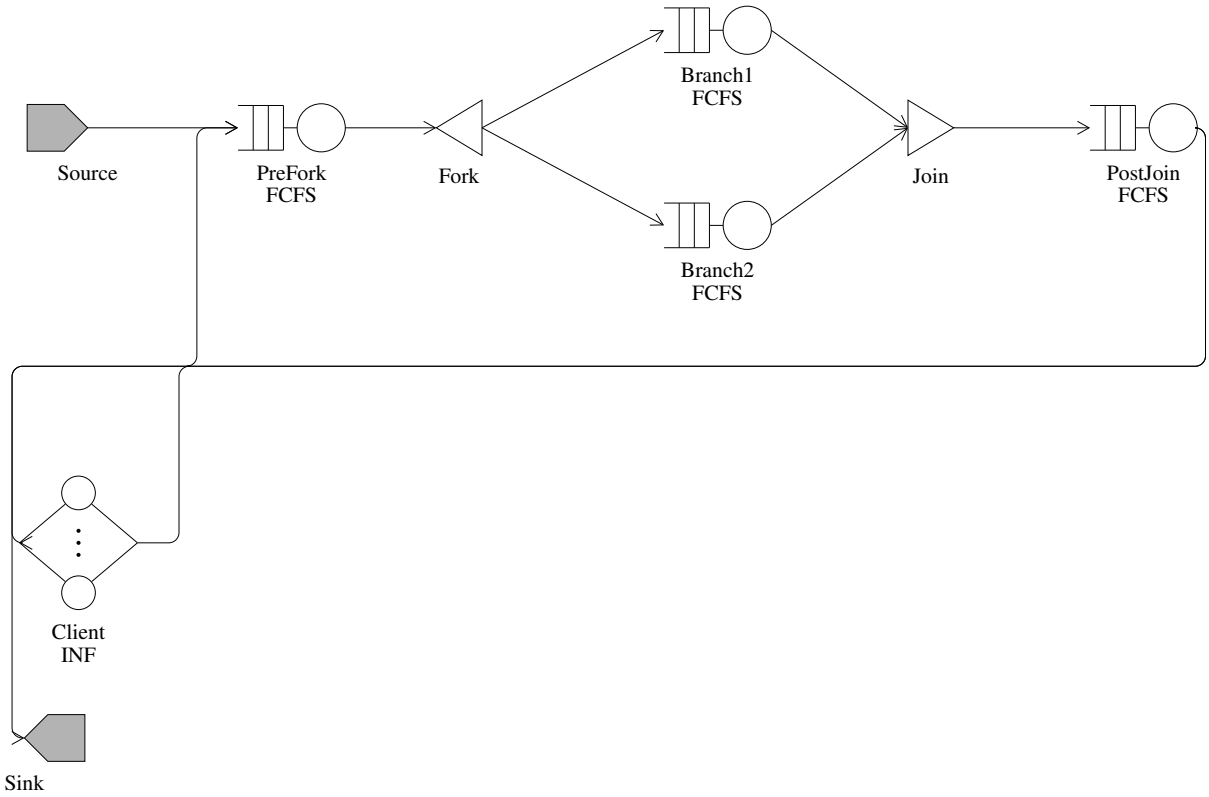


Figure 2.54: Topology of `fj_mixed_openclosed`: an open network over 5 queueing stations with a fork-join block.

Nodes	9: Source, Queue × 4, Delay, Fork, Join, Sink
Classes	2: Open (open), Closed (closed, $N = 1$)
Scheduling	Client: INF; PreFork: FCFS; Branch1: FCFS; Branch2: FCFS; PostJoin: FCFS
Arrivals	Source – Open: Exp(0.1); Closed: Disabled
Service	Client – Open: Disabled; Closed: Exp(1) PreFork – Open: Exp(5); Closed: Exp(5) Branch1 – Open: Exp(3.333); Closed: Exp(3.333) Branch2 – Open: Exp(2.5); Closed: Exp(2.5) PostJoin – Open: Exp(10); Closed: Exp(10)
Features	class switching on 2 links
Solvers	MVA, JMT

Table 2.54: Features of `fj_mixed_openclosed`.

2.6.12 `fj_nojoin`

A fork with no join, so the siblings leave through the sink independently.

2.6.13 `fj_route_overlap`

Branches that share a station, so their visits overlap.

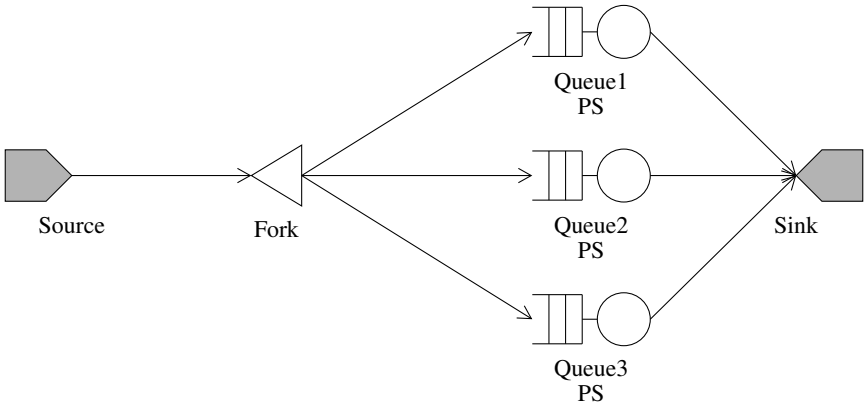


Figure 2.55: Topology of `fj_nojoin`: an open network over 3 queueing stations with a fork-join block.

Nodes	6: Source, Queue × 3, Fork, Sink
Classes	1: class1 (open)
Scheduling	Queue1: PS; Queue2: PS; Queue3: PS
Arrivals	Source – class1: Exp(0.5)
Service	Queue1 – class1: Exp(1) Queue2 – class1: Exp(2) Queue3 – class1: Exp(3)
Solvers	MVA, JMT

Table 2.55: Features of `fj_nojoin`.

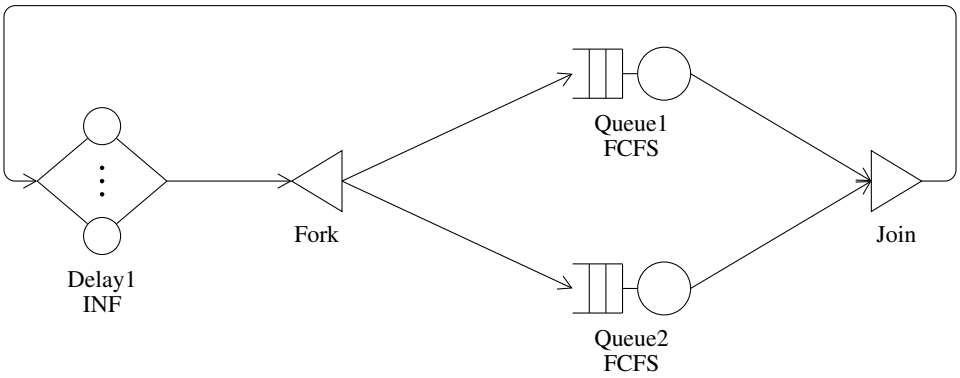


Figure 2.56: Topology of `fj_route_overlap`: a closed network over 3 queueing stations with a fork-join block.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	2: class1 (closed, $N = 10$), class2 (closed, $N = 10$)
Scheduling	Delay1: INF; Queue1: FCFS; Queue2: FCFS
Service	Delay1 – class1: Exp(0.5); class2: Exp(0.2) Queue1 – class1: Exp(1); class2: Exp(1) Queue2 – class1: Exp(2); class2: Exp(2)
Solvers	JMT, MVA

Table 2.56: Features of `fj_route_overlap`.

2.6.14 `fj_serialfjs_closed`

The closed version.

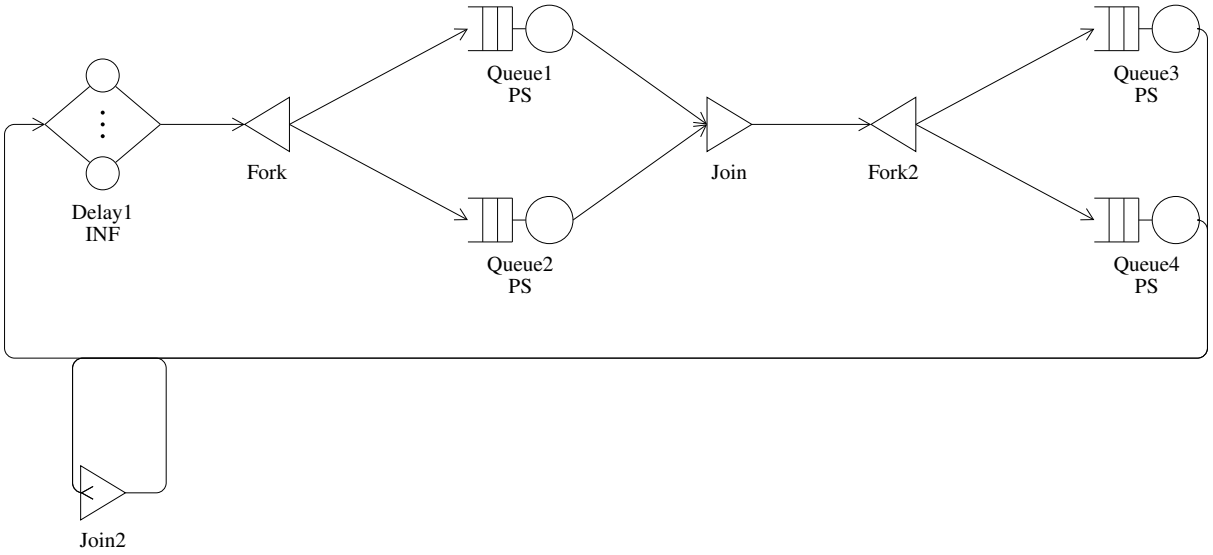


Figure 2.57: Topology of `fj_serialfjs_closed`: a closed network over 5 queueing stations with a fork-join block.

Nodes	9: Queue × 4, Delay, Fork × 2, Join × 2
Classes	1: class1 (closed, $N = 10$)
Scheduling	Delay1: INF; Queue1: PS; Queue2: PS; Queue3: PS; Queue4: PS
Service	Delay1 – class1: Exp(0.5) Queue1 – class1: Exp(1) Queue2 – class1: Exp(1) Queue3 – class1: Exp(1) Queue4 – class1: Exp(1)
Solvers	JMT, MVA

Table 2.57: Features of `fj_serialfjs_closed`.

2.6.15 `fj_serialfjs_open`

A chain of fork-join blocks, open.

Nodes	10: Source, Queue × 4, Fork × 2, Join × 2, Sink
Classes	1: class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS; Queue4: FCFS
Arrivals	Source – class1: Exp(0.4)
Service	Queue1 – class1: Exp(1) Queue2 – class1: Exp(1) Queue3 – class1: Exp(1) Queue4 – class1: Exp(1)
Solvers	JMT, MVA

Table 2.58: Features of `fj_serialfjs_open`.

2.6.16 `fj_threebranches`

Three parallel branches rather than two.

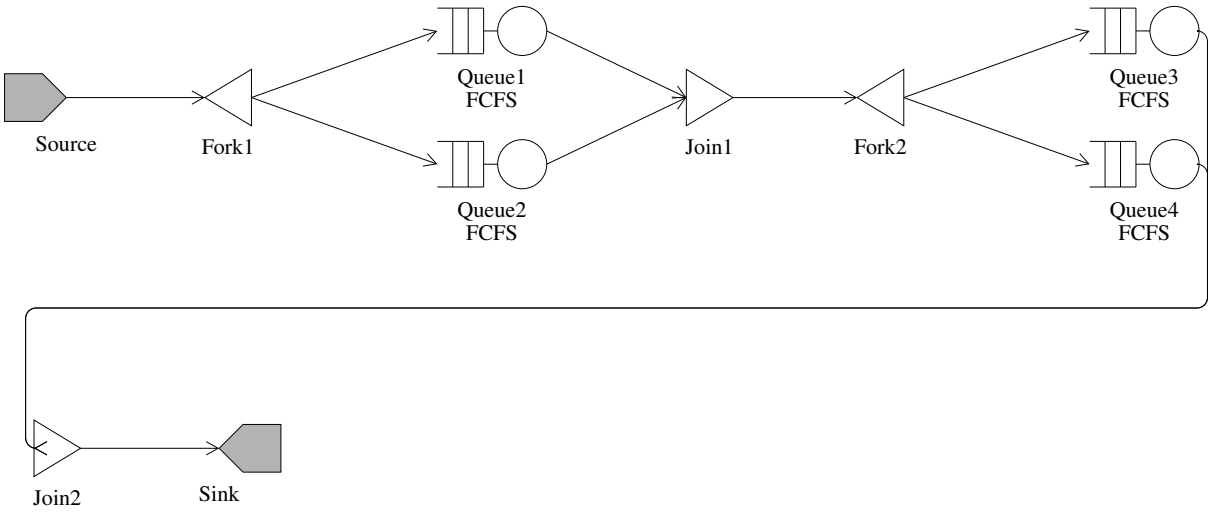


Figure 2.58: Topology of `fj_serialfjs_open`: an open network over 4 queueing stations with a fork-join block.

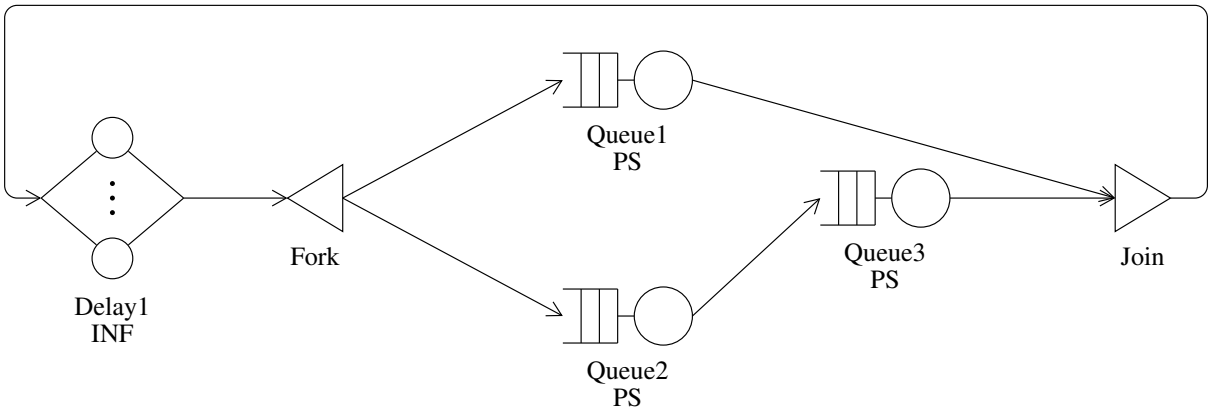


Figure 2.59: Topology of `fj_threebranches`: a closed network over 4 queueing stations with a fork-join block.

Nodes	6: Queue $\times$ 3, Delay, Fork, Join
Classes	2: class1 (closed, $N = 10$), class2 (closed, $N = 10$)
Scheduling	Delay1: INF; Queue1: PS; Queue2: PS; Queue3: PS
Service	Delay1 – class1: Exp(0.5); class2: Exp(0.8) Queue1 – class1: Exp(1.5); class2: Exp(2.8) Queue2 – class1: Exp(1.1); class2: Exp(3) Queue3 – class1: Exp(2.5); class2: Exp(1)
Solvers	MVA

Table 2.59: Features of `fj_threebranches`.

2.6.17 `fj_tiny_closed`

The smallest closed fork-join model, small enough for an exact CTMC solve.

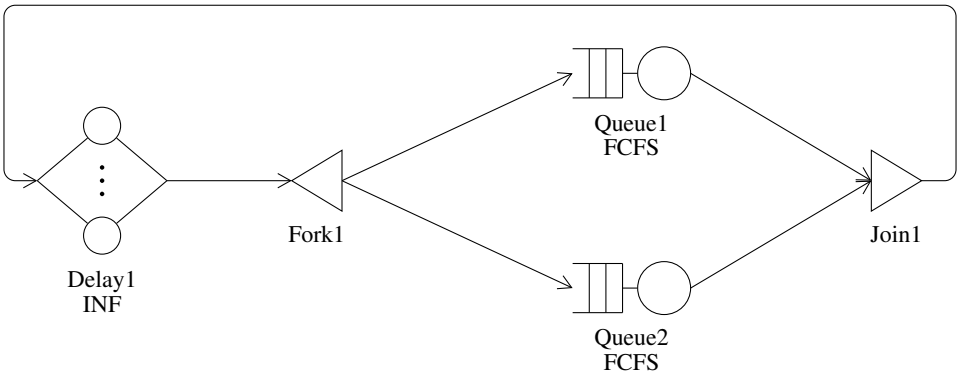


Figure 2.60: Topology of `fj_tiny_closed`: a closed network over 3 queueing stations with a fork-join block.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	1: Class1 (closed, $N = 1$)
Scheduling	Delay1: INF; Queue1: FCFS; Queue2: FCFS
Service	Delay1 – Class1: Exp(1) Queue1 – Class1: Exp(2) Queue2 – Class1: Exp(3)
Solvers	CTMC, SSA, JMT

Table 2.60: Features of `fj_tiny_closed`.

2.6.18 `fj_twoclasses_forked`

Two classes forked independently through the same block.

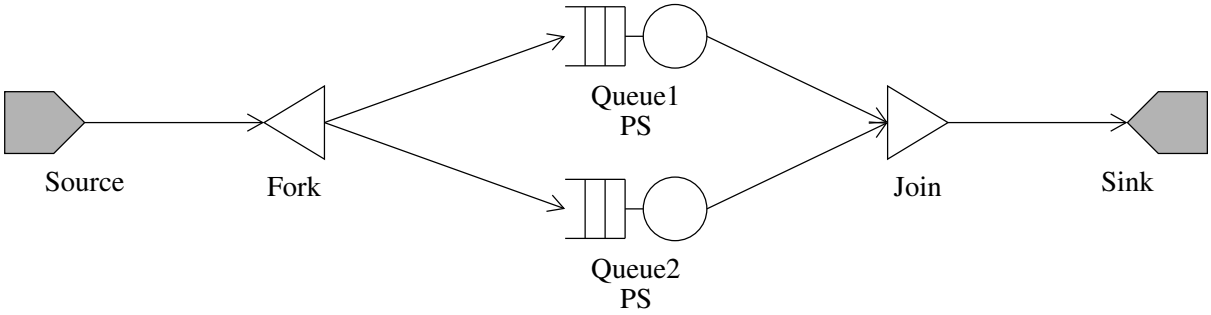


Figure 2.61: Topology of `fj_twoclasses_forked`: an open network over 2 queueing stations with a fork-join block.

2.6.19 `fj_variable_fanout`

A forking level that is not one fixed number: a per-link vector, a random degree, or a branch that fires only part of the time.

Nodes	6: Source, Queue × 2, Fork, Join, Sink
Classes	2: class1 (open), class2 (open)
Scheduling	Queue1: PS; Queue2: PS
Arrivals	Source – class1: Exp(0.25); class2: Exp(0.25)
Service	Queue1 – class1: Exp(1); class2: Immediate Queue2 – class1: Exp(0.75); class2: Exp(2)
Solvers	JMT, MVA

Table 2.61: Features of `fj_twoclasses_forked`.

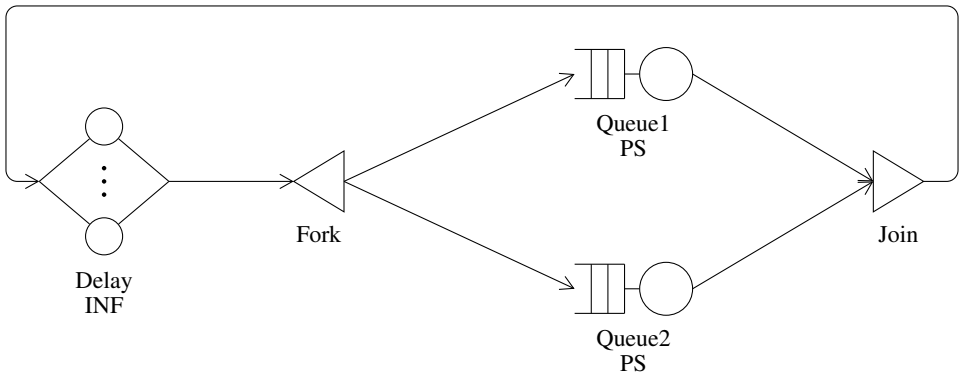


Figure 2.62: Topology of `fj_variable_fanout`: a closed network over 3 queueing stations with a fork-join block.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	1: class1 (closed, $N = 4$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – class1: Exp(1) Queue1 – class1: Exp(2) Queue2 – class1: Exp(2)
Solvers	JMT

Table 2.62: Features of `fj_variable_fanout`.

2.7 Layered queueing networks

A layered queueing network (LQN) models software resources explicitly. A `Processor` hosts `Tasks`, each task exposes `Entry` points, and each entry runs an `Activity` graph that consumes host demand and makes synchronous or asynchronous calls to the entries of other tasks. A synchronous call blocks the caller until the reply arrives, which is what creates the layers: a task is simultaneously a server to its callers and a client of the tasks it calls.

The models are solved by `SolverLN`, which decomposes the layers into queueing submodels and iterates to a fixed point, and by the external `lqns` and `lqsim` tools. The same model is also readable from, and writable to, the `.lqnx` XML interchange format.

2.7.1 `lqn_basic`

One reference task calling one server task, the smallest layered model.

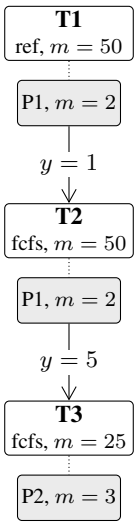


Figure 2.63: Call graph of `lqn_basic`: 3 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 2$), P2 (ps, $m = 3$)
Tasks	3: T1 (ref, $m = 50$), T2 (fcfs, $m = 50$), T3 (fcfs, $m = 25$)
Entries	3: E1, E2, E3
Activities	3, host demands AS1: 0.1, AS2: 0.05, AS3: 0.02
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 2.0

Table 2.63: Features of `lqn_basic`.

2.7.2 `lqn_bpmn`

A BPMN process translated into a layered model with forks, joins and immediate activities.

2.7.3 `lqn_bpmn_trace`

The BPMN translation with the activity trace printed.

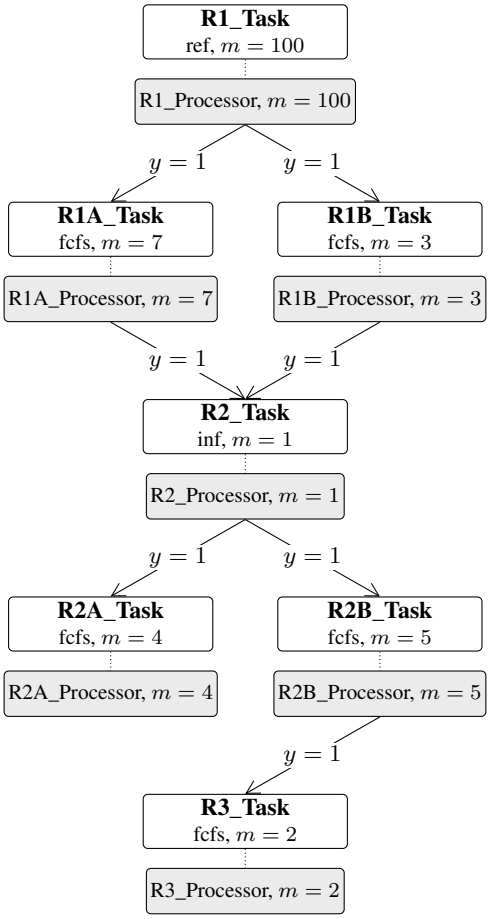


Figure 2.64: Call graph of `lqn_bpmn`: 7 tasks over 7 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	7: R1_Processor (fcfs, $m = 100$), R2_Processor (inf, $m = 1$), R3_Processor (fcfs, $m = 2$), R1A_Processor (fcfs, $m = 7$), R1B_Processor (fcfs, $m = 3$), R2A_Processor (fcfs, $m = 4$), R2B_Processor (fcfs, $m = 5$)
Tasks	7: R1_Task (ref, $m = 100$), R2_Task (inf, $m = 1$), R3_Task (fcfs, $m = 2$), R1A_Task (fcfs, $m = 7$), R1B_Task (fcfs, $m = 3$), R2A_Task (fcfs, $m = 4$), R2B_Task (fcfs, $m = 5$)
Entries	16: R1_Ref_Entry, R2_Synch_A2_Entry, R2_Synch_A5_Entry, R3_Synch_A9_Entry, R1A_Synch_A1_Entry, R1A_Synch_A2_Entry, R1A_Synch_A3_Entry, R1B_Synch_A4_Entry, R1B_Synch_A5_Entry, R1B_Synch_A6_Entry, R2A_Synch_A7_Entry, R2A_Synch_A8_Entry, R2A_Synch_A11_Entry, R2B_Synch_A9_Entry, R2B_Synch_A10_Entry, R2B_Synch_A12_Entry
Activities	29, host demands A1_Empty: 0.0, A2_Empty: 0.0, A5_Empty: 0.0, A6_Empty: 0.0, A3_Empty: 0.0, A4_Empty: 0.0, E4_Empty: 0.0, A7_Empty: 0.0, ...
Calls	15 (15 synchronous, 0 asynchronous)
Think times	R1_Task: 20.0

Table 2.64: Features of `lqn_bpmn`.

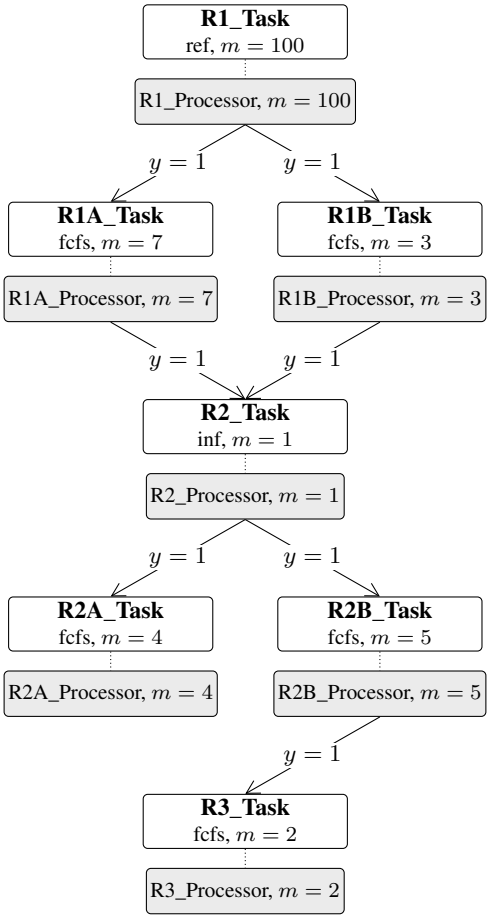


Figure 2.65: Call graph of `lqn_bpmn_trace`: 7 tasks over 7 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	7: R1_Processor (fcfs, $m = 100$), R2_Processor (inf, $m = 1$), R3_Processor (fcfs, $m = 2$), R1A_Processor (fcfs, $m = 7$), R1B_Processor (fcfs, $m = 3$), R2A_Processor (fcfs, $m = 4$), R2B_Processor (fcfs, $m = 5$)
Tasks	7: R1_Task (ref, $m = 100$), R2_Task (inf, $m = 1$), R3_Task (fcfs, $m = 2$), R1A_Task (fcfs, $m = 7$), R1B_Task (fcfs, $m = 3$), R2A_Task (fcfs, $m = 4$), R2B_Task (fcfs, $m = 5$)
Entries	16: R1_Ref_Entry, R2_Synch_A2_Entry, R2_Synch_A5_Entry, R3_Synch_A9_Entry, R1A_Synch_A1_Entry, R1A_Synch_A2_Entry, R1A_Synch_A3_Entry, R1B_Synch_A4_Entry, R1B_Synch_A5_Entry, R1B_Synch_A6_Entry, R2A_Synch_A7_Entry, R2A_Synch_A8_Entry, R2A_Synch_A11_Entry, R2B_Synch_A9_Entry, R2B_Synch_A10_Entry, R2B_Synch_A12_Entry
Activities	29, host demands A1_Empty: 0.0, A2_Empty: 0.0, A5_Empty: 0.0, A6_Empty: 0.0, A3_Empty: 0.0, A4_Empty: 0.0, E4_Empty: 0.0, A7_Empty: 0.0, ...
Calls	15 (15 synchronous, 0 asynchronous)
Think times	R1_Task: 20.0

Table 2.65: Features of `lqn_bpmn_trace`.

2.7.4 lqn_flatph

Method `flat.ph`: the squashed layering carrying the same composed phase-type encoding, one submodel holding every processor and called task.

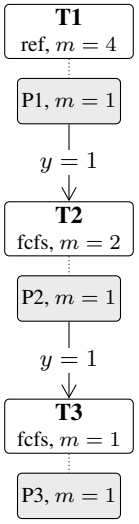


Figure 2.66: Call graph of `lqn_flatph`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	3: P1 (ps, $m = 1$), P2 (ps, $m = 1$), P3 (ps, $m = 1$)
Tasks	3: T1 (ref, $m = 4$), T2 (fcfs, $m = 2$), T3 (fcfs, $m = 1$)
Entries	3: E1, E2, E3
Activities	3, host demands A1: 0.2, A2: 0.2, A3: 0.2
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 1.0

Table 2.66: Features of `lqn_flatph`.

2.7.5 lqn_fork_open_arrival

An AND-fork under an open arrival stream.

Processors	2: P1 (inf, $m = 1$), P2 (inf, $m = 1$)
Tasks	2: Client (ref, $m = 1$), Server (fcfs, $m = 1$)
Entries	3: CE, SE, OE
Activities	9, host demands CA: 0.5, RA1: 0.2, RA2: 0.3, RA3: 0.4, RA4: 0.1, OA1: 0.2, OA2: 0.3, OA3: 0.4, ...
Calls	1 (1 synchronous, 0 asynchronous)
Think times	Client: 1.0

Table 2.67: Features of `lqn_fork_open_arrival`.

2.7.6 lqn_init

A layered model whose `SolverLN` run is warm-started from the `LQNS` solution, and the elapsed time that initialization saves.

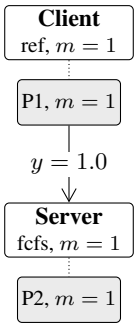


Figure 2.67: Call graph of `lqn_fork_open_arrival`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

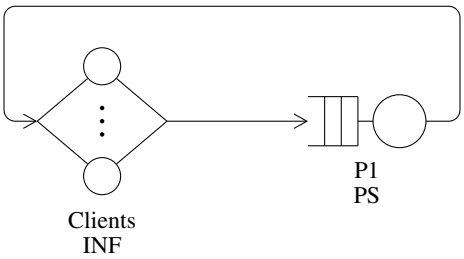


Figure 2.68: Topology of `lqn_init`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: T1 (closed, $N = 10$)
Scheduling	Clients: INF; P1: PS
Service	Clients – T1: Exp(0.008971) P1 – T1: APH

Table 2.68: Features of `lqn_init`.

2.7.7 `lqn_jsq`

Join-the-shortest-queue dispatching between replicated tasks.

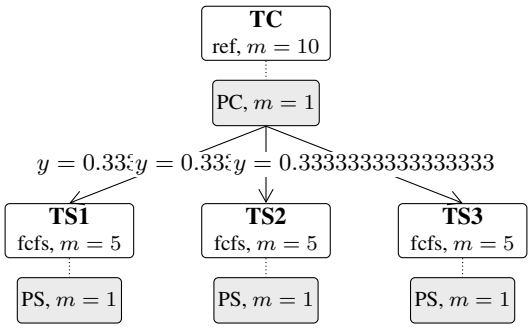


Figure 2.69: Call graph of `lqn_jsq`: 4 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: PC (inf, $m = 1$), PS (ps, $m = 1$)
Tasks	4: TC (ref, $m = 10$), TS1 (fcfs, $m = 5$), TS2 (fcfs, $m = 5$), TS3 (fcfs, $m = 5$)
Entries	4: EC, ES1, ES2, ES3
Activities	4, host demands AC: 0.5, AS1: 1.0, AS2: 1.0, AS3: 1.0
Calls	3 (3 synchronous, 0 asynchronous)
Think times	TC: 5.0

Table 2.69: Features of `lqn_jsq`.

2.7.8 `lqn_moment3`

The third moment of the entry service time, and how it propagates up the layers.

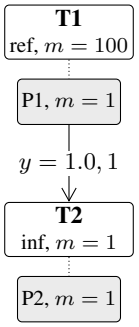


Figure 2.70: Call graph of `lqn_moment3`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

2.7.9 `lqn_multi_solvers`

One layered model solved by every layered solver, side by side.

2.7.10 `lqn_ofbiz`

The Apache OFBiz case study, read from its `.lqnx` file.

Processors	2: P1 (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 100$), T2 (inf, $m = 1$)
Entries	3: E1, E2, E3
Activities	5, host demands A1: 1.0, A20: 1.0, A21: 1.0, A22: 1.0, A3: 1.0
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 10.0
Solvers	FLD

Table 2.70: Features of `lqn_moment3`.

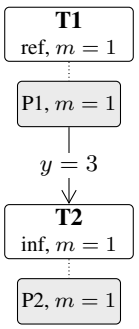


Figure 2.71: Call graph of `lqn_multi_solvers`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (inf, $m = 1$), P2 (inf, $m = 1$)
Tasks	2: T1 (ref, $m = 1$), T2 (inf, $m = 1$)
Entries	2: E1, E2
Activities	2, host demands A1: 1.0, A2: 1.0
Calls	1 (1 synchronous, 0 asynchronous)
Think times	T1: 0.0001

Table 2.71: Features of `lqn_multi_solvers`.

Processors	9: FrontEnd_CPU_Processor (ps, $m = 1$), USAGE_DELAY_Processor (inf, $m = 1$), UsageScenario_userType1_1_Processor (fcfs, $m = 1$), RequestHandler_HandlerIF_main_345_Processor (fcfs, $m = 1$), RequestHandler_HandlerIF_login_345_Processor (fcfs, $m = 1$), RequestHandler_HandlerIF_checkLogin_345_Processor (fcfs, $m = 1$), RequestHandler_HandlerIF_logout_345_Processor (fcfs, $m = 1$), UsageScenario_userType2_7_Processor (fcfs, $m = 1$), RequestHandler_HandlerIF_quickadd_345_Processor (fcfs, $m = 1$)
Tasks	9: FrontEnd_CPU_Task (inf, $m = 1$), USAGE_DELAY_Task (inf, $m = 1$), UsageScenario_userType1_1_Task (ref, $m = 10$), RequestHandler_HandlerIF_main_345_Task (inf, $m = 1$), RequestHandler_HandlerIF_login_345_Task (inf, $m = 1$), RequestHandler_HandlerIF_checkLogin_345_Task (inf, $m = 1$), RequestHandler_HandlerIF_logout_345_Task (inf, $m = 1$), UsageScenario_userType2_7_Task (ref, $m = 10$), RequestHandler_HandlerIF_quickadd_345_Task (inf, $m = 1$)
Entries	14: FrontEnd_CPU_Entry, InternalAction_main__Iu1-wMhoEeKON4DtRoKCMw_34_50_Entry, InternalAction_login__YgxRGw26EeSPwb7XgvxhWQ_34_50_Entry, InternalAction_CheckLogin__vHMZsMhoEeKON4DtRoKCMw_34_50_Entry, InternalAction_logout__j-2E4MhrEeKON4DtRoKCMw_34_50_Entry, InternalAction_addcart__5JEHQMhoEeKON4DtRoKCMw_34_50_Entry, USAGE_DELAY0_Entry, UsageScenario_userType1_1_Entry, RequestHandler_HandlerIF_main_345_Entry, RequestHandler_HandlerIF_login_345_Entry, RequestHandler_HandlerIF_checkLogin_345_Entry, RequestHandler_HandlerIF_logout_345_Entry, UsageScenario_userType2_7_Entry, RequestHandler_HandlerIF_quickadd_345_Entry
Activities	40, host demands FrontEnd_CPU_Activity: 0.0, InternalAction_main__Iu1-wMhoEeKON4DtRoKCMw_34_50_Activity: 0.01, InternalAction_login__YgxRGw26EeSPwb7XgvxhWQ_34_50_Activity: 0.01, InternalAction_CheckLogin__vHMZsMhoEeKON4DtRoKCMw_34_50_Activity: 0.01, InternalAction_logout__j-2E4MhrEeKON4DtRoKCMw_34_50_Activity: 0.01, InternalAction_addcart__5JEHQMhoEeKON4DtRoKCMw_34_50_Activity: 0.01, USAGE_DELAY0_Activity: 0.0, Start2: 0.0, ...
Calls	19 (19 synchronous, 0 asynchronous)
Think times	UsageScenario_userType1_1_Task: 10.0, UsageScenario_userType2_7_Task: 14.0

Table 2.72: Features of `lqn_ofbiz`.

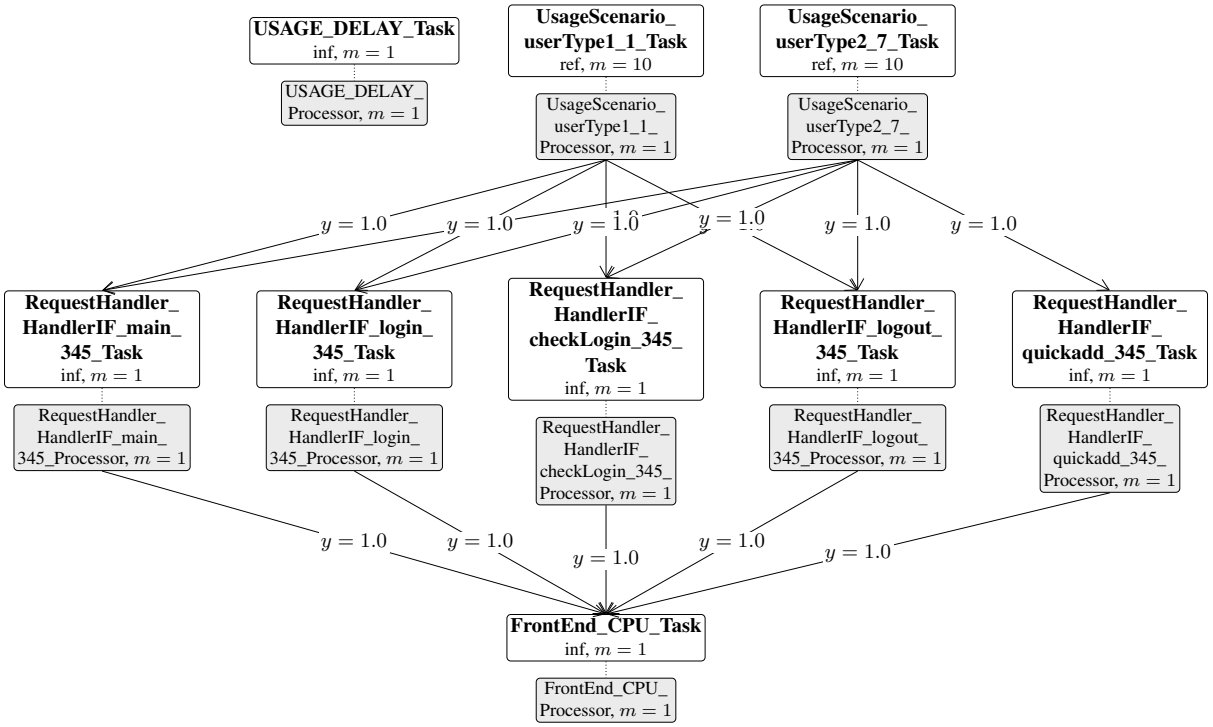


Figure 2.72: Call graph of `lqn_ofbiz`: 9 tasks over 9 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

2.7.11 `lqn_open_arrival`

An open arrival stream into an entry, instead of a closed reference task.

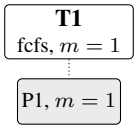


Figure 2.73: Call graph of `lqn_open_arrival`: 1 task over 1 processor, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	1: P1 (ps, $m = 1$)
Tasks	1: T1 (fcfs, $m = 1$)
Entries	1: E1
Activities	1, host demands A1: 1.6

Table 2.73: Features of `lqn_open_arrival`.

2.7.12 `lqn_paramident`

Parameter identification on a layered model, fitting host demands to measurements.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.7.13 `lqn_rrobin`

Round-robin dispatching between replicated tasks.

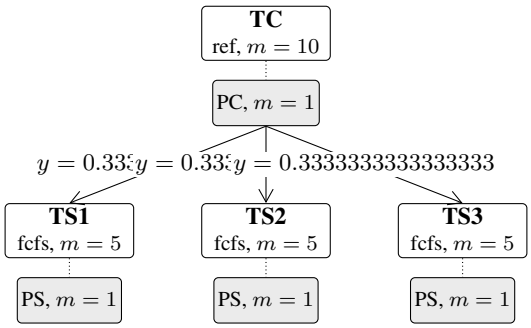


Figure 2.74: Call graph of `lqn_rrobin`: 4 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: PC (inf, $m = 1$), PS (ps, $m = 1$)
Tasks	4: TC (ref, $m = 10$), TS1 (fcfs, $m = 5$), TS2 (fcfs, $m = 5$), TS3 (fcfs, $m = 5$)
Entries	4: EC, ES1, ES2, ES3
Activities	4, host demands AC: 0.5, AS1: 1.0, AS2: 1.0, AS3: 1.0
Calls	3 (3 synchronous, 0 asynchronous)
Think times	TC: 5.0

Table 2.74: Features of `lqn_rrobin`.

2.7.14 `lqn_serial`

A serial call chain across three layers, with an immediate activity.

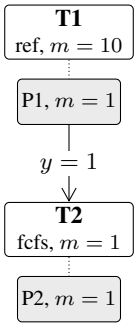


Figure 2.75: Call graph of `lqn_serial`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

2.7.15 `lqn_server_pools`

A layered model with heterogeneous server pools, compared with the compatibility representation that treats the servers as interchangeable.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Processors	2: P1 (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 10$), T2 (fcfs, $m = 1$)
Entries	2: E1, E2
Activities	4, host demands AS1: 1.6, AS2: 0.0, AS3: 5.0, AS4: 1.0
Calls	1 (1 synchronous, 0 asynchronous)
Think times	T1: 100.0

Table 2.75: Features of `lqn_serial`.

2.7.16 `lqn_setup`

Task setup and initialization activities before service.

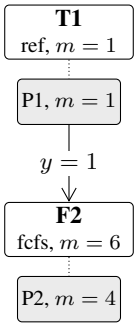


Figure 2.76: Call graph of `lqn_setup`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (inf, $m = 1$), P2 (fcfs, $m = 4$)
Tasks	2: T1 (ref, $m = 1$), F2 (fcfs, $m = 6$)
Entries	2: E1, E2
Activities	2, host demands A1: 1.0, A2: 0.3333333333333333
Calls	1 (1 synchronous, 0 asynchronous)

Table 2.76: Features of `lqn_setup`.

2.7.17 `lqn_sockshop`

The Sock Shop microservice benchmark as a layered model.

2.7.18 `lqn_srvnph`

Method `srvnph`: each entry activity graph composed into one phase-type service law, so an AND fork-join and a loop survive as a distribution rather than as routing.

2.7.19 `lqn_transient`

The transient response of a layered model, through `getTranAvg`.

2.7.20 `lqn_twotasks`

Two tasks over separate processors, the two-layer archetype.

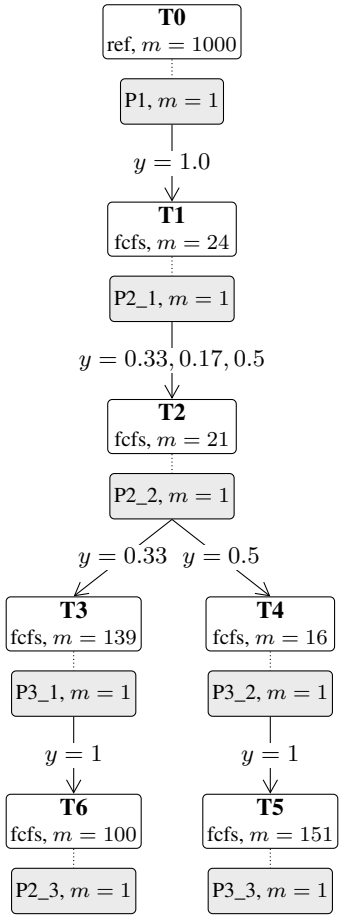


Figure 2.77: Call graph of `lqn_sockshop`: 7 tasks over 7 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	7: P1 (inf, $m = 1$), P2_1 (ps, $m = 1$), P2_2 (ps, $m = 1$), P2_3 (ps, $m = 1$), P3_1 (ps, $m = 1$), P3_2 (ps, $m = 1$), P3_3 (ps, $m = 1$)
Tasks	7: T0 (ref, $m = 1000$), T1 (fcfs, $m = 24$), T2 (fcfs, $m = 21$), T6 (fcfs, $m = 100$), T3 (fcfs, $m = 139$), T4 (fcfs, $m = 16$), T5 (fcfs, $m = 151$)
Entries	12: E0, E1, E2, E3, E4, E11, E5, E6, E7, E8, E9, E10
Activities	24, host demands AS: 5e-08, AS0: 5e-08, AS1: 5e-08, AS2: 0.0022216, AH1: 5e-08, AH2: 0.0021319, AH3: 5e-08, AH4: 0.0037561, ...
Calls	14 (14 synchronous, 0 asynchronous)
Think times	T0: 7.0

Table 2.77: Features of `lqn_sockshop`.

Processors	3: P1 (inf, $m = 1$), P2 (ps, $m = 1$), P3 (ps, $m = 1$)
Tasks	3: T1 (ref, $m = 20$), T2 (fcfs, $m = 5$), T3 (fcfs, $m = 3$)
Entries	4: E1, E2, E3, E4
Activities	10, host demands A1: 0.1, A20: 0.2, A21: 0.3, A22: 0.2, A23: 0.1, A30: 0.1, A31: 0.2, A32: 0.1, ...
Calls	3 (3 synchronous, 0 asynchronous)
Think times	T1: 1.0

Table 2.78: Features of `lqn_srvnph`.

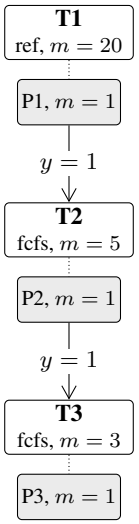


Figure 2.78: Call graph of `lqn_srvnph`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

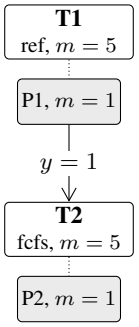


Figure 2.79: Call graph of `lqn_transient`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 5$), T2 (fcfs, $m = 5$)
Entries	2: E1, E2
Activities	2, host demands A1: 0.5, A2: 0.3333333333333333
Calls	1 (1 synchronous, 0 asynchronous)
Think times	T1: 1.0

Table 2.79: Features of `lqn_transient`.

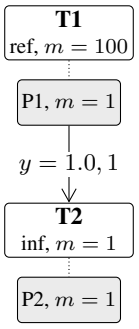


Figure 2.80: Call graph of `lqn_twotasks`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 100$), T2 (inf, $m = 1$)
Entries	3: E1, E2, E3
Activities	5, host demands A1: 1.0, A20: 1.0, A21: 1.0, A22: 1.0, A3: 1.0
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 10.0

Table 2.80: Features of `lqn_twotasks`.

2.7.21 `lqn_workflows`

Workflow patterns (sequence, AND-fork and join, loop) expressed as activity graphs.

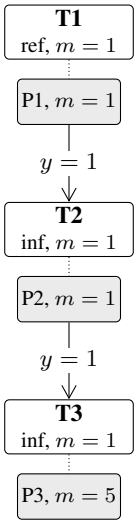


Figure 2.81: Call graph of `lqn_workflows`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	3: P1 (inf, $m = 1$), P2 (inf, $m = 1$), P3 (ps, $m = 5$)
Tasks	3: T1 (ref, $m = 1$), T2 (inf, $m = 1$), T3 (inf, $m = 1$)
Entries	3: Entry, E2, E1
Activities	14, host demands A1: 1.0, A2: 2.0, A3: 3.0, B1: 0.1, B2: 0.2, B3: 0.3, B4: 0.4, B5: 0.5, ...
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 0.0

Table 2.81: Features of `lqn_workflows`.

2.8 Cache models

A `Cache` node holds a finite set of items under a replacement policy. Each request names an item drawn from a popularity distribution, typically Zipf; a hit is served at once, a miss must be fetched and inserted, which evicts an item. The metric of interest is the hit ratio, and everything else in the model follows from it.

The first group of scripts fixes the workload and varies the replacement policy (LRU, FIFO, RR, CLIMB, H-LRU, Q-LRU), so the hit ratios are directly comparable. The second group builds retrieval systems, in which the cache sits in front of a backing store and the miss path is itself a queueing network,

and the delayed-hit effect appears: a request for an item already being fetched waits for that fetch rather than starting another.

2.8.1 cache_compare_replc

RR, FIFO and LRU on one item popularity. CTMC and MVA report all three hit ratios; NC does not support LRU and reports that cell as NaN rather than implying support.

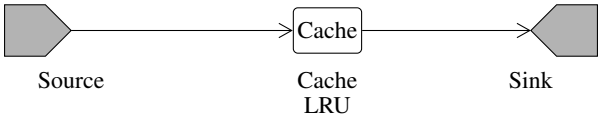


Figure 2.82: Topology of cache_compare_replc: an open network with a cache node.

Nodes	3: Source, Cache, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Arrivals	Source – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled
Features	cache: LRU replacement, 5 items, capacity 2,1; class switching on 2 links

Table 2.82: Features of cache_compare_replc.

2.8.2 cache_itemsize_costcap

Per-item storage costs and a cap on the cost each list may hold, so an insertion that would breach the cap serves the request without changing the cache state.

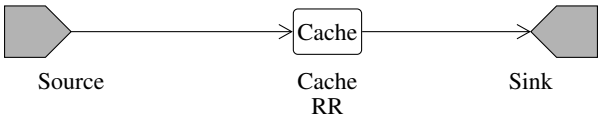


Figure 2.83: Topology of cache_itemsize_costcap: an open network with a cache node.

2.8.3 cache_mmap_rr_env

A marked MAP request stream into a random-replacement cache inside a random environment.

2.8.4 cache_replc_climb

The CLIMB policy, which promotes a hit by one list only.

2.8.5 cache_replc_fifo

First-in first-out replacement.

2.8.6 cache_replc_hlru

H-LRU, the multilist generalization of LRU.

2.8.7 cache_replc_lru

Least-recently-used replacement, with a closed class circulating between a Delay and the cache.

Nodes	3: Source, Cache, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Arrivals	Source – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled
Features	cache: RR replacement, 6 items, capacity 1,1; class switching on 2 links
Solvers	NC

Table 2.83: Features of `cache_itemsize_costcap`.

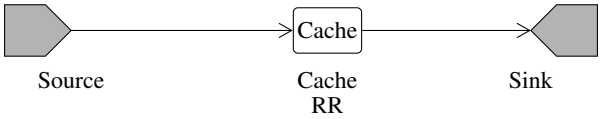


Figure 2.84: Topology of `cache_mmap_rr_env`: an open network with a cache node.

Nodes	3: Source, Cache, Sink
Classes	6: Read1 (open), Read2 (open), Hit1 (open), Miss1 (open), Hit2 (open), Miss2 (open)
Arrivals	Source – Read1: MMAP; Read2: MMAP; Hit1: Disabled; Miss1: Disabled; Hit2: Disabled; Miss2: Disabled
Features	cache: RR replacement, 4 items, capacity 2; class switching on 2 links
Solvers	LDES, CTMC

Table 2.84: Features of `cache_mmap_rr_env`.

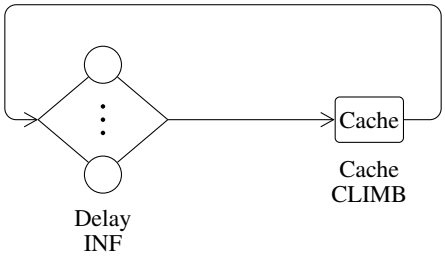


Figure 2.85: Topology of `cache_replc_climb`: a closed network over 1 queueing station with a cache node.

Nodes	2: Delay, Cache
Classes	3: JobClass (closed, $N = 1$), HitClass (closed, $N = 0$), MissClass (closed, $N = 0$)
Scheduling	Delay: INF
Service	Delay – JobClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Delay; cache: CLIMB replacement, 5 items, capacity 2; class switching on 2 links
Solvers	CTMC

Table 2.85: Features of `cache_replc_climb`.

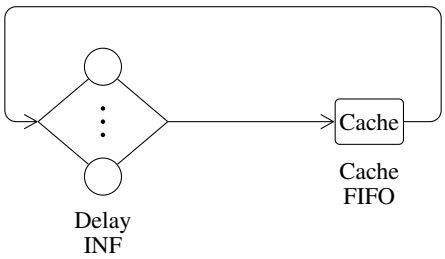


Figure 2.86: Topology of `cache_replc_fifo`: a closed network over 1 queueing station with a cache node.

Nodes	2: Delay, Cache
Classes	3: JobClass (closed, $N = 1$), HitClass (closed, $N = 0$), MissClass (closed, $N = 0$)
Scheduling	Delay: INF
Service	Delay – JobClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Delay; cache: FIFO replacement, 5 items, capacity 2; class switching on 2 links
Solvers	CTMC, SSA, MVA, FLD

Table 2.86: Features of `cache_replc_fifo`.

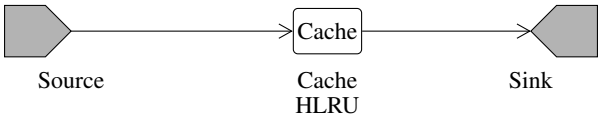


Figure 2.87: Topology of `cache_replc_hlru`: an open network with a cache node.

Nodes	3: Source, Cache, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	cache: HLRU replacement, 6 items, capacity 2,1; class switching on 2 links
Solvers	CTMC, MVA, SSA

Table 2.87: Features of `cache_replc_hlru`.

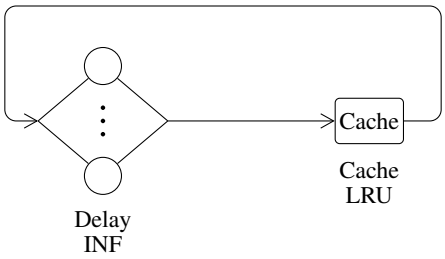


Figure 2.88: Topology of `cache_replc_lru`: a closed network over 1 queueing station with a cache node.

Nodes	2: Delay, Cache
Classes	3: JobClass (closed, $N = 1$), HitClass (closed, $N = 0$), MissClass (closed, $N = 0$)
Scheduling	Delay: INF
Service	Delay – JobClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Delay; cache: LRU replacement, 5 items, capacity 2; class switching on 2 links
Solvers	CTMC, SSA, MVA

Table 2.88: Features of `cache_replc_lru`.

2.8.8 cache_replc_qlru

Q-LRU, which promotes a hit with probability q.

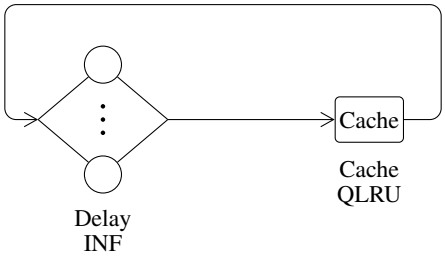


Figure 2.89: Topology of cache_replc_qlru: a closed network over 1 queueing station with a cache node.

Nodes	2: Delay, Cache
Classes	3: JobClass (closed, $N = 1$), HitClass (closed, $N = 0$), MissClass (closed, $N = 0$)
Scheduling	Delay: INF
Service	Delay – JobClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Delay; cache: QLRU replacement, 5 items, capacity 2; class switching on 2 links
Solvers	CTMC

Table 2.89: Features of cache_replc_qlru.

2.8.9 cache_replc_routing

Cache-aware routing: hits and misses leave the cache on different arcs, under RAND and round-robin routing.

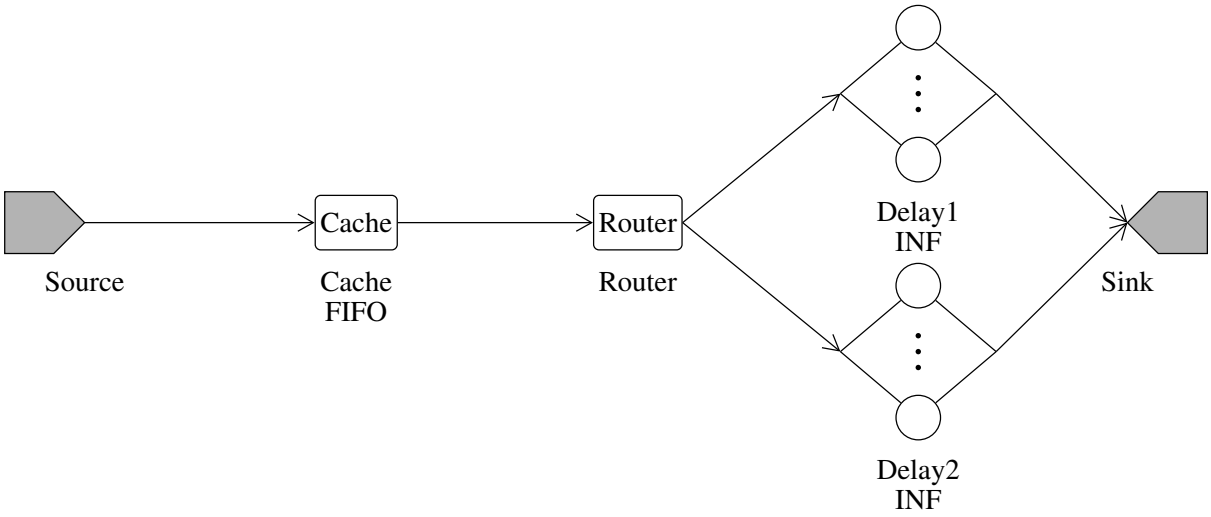


Figure 2.90: Topology of cache_replc_routing: an open network over 2 queueing stations with a cache node and a router.

2.8.10 cache_replc_rr

Random replacement.

Nodes	6: Source, Delay × 2, Cache, Router, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Scheduling	Delay1: INF; Delay2: INF
Arrivals	Source – InitClass: Exp(2)
Service	Delay1 – HitClass: Exp(10); MissClass: Exp(1) Delay2 – HitClass: Exp(20); MissClass: Exp(2)
Features	cache: FIFO replacement, 5 items, capacity 2; class switching on 6 links
Solvers	CTMC, SSA, MVA, NC, FLD

Table 2.90: Features of `cache_replc_routing`.

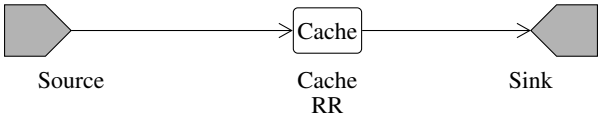


Figure 2.91: Topology of `cache_replc_rr`: an open network with a cache node.

2.8.11 `cache_rmf_transient`

The transient hit ratio from the refined mean field approximation.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.8.12 `cachenet_tandem`

A tandem of two caches over one item set, where a miss at the first is a read of the same item at the second. Item identity is carried by a dedicated job class per item, so no class switching happens on the arc between the caches.

2.8.13 `retrieval_chain`

A chain of caches, each backing the one before it.

2.8.14 `retrieval_default`

The same system with a PS backing store and the default retrieval parameters.

2.8.15 `retrieval_ps`

The PS variant, with the delayed-hit behaviour made explicit.

2.8.16 `retrieval_routing`

Retrieval with routing that separates the hit and miss paths.

Nodes	3: Source, Cache, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Arrivals	Source – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled
Features	cache: RR replacement, 5 items, capacity 2; class switching on 2 links
Solvers	CTMC, SSA, MVA, NC, FLD

Table 2.91: Features of `cache_replc_rr`.

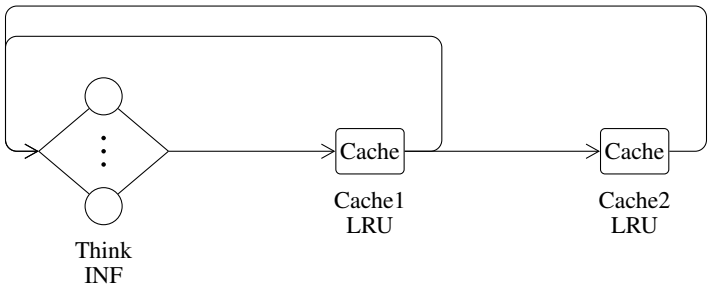


Figure 2.92: Topology of `cachenet_tandem`: a closed network over 1 queueing station with a cache node.

Nodes	3: Delay, Cache $\times$ 2
Classes	15: Read1 (closed, $N = 1$), Hit1_1 (closed, $N = 0$), Hit2_1 (closed, $N = 0$), Miss_1 (closed, $N = 0$), Read2 (closed, $N = 1$), Hit1_2 (closed, $N = 0$), Hit2_2 (closed, $N = 0$), Miss_2 (closed, $N = 0$), Read3 (closed, $N = 1$), Hit1_3 (closed, $N = 0$), Hit2_3 (closed, $N = 0$), Miss_3 (closed, $N = 0$), Cache2_item1 (closed, $N = 0$), Cache2_item2 (closed, $N = 0$), Cache2_item3 (closed, $N = 0$)
Scheduling	Think: INF
Service	Think – Read1: Exp(0.5); Hit1_1: Disabled; Hit2_1: Disabled; Miss_1: Disabled; Read2: Exp(0.3); Hit1_2: Disabled; Hit2_2: Disabled; Miss_2: Disabled; Read3: Exp(0.2); Hit1_3: Disabled; Hit2_3: Disabled; Miss_3: Disabled; Cache2_item1: Disabled; Cache2_item2: Disabled; Cache2_item3: Disabled
Features	drop rule {'Hit1_1': 'waitingQueue', 'Hit2_1': 'waitingQueue', 'Miss_1': 'waitingQueue', 'Hit1_2': 'waitingQueue', 'Hit2_2': 'waitingQueue', 'Miss_2': 'waitingQueue', 'Hit1_3': 'waitingQueue', 'Hit2_3': 'waitingQueue', 'Miss_3': 'waitingQueue', 'Cache2_item1': 'waitingQueue', 'Cache2_item2': 'waitingQueue', 'Cache2_item3': 'waitingQueue'} at Think; cache: LRU replacement, 3 items, capacity 1; cache: LRU replacement, 3 items, capacity 2; class switching on 3 links
Solvers	CTMC, SSA, LDES

Table 2.92: Features of `cachenet_tandem`.

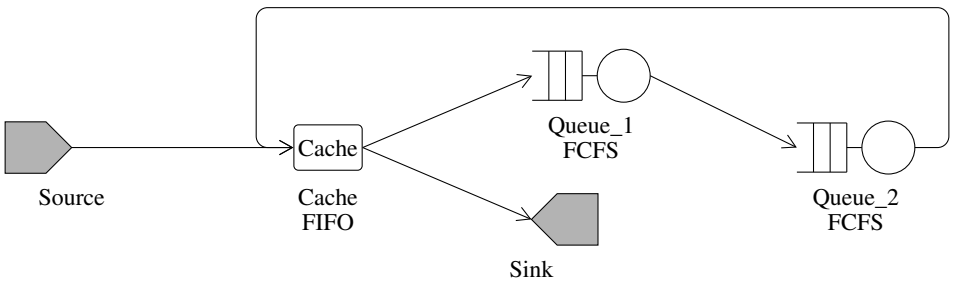


Figure 2.93: Topology of `retrieval_chain`: an open network over 2 queueing stations with a cache node.

Nodes	5: Source, Queue × 2, Cache, Sink
Classes	6: InitClass (open), HitClass (open), MissClass (open), InitClass_retrievalClass_1 (open), InitClass_retrievalClass_2 (open), InitClass_retrievalClass_3 (open)
Scheduling	Queue_1: FCFS; Queue_2: FCFS
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Disabled; InitClass_retrievalClass_2: Disabled; InitClass_retrievalClass_3: Disabled
Service	Queue_1 – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(2); InitClass_retrievalClass_2: Exp(2); InitClass_retrievalClass_3: Exp(2) Queue_2 – InitClass: Exp(3); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(3); InitClass_retrievalClass_2: Exp(3); InitClass_retrievalClass_3: Exp(3)
Features	cache: FIFO replacement, 3 items, capacity 1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue_1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue_2; class switching on 4 links
Solvers	SSA, LDES, MVA, NC

Table 2.93: Features of `retrieval_chain`.

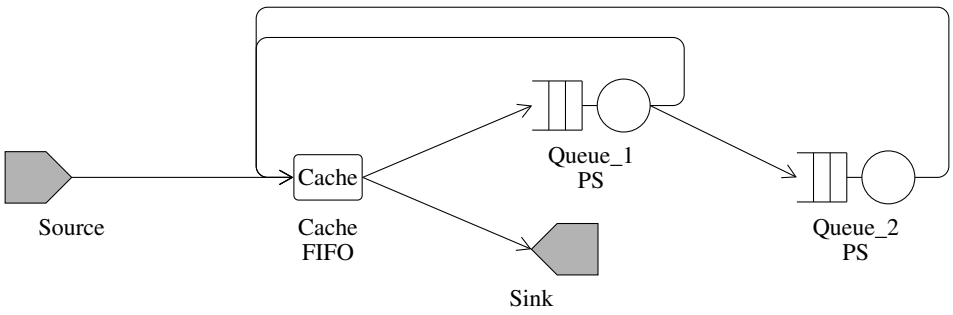


Figure 2.94: Topology of `retrieval_default`: an open network over 2 queueing stations with a cache node.

Nodes	5: Source, Queue × 2, Cache, Sink
Classes	6: InitClass (open), HitClass (open), MissClass (open), InitClass_retrievalClass_1 (open), InitClass_retrievalClass_2 (open), InitClass_retrievalClass_3 (open)
Scheduling	Queue_1: PS; Queue_2: PS
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Disabled; InitClass_retrievalClass_2: Disabled; InitClass_retrievalClass_3: Disabled
Service	Queue_1 – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(5); InitClass_retrievalClass_2: Exp(2); InitClass_retrievalClass_3: Exp(2) Queue_2 – InitClass: Exp(3); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(3); InitClass_retrievalClass_2: Exp(3); InitClass_retrievalClass_3: Exp(3)
Features	cache: FIFO replacement, 3 items, capacity 1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue_1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue_2; class switching on 4 links
Solvers	SSA, LDES, MVA, NC

Table 2.94: Features of `retrieval_default`.

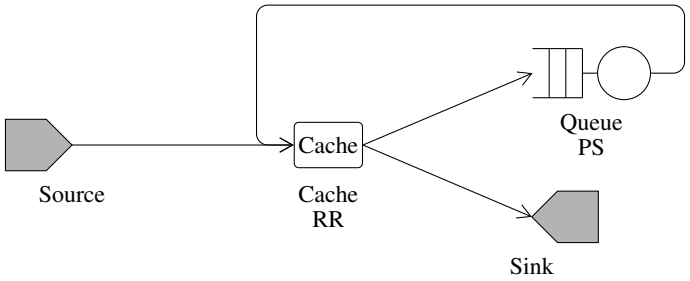


Figure 2.95: Topology of `retrieval_ps`: an open network over 1 queueing station with a cache node.

Nodes	4: Source, Queue, Cache, Sink
Classes	10: InitClass (open), HitClass (open), MissClass (open), InitClass_retrievalClass_1 (open), InitClass_retrievalClass_2 (open), InitClass_retrievalClass_3 (open), InitClass_retrievalClass_4 (open), InitClass_retrievalClass_5 (open), InitClass_retrievalClass_6 (open), InitClass_retrievalClass_7 (open)
Scheduling	Queue: PS
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Disabled; InitClass_retrievalClass_2: Disabled; InitClass_retrievalClass_3: Disabled; InitClass_retrievalClass_4: Disabled; InitClass_retrievalClass_5: Disabled; InitClass_retrievalClass_6: Disabled; InitClass_retrievalClass_7: Disabled
Service	Queue – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(1); InitClass_retrievalClass_2: Exp(1); InitClass_retrievalClass_3: Exp(1); InitClass_retrievalClass_4: Exp(1); InitClass_retrievalClass_5: Exp(1); InitClass_retrievalClass_6: Exp(1); InitClass_retrievalClass_7: Exp(1)
Features	cache: RR replacement, 7 items, capacity 6; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue; class switching on 3 links
Solvers	SSA, LDES, MVA, NC

Table 2.95: Features of `retrieval_ps`.

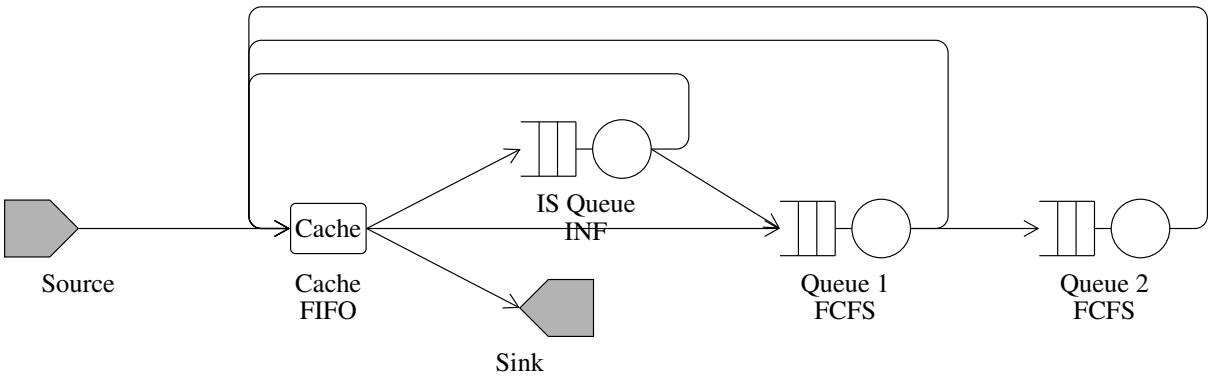


Figure 2.96: Topology of `retrieval_routing`: an open network over 3 queueing stations with a cache node.

Nodes	6: Source, Queue × 3, Cache, Sink
Classes	6: InitClass (open), HitClass (open), MissClass (open), InitClass_retrievalClass_1 (open), InitClass_retrievalClass_2 (open), InitClass_retrievalClass_3 (open)
Scheduling	IS Queue: INF; Queue 1: FCFS; Queue 2: FCFS
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Disabled; InitClass_retrievalClass_2: Disabled; InitClass_retrievalClass_3: Disabled
Service	IS Queue – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(2); InitClass_retrievalClass_2: Exp(2); InitClass_retrievalClass_3: Exp(2) Queue 1 – InitClass: Exp(3); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(3); InitClass_retrievalClass_2: Exp(3); InitClass_retrievalClass_3: Exp(3) Queue 2 – InitClass: Exp(3); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(3); InitClass_retrievalClass_2: Exp(3); InitClass_retrievalClass_3: Exp(3)
Features	cache: FIFO replacement, 3 items, capacity 2; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at IS Queue; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue 1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue 2; class switching on 5 links
Solvers	SSA, LDES, MVA, NC

Table 2.96: Features of `retrieval_routing`.

2.8.17 `retrieval_simple`

A retrieval system: a cache in front of an infinite-server backing store, hits served at once and misses fetched.

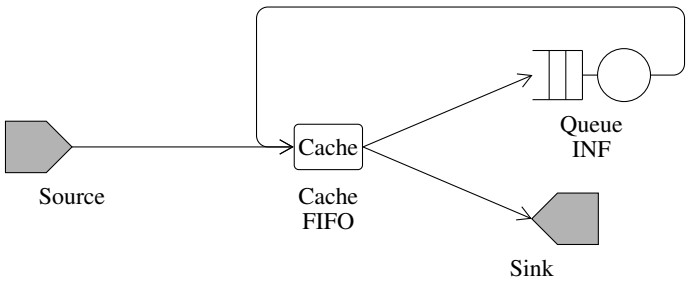


Figure 2.97: Topology of `retrieval_simple`: an open network over 1 queueing station with a cache node.

2.9 Cluster models

A cluster is a dispatcher in front of a pool of servers. It is not a new node type: the shorthand builds a Router and a set of stations, and the interest is in the dispatching policy and in how the response time scales with the pool size.

2.9.1 `cl_basic`

A dispatcher in front of a pool of identical servers, built with the cluster shorthand.

2.9.2 `cl_closed`

The closed version, with a fixed population of requests.

Nodes	4: Source, Queue, Cache, Sink
Classes	6: InitClass (open), HitClass (open), MissClass (open), InitClass_retrievalClass_1 (open), InitClass_retrievalClass_2 (open), InitClass_retrievalClass_3 (open)
Scheduling	Queue: INF
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Disabled; InitClass_retrievalClass_2: Disabled; InitClass_retrievalClass_3: Disabled
Service	Queue – InitClass: Exp(2); HitClass: Disabled; MissClass: Disabled; InitClass_retrievalClass_1: Exp(2); InitClass_retrievalClass_2: Exp(2); InitClass_retrievalClass_3: Exp(2)
Features	cache: FIFO replacement, 3 items, capacity 1; drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Queue; class switching on 3 links
Solvers	SSA, LDES, MVA, NC

Table 2.97: Features of `retrieval_simple`.

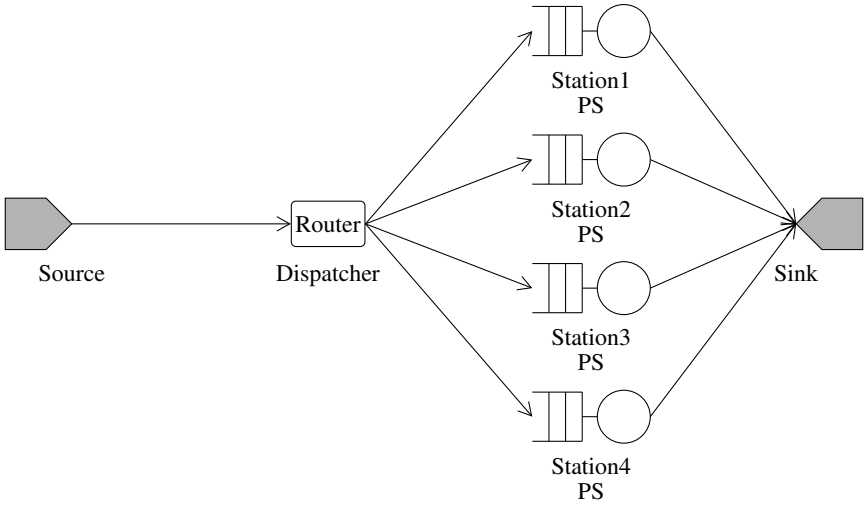


Figure 2.98: Topology of `cl_basic`: an open network over 4 queueing stations with a router.

Nodes	7: Source, Queue × 4, Router, Sink
Classes	1: Class1 (open)
Scheduling	Station1: PS; Station2: PS; Station3: PS; Station4: PS
Arrivals	Source – Class1: Exp(0.4)
Service	Station1 – Class1: Exp(1) Station2 – Class1: Exp(1) Station3 – Class1: Exp(1) Station4 – Class1: Exp(1)
Features	class switching on 7 links
Solvers	MVA

Table 2.98: Features of `cl_basic`.

Nodes	5: Queue × 3, Delay, Router
Classes	1: Class1 (closed, $N = 8$)
Scheduling	Think: INF; Station1: PS; Station2: PS; Station3: PS
Service	Think – Class1: Exp(1) Station1 – Class1: Exp(1) Station2 – Class1: Exp(1) Station3 – Class1: Exp(1)
Features	class switching on 5 links
Solvers	MVA

Table 2.99: Features of `cl_closed`.

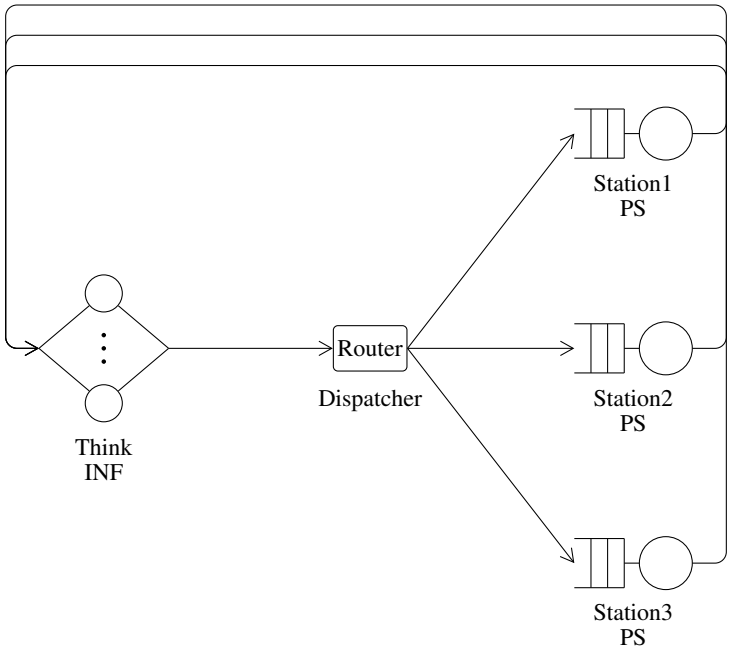


Figure 2.99: Topology of `cl_closed`: a closed network over 4 queueing stations with a router.

2.9.3 `cl_compare`

RAND against round-robin dispatching on one cluster.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Solvers	SSA

Table 2.100: Features of `cl_compare`.

2.9.4 `cl_multiclass`

Several request classes through one cluster.

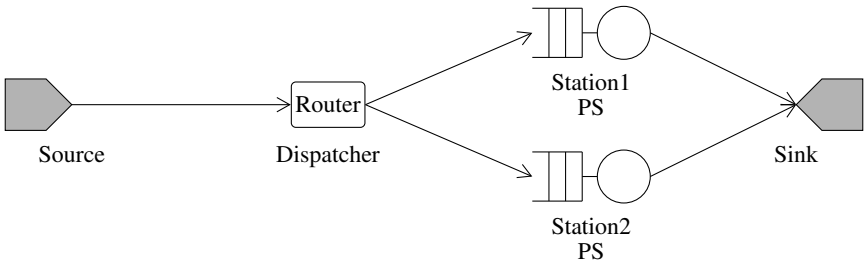


Figure 2.100: Topology of `cl_multiclass`: an open network over 2 queueing stations with a router.

2.9.5 `cl_scheduling`

The same cluster under FCFS and under PS.

Nodes	5: Source, Queue × 2, Router, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Station1: PS; Station2: PS
Arrivals	Source – Class1: Exp(0.3); Class2: Exp(0.2)
Service	Station1 – Class1: Exp(1); Class2: Exp(2) Station2 – Class1: Exp(1); Class2: Exp(2)
Features	class switching on 5 links
Solvers	MVA

Table 2.101: Features of `cl_multiclass`.

No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Solvers	SSA

Table 2.102: Features of `cl_scheduling`.

2.9.6 `cl_stations`

Clusters whose members are declared station by station.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Solvers	MVA

Table 2.103: Features of `cl_stations`.

2.9.7 `cl_sweep`

A sweep over the cluster size, and the response time curve it traces.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.9.8 `dispatch_closed`

The closed version.

2.9.9 `dispatch_mixed`

The mixed version, open and closed classes through the same dispatcher.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.9.10 `dispatch_open`

The dispatcher idiom written out in full, open workload.

Solvers	MVA

Table 2.104: Features of `cl_sweep`.

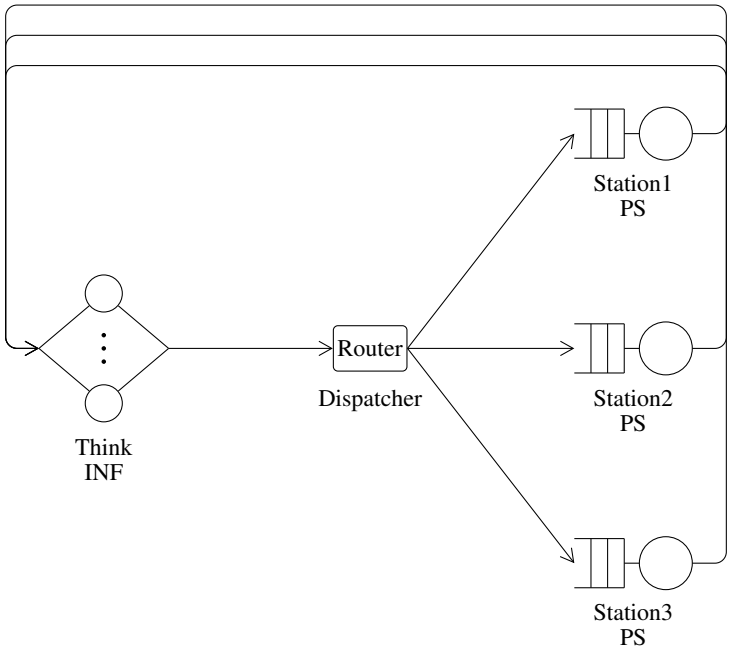


Figure 2.101: Topology of `dispatch_closed`: a closed network over 4 queueing stations with a router.

Nodes	5: Queue × 3, Delay, Router
Classes	1: Class1 (closed, $N = 10$)
Scheduling	Think: INF; Station1: PS; Station2: PS; Station3: PS
Service	Think – Class1: Exp(1) Station1 – Class1: Exp(1) Station2 – Class1: Exp(1) Station3 – Class1: Exp(1)
Features	class switching on 5 links
Solvers	MVA, SSA

Table 2.105: Features of `dispatch_closed`.

Solvers	MVA, LDES

Table 2.106: Features of `dispatch_mixed`.

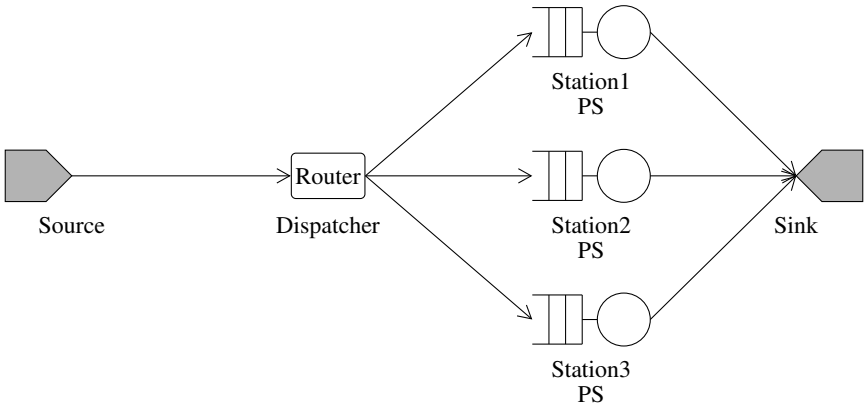


Figure 2.102: Topology of `dispatch_open`: an open network over 3 queueing stations with a router.

Nodes	6: Source, Queue × 3, Router, Sink
Classes	1: Class1 (open)
Scheduling	Station1: PS; Station2: PS; Station3: PS
Arrivals	Source – Class1: Exp(0.4)
Service	Station1 – Class1: Exp(1) Station2 – Class1: Exp(1) Station3 – Class1: Exp(1)
Features	class switching on 6 links
Solvers	MVA, SSA

Table 2.107: Features of `dispatch_open`.

2.10 Stochastic Petri nets

A stochastic Petri net is built from `Place` and `Transition` nodes. A transition fires when its input places hold enough tokens, consuming and producing tokens according to the arc weights; timed transitions fire after a random delay, immediate transitions fire at once, and an inhibitor arc disables a transition while its place is marked. A transition may have several firing modes, each with its own rate and arc weights.

Petri nets and queueing models are two notations for the same processes, and two of the scripts make that explicit by building the same system both ways and comparing the generators. Because the firing time need not be exponential, the family also carries the non-Markovian cases.

The figures of this section are drawn in the notation of the net rather than in that of a queueing network. A place is a circle, and the tokens or the number inside it are the marking the model starts from; a transition is a bar, hollow when it fires after a random delay and solid when it fires at once; an arc is plain when it moves one token and labelled with its multiplicity otherwise, and an arc ending in a small circle is an inhibitor, which holds its transition disabled for as long as its place is marked. A transition with more than one firing mode is named with the number of modes, since no single distribution describes it. A circle bisected by a bar is a queueing place, whose tokens are served by a scheduling station embedded in the place.

2.10.1 `spn_basic_closed`

The closed counterpart, a token circulating between two places.

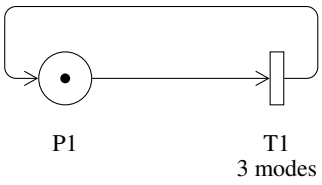


Figure 2.103: Topology of `spn_basic_closed`: a stochastic Petri net over 1 place and 1 transition; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; a transition offering several firing modes is named with their number in place of a distribution.

Nodes	2: Place, Transition
Classes	1: Class1 (closed, $N = 1$)
Features	1 transition with 3 firing modes each
Solvers	JMT

Table 2.108: Features of `spn_basic_closed`.

2.10.2 spn_basic_open

One place and one timed transition fed by an open class, the Petri net form of an M/M/1 queue.

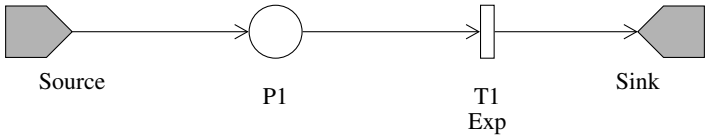


Figure 2.104: Topology of spn_basic_open: a stochastic Petri net over 1 place and 1 transition; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the net is open, fed by a Source and drained into a Sink.

Nodes	4: Source, Place, Transition, Sink
Classes	1: Class1 (open)
Arrivals	Source – Class1: Exp(1)
Features	1 transition with 1 firing mode each
Solvers	JMT

Table 2.109: Features of spn_basic_open.

2.10.3 spn_closed_fourplaces

Four places with Erlang, Coxian and hyperexponential firing times.

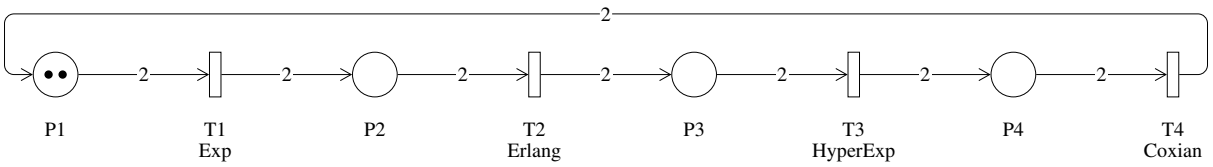


Figure 2.105: Topology of spn_closed_fourplaces: a stochastic Petri net over 4 places and 4 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; an arc that moves more than one token carries its multiplicity.

2.10.4 spn_closed_twoplaces

A closed net over two places with Erlang firing times.

2.10.5 spn_colored_gspn

A colored generalized stochastic Petri net: one firing mode per token color, an immediate transition whose inhibitor arcs make the server exclusive, and firing weights that arbitrate the two colors competing for it.

2.10.6 spn_fluid_dae

Fluid DAE analysis of stochastic Petri nets, including invariant constraints, immediate-transition flows, bounded markings and marking covariances.

Nodes	8: Place × 4, Transition × 4
Classes	1: Class1 (closed, $N = 2$)
Features	4 transitions with 1 firing mode each
Solvers	JMT

Table 2.110: Features of `spn_closed_fourplaces`.

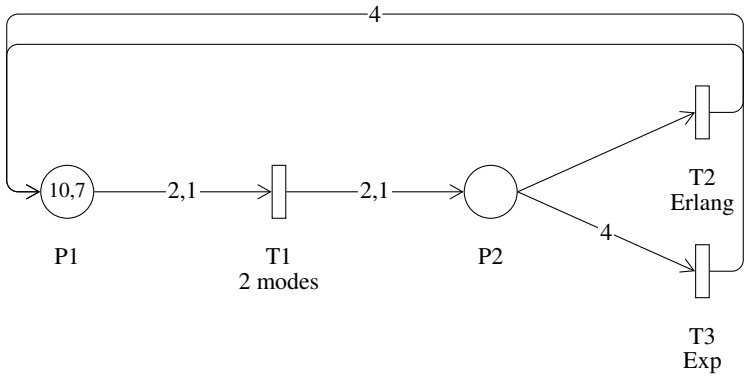


Figure 2.106: Topology of `spn_closed_twoplaces`: a stochastic Petri net over 2 places and 3 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; an arc that moves more than one token carries its multiplicity; a transition offering several firing modes is named with their number in place of a distribution.

Nodes	5: Place × 2, Transition × 3
Classes	2: Class1 (closed, $N = 10$), Class2 (closed, $N = 7$)
Features	3 transitions, 1 to 2 firing modes; class switching on 2 links
Solvers	JMT

Table 2.111: Features of `spn_closed_twoplaces`.

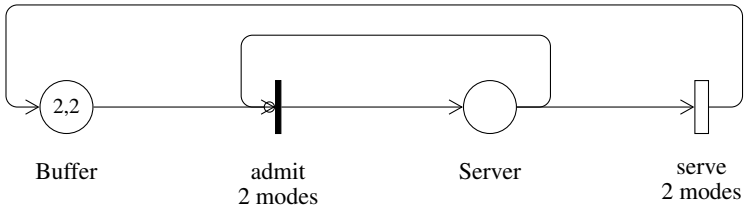


Figure 2.107: Topology of `spn_colored_gspn`: a stochastic Petri net over 2 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the arc ending in a circle is an inhibitor and holds admit disabled while Server is marked; a transition offering several firing modes is named with their number in place of a distribution.

Nodes	4: Place × 2, Transition × 2
Classes	2: Gold (closed, $N = 2$), Silver (closed, $N = 2$)
Features	2 transitions with 2 firing modes each
Solvers	CTMC, SSA

Table 2.112: Features of `spn_colored_gspn`.

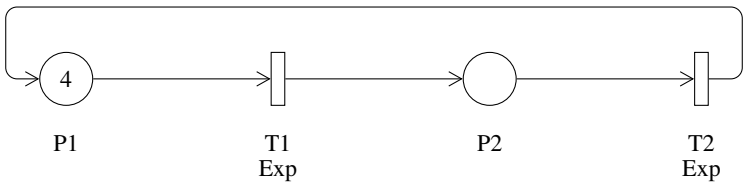


Figure 2.108: Topology of `spn_fluid_dae`: a stochastic Petri net over 2 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once.

Nodes	4: Place × 2, Transition × 2
Classes	1: Class1 (closed, $N = 4$)
Features	2 transitions with 1 firing mode each
Solvers	FLD, CTMC

Table 2.113: Features of `spn_fluid_dae`.

2.10.7 `spn_fluid_onoff`

Fluid stochastic Petri net: an on-off source feeding a fluid buffer, with flows, a level guard and an impulse, solved by the fluid DAE and by piecewise-deterministic simulation against the exact level.

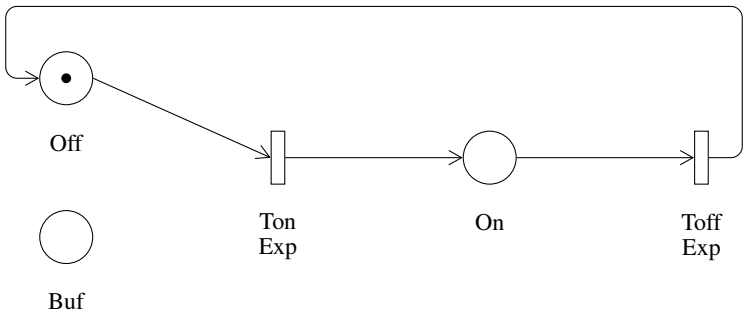


Figure 2.109: Topology of `spn_fluid_onoff`: a stochastic Petri net over 3 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once.

2.10.8 `spn_fourmodes`

Four firing modes, and the mode selection probabilities.

2.10.9 `spn_inhibiting`

An inhibitor arc, which disables a transition while its place is marked.

2.10.10 `spn_lpbounds`

Linear-programming bounds on a Petri net with SolverBA: the stationary chain is relaxed to a moment polytope and every reported measure minimised and maximised over it, reproducing Table 2 of Liu (1998) on a blocking production line and then bracketing the exact marking of an inhibitor net.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Nodes	5: Place × 3, Transition × 2
Classes	1: Source (closed, $N = 1$)
Features	2 transitions with 1 firing mode each; class switching on 1 link
Solvers	FLD, SSA

Table 2.114: Features of `spn_fluid_onoff`.

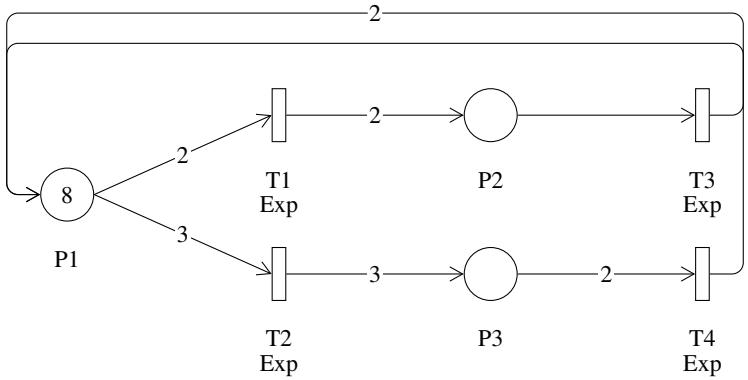


Figure 2.110: Topology of `spn_fourmodes`: a stochastic Petri net over 3 places and 4 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; an arc that moves more than one token carries its multiplicity.

Nodes	7: Place × 3, Transition × 4
Classes	1: Class1 (closed, $N = 8$)
Features	4 transitions with 1 firing mode each
Solvers	JMT

Table 2.115: Features of `spn_fourmodes`.

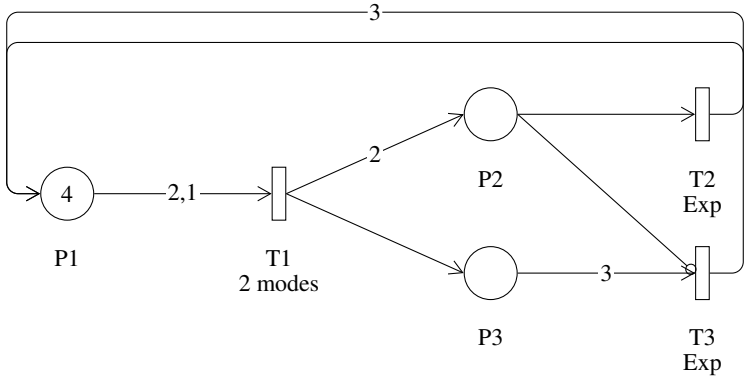


Figure 2.111: Topology of `spn_inhibiting`: a stochastic Petri net over 3 places and 3 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; an arc that moves more than one token carries its multiplicity; the arc ending in a circle is an inhibitor and holds T3 disabled while P2 is marked; a transition offering several firing modes is named with their number in place of a distribution.

Nodes	6: Place × 3, Transition × 3
Classes	1: Class1 (closed, $N = 4$)
Features	3 transitions, 1 to 2 firing modes
Solvers	JMT

Table 2.116: Features of `spn_inhibiting`.

Solvers	CTMC

Table 2.117: Features of `spn_lpbounds`.

2.10.11 `spn_open_sevenplaces`

A seven-place open net, large enough that the state space matters.

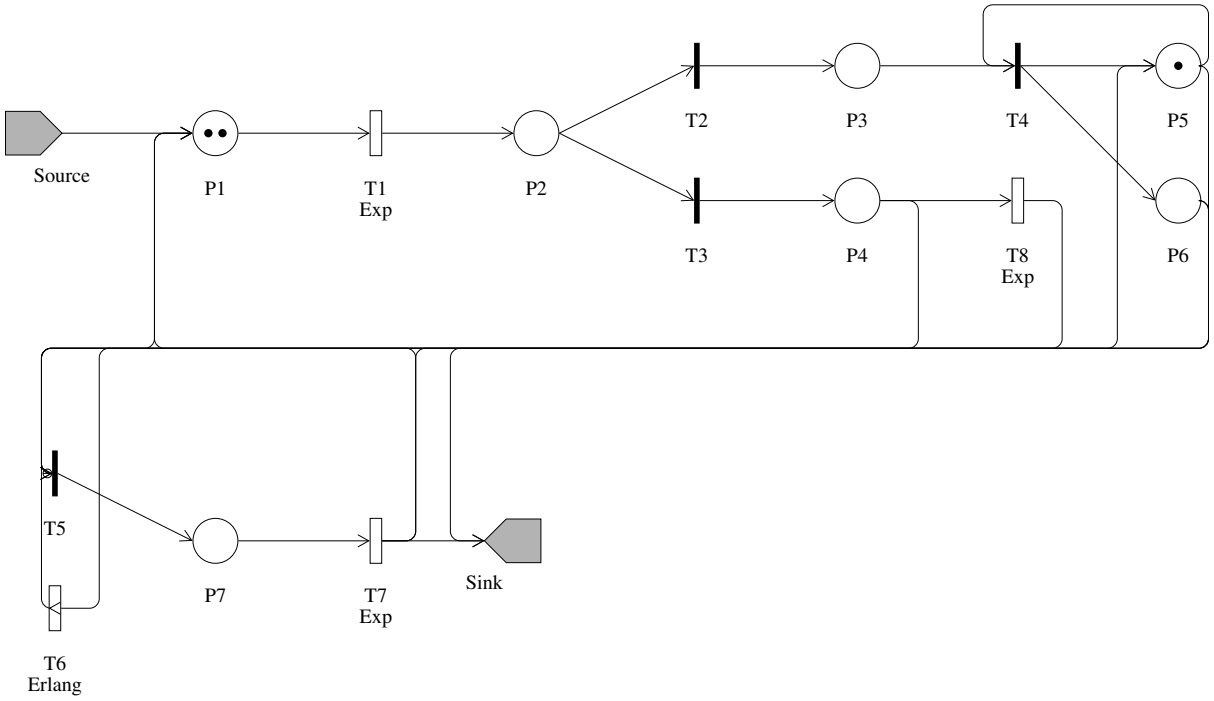


Figure 2.112: Topology of `spn_open_sevenplaces`: a stochastic Petri net over 7 places and 8 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the arc ending in a circle is an inhibitor and holds T5 disabled while P6 is marked; the net is open, fed by a Source and drained into a Sink.

Nodes	17: Source, Place × 7, Transition × 8, Sink
Classes	1: Class1 (open)
Arrivals	Source – Class1: Exp(1)
Features	8 transitions with 1 firing mode each
Solvers	JMT

Table 2.118: Features of `spn_open_sevenplaces`.

2.10.12 `spn_pareto_service`

A Pareto firing time, a heavy-tailed non-Markovian net.

2.10.13 `spn_productform_nc`

Product-form analysis of cyclic and fork–join stochastic Petri nets with SolverNC, compared against an explicit CTMC and accompanied by the structural certificate and marking invariants.

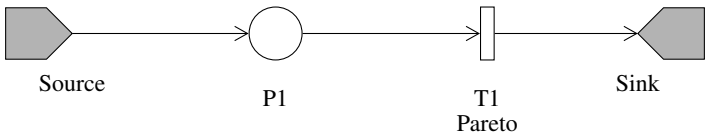


Figure 2.113: Topology of `spn_pareto_service`: a stochastic Petri net over 1 place and 1 transition; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the net is open, fed by a Source and drained into a Sink.

Nodes	4: Source, Place, Transition, Sink
Classes	1: Class1 (open)
Arrivals	Source – Class1: Exp(0.5)
Features	1 transition with 1 firing mode each
Solvers	JMT

Table 2.119: Features of `spn_pareto_service`.

No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Solvers	NC, CTMC

Table 2.120: Features of `spn_productform_nc`.

2.10.14 `spn_queueing_place`

A queueing station and a Petri net in one model, the two formalisms side by side.

2.10.15 `spn_twomodes`

A transition with two firing modes.

2.10.16 `test_spn_nrm_open`

The open net solved by the next-reaction method, against the exact generator.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11 Workflow models

The workflow models express the standard control-flow patterns – sequence, probabilistic branch, parallel split and join, loop – as activity graphs, and solve them for the completion time of the whole workflow rather than for station metrics.

2.11.1 `wf_aph`

Acyclic phase-type activity durations.

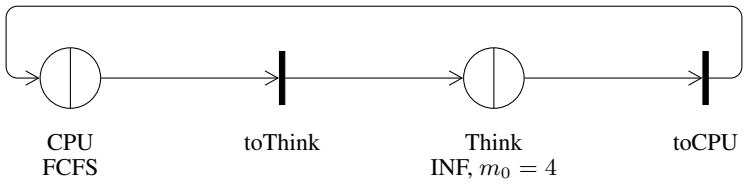


Figure 2.114: Topology of `spn_queueing_place`: a stochastic Petri net over 2 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the 2 bisected circles are queueing places, whose tokens are served by an embedded station.

Nodes	4: Place × 2, Transition × 2
Classes	1: Jobs (closed, $N = 4$)
Service	CPU – Jobs: Exp(1.5) Think – Jobs: Exp(0.5)
Features	2 transitions with 1 firing mode each
Solvers	LDES, MVA

Table 2.121: Features of `spn_queueing_place`.

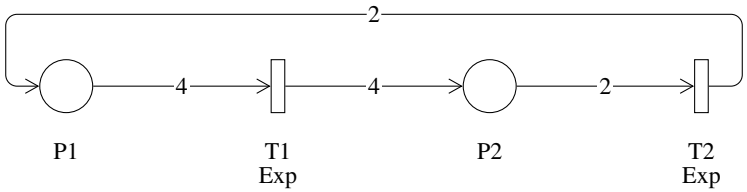


Figure 2.115: Topology of `spn_twomodes`: a stochastic Petri net over 2 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; an arc that moves more than one token carries its multiplicity.

Nodes	4: Place × 2, Transition × 2
Classes	1: Class1 (closed, $N = 10$)
Features	2 transitions with 1 firing mode each
Solvers	JMT

Table 2.122: Features of `spn_twomodes`.

Solvers	JMT

Table 2.123: Features of `test_spn_nrm_open`.

No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.2 `wf_branch`

A probabilistic branch and its merge.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.3 `wf_complex`

Nested branches, forks and loops in one workflow.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.4 `wf_erlang`

Erlang activity durations.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.5 `wf_loop`

A loop with a repetition count.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.6 `wf_parallel`

An AND-fork, parallel activities and a join.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.11.7 `wf_serial`

A sequence of activities.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

2.12 Workflow nets

The workflow nets describe a business process as a Petri net with one source place and one sink place. Each example decides soundness, computes the exact cycle-time law of one case, or compiles the net to a queueing or Petri network for a stream of cases, and checks the result against a closed form.

2.12.1 wfnet_and_xor

AND and XOR blocks with choice weights, compiled to a fork-join network and read back as case metrics.

Solvers	MVA

Table 2.124: Features of `wfnet_and_xor`.

2.12.2 wfnet_closed

A closed population of cases: k-soundness and the closed analysis of a saturated server.

Solvers	MVA

Table 2.125: Features of `wfnet_closed`.

2.12.3 wfnet_nonsp_sound

A sound workflow net that is not block-structured: exact analysis and the SPN target work, toWorkflow refuses.

2.12.4 wfnet_pnml

Round trip of a workflow net through PNML.

2.12.5 wfnet_resource

Tasks sharing a resource under a Poisson stream of cases, on both network targets.

Solvers	MVA

Table 2.126: Features of `wfnet_resource`.

2.12.6 wfnet_serial

A serial workflow net: soundness check, exact cycle-time law, conversion to a Workflow.

2.12.7 wfnet_unsound

Soundness reports with witnesses: improper completion, deadlock with a dead transition, unboundedness.

Chapter 3

Advanced models

The advanced examples cover the features that leave the product-form setting: load dependence, blocking, state-dependent routing, rewards, random environments and the disciplines that no closed form covers. They live under no R twin; load the document and solve it.

Each of these features narrows the set of solvers that can be used, and the scripts say which. A model that a solver cannot represent is refused rather than approximated silently, so the feature set of a solver is discoverable from the model itself.

3.1 Load-dependent stations

A load-dependent station has a service rate that is a function of the number of jobs it holds, declared through `setLoadDependence` as a rate vector $\mu(n)$. A multiserver station is the special case $\mu(n) = \min(n, c)\mu$, and writing it out as a load-dependent station is how the scripts check the two against each other. Class dependence, declared through `setClassDependence`, lets the rate depend on the class mix rather than on the total only.

The same mechanism supports the flow-equivalent server construction: solve a subnetwork in isolation for every population, tabulate its throughput, and install that table as the rate vector of a single station that replaces it.

3.1.1 `fes_aggregation`

Aggregating a subnetwork into one station and checking the aggregate against the full model.

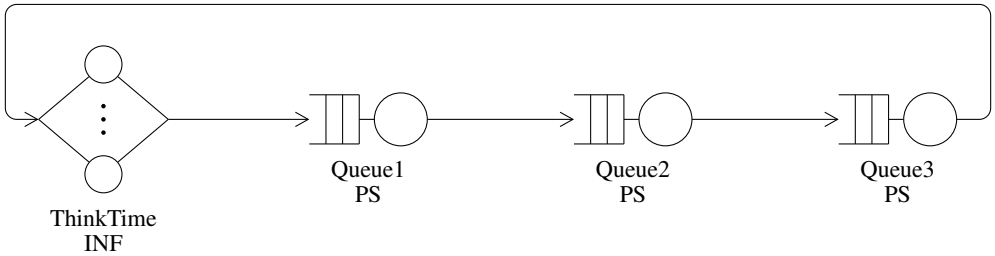


Figure 3.1: Topology of `fes_aggregation`: a closed network over 4 queueing stations.

3.1.2 `fes_single_class`

The flow-equivalent construction step by step: solve the subnetwork, tabulate its throughput, install it as a load-dependent station.

Nodes	4: Queue × 3, Delay
Classes	2: Class1 (closed, $N = 3$), Class2 (closed, $N = 2$)
Scheduling	ThinkTime: INF; Queue1: PS; Queue2: PS; Queue3: PS
Service	ThinkTime – Class1: Exp(0.2); Class2: Exp(0.25) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.5) Queue2 – Class1: Exp(1); Class2: Exp(0.8333) Queue3 – Class1: Exp(1.25); Class2: Exp(1)
Solvers	MVA

Table 3.1: Features of `fes_aggregation`.

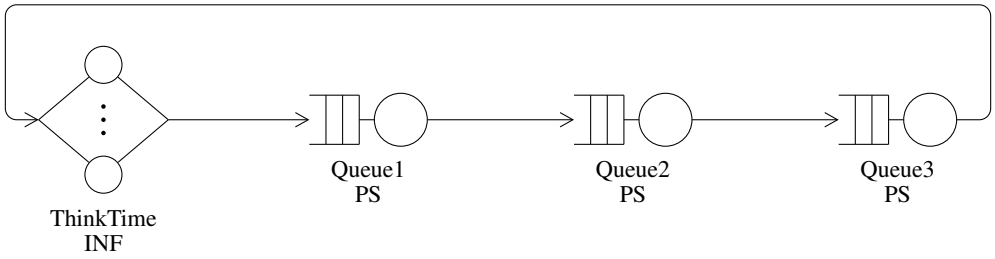


Figure 3.2: Topology of `fes_single_class`: a closed network over 4 queueing stations.

3.1.3 `ld_class_dependence`

Service rates that depend on the class mix at the station, not only on the total population.

3.1.4 `ld_fes_multiclass`

The multiclass flow-equivalent server.

3.1.5 `ld_fes_singleclass`

A flow-equivalent server replacing a subnetwork, single class.

3.1.6 `ld_global_dependence`

A globally state-dependent (Whittle) model: `setGlobalDependence` scales the service rate by a function of the FULL station-class population matrix rather than of the population local to one station. The scaling used here satisfies the Whittle balance property, so the chain is reversible, keeps a product form and is insensitive to the service distribution beyond its mean.

3.1.7 `ld_joint_dependence`

Joint dependence on the populations of several classes.

Nodes	4: Queue × 3, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	ThinkTime: INF; Queue1: PS; Queue2: PS; Queue3: PS
Service	ThinkTime – Class1: Exp(0.2) Queue1 – Class1: Exp(0.6667) Queue2 – Class1: Exp(1) Queue3 – Class1: Exp(1.25)
Solvers	MVA

Table 3.2: Features of `fes_single_class`.

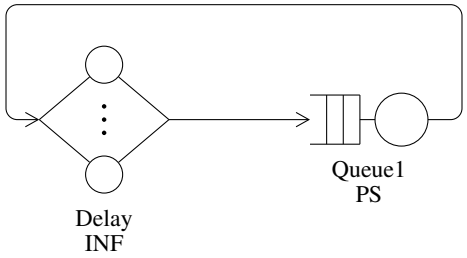


Figure 3.3: Topology of `ld_class_dependence`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 16$), Class2 (closed, $N = 8$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(0.5) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.4)
Solvers	CTMC, MVA

Table 3.3: Features of `ld_class_dependence`.

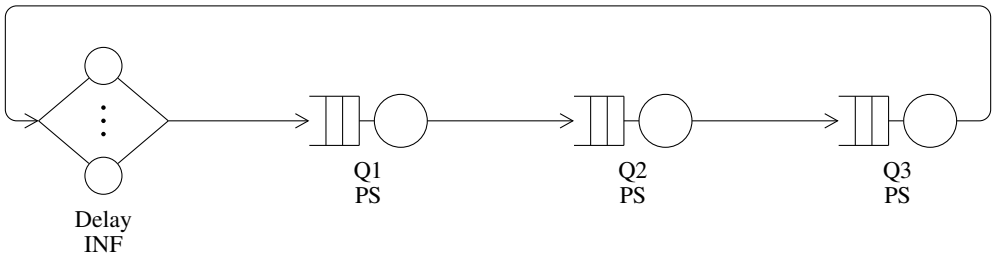


Figure 3.4: Topology of `ld_fes_multiclass`: a closed network over 4 queueing stations.

Nodes	4: Queue $\times$ 3, Delay
Classes	2: Class1 (closed, $N = 3$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Q1: PS; Q2: PS; Q3: PS
Service	Delay – Class1: Exp(1); Class2: Exp(0.6667) Q1 – Class1: Exp(2); Class2: Exp(1.25) Q2 – Class1: Exp(3.333); Class2: Exp(1.667) Q3 – Class1: Exp(2.5); Class2: Exp(1.429)
Solvers	MVA, NC

Table 3.4: Features of `ld_fes_multiclass`.

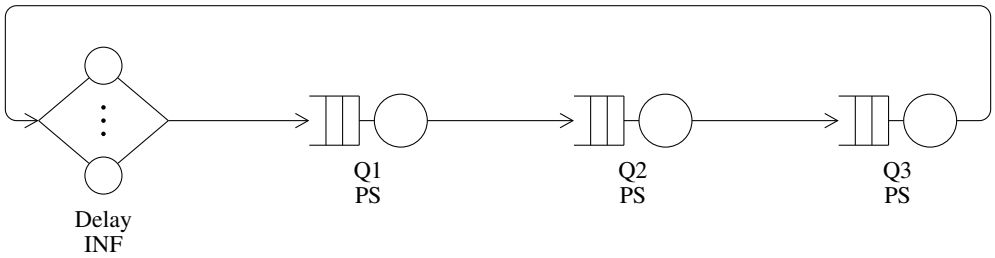


Figure 3.5: Topology of `ld_fes_singleclass`: a closed network over 4 queueing stations.

Nodes	4: Queue $\times$ 3, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Q1: PS; Q2: PS; Q3: PS
Service	Delay – Class1: Exp(0.2) Q1 – Class1: Exp(0.6667) Q2 – Class1: Exp(1) Q3 – Class1: Exp(1.25)
Solvers	MVA, NC

Table 3.5: Features of `ld_fes_singleclass`.

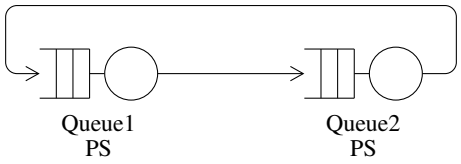


Figure 3.6: Topology of `ld_global_dependence`: a closed network over 2 queueing stations.

Nodes	2: Queue $\times$ 2
Classes	1: Class1 (closed, $N = 3$)
Scheduling	Queue1: PS; Queue2: PS
Service	Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(2)
Solvers	CTMC, SSA

Table 3.6: Features of `ld_global_dependence`.

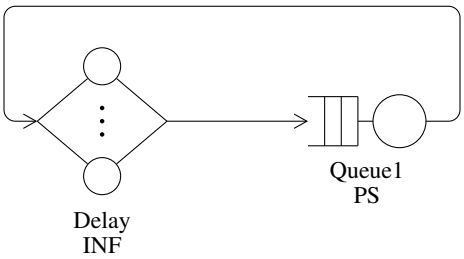


Figure 3.7: Topology of `ld_joint_dependence`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 16$), Class2 (closed, $N = 8$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(0.5) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.4)
Solvers	CTMC, MVA

Table 3.7: Features of `ld_joint_dependence`.

3.1.8 ld_multiserver_fcfs

The FCFS counterpart, with class-dependent scaling on top.

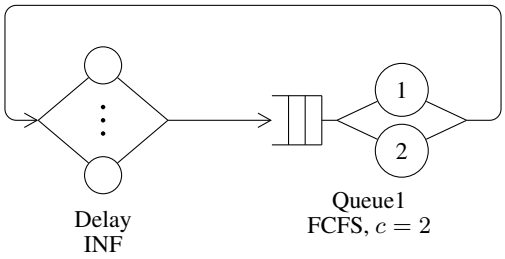


Figure 3.8: Topology of `ld_multiserver_fcfs`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 16$)
Scheduling	Delay: INF; Queue1: FCFS ($c = 2$)
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(0.6667)
Solvers	NC, CTMC, MVA, JMT

Table 3.8: Features of `ld_multiserver_fcfs`.

3.1.9 ld_multiserver_ps

A multiserver PS station written as a load-dependent station, and the rate vector that reproduces it.

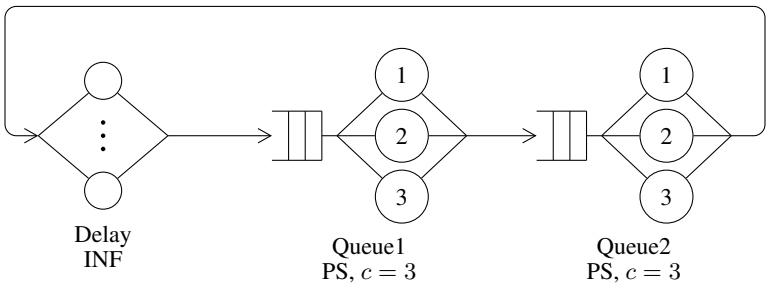


Figure 3.9: Topology of `ld_multiserver_ps`: a closed network over 3 queueing stations.

3.1.10 ld_multiserver_ps_twoclasses

The two-class version.

3.1.11 ld_whittle_bandwidth

An open bandwidth-sharing network in which one route holds several links at once, which no per-station rate scaling can express. Capacity is shared by balanced fairness, whose rate function satisfies the Whittle property by construction, so the stationary law is again of product form and insensitive.

3.2 Finite capacity regions

A finite capacity region caps the number of jobs inside a set of stations. A job that would breach the cap is either dropped or made to wait outside the region, per class, and per-class caps may be declared

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 4$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS ($c = 3$); Queue2: PS ($c = 3$)
Service	Delay – Class1: Exp(1); Class2: Exp(0.5) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.4) Queue2 – Class1: Exp(0.2857); Class2: Exp(0.2222)
Solvers	MVA, CTMC, NC, JMT

Table 3.9: Features of `ld_multiserver_ps`.

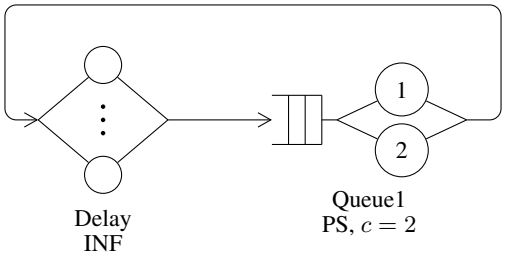


Figure 3.10: Topology of `ld_multiserver_ps_twoclasses`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 4$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(0.5) Queue1 – Class1: Exp(0.6667); Class2: Exp(0.4)
Solvers	MVA, CTMC, NC, JMT

Table 3.10: Features of `ld_multiserver_ps_twoclasses`.

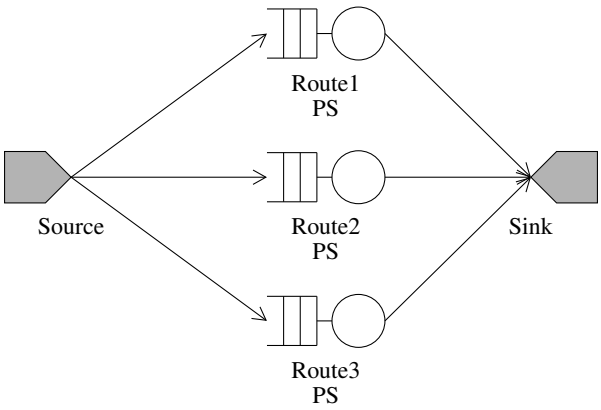


Figure 3.11: Topology of `ld_whittle_bandwidth`: an open network over 3 queueing stations.

Nodes	5: Source, Queue × 3, Sink
Classes	3: Route1Flows (open), Route2Flows (open), Route3Flows (open)
Scheduling	Route1: PS; Route2: PS; Route3: PS
Arrivals	Source – Route1Flows: Exp(0.2); Route2Flows: Exp(0.3); Route3Flows: Exp(0.3)
Service	Route1 – Route1Flows: Exp(1); Route2Flows: Disabled; Route3Flows: Disabled Route2 – Route1Flows: Disabled; Route2Flows: Exp(1); Route3Flows: Disabled Route3 – Route1Flows: Disabled; Route2Flows: Disabled; Route3Flows: Exp(1)
Solvers	CTMC

Table 3.11: Features of `ld_whittle_bandwidth`.

alongside the global one. The region is a property of the model, not of a station, which is what lets it span several stations.

The flagship comparison is `fcr_mm1kdrop`: a region of capacity K around a single queue, with dropping, is an M/M/1/K queue, and the script builds both models and solves them side by side.

3.2.1 `fcr_constraints`

Per-class capacities inside one region, and the interaction with the global limit.

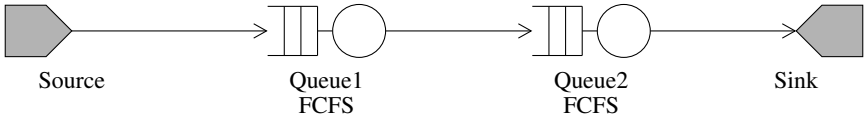


Figure 3.12: Topology of `fcr_constraints`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	2: HighPriority (open), LowPriority (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – HighPriority: Exp(0.3); LowPriority: Exp(0.5)
Service	Queue1 – HighPriority: Exp(1); LowPriority: Exp(0.8) Queue2 – HighPriority: Exp(1.2); LowPriority: Exp(1)
Solvers	JMT

Table 3.12: Features of `fcr_constraints`.

3.2.2 `fcr_lincon`

A linear constraint over the class populations, the general form of a region limit.

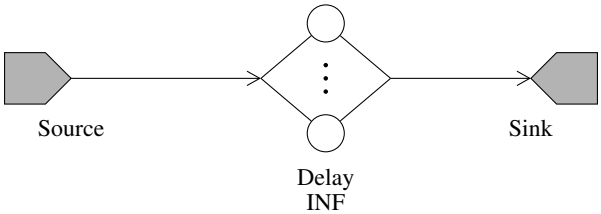


Figure 3.13: Topology of `fcr_lincon`: an open network over 1 queueing station.

3.2.3 `fcr_lossn`

A loss network: every request that cannot be admitted is lost, and the blocking probabilities are the quantity of interest.

3.2.4 `fcr_mm1kdrop`

A region of capacity K around one queue with dropping, which reproduces an M/M/1/K queue.

3.2.5 `fcr_mm1waitq`

The same region with waiting instead of dropping, so blocked jobs queue outside the region.

Nodes	3: Source, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF
Arrivals	Source – Class1: Exp(0.3); Class2: Exp(0.2)
Service	Delay – Class1: Exp(1); Class2: Exp(0.8)
Solvers	LDES

Table 3.13: Features of `fcr_lincon`.

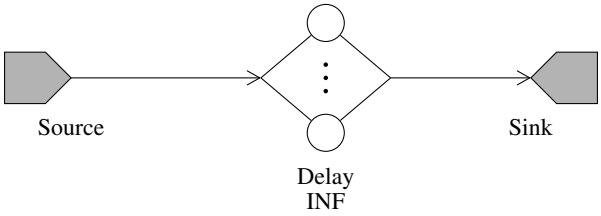


Figure 3.14: Topology of `fcr_lossn`: an open network over 1 queueing station.

Nodes	3: Source, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF
Arrivals	Source – Class1: Exp(0.3); Class2: Exp(0.2)
Service	Delay – Class1: Exp(1); Class2: Exp(0.8)
Solvers	JMT

Table 3.14: Features of `fcr_lossn`.

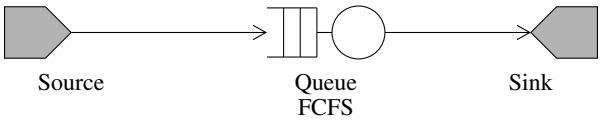


Figure 3.15: Topology of `fcr_mm1kdrop`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(0.8)
Service	Queue – Class1: Exp(1)
Solvers	JMT

Table 3.15: Features of `fcr_mm1kdrop`.

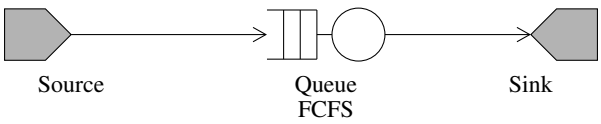


Figure 3.16: Topology of `fcr_mm1waitq`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(0.5)
Service	Queue – Class1: Exp(1)
Solvers	JMT

Table 3.16: Features of `fcr_mm1waitq`.

3.2.6 `fcr_oqndrop`

A region spanning several stations of an open network, with dropping.

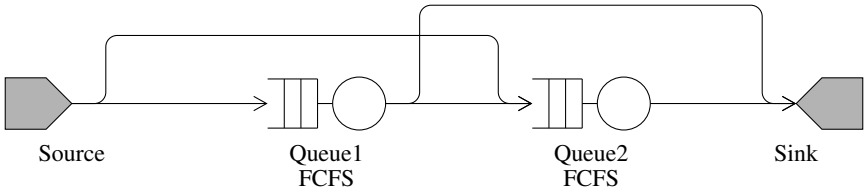


Figure 3.17: Topology of `fcr_oqndrop`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(0.4); Class2: Exp(0.3)
Service	Queue1 – Class1: Exp(1); Class2: Exp(0.9) Queue2 – Class1: Exp(1.1); Class2: Exp(1)
Solvers	JMT

Table 3.17: Features of `fcr_oqndrop`.

3.2.7 `fcr_oqnwaitq`

The same region with waiting.

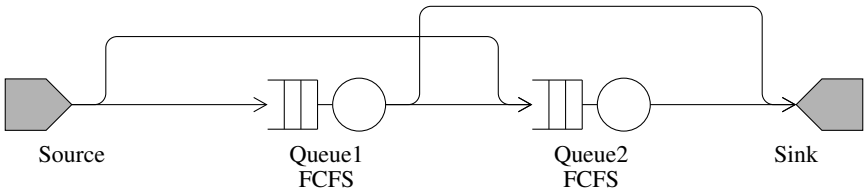


Figure 3.18: Topology of `fcr_oqnwaitq`: an open network over 2 queueing stations.

3.3 State probabilities

Beyond the averages, the solvers that build a state space can return probabilities. `getProb` gives the probability of a detailed station state, `getProbAggr` of an aggregate one (the queue length only, with the phases and the in-service order summed out), and the `Sys` variants give the joint state of the whole network.

The detailed state is discipline-dependent: under FCFS it records the order of the jobs in service, under PS it does not, and with nonexponential service it carries the phase of each job. That is why the same script exists in an FCFS and a PS version.

3.3.1 `statepr_aggr`

The marginal probability of an aggregate station state, through `getProbAggr`.

3.3.2 `statepr_aggr_large`

The same on a model whose state space is large enough to need the aggregate form.

Nodes	4: Source, Queue × 2, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(0.4); Class2: Exp(0.3)
Service	Queue1 – Class1: Exp(1); Class2: Exp(0.9) Queue2 – Class1: Exp(1.1); Class2: Exp(1)
Solvers	JMT

Table 3.18: Features of `fcr_oqnwaitq`.

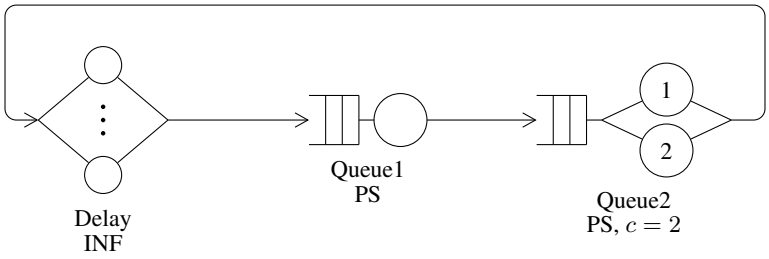


Figure 3.19: Topology of `statepr_aggr`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 0$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(3); Class2: Exp(4) Queue2 – Class1: Exp(1); Class2: Exp(3)
Solvers	CTMC, NC

Table 3.19: Features of `statepr_aggr`.

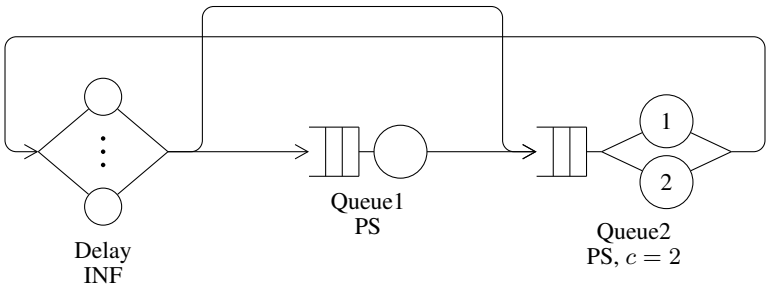


Figure 3.20: Topology of `statepr_aggr_large`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	4: Class1 (closed, $N = 1$), Class2 (closed, $N = 0$), Class3 (closed, $N = 4$), Class4 (closed, $N = 0$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(2); Class3: Exp(1); Class4: Exp(1) Queue1 – Class1: Exp(3); Class2: Exp(4); Class3: Exp(5); Class4: Exp(1) Queue2 – Class1: Exp(1); Class2: Exp(3); Class3: Exp(5); Class4: Exp(2)
Features	class switching on 1 link
Solvers	CTMC, NC

Table 3.20: Features of `statepr_aggr_large`.

3.3.3 statepr_allprobs_fcfs

The FCFS version, where the detailed state carries the order of the jobs in service.

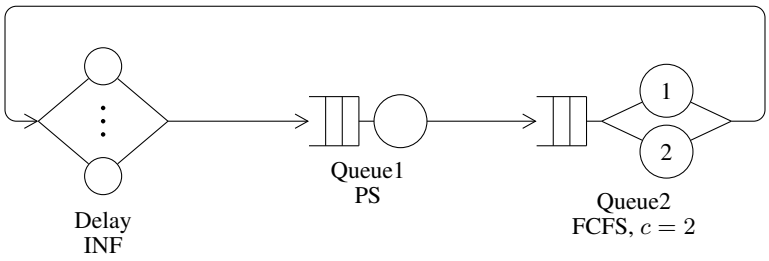


Figure 3.21: Topology of `statepr_allprobs_fcfs`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 0$)
Scheduling	Delay: INF; Queue1: PS; Queue2: FCFS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(3); Class2: Exp(4) Queue2 – Class1: Exp(3); Class2: Exp(3)
Solvers	CTMC, NC, SSA, JMT

Table 3.21: Features of `statepr_allprobs_fcfs`.

3.3.4 statepr_allprobs_ps

Every probability API on one PS model: detailed and aggregate, per station and system-wide.

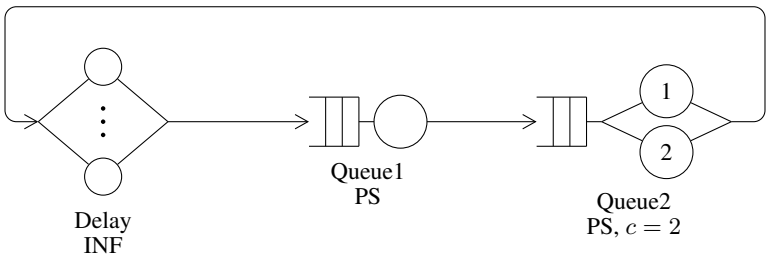


Figure 3.22: Topology of `statepr_allprobs_ps`: a closed network over 3 queueing stations.

3.3.5 statepr_sys_aggr

The joint aggregate state of the whole network, through `getProbSysAggr`.

3.3.6 statepr_sys_aggr_large

The large-model version.

3.3.7 statepr_sys_marg

The joint law of the per-station TOTAL queue lengths, all classes summed out, through `getProbSysMarg`.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 0$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(3); Class2: Exp(4) Queue2 – Class1: Exp(1); Class2: Exp(3)
Solvers	CTMC, NC, SSA, JMT

Table 3.22: Features of `statepr_allprobs_ps`.

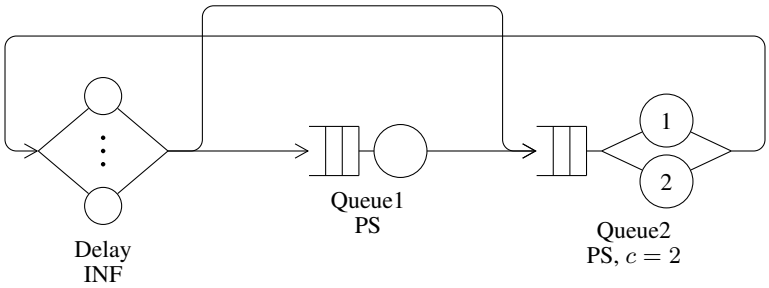


Figure 3.23: Topology of `statepr_sys_aggr`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	4: Class1 (closed, $N = 1$), Class2 (closed, $N = 0$), Class3 (closed, $N = 3$), Class4 (closed, $N = 0$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS ($c = 2$)
Service	Delay – Class1: Exp(1); Class2: Exp(2); Class3: Exp(1); Class4: Exp(1) Queue1 – Class1: Exp(3); Class2: Exp(4); Class3: Exp(5); Class4: Exp(1) Queue2 – Class1: Exp(1); Class2: Exp(3); Class3: Exp(5); Class4: Exp(2)
Features	class switching on 1 link
Solvers	CTMC, NC, JMT

Table 3.23: Features of `statepr_sys_aggr`.

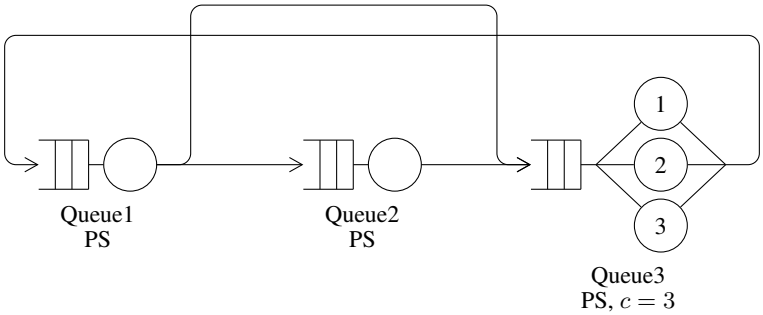


Figure 3.24: Topology of `statepr_sys_aggr_large`: a closed network over 3 queueing stations.

Nodes	3: Queue × 3
Classes	4: Class1 (closed, $N = 1$), Class2 (closed, $N = 1$), Class3 (closed, $N = 1$), Class4 (closed, $N = 1$)
Scheduling	Queue1: PS; Queue2: PS; Queue3: PS ($c = 3$)
Service	Queue1 – Class1: Exp(1); Class2: Exp(2); Class3: Exp(1); Class4: Exp(1) Queue2 – Class1: Exp(3); Class2: Exp(4); Class3: Exp(5); Class4: Exp(1) Queue3 – Class1: Exp(1); Class2: Exp(3); Class3: Exp(5); Class4: Exp(2)
Features	class switching on 1 link
Solvers	CTMC

Table 3.24: Features of `statepr_sys_aggr_large`.

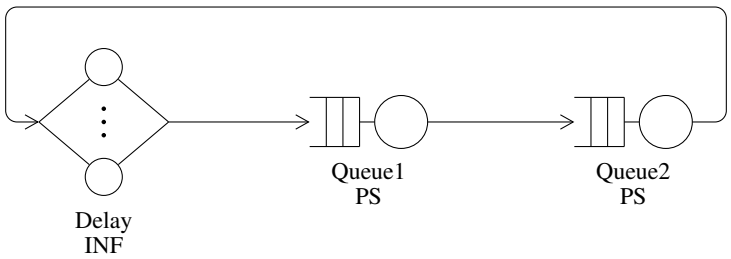


Figure 3.25: Topology of `statepr_sys_marg`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – Class1: Exp(0.6667); Class2: Exp(0.5) Queue1 – Class1: Exp(1.429); Class2: Exp(2.5) Queue2 – Class1: Exp(3.333); Class2: Exp(1.111)
Solvers	CTMC

Table 3.25: Features of `statepr_sys_marg`.

3.4 Initial state and warm start

Transient analysis needs an initial state. `initFromMarginal` builds one from marginal queue lengths, filling in the phases and the in-service order in a way consistent with the discipline; a state may also be set explicitly. For closed classes the initial state must conserve the population, which is what makes the ordered form necessary rather than convenient.

A simulation can also be started from the steady state computed by another solver, which removes the warmup transient from the sample path.

3.4.1 `init_state_fcfs_exp`

An initial state set from marginal queue lengths, and the transient it produces on an FCFS network.

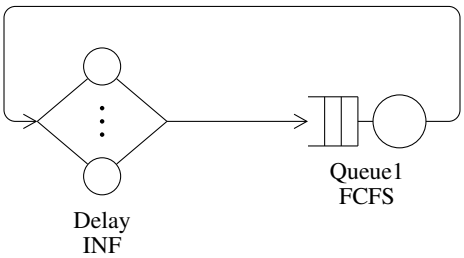


Figure 3.26: Topology of `init_state_fcfs_exp`: a closed network over 2 queueing stations.

3.4.2 `init_state_fcfs_nonexp`

The same with nonexponential service, where the phase of the job in service is part of the state.

3.4.3 `init_state_ps`

The processor sharing version.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: FCFS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(0.7)

Table 3.26: Features of `init_state_fcfs_exp`.

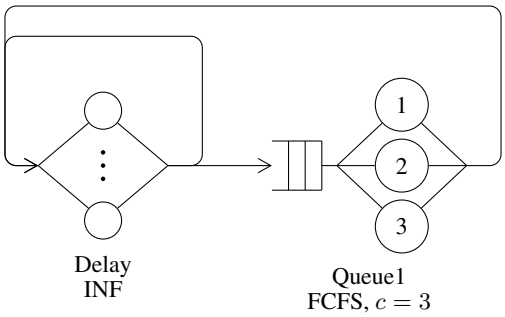


Figure 3.27: Topology of `init_state_fcfs_nonexp`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 3$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: FCFS ($c = 3$)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(1.2); Class2: Erlang(2,2)
Solvers	CTMC, FLD

Table 3.27: Features of `init_state_fcfs_nonexp`.

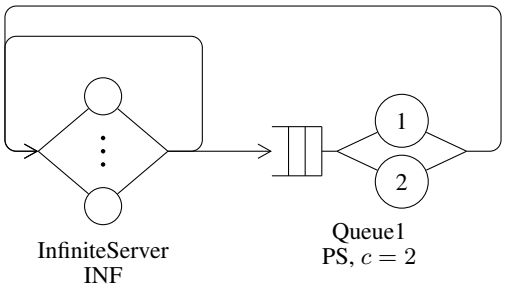


Figure 3.28: Topology of `init_state_ps`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 3$), Class2 (closed, $N = 1$)
Scheduling	InfiniteServer: INF; Queue1: PS ($c = 2$)
Service	InfiniteServer – Class1: Exp(3); Class2: Exp(0.5) Queue1 – Class1: Exp(0.1); Class2: Exp(1)
Solvers	CTMC, JMT, SSA, FLD, MVA, NC

Table 3.28: Features of `init_state_ps`.

3.4.4 `ldes_warmstart`

Seeding a simulation from the steady state of another solver, so the warmup phase can be skipped.

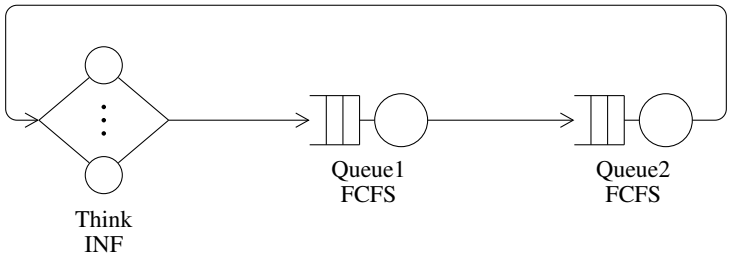


Figure 3.29: Topology of `ldes_warmstart`: a closed network over 3 queueing stations.

Nodes	3: Queue $\times$ 2, Delay
Classes	1: Jobs (closed, $N = 100$)
Scheduling	Think: INF; Queue1: FCFS; Queue2: FCFS
Service	Think – Jobs: Exp(1) Queue1 – Jobs: Exp(1) Queue2 – Jobs: Exp(0.98)
Solvers	MVA, CTMC, LDES

Table 3.29: Features of `ldes_warmstart`.

3.5 State-dependent routing

Under RAND or probabilistic routing the destination is drawn from a fixed distribution. State-dependent policies read the network state instead: JSQ sends the job to the shortest queue, k-choices samples k queues and picks the shortest of those, and round-robin and its weighted form cycle deterministically. Because the routing depends on the state, the network is no longer product-form, and JSQ in particular is only available to the simulators.

3.5.1 `sdroute_closed`

The closed counterpart.

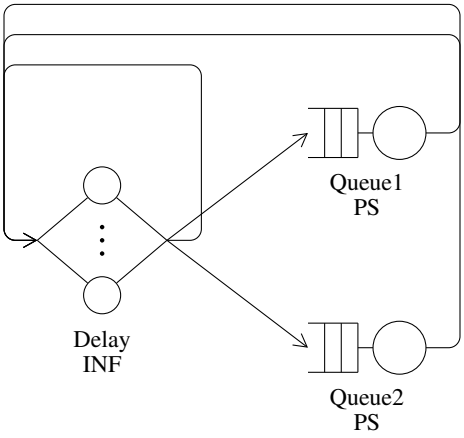


Figure 3.30: Topology of `sdroute_closed`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	1: Class1 (closed, $N = 1$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – Class1: HyperExp(0.99,0.01;1.534,0.02819) Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(2)
Features	class switching on 1 link
Solvers	CTMC, JMT, SSA

Table 3.30: Features of `sdroute_closed`.

3.5.2 `sdroute_jsq`

Join-the-shortest-queue dispatching over a pool of FCFS queues.

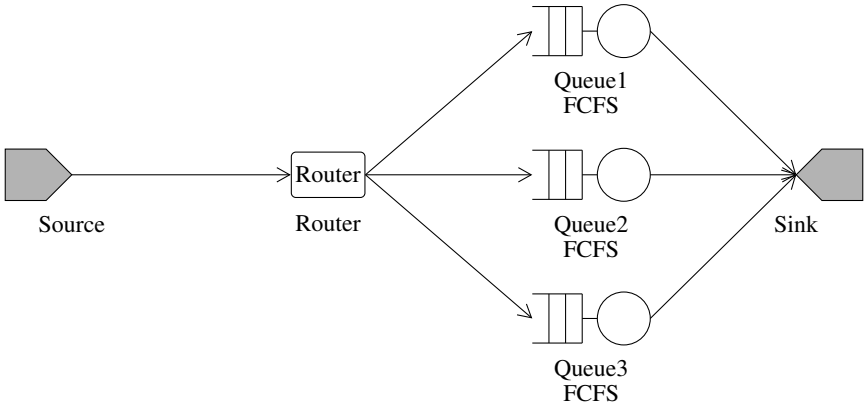


Figure 3.31: Topology of `sdroute_jsq`: an open network over 3 queueing stations with a router.

Nodes	6: Source, Queue × 3, Router, Sink
Classes	1: Class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Arrivals	Source – Class1: Exp(2.1)
Service	Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(1) Queue3 – Class1: Exp(1)
Features	class switching on 6 links
Solvers	JMT, LDES

Table 3.31: Features of `sdroute_jsq`.

3.5.3 `sdroute_krzesinski`

Adaptive routing with balanced loading (Krzesinski 1987): a central server and four single-center branches whose routing probabilities depend on the branch populations, and which keep a product form.

3.5.4 `sdroute_multibranch`

The same product form on branches holding several centers, where the coefficients are the branch traffic equations.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

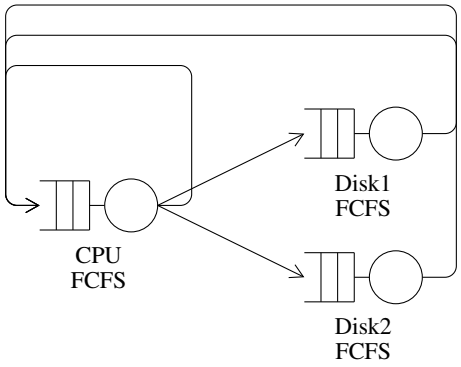


Figure 3.32: Topology of `sdroute_krzesinski`: a closed network over 3 queueing stations.

Nodes	3: Queue × 3
Classes	1: Class1 (closed, $N = 3$)
Scheduling	CPU: FCFS; Disk1: FCFS; Disk2: FCFS
Service	CPU – Class1: Exp(1) Disk1 – Class1: Exp(0.8) Disk2 – Class1: Exp(0.5)
Solvers	NC, CTMC

Table 3.32: Features of `sdroute_krzesinski`.

3.5.5 `sdroute_open`

Round-robin dispatching from a Router node in an open network.

3.5.6 `sdroute_twoclasses_closed`

Two classes with MAP, MMPP(2) and phase-type service under round-robin dispatching.

3.6 Cyclic polling

A polling server visits its queues in a cycle. Under gated service it serves the jobs that were waiting when it arrived, under exhaustive service it stays until the queue empties, and under k-limited service it serves at most k jobs per visit. The discipline is declared as `SchedStrategy.POLLING` and the variant is a parameter of the station.

The polling server parks at a queue rather than roving: the visit order and the ratio between the queue lengths are what a polling model must be validated on, not the aggregate throughput, which is insensitive to the discipline.

3.6.1 `polling_exhaustive_det`

Exhaustive service with deterministic (immediate) switchover.

3.6.2 `polling_exhaustive_exp`

Exhaustive service with exponential switchover times.

Solvers	CTMC

Table 3.33: Features of `sdroute_multibranch`.

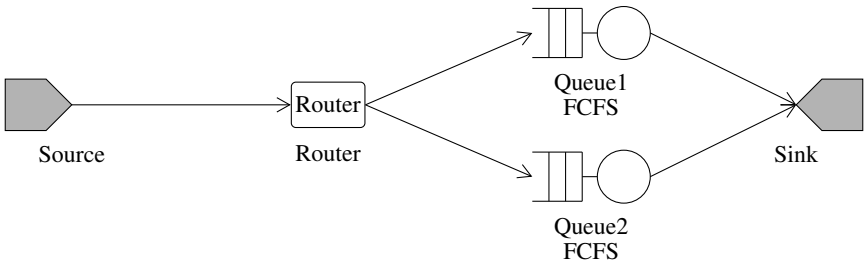


Figure 3.33: Topology of `sdroute_open`: an open network over 2 queueing stations with a router.

Nodes	5: Source, Queue × 2, Router, Sink
Classes	1: Class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Queue1 – Class1: Exp(2) Queue2 – Class1: Exp(2)
Features	class switching on 5 links
Solvers	JMT, CTMC

Table 3.34: Features of `sdroute_open`.

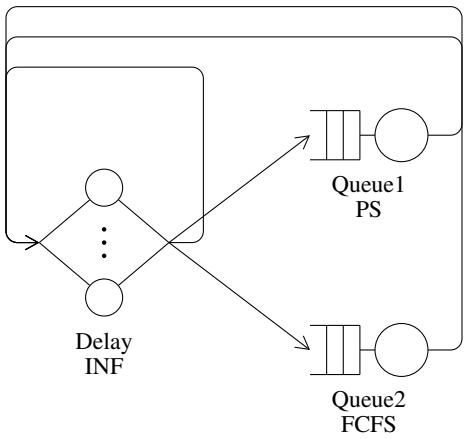


Figure 3.34: Topology of `sdroute_twoclasses_closed`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS; Queue2: FCFS
Service	Delay – Class1: HyperExp(0.99,0.01;1.534,0.02819); Class2: HyperExp(0.99,0.01;1.534,0.02819) Queue1 – Class1: APH; Class2: PH Queue2 – Class1: MAP; Class2: MAP
Features	class switching on 1 link
Solvers	CTMC, JMT, SSA

Table 3.35: Features of `sdroute_twoclasses_closed`.

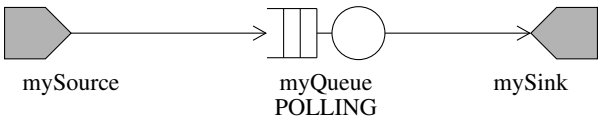


Figure 3.35: Topology of `polling_exhaustive_det`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: POLLING
Arrivals	mySource – myClass1: Det(1); myClass2: Det(1)
Service	myQueue – myClass1: Det(0.001); myClass2: Det(0.001)
Solvers	LDES

Table 3.36: Features of `polling_exhaustive_det`.

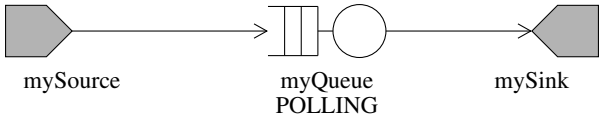


Figure 3.36: Topology of `polling_exhaustive_exp`: an open network over 1 queueing station.

3.6.3 `polling_gated`

A gated polling server: at each visit it serves the jobs that were waiting when it arrived.

3.6.4 `polling_klimited`

k-limited service, at most k jobs per visit.

3.7 Switchover times

Moving the server between queues costs time. A switchover distribution is declared per arc of the polling cycle, and its mean adds directly to the cycle time, so the waiting time grows even though no extra work is done.

3.7.1 `switchover_basic`

Pairwise switchover metadata on an ordinary FCFS queue. The JMT run warns that nonpolling queues do not consume these times and solves the underlying saturated two-class queue without them; use the cyclic-polling examples for honoured switchover delays.

3.8 Response time distribution

`getCdfRespT` returns the cumulative distribution of the response time, per station and per class, rather than its mean. The distribution is what a percentile service level objective is written against, and it is far more sensitive to the service distribution than the mean is: two models with the same mean response time can have very different tails.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: POLLING
Arrivals	mySource – myClass1: Exp(0.1); myClass2: Exp(0.1)
Service	myQueue – myClass1: Exp(1); myClass2: Exp(1.5)
Solvers	LDES

Table 3.37: Features of `polling_exhaustive_exp`.

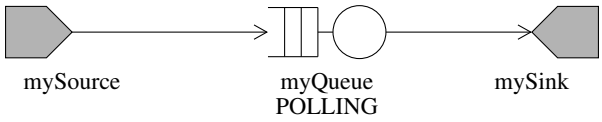


Figure 3.37: Topology of `polling_gated`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: POLLING
Arrivals	mySource – myClass1: Exp(1); myClass2: Exp(0.8)
Service	myQueue – myClass1: Exp(4); myClass2: Exp(1.5)
Solvers	JMT

Table 3.38: Features of `polling_gated`.

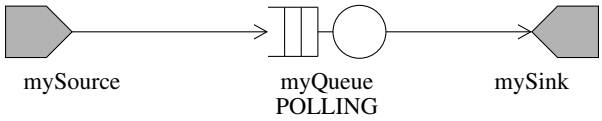


Figure 3.38: Topology of `polling_klimited`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: POLLING
Arrivals	mySource – myClass1: Exp(0.2); myClass2: Exp(0.3)
Service	myQueue – myClass1: Exp(1); myClass2: Exp(1.5)
Solvers	LDES

Table 3.39: Features of `polling_klimited`.

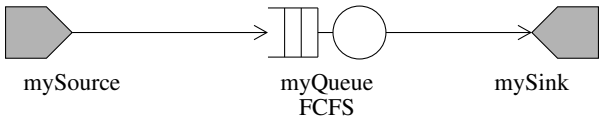


Figure 3.39: Topology of `switchover_basic`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass1: Exp(0.2); myClass2: Exp(0.8)
Service	myQueue – myClass1: Exp(0.1); myClass2: Exp(1.5)
Solvers	JMT

Table 3.40: Features of `switchover_basic`.

3.8.1 cdf_respt_closed

The response time CDF at a station of a closed PS network.

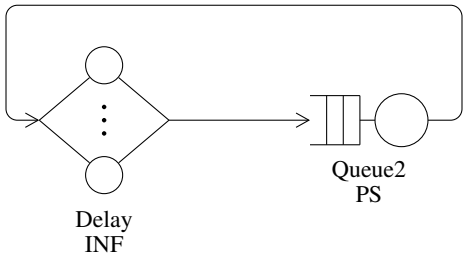


Figure 3.40: Topology of `cdf_respt_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 1$)
Scheduling	Delay: INF; Queue2: PS
Service	Delay – Class1: Exp(10) Queue2 – Class1: Erlang(3,3)
Solvers	JMT, FLD

Table 3.41: Features of `cdf_respt_closed`.

3.8.2 cdf_respt_closed_threeclasses

The three-class version, with Erlang service.

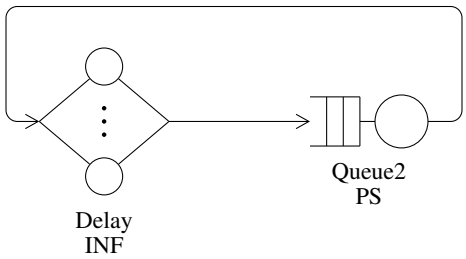


Figure 3.41: Topology of `cdf_respt_closed_threeclasses`: a closed network over 2 queueing stations.

3.8.3 cdf_respt_distrib

The distribution object returned by `getCdfRespT` and the percentiles read from it.

3.8.4 cdf_respt_open_twoclasses

The open two-class version, on FCFS stations.

3.8.5 cdf_respt_populations

How the response time distribution moves as the closed population grows.

Nodes	2: Queue, Delay
Classes	3: Class1 (closed, $N = 1$), Class2 (closed, $N = 0$), Class3 (closed, $N = 0$)
Scheduling	Delay: INF; Queue2: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1) Queue2 – Class1: Exp(1); Class2: Erlang(0.5,2); Class3: Exp(100)
Solvers	FLD

Table 3.42: Features of `cdf_respt_closed_threeclasses`.

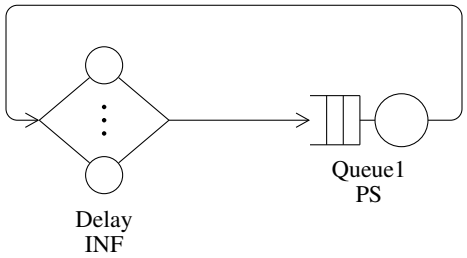


Figure 3.42: Topology of `cdf_respt_distrib`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 3$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Erlang(0.5,2) Queue1 – Class1: Exp(0.5); Class2: HyperExp(0.99,0.01;0.324,0.005143)
Solvers	FLD, JMT

Table 3.43: Features of `cdf_respt_distrib`.

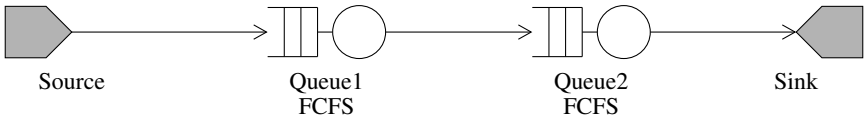


Figure 3.43: Topology of `cdf_respt_open_twoclasses`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(0.25); Class2: Exp(0.25)
Service	Queue1 – Class1: Exp(1); Class2: Exp(1) Queue2 – Class1: Exp(1); Class2: Exp(1)
Solvers	FLD, JMT

Table 3.44: Features of `cdf_respt_open_twoclasses`.

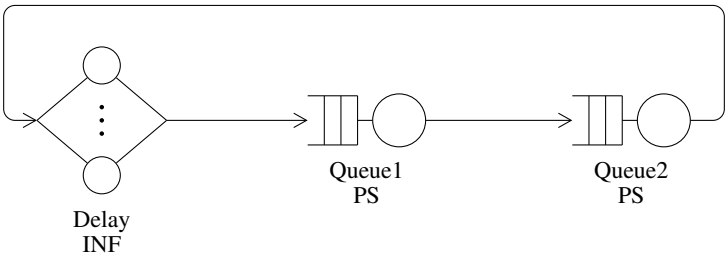


Figure 3.44: Topology of `cdf_respt_populations`: a closed network over 3 queueing stations.

Nodes	3: Queue × 2, Delay
Classes	1: Class1 (closed, $N = 8$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(0.5) Queue2 – Class1: Exp(0.5)
Solvers	FLD

Table 3.45: Features of `cdf_respt_populations`.

3.9 Busy periods

A busy period of order n runs from the instant an arrival raises the number of jobs held by the target to n until it falls back below n. The target may be a station, a station-class pair, or a declared set of stations, and the order n ranges over $1..K$. The analytical reference is the busy period formula of Daduna, exposed by the normalizing constant solver; the simulator measures the same quantity from the sample path.

3.9.1 `busyp_subnetwork`

Mean busy periods of order n for a station, for a station-class pair and for a declared subnetwork of stations.

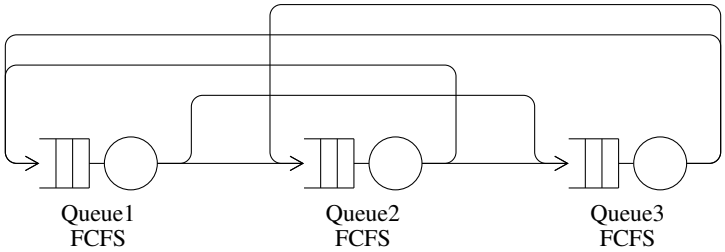


Figure 3.45: Topology of `busyp_subnetwork`: a closed network over 3 queueing stations.

Nodes	3: Queue × 3
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Queue1 – Class1: Exp(1.5) Queue2 – Class1: Exp(0.9) Queue3 – Class1: Exp(2)
Solvers	LDES

Table 3.46: Features of `busyp_subnetwork`.

3.10 Stochastic network calculus

Every other family in these examples reports a mean. The stochastic network calculus reports a *tail*: given a violation probability ε it returns a delay or a queue length that is exceeded with probability at most ε, under any work-conserving scheduling policy at the station. Each flow is carried by a moment-generating-function envelope and each server by the counting process of its service; the min-plus operations compose them across a feedforward network, at the cost of the burstiness a server adds to the flow it forwards. The bound on the mean that follows by integration is loose, and the decay rate of the tail is asymptotically exact, so the quantile is what these examples read.

3.10.1 snc_delay_quantile

A delay quantile with a certified violation probability, and how it tightens deep in the tail.

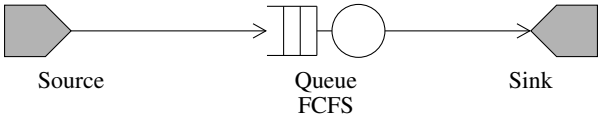


Figure 3.46: Topology of snc_delay_quantile: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(0.6)
Service	Queue – Class1: Exp(1)

Table 3.47: Features of snc_delay_quantile.

3.10.2 snc_tandem_multiclass

Envelope propagation across a feedforward tandem, pay-bursts-only-once, blind multiplexing, and the model classes the family refuses.

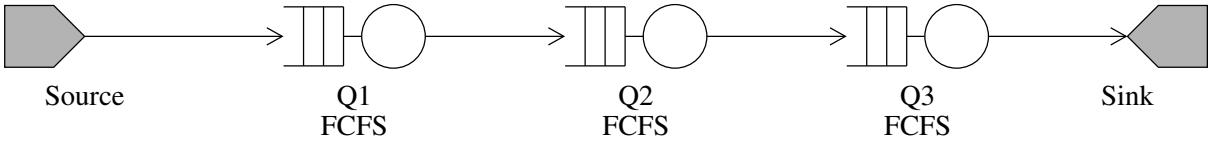


Figure 3.47: Topology of snc_tandem_multiclass: an open network over 3 queueing stations.

3.11 Random environments

A random environment is a Markov process over stages, with one model per stage. The environment moves between stages according to a transition process, and the network parameters change with it; the metrics reported are the environment average, weighted by the stage probabilities, but the transient handoff between stages matters when the stage sojourn is comparable with the relaxation time of the network.

Server breakdown and repair is the smallest instance: two stages, one with the server up and one with it down. The case studies at the end of the family are larger, and are the models on which the environment solver is validated.

3.11.1 example_mapqn2renv

Converting a queueing network with MMPP(2) service into an equivalent random environment model.

3.11.2 renv_basic

The smallest random environment: two stages, one model per stage, and the stage transition process. No topology is drawn for this example: the captured run yielded no Network or LayeredNetwork to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Nodes	5: Source, Queue $\times$ 3, Sink
Classes	1: Class1 (open)
Scheduling	Q1: FCFS; Q2: FCFS; Q3: FCFS
Arrivals	Source – Class1: Exp(0.6)
Service	Q1 – Class1: Exp(1.5) Q2 – Class1: Exp(1.2) Q3 – Class1: Exp(1)

Table 3.48: Features of `snc_tandem_multiclass`.

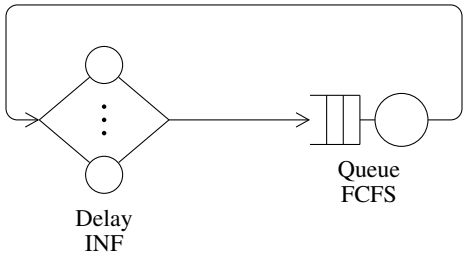


Figure 3.48: Topology of `example_mapqn2renv`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue: FCFS
Service	Delay – Class1: Exp(1) Queue – Class1: MMPP2
Solvers	JMT

Table 3.49: Features of `example_mapqn2renv`.

Solvers	FLD, MVA

Table 3.50: Features of `renv_basic`.

3.11.3 renv_container_terminal

A container terminal case study: MAP arrivals, multiserver and load-dependent stations, several environment stages.

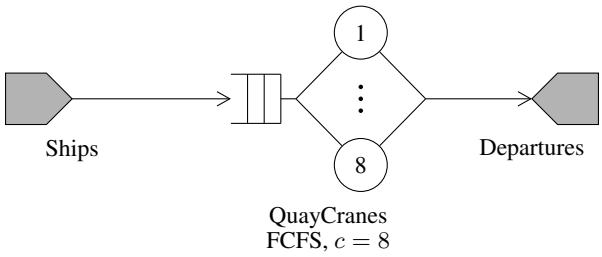


Figure 3.49: Topology of `renv_container_terminal`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Containers (open)
Scheduling	QuayCranes: FCFS ($c = 8$)
Arrivals	Ships – Containers: MAP
Service	QuayCranes – Containers: Exp(16)

Table 3.51: Features of `renv_container_terminal`.

3.11.4 renv_fourstages_repairmen

The four-stage version.

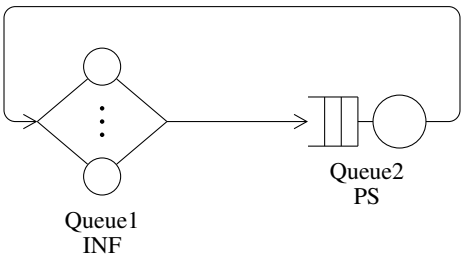


Figure 3.50: Topology of `renv_fourstages_repairmen`: a closed network over 2 queueing stations.

3.11.5 renv_genqn

A randomly generated network placed in an environment.

3.11.6 renv_lqn_attackgraph

A three-tier layered model driven by an absorbing attack-graph environment, solved by stage occupation.

3.11.7 renv_lqn_twostages

A layered model in a two-stage environment, including its transient.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 30$)
Scheduling	Queue1: INF; Queue2: PS
Service	Queue1 – Class1: Exp(4) Queue2 – Class1: Exp(1)
Solvers	ENV

Table 3.52: Features of `renv_fourstages_repairmen`.

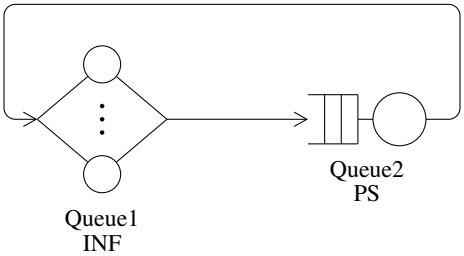


Figure 3.51: Topology of `renv_genqn`: a closed network over 2 queueing stations.

3.11.8 `renv_map_fallback`

An MMPP(2) stage transition process, and the fallback taken when it cannot be represented exactly.

3.11.9 `renv_node_breakdown`

Server breakdown and repair as an environment over a queueing model.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

3.11.10 `renv_rotterdam_blending`

The Rotterdam blending case study.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

3.11.11 `renv_threestages_repairmen`

The three-stage version.

3.11.12 `renv_twostages_repairmen`

A repairman model in a two-stage random environment.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 4$)
Scheduling	Queue1: INF; Queue2: PS
Service	Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(0.5)
Solvers	MVA

Table 3.53: Features of `renv_genqn`.

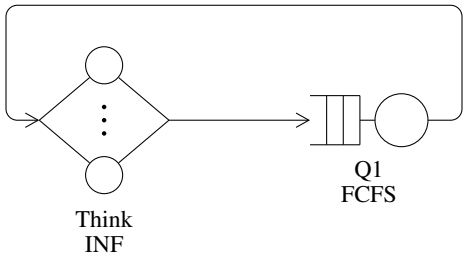


Figure 3.52: Topology of `renv_map_fallback`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: C1 (closed, $N = 5$)
Scheduling	Think: INF; Q1: FCFS
Service	Think – C1: Exp(1) Q1 – C1: MMPP2
Solvers	ENV, CTMC

Table 3.54: Features of `renv_map_fallback`.

Solvers	ENV

Table 3.55: Features of `renv_node_breakdown`.

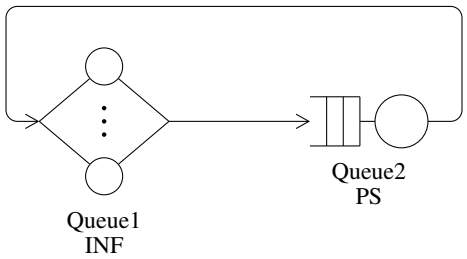


Figure 3.53: Topology of `renv_threestages_repairmen`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 2$)
Scheduling	Queue1: INF; Queue2: PS
Service	Queue1 – Class1: Exp(3) Queue2 – Class1: Exp(1)
Solvers	ENV

Table 3.56: Features of `renv_threestages_repairmen`.

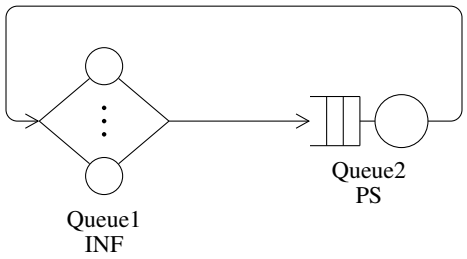


Figure 3.54: Topology of `renv_twostages_repairmen`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 1$)
Scheduling	Queue1: INF; Queue2: PS
Service	Queue1 – Class1: Exp(2) Queue2 – Class1: Exp(1)
Solvers	ENV

Table 3.57: Features of `renv_twostages_repairmen`.

3.12 Layered cache-queueing models

A `CacheTask` inside a layered model exposes `ItemEntry` points, and a call to one is a cache request whose hit or miss outcome selects the downstream branch. The replacement policy is simulated directly, so hit and miss are exact sample-path events rather than a hit-ratio parameter.

3.12.1 `lcq_async_cache`

An ASYNCHRONOUS (nonblocking) call into the cache task, as a prefetch or a warmup makes: the client continues without waiting for the reply while the request still resolves through POST_CACHE hit/miss branching.

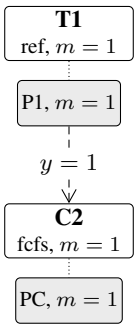


Figure 3.55: Call graph of `lcq_async_cache`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 1$), PC (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 1$), C2 (fcfs, $m = 1$)
Entries	2: E1, I2
Activities	4, host demands A1: 0.0, AC2: 0.0, AC2h: 1.0, AC2m: 2.0
Calls	1 (0 synchronous, 1 asynchronous)

Table 3.58: Features of `lcq_async_cache`.

3.12.2 `lcq_singlehost`

A cache task and its item entries inside a layered model on one processor.

3.12.3 `lcq_threehosts`

The three-processor version, where the cache and its clients are on separate hosts.

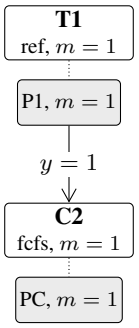


Figure 3.56: Call graph of `lcq_singlehost`: 2 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 1$), PC (ps, $m = 1$)
Tasks	2: T1 (ref, $m = 1$), C2 (fcfs, $m = 1$)
Entries	2: E1, I2
Activities	4, host demands A1: 0.0, AC2: 0.0, AC2h: 1.0, AC2m: 2.0
Calls	1 (1 synchronous, 0 asynchronous)

Table 3.59: Features of `lcq_singlehost`.

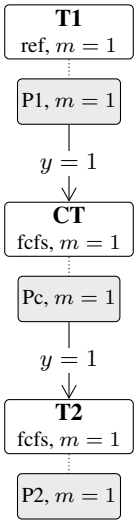


Figure 3.57: Call graph of `lcq_threehosts`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	3: P1 (ps, $m = 1$), Pc (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	3: T1 (ref, $m = 1$), CT (fcfs, $m = 1$), T2 (fcfs, $m = 1$)
Entries	3: E1, IE, E2
Activities	5, host demands A1: 0.0, Ac: 0.0, Ac_hit: 1.0, Ac_miss: 2.0, A2: 0.2
Calls	2 (2 synchronous, 0 asynchronous)

Table 3.60: Features of `lcq_threehosts`.

3.13 Pass and swap

Under the pass-and-swap discipline a job that finds an incompatible job in service swaps with it rather than waiting, subject to a compatibility relation between classes. The initial placement of a closed population is therefore part of the model: the state must be set in the order the discipline assumes, and a placement that ignores it changes the answer.

3.13.1 `pas_closed_tandem_fig6`

The closed tandem of the reference paper, figure 6.

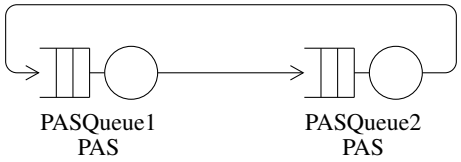


Figure 3.58: Topology of `pas_closed_tandem_fig6`: a closed network over 2 queueing stations.

Nodes	2: Queue × 2
Classes	6: Class1 (closed, $N = 1$), Class2 (closed, $N = 1$), Class3 (closed, $N = 1$), Class4 (closed, $N = 1$), Class5 (closed, $N = 1$), Class6 (closed, $N = 1$)
Scheduling	PASQueue1: PAS; PASQueue2: PAS
Service	PASQueue1 – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1); Class4: Exp(1); Class5: Exp(1); Class6: Exp(1) PASQueue2 – Class1: Exp(1.3); Class2: Exp(1.3); Class3: Exp(1.3); Class4: Exp(1.3); Class5: Exp(1.3); Class6: Exp(1.3)
Features	finite buffer 6 at PASQueue1; finite buffer 6 at PASQueue2; class switching on 2 links
Solvers	CTMC

Table 3.61: Features of `pas_closed_tandem_fig6`.

3.13.2 `pas_compatibility_5class`

Five classes and the compatibility matrix that says which may swap with which.

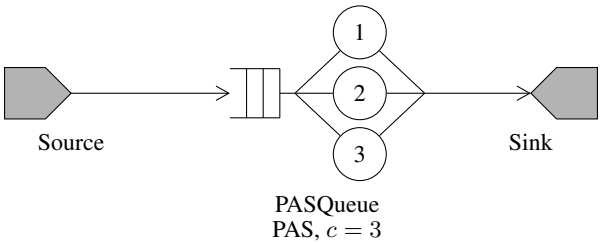


Figure 3.59: Topology of `pas_compatibility_5class`: an open network over 1 queueing station.

3.13.3 `pas_cyclic_ctmc_vs_bruteforce`

A cyclic pass-and-swap network solved by CTMC and by brute-force enumeration, as a cross-check. No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

Nodes	3: Source, Queue, Sink
Classes	5: Class1 (open), Class2 (open), Class3 (open), Class4 (open), Class5 (open)
Scheduling	PASQueue: PAS ($c = 3$)
Arrivals	Source – Class1: Exp(0.5); Class2: Exp(0.4); Class3: Exp(0.3); Class4: Exp(0.2); Class5: Exp(0.1)
Service	PASQueue – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1); Class4: Exp(2); Class5: Exp(1)
Features	finite buffer 3 at PASQueue
Solvers	CTMC

Table 3.62: Features of `pas_compatibility_5class`.

Solvers	CTMC, LDES

Table 3.63: Features of `pas_cyclic_ctmc_vs_bruteforce`.

3.13.4 `pas_cyclic_normconst_bruteforce`

The same cross-check on the normalizing constant.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

3.13.5 `pas_mmk`

The pass-and-swap discipline on an M/M/k station.

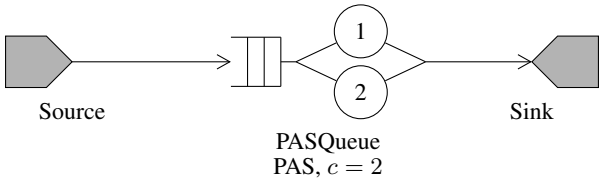


Figure 3.60: Topology of `pas_mmk`: an open network over 1 queueing station.

3.13.6 `pas_nc_sampling`

The normalizing constant of a pass-and-swap network, by sampling.

3.13.7 `pas_saturation`

The saturated regime, where the swap rule sets the throughput.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

3.13.8 `pas_selfloop`

A self-looping class under pass-and-swap.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	PASQueue: PAS ($c = 2$)
Arrivals	Source – Class1: Exp(0.7); Class2: Exp(0.5)
Service	PASQueue – Class1: Exp(1); Class2: Exp(1)
Features	finite buffer 4 at PASQueue
Solvers	CTMC

Table 3.64: Features of `pas_mmk`.

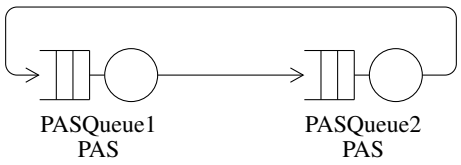


Figure 3.61: Topology of `pas_nc_sampling`: a closed network over 2 queueing stations.

Nodes	2: Queue $\times$ 2
Classes	6: Class1 (closed, $N = 1$), Class2 (closed, $N = 1$), Class3 (closed, $N = 1$), Class4 (closed, $N = 1$), Class5 (closed, $N = 1$), Class6 (closed, $N = 1$)
Scheduling	PASQueue1: PAS; PASQueue2: PAS
Service	PASQueue1 – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1); Class4: Exp(1); Class5: Exp(1); Class6: Exp(1) PASQueue2 – Class1: Exp(1.3); Class2: Exp(1.3); Class3: Exp(1.3); Class4: Exp(1.3); Class5: Exp(1.3); Class6: Exp(1.3)
Features	finite buffer 6 at PASQueue1; finite buffer 6 at PASQueue2

Table 3.65: Features of `pas_nc_sampling`.

Solvers	CTMC, LDES

Table 3.66: Features of `pas_saturation`.

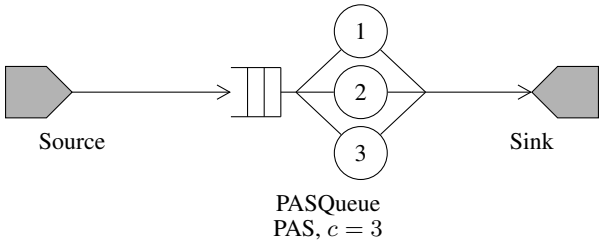


Figure 3.62: Topology of `pas_selfloop`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	3: Class1 (open), Class2 (open), Class3 (open)
Scheduling	PASQueue: PAS ($c = 3$)
Arrivals	Source – Class1: Exp(0.6); Class2: Exp(0.4); Class3: Exp(0.3)
Service	PASQueue – Class1: Exp(1); Class2: Exp(1); Class3: Exp(1)
Features	finite buffer 3 at PASQueue
Solvers	CTMC

Table 3.67: Features of `pas_selfloop`.

3.14 Reward models

A reward model attaches a rate to each state, or a value to each transition, and reports the expected accumulated reward. Cost per waiting job, revenue per completion and penalty per loss are the usual three, and together they turn a performance model into an economic one. Rewards may be declared over aggregate states when the detailed state space is too large to enumerate.

3.14.1 `rewardModel_aggregation`

Rewards over aggregated states, when the detailed state is too large to enumerate.

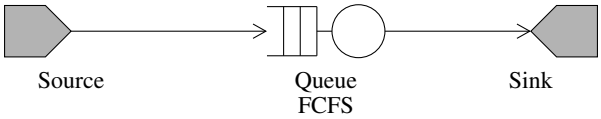


Figure 3.63: Topology of `rewardModel_aggregation`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: HighPriority (open), LowPriority (open)
Scheduling	Queue: FCFS
Arrivals	Source – HighPriority: Exp(1); LowPriority: Exp(0.8)
Service	Queue – HighPriority: Exp(3); LowPriority: Exp(3)
Features	finite buffer 4 at Queue

Table 3.68: Features of `rewardModel_aggregation`.

3.14.2 `rewardModel_mm1k`

State rewards on an M/M/1/K queue: cost per waiting job, revenue per completion, penalty per loss.

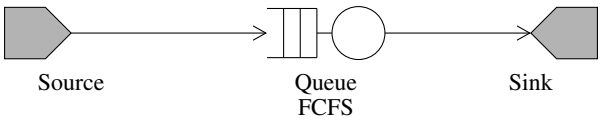


Figure 3.64: Topology of `rewardModel_mm1k`: an open network over 1 queueing station.

3.14.3 `rewardModel_multiclass`

Per-class rewards on a multiclass PS station.

3.14.4 `rewardModel_templates`

The reward templates that cover the common cases without writing a reward function.

3.15 Agent models

The agent interface builds a network from the point of view of the entities that move through it rather than from the stations. The resulting models are ordinary networks – Jackson networks, closed multiclass networks, G-networks with their negative customers – and are solved by the same solvers.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(2)
Service	Queue – Class1: Exp(3)
Features	finite buffer 3 at Queue
Solvers	CTMC

Table 3.69: Features of `rewardModel_mm1k`.

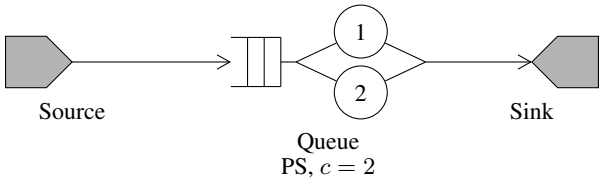


Figure 3.65: Topology of `rewardModel_multiclass`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Interactive (open), Batch (open)
Scheduling	Queue: PS ($c = 2$)
Arrivals	Source – Interactive: Exp(2); Batch: Exp(1.5)
Service	Queue – Interactive: Exp(0.5); Batch: Exp(1)
Features	finite buffer 6 at Queue

Table 3.70: Features of `rewardModel_multiclass`.

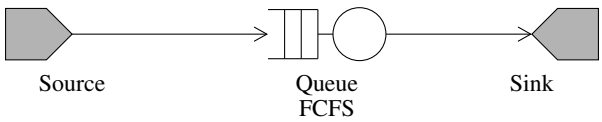


Figure 3.66: Topology of `rewardModel_templates`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1.5)
Service	Queue – Class1: Exp(2)
Features	finite buffer 10 at Queue

Table 3.71: Features of `rewardModel_templates`.

3.15.1 ag_closed_network

The closed counterpart.

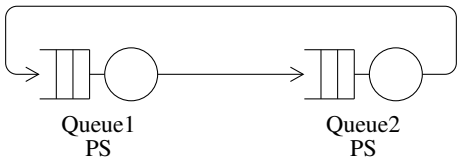


Figure 3.67: Topology of `ag_closed_network`: a closed network over 2 queueing stations.

Nodes	2: Queue × 2
Classes	1: Class1 (closed, $N = 10$)
Scheduling	Queue1: PS; Queue2: PS
Service	Queue1 – Class1: Exp(2) Queue2 – Class1: Exp(1)
Solvers	MVA, CTMC

Table 3.72: Features of `ag_closed_network`.

3.15.2 ag_gnetwork

A G-network, with the negative customers that remove jobs on arrival.

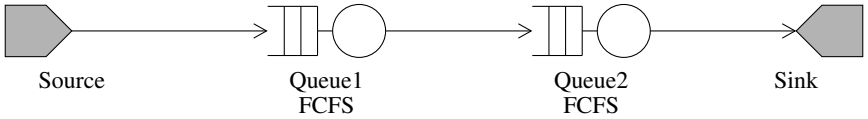


Figure 3.68: Topology of `ag_gnetwork`: an open network over 2 queueing stations.

3.15.3 ag_jackson_network

A Jackson network built agent by agent.

3.15.4 ag_multiclass_closed

The multiclass closed version.

3.15.5 ag_tandem_open

An open tandem written through the agent interface.

3.15.6 ag_tandem_phasetype

An open tandem whose two queues share a mean service time and differ only in variability (Erlang-2 against a hyperexponential), solved by the RCAT methods, which carry the phase-type law exactly where an M/M/1 reading sees no difference.

Nodes	4: Source, Queue × 2, Sink
Classes	2: Positive (open), Negative (signal)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Positive: Exp(1); Negative: Exp(0.3)
Service	Queue1 – Positive: Exp(2); Negative: Exp(2) Queue2 – Positive: Exp(3); Negative: Exp(3)

Table 3.73: Features of `ag_gnetwork`.

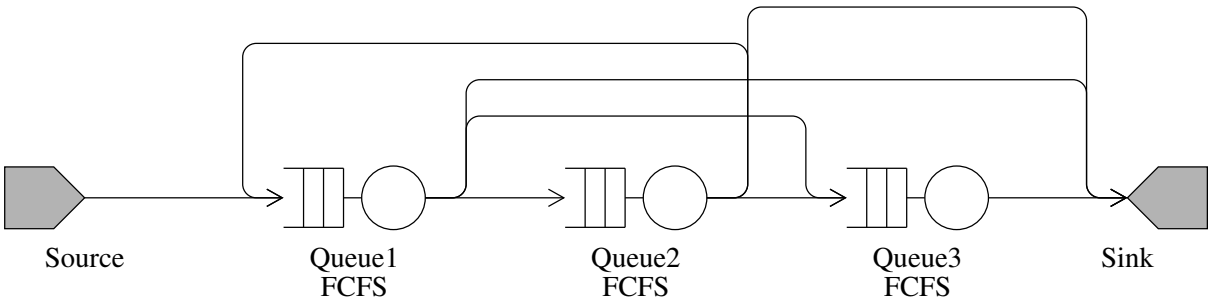


Figure 3.69: Topology of `ag_jackson_network`: an open network over 3 queueing stations.

Nodes	5: Source, Queue × 3, Sink
Classes	1: Class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Queue1 – Class1: Exp(2) Queue2 – Class1: Exp(3) Queue3 – Class1: Exp(2.5)
Solvers	MVA

Table 3.74: Features of `ag_jackson_network`.

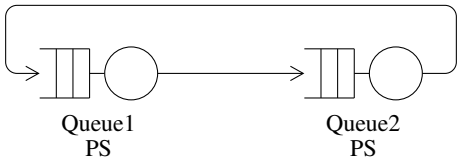


Figure 3.70: Topology of `ag_multiclass_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue × 2
Classes	2: Class1 (closed, $N = 5$), Class2 (closed, $N = 3$)
Scheduling	Queue1: PS; Queue2: PS
Service	Queue1 – Class1: Exp(2); Class2: Exp(1.5) Queue2 – Class1: Exp(1); Class2: Exp(0.8)
Solvers	MVA

Table 3.75: Features of `ag_multiclass_closed`.

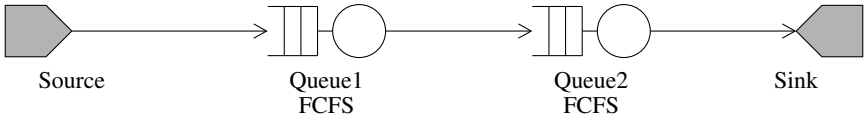


Figure 3.71: Topology of `ag_tandem_open`: an open network over 2 queueing stations.

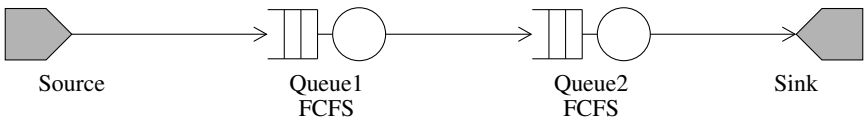


Figure 3.72: Topology of `ag_tandem_phasetype`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: Class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(0.5)
Service	Queue1 – Class1: Exp(1) Queue2 – Class1: Exp(1.5)
Solvers	MVA

Table 3.76: Features of `ag_tandem_open`.

Nodes	4: Source, Queue × 2, Sink
Classes	1: Class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – Class1: Exp(0.5)
Service	Queue1 – Class1: Erlang(2,2) Queue2 – Class1: HyperExp(0.99,0.01;1.14,0.07584)

Table 3.77: Features of `ag_tandem_phasetype`.

3.16 Large-scale models

When the state space cannot be enumerated, the exact solvers are out and the question becomes how much accuracy the approximations retain. The script here is a large closed PS network solved by several approximate methods.

3.16.1 `largescale_tbi`

A large closed PS network, and the accuracy the approximate solvers retain on it.

3.17 Custom solvers

A solver is a class with an analyzer and a feature set. The feature set is a claim about what the solver can represent, and the dispatcher uses it to decide whether a model may be given to it; the analyzer computes the metrics. These two scripts show the smallest pair that the dispatcher will accept.

3.17.1 `solver_custom`

A user-defined solver registered with the dispatcher.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

3.17.2 `solver_custom_analyzer`

Its analyzer, the function that actually computes the metrics.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

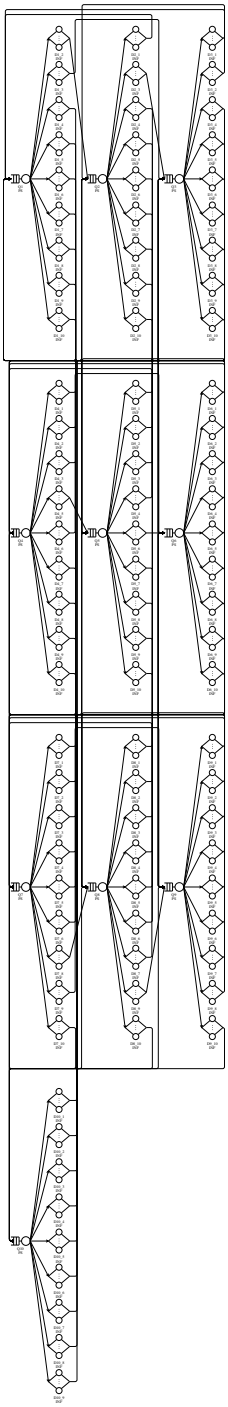


Figure 3.73: Topology of `largescale_tbi`: a closed network over 100 queueing stations.

Nodes	100: Queue × 10, Delay × 90
Classes	1: C1 (closed, $N = 250$)
Scheduling	Q1: PS; Q2: PS; Q3: PS; Q4: PS; Q5: PS; Q6: PS; Q7: PS; Q8: PS; Q9: PS; Q10: PS; D1_2: INF; D1_3: INF; D1_4: INF; D1_5: INF; D1_6: INF; D1_7: INF; D1_8: INF; D1_9: INF; D1_10: INF; D2_1: INF; D2_3: INF; D2_4: INF; D2_5: INF; D2_6: INF; D2_7: INF; D2_8: INF; D2_9: INF; D2_10: INF; D3_1: INF; D3_2: INF; D3_4: INF; D3_5: INF; D3_6: INF; D3_7: INF; D3_8: INF; D3_9: INF; D3_10: INF; D4_1: INF; D4_2: INF; D4_3: INF; D4_5: INF; D4_6: INF; D4_7: INF; D4_8: INF; D4_9: INF; D4_10: INF; D5_1: INF; D5_2: INF; D5_3: INF; D5_4: INF; D5_6: INF; D5_7: INF; D5_8: INF; D5_9: INF; D5_10: INF; D6_1: INF; D6_2: INF; D6_3: INF; D6_4: INF; D6_5: INF; D6_7: INF; D6_8: INF; D6_9: INF; D6_10: INF; D7_1: INF; D7_2: INF; D7_3: INF; D7_4: INF; D7_5: INF; D7_6: INF; D7_8: INF; D7_9: INF; D7_10: INF; D8_1: INF; D8_2: INF; D8_3: INF; D8_4: INF; D8_5: INF; D8_6: INF; D8_7: INF; D8_9: INF; D8_10: INF; D9_1: INF; D9_2: INF; D9_3: INF; D9_4: INF; D9_5: INF; D9_6: INF; D9_7: INF; D9_8: INF; D9_10: INF; D10_1: INF; D10_2: INF; D10_3: INF; D10_4: INF; D10_5: INF; D10_6: INF; D10_7: INF; D10_8: INF; D10_9: INF
Service	Q1 – C1: Erlang(10.14,4) Q2 – C1: Erlang(7.148,4) Q3 – C1: Erlang(9.512,4) Q4 – C1: Erlang(14.34,4) Q5 – C1: Erlang(12.08,4) Q6 – C1: Erlang(9.054,4) Q7 – C1: Erlang(12.37,4) Q8 – C1: Erlang(13.3,4) Q9 – C1: Erlang(7.543,4) Q10 – C1: Erlang(8.317,4) D1_2 – C1: Erlang(7.616,16) D1_3 – C1: Erlang(7.629,16) D1_4 – C1: Erlang(15.29,16) D1_5 – C1: Erlang(15.71,16) D1_6 – C1: Erlang(62.72,16) D1_7 – C1: Erlang(12.54,16) D1_8 – C1: Erlang(9.005,16) D1_9 – C1: Erlang(14.45,16) D1_10 – C1: Erlang(15.9,16) D2_1 – C1: Erlang(9.408,16) D2_3 – C1: Erlang(13.67,16) D2_4 – C1: Erlang(7.535,16) D2_5 – C1: Erlang(12.48,16) D2_6 – C1: Erlang(30.69,16) D2_7 – C1: Erlang(12.99,16) D2_8 – C1: Erlang(11.06,16) D2_9 – C1: Erlang(11.22,16) D2_10 – C1: Erlang(57.31,16) D3_1 – C1: Erlang(49.28,16) D3_2 – C1: Erlang(8.398,16) D3_4 – C1: Erlang(22.22,16) D3_5 – C1: Erlang(13.13,16) D3_6 – C1: Erlang(9.378,16) D3_7 – C1: Erlang(8.699,16) D3_8 – C1: Erlang(9.018,16) D3_9 – C1: Erlang(8.866,16) D3_10 – C1: Erlang(44.06,16) D4_1 – C1: Erlang(8.192,16) D4_2 – C1: Erlang(21.39,16) D4_3 – C1: Erlang(10.73,16) D4_5 – C1: Erlang(8.551,16) D4_6 – C1: Erlang(25.38,16) D4_7 – C1: Erlang(8.839,16) D4_8 – C1: Erlang(31.93,16) D4_9 – C1: Erlang(8.042,16) D4_10 – C1: Erlang(11.6,16) D5_1 – C1: Erlang(7.85,16) D5_2 – C1: Erlang(8.118,16) D5_3 – C1: Erlang(23.15,16) D5_4 – C1: Erlang(25.67,16) D5_6 – C1: Erlang(49.16,16) D5_7 – C1: Erlang(30.24,16) D5_8 – C1: Erlang(19.2,16) D5_9 – C1: Erlang(28.72,16) D5_10 – C1: Erlang(75.6,16) D6_1 – C1: Erlang(38.85,16) D6_2 – C1: Erlang(16.65,16) D6_3 – C1: Erlang(10.61,16) D6_4 – C1: Erlang(8.27,16) D6_5 – C1: Erlang(8.789,16) D6_7 – C1: Erlang(12.43,16) D6_8 – C1: Erlang(7.298,16) D6_9 – C1: Erlang(22.42,16) D6_10 – C1: Erlang(22.36,16) D7_1 – C1: Erlang(34.98,16) D7_2 – C1: Erlang(15.36,16) D7_3 – C1: Erlang(14.39,16) D7_4 – C1: Erlang(8.513,16) D7_5 – C1: Erlang(17.2,16) D7_6 – C1: Erlang(17.1,16) D7_8 – C1: Erlang(26.38,16) D7_9 – C1: Erlang(19.32,16) D7_10 – C1: Erlang(11.82,16) D8_1 – C1: Erlang(8.977,16) D8_2 – C1: Erlang(11.48,16) D8_3 – C1: Erlang(10.13,16) D8_4 – C1: Erlang(45.18,16) D8_5 – C1: Erlang(25.57,16) D8_6 – C1: Erlang(13.2,16) D8_7 – C1: Erlang(20.25,16) D8_9 – C1: Erlang(12.79,16) D8_10 – C1: Erlang(7.512,16) D9_1 – C1: Erlang(8.448,16) D9_2 – C1: Erlang(9.597,16) D9_3 – C1: Erlang(16.24,16) D9_4 – C1: Erlang(8.5,16) D9_5 – C1: Erlang(7.444,16) D9_6 – C1: Erlang(10.08,16) D9_7 – C1: Erlang(19.56,16) D9_8 – C1: Erlang(7.685,16) D9_10 – C1: Erlang(7.351,16) D10_1 – C1: Erlang(10.79,16) D10_2 – C1: Erlang(16.61,16) D10_3 – C1: Erlang(68.54,16) D10_4 – C1: Erlang(7.465,16) D10_5 – C1: Erlang(9.432,16) D10_6 – C1: Erlang(25.87,16) D10_7 – C1: Erlang(68.48,16) D10_8 – C1: Erlang(7.279,16) D10_9 – C1: Erlang(8.43,16)

Table 3.78: Features of `largescale_tbi`.

Chapter 4

Model gallery

The gallery is a catalogue of small named models, one function each, returning a model rather than solving it. It exists twice over: as documentation, because a name in Kendall notation is the fastest way to find the model one means, and as the cross-codebase parity suite, because the same 71 functions exist in all four codebases and their metrics are compared numerically.

The models live in `r/linesolver/inst/examples/`, for the ported subset. Calling a gallery function builds and returns the model without solving it, so the caller chooses the solver. Running a Python gallery file as a script prints a short confirmation of the model it built.

4.1 Reading the names

The single-station entries are named in Kendall notation, with the distributions abbreviated: `m` exponential, `d` deterministic, `erl` Erlang, `hyp` hyperexponential, `aph` acyclic phase-type, `cox` Coxian, `me` matrix-exponential, `map` Markovian arrival process, `mmap` marked MAP, `gam` Gamma, `par` Pareto, `u` uniform. So `gallery_erlm1` is an Erl/M/1 queue and `gallery_mhypk` an M/HyperExp/k. A trailing `_ps` replaces FCFS by processor sharing, `k` in place of `1` makes the station multiserver, and the suffixes `_tandem`, `_linear`, `_reentrant`, `_feedback`, `_multiclass` and `_prio` describe the topology or the workload around the station.

The remaining entries are named after what they model rather than after a queueing notation: `gallery_cqn` and `gallery_repairmen` for the closed archetypes, `gallery_fj_open` and `gallery_fj_closed` for fork-join, `gallery_fcr` for a finite capacity region, `gallery_cache_lru` and `gallery_cache_routing` for caches, `gallery_renv_breakdown` for a random environment, `gallery_lqn_basic`, `gallery_lqn_workflows` and `gallery_lqn_random` for layered models, `gallery_multitier` and `gallery_multitier_storage` for multitier applications, `gallery_lukumar_reentrant` for the Lu-Kumar reentrant line, and `gallery_qn_random` for a generated network.

4.2 The catalogue

The complete list, in the order in which the models are registered:

```
gallery_aphm1
gallery_cache_lru
gallery_cache_routing
gallery_coxm1
gallery_cqn
gallery_cqn_multiclass
gallery_merl1_linear
gallery_merl1_reentrant
gallery_merl1_tandem
gallery_merlk
gallery_mhyp1
gallery_mhyp1_linear
```

```
gallery_detm1
gallery_dm1
gallery_erldk
gallery_erlerl1
gallery_erlerl1_reentrant
gallery_erlm1
gallery_erlm1_ps
gallery_erlm1_reentrant
gallery_erlm1ps
gallery_fcr
gallery_fj_closed
gallery_fj_open
gallery_fj_quorum
gallery_gamm1
gallery_hyperl1_feedback
gallery_hyperl1_reentrant
gallery_hyperlk
gallery_hyphyp1_linear
gallery_hyphyp1_reentrant
gallery_hyphyp1_tandem
gallery_hypm1
gallery_hypm1_reentrant
gallery_lqn_basic
gallery_lqn_random
gallery_lqn_workflows
gallery_lukumar_reentrant
gallery_mapm1
gallery_mapmk
gallery_mdk
gallery_merl1
gallery_mhyp1_reentrant
gallery_mhyp1_tandem
gallery_mhypk
gallery_mm1
gallery_mm1_feedback
gallery_mm1_linear
gallery_mm1_multiclass
gallery_mm1_prio
gallery_mm1_ps
gallery_mm1_ps_feedback
gallery_mm1_ps_multiclass
gallery_mm1_ps_reentrant
gallery_mm1_reentrant
gallery_mm1_tandem
gallery_mm1_tandem_multiclass
gallery_mm1k
gallery_mmap1
gallery_mmap1_multiclass
gallery_mmapk
gallery_mmk
gallery_mpar1
gallery_multitier
gallery_multitier_storage
gallery_parm1
gallery_qn_random
gallery_renv_breakdown
gallery_repairmen
gallery_replayerm1
gallery_um1
```

4.2.1 gallery_aphm1

The APH/M/1 queue: acyclic phase-type inter-arrival times, exponential service, and a single server.

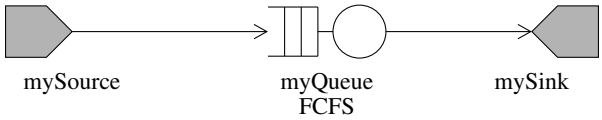


Figure 4.1: Topology of gallery_aphm1: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: APH
Service	myQueue – myClass: Exp(2)

Table 4.2: Features of gallery_aphm1.

4.2.2 gallery_cache_lru

A cache node under least-recently-used replacement, read by a closed class.

4.2.3 gallery_cache_routing

A cache whose hit and miss outcomes route the job to different downstream stations.

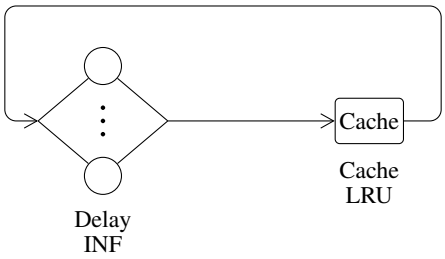


Figure 4.2: Topology of `gallery_cache_lru`: a closed network over 1 queueing station with a cache node.

Nodes	2: Delay, Cache
Classes	3: JobClass (closed, $N = 1$), HitClass (closed, $N = 0$), MissClass (closed, $N = 0$)
Scheduling	Delay: INF
Service	Delay – JobClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Features	drop rule {'HitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at Delay; cache: LRU replacement, 5 items, capacity 2; class switching on 2 links

Table 4.3: Features of `gallery_cache_lru`.

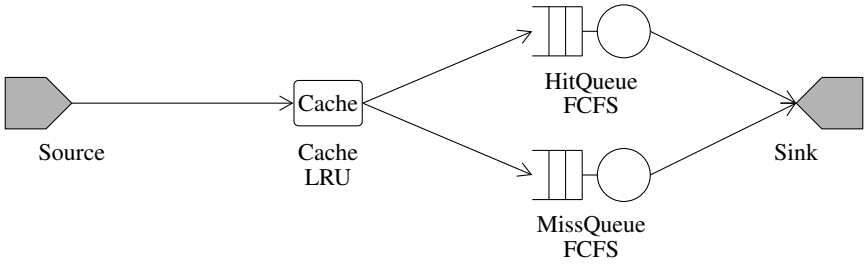


Figure 4.3: Topology of `gallery_cache_routing`: an open network over 2 queueing stations with a cache node.

Nodes	5: Source, Queue × 2, Cache, Sink
Classes	3: InitClass (open), HitClass (open), MissClass (open)
Scheduling	HitQueue: FCFS; MissQueue: FCFS
Arrivals	Source – InitClass: Exp(1); HitClass: Disabled; MissClass: Disabled
Service	HitQueue – InitClass: Disabled; HitClass: Exp(2); MissClass: Disabled MissQueue – InitClass: Disabled; HitClass: Disabled; MissClass: Exp(1)
Features	cache: LRU replacement, 4 items, capacity 2; drop rule {'InitClass': 'waitingQueue', 'MissClass': 'waitingQueue'} at HitQueue; drop rule {'InitClass': 'waitingQueue', 'HitClass': 'waitingQueue'} at MissQueue; class switching on 4 links

Table 4.4: Features of `gallery_cache_routing`.

4.2.4 gallery_coxm1

The Coxian/M/1 queue: Coxian inter-arrival times, exponential service, and a single server.

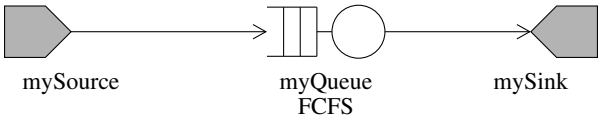


Figure 4.4: Topology of `gallery_coxm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Coxian
Service	myQueue – myClass: Exp(2)

Table 4.5: Features of `gallery_coxm1`.

4.2.5 gallery_cqn

The closed queueing network archetype: a fixed job population circulating between a Delay station and a queue.

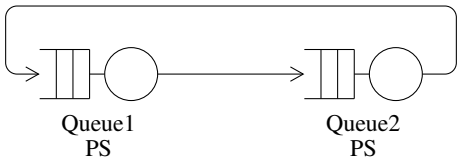


Figure 4.5: Topology of `gallery_cqn`: a closed network over 2 queueing stations.

4.2.6 gallery_cqn_multiclass

The closed archetype with several classes, each with its own population.

4.2.7 gallery_detm1

The D/M/1 queue: deterministic inter-arrival times, exponential service, and a single server.

4.2.8 gallery_dm1

The D/M/1 queue: deterministic inter-arrival times, exponential service, and a single server.

4.2.9 gallery_erldk

The Erl/D/k queue: Erlang inter-arrival times, deterministic service, and k parallel servers.

4.2.10 gallery_erlerl1

The Erl/Erl/1 queue: Erlang inter-arrival times, Erlang service, and a single server.

Nodes	2: Queue $\times$ 2
Classes	1: Class1 (closed, $N = 3$)
Scheduling	Queue1: PS; Queue2: PS
Service	Queue1 – Class1: Exp(0.525) Queue2 – Class1: Exp(0.3712)

Table 4.6: Features of `gallery_cqn`.

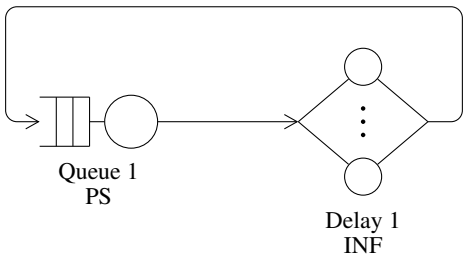


Figure 4.6: Topology of `gallery_cqn_multiclass`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 5$), Class2 (closed, $N = 5$)
Scheduling	Queue 1: PS; Delay 1: INF
Service	Queue 1 – Class1: Exp(0.03846); Class2: Exp(0.02857) Delay 1 – Class1: Exp(0.01111); Class2: Exp(0.01754)

Table 4.7: Features of `gallery_cqn_multiclass`.

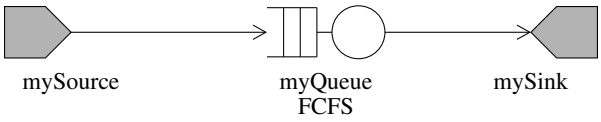


Figure 4.7: Topology of `gallery_detm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Det(1)
Service	myQueue – myClass: Exp(2)

Table 4.8: Features of `gallery_detm1`.

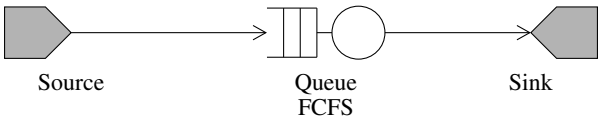


Figure 4.8: Topology of `gallery_dm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Det(1)
Service	Queue – Class1: Exp(2)

Table 4.9: Features of `gallery_dm1`.

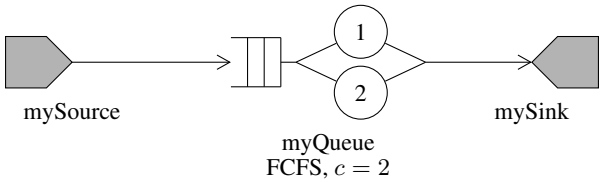


Figure 4.9: Topology of `gallery_erldk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Erlang(5,5)
Service	myQueue – myClass: Det(1)

Table 4.10: Features of `gallery_erldk`.

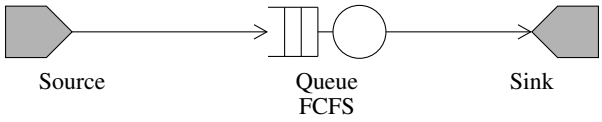


Figure 4.10: Topology of `gallery_erlerl1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Erlang(5,5); Class2: Disabled
Service	Queue – Class1: Erlang(10,5); Class2: Exp(3)
Features	class switching on 2 links

Table 4.11: Features of `gallery_erlerl1`.

4.2.11 gallery_erlerl1_reentrant

An Erl/Erl/1 queue with reentrant visits.

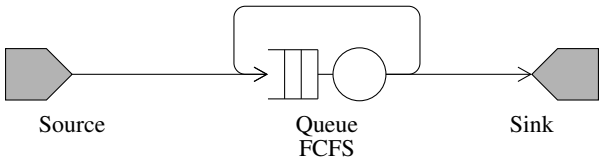


Figure 4.11: Topology of `gallery_erlerl1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Erlang(5,5); Class2: Disabled
Service	Queue – Class1: Erlang(50,5); Class2: Exp(10)
Features	class switching on 1 link

Table 4.12: Features of `gallery_erlerl1_reentrant`.

4.2.12 gallery_erlm1

The Erl/M/1 queue: Erlang inter-arrival times, exponential service, and a single server.

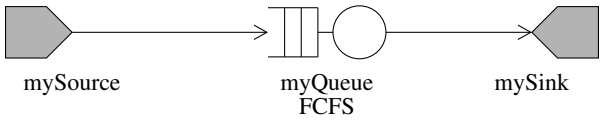


Figure 4.12: Topology of `gallery_erlm1`: an open network over 1 queueing station.

4.2.13 gallery_erlm1_ps

The Erl/M/1 queue: Erlang inter-arrival times, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS.

4.2.14 gallery_erlm1_reentrant

The Erl/M/1 queue: Erlang inter-arrival times, exponential service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

4.2.15 gallery_erlm1ps

The Erl/M/1 queue: Erlang inter-arrival times, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS.

4.2.16 gallery_fcr

A finite capacity region, which caps the number of jobs admitted to a set of stations irrespective of their individual buffers.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Erlang(5,5)
Service	myQueue – myClass: Exp(2)

Table 4.13: Features of `gallery_erlm1`.

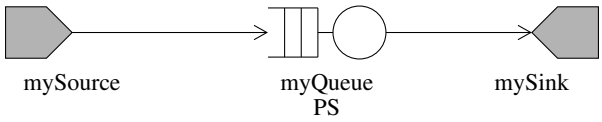


Figure 4.13: Topology of `gallery_erlm1_ps`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: PS
Arrivals	mySource – myClass: Erlang(5,5)
Service	myQueue – myClass: Exp(2)

Table 4.14: Features of `gallery_erlm1_ps`.

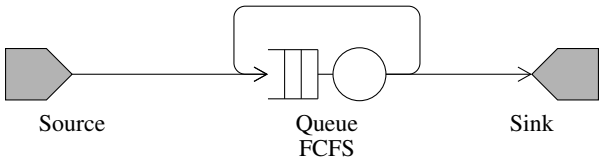


Figure 4.14: Topology of `gallery_erlm1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Erlang(5,5); Class2: Disabled
Service	Queue – Class1: Exp(2); Class2: Exp(3)
Features	class switching on 1 link

Table 4.15: Features of `gallery_erlm1_reentrant`.

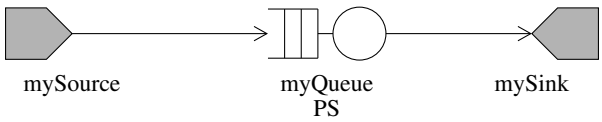


Figure 4.15: Topology of `gallery_erlm1ps`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: PS
Arrivals	mySource – myClass: Erlang(5,5)
Service	myQueue – myClass: Exp(2)

Table 4.16: Features of `gallery_erlm1ps`.

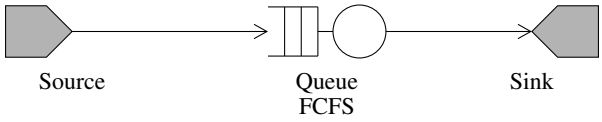


Figure 4.16: Topology of `gallery_fcr`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(0.8)
Service	Queue – Class1: Exp(1)

Table 4.17: Features of `gallery_fcr`.

4.2.17 `gallery_fj_closed`

The closed fork-join counterpart, with a fixed population circulating through the block.

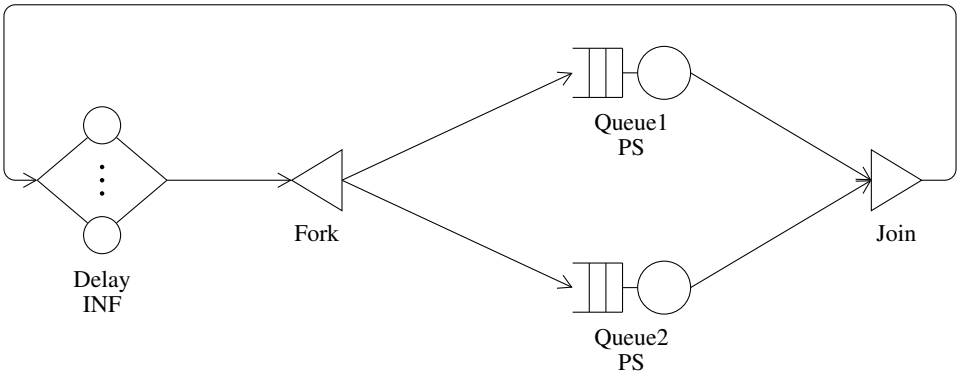


Figure 4.17: Topology of `gallery_fj_closed`: a closed network over 3 queueing stations with a fork-join block.

4.2.18 `gallery_fj_open`

An open fork-join block: a job forks into parallel tasks and its siblings are joined before departure.

4.2.19 `gallery_fj_quorum`

A fork-join block whose join fires on a quorum of the siblings, the later ones being discarded when they arrive.

4.2.20 `gallery_gamm1`

The Gamma/M/1 queue: Gamma inter-arrival times, exponential service, and a single server.

4.2.21 `gallery_hyperl1_feedback`

A HyperExp/Erl/1 queue with probabilistic feedback.

Nodes	5: Queue × 2, Delay, Fork, Join
Classes	1: class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS
Service	Delay – class1: Exp(1) Queue1 – class1: Exp(1) Queue2 – class1: Exp(1)

Table 4.18: Features of `gallery_fj_closed`.

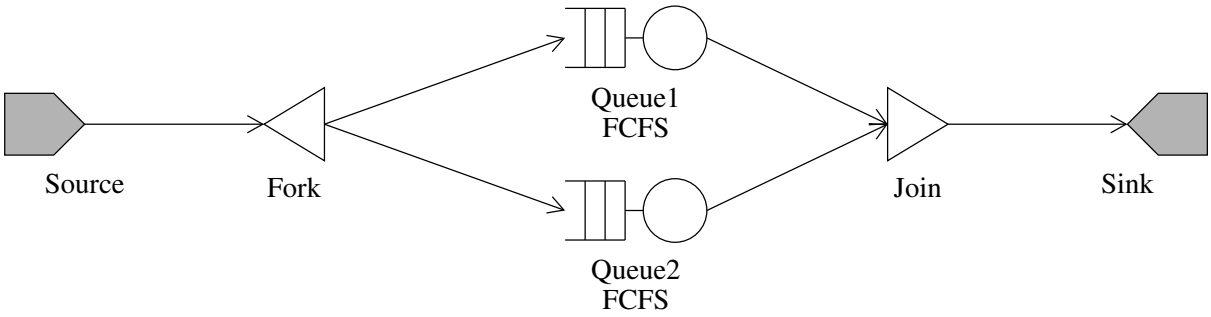


Figure 4.18: Topology of `gallery_fj_open`: an open network over 2 queueing stations with a fork-join block.

Nodes	6: Source, Queue × 2, Fork, Join, Sink
Classes	1: class1 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – class1: Exp(0.05)
Service	Queue1 – class1: Exp(1) Queue2 – class1: Exp(2)

Table 4.19: Features of `gallery_fj_open`.

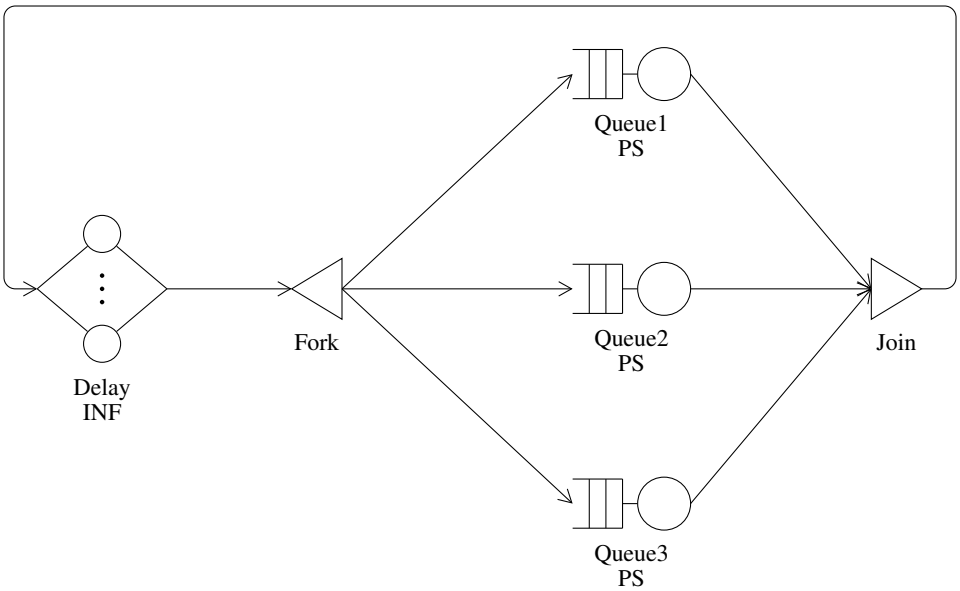


Figure 4.19: Topology of `gallery_fj_quorum`: a closed network over 4 queueing stations with a fork-join block.

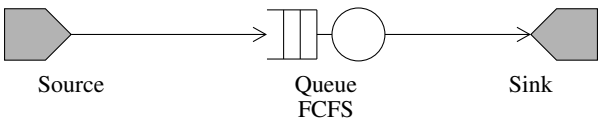


Figure 4.20: Topology of `gallery_gamm1`: an open network over 1 queueing station.

Nodes	6: Queue × 3, Delay, Fork, Join
Classes	1: class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: PS; Queue2: PS; Queue3: PS
Service	Delay – class1: Exp(1) Queue1 – class1: Exp(2) Queue2 – class1: Exp(2) Queue3 – class1: Exp(2)

Table 4.20: Features of `gallery_fj_quorum`.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Gamma(5,0.2)
Service	Queue – Class1: Exp(2)

Table 4.21: Features of `gallery_gamm1`.

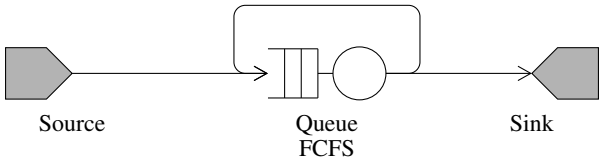


Figure 4.21: Topology of `gallery_hyperl1_feedback`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: HyperExp(0.99,0.01;2.294,0.01759)
Service	Queue – Class1: Erlang(100,5)

Table 4.22: Features of `gallery_hyperl1_feedback`.

4.2.22 gallery_hyperl1_reentrant

The HyperExp/Erl/1 queue: hyperexponential inter-arrival times, Erlang service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

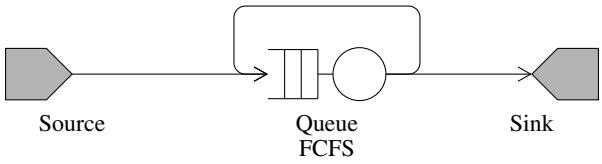


Figure 4.22: Topology of `gallery_hyperl1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: HyperExp(0.99,0.01;2.294,0.01759); Class2: Disabled
Service	Queue – Class1: Erlang(10,5); Class2: Exp(3)
Features	class switching on 1 link

Table 4.23: Features of `gallery_hyperl1_reentrant`.

4.2.23 gallery_hyperlk

The HyperExp/Erl/k queue: hyperexponential inter-arrival times, Erlang service, and k parallel servers.

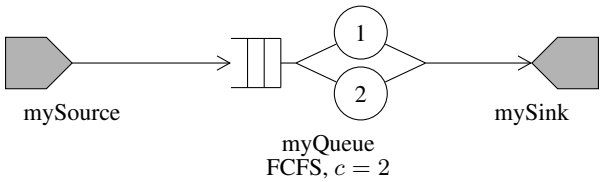


Figure 4.23: Topology of `gallery_hyperlk`: an open network over 1 queueing station.

4.2.24 gallery_hyphyp1_linear

The HyperExp/HyperExp/1 queue: hyperexponential inter-arrival times, hyperexponential service, and a single server. In this variant, the model is a linear chain of such stations.

4.2.25 gallery_hyphyp1_reentrant

The HyperExp/HyperExp/1 queue: hyperexponential inter-arrival times, hyperexponential service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

4.2.26 gallery_hyphyp1_tandem

Two HyperExp/HyperExp/1 queues in series.

4.2.27 gallery_hypm1

The HyperExp/M/1 queue: hyperexponential inter-arrival times, exponential service, and a single server.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: HyperExp(0.1127,0.8873;0.4057,3.194)
Service	myQueue – myClass: Erlang(4,4)

Table 4.24: Features of `gallery_hyperlk`.

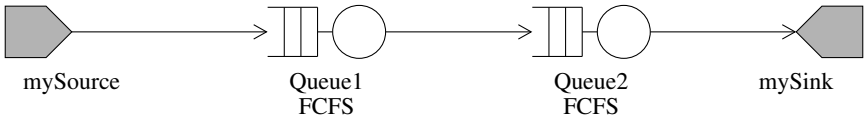


Figure 4.24: Topology of `gallery_hyphyp1_linear`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: HyperExp(0.99,0.01;1.077,0.1244)
Service	Queue1 – myClass: HyperExp(0.99,0.01;1.196,0.1383) Queue2 – myClass: HyperExp(0.99,0.01;1.235,0.1015)

Table 4.25: Features of `gallery_hyphyp1_linear`.

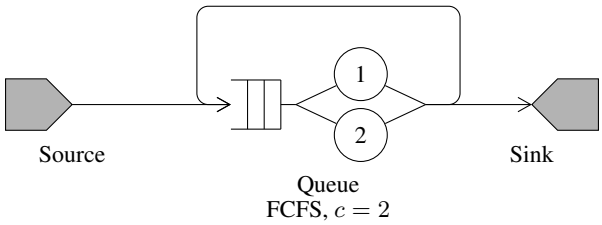


Figure 4.25: Topology of `gallery_hyphyp1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS ($c = 2$)
Arrivals	Source – Class1: HyperExp(0.99,0.01;2.294,0.01759); Class2: Disabled
Service	Queue – Class1: HyperExp(0.99,0.01;2.281,0.1517); Class2: Exp(3)
Features	class switching on 1 link

Table 4.26: Features of `gallery_hyphyp1_reentrant`.

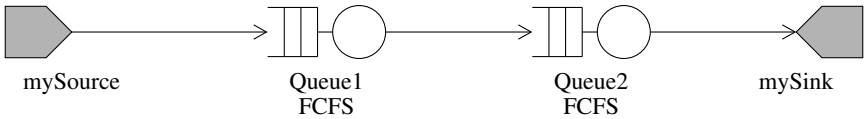


Figure 4.26: Topology of `gallery_hyphyp1_tandem`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: HyperExp(0.99,0.01;1.077,0.1244)
Service	Queue1 – myClass: HyperExp(0.99,0.01;1.196,0.1383) Queue2 – myClass: HyperExp(0.99,0.01;1.235,0.1015)

Table 4.27: Features of `gallery_hyphyp1_tandem`.

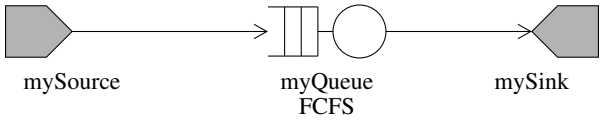


Figure 4.27: Topology of `gallery_hypm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: HyperExp(0.99,0.01;2.294,0.01759)
Service	myQueue – myClass: Exp(2)

Table 4.28: Features of `gallery_hypm1`.

4.2.28 `gallery_hypm1_reentrant`

The HyperExp/M/1 queue: hyperexponential inter-arrival times, exponential service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

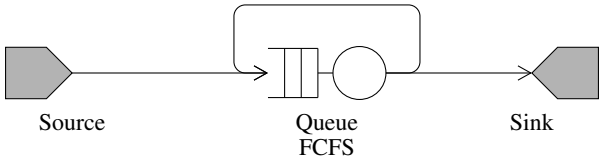


Figure 4.28: Topology of `gallery_hypm1_reentrant`: an open network over 1 queueing station.

4.2.29 `gallery_lqn_basic`

The smallest layered queueing network: one reference task calling one server task.

4.2.30 `gallery_lqn_random`

A randomly generated layered model, used to exercise the layered solvers beyond handwritten topologies.

4.2.31 `gallery_lqn_workflows`

Workflow patterns (sequence, AND-fork and join, loop) as an activity graph in a layered model.

4.2.32 `gallery_lukumar_reentrant`

The Lu-Kumar reentrant line, the standard counterexample in which a work-conserving policy destabilizes a network whose stations are individually underloaded.

4.2.33 `gallery_mapm1`

The MAP/M/1 queue: a Markovian arrival process, exponential service, and a single server.

4.2.34 `gallery_mapmk`

The MAP/M/k queue: a Markovian arrival process, exponential service, and k parallel servers.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: HyperExp(0.99,0.01;1.14,0.07584); Class2: Disabled
Service	Queue – Class1: Exp(2); Class2: Exp(3)
Features	class switching on 1 link

Table 4.29: Features of `gallery_hypm1_reentrant`.

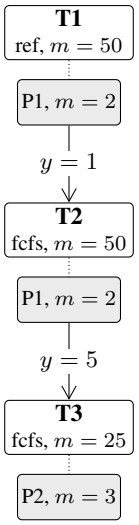


Figure 4.29: Call graph of `gallery_lqn_basic`: 3 tasks over 2 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

Processors	2: P1 (ps, $m = 2$), P2 (ps, $m = 3$)
Tasks	3: T1 (ref, $m = 50$), T2 (fcfs, $m = 50$), T3 (fcfs, $m = 25$)
Entries	3: E1, E2, E3
Activities	3, host demands AS1: 0.1, AS2: 0.05, AS3: 0.02
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 2.0

Table 4.30: Features of `gallery_lqn_basic`.

Processors	3: c_processor_1 (inf, $m = 1$), processor_1 (ps, $m = 1$), processor_2 (ps, $m = 1$)
Tasks	5: c_task_1 (ref, $m = 1$), task_1 (fcfs, $m = 1$), task_2 (fcfs, $m = 1$), task_3 (fcfs, $m = 1$), task_4 (fcfs, $m = 1$)
Entries	5: c_entry_1, entry_1, entry_2, entry_3, entry_4
Activities	5, host demands c_activity_1: 1.0, activity_1: 1.0, activity_2: 1.0, activity_3: 1.0, activity_4: 1.0
Calls	4 (4 synchronous, 0 asynchronous)

Table 4.31: Features of `gallery_lqn_random`.

Processors	3: P1 (inf, $m = 1$), P2 (inf, $m = 1$), P3 (ps, $m = 5$)
Tasks	3: T1 (ref, $m = 1$), T2 (inf, $m = 1$), T3 (inf, $m = 1$)
Entries	3: Entry, E2, E3
Activities	14, host demands A1: 1.0, A2: 2.0, A3: 3.0, B1: 0.1, B2: 0.2, B3: 0.3, B4: 0.4, B5: 0.5, ...
Calls	2 (2 synchronous, 0 asynchronous)
Think times	T1: 0.0

Table 4.32: Features of `gallery_lqn_workflows`.

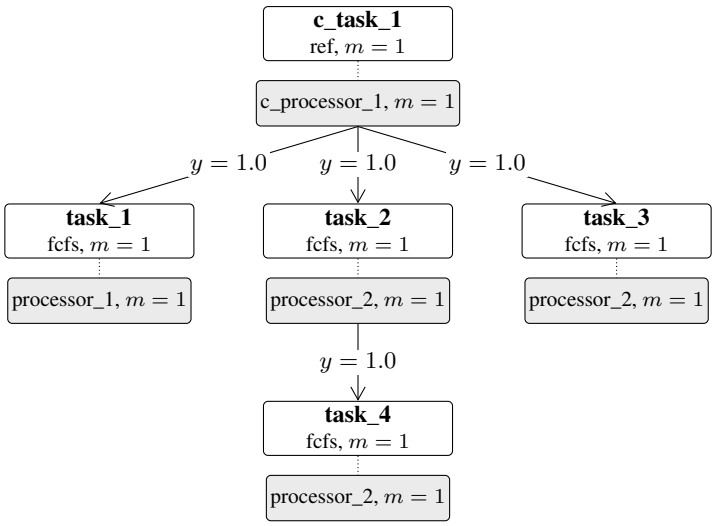


Figure 4.30: Call graph of `gallery_lqn_random`: 5 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

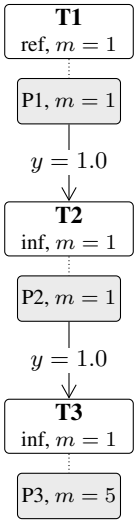


Figure 4.31: Call graph of `gallery_lqn_workflows`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

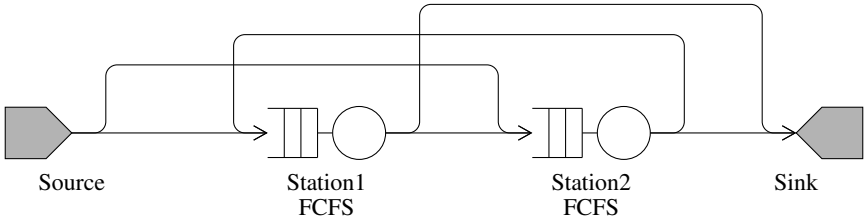


Figure 4.32: Topology of `gallery_lukumar_reentrant`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	4: Class1 (open), Class2 (open), Class3 (open), Class4 (open)
Scheduling	Station1: FCFS; Station2: FCFS
Arrivals	Source – Class1: Exp(0.08); Class2: Disabled; Class3: Exp(0.08); Class4: Disabled
Service	Station1 – Class1: Exp(0.1); Class2: Disabled; Class3: Disabled; Class4: Exp(1) Station2 – Class1: Disabled; Class2: Exp(1); Class3: Exp(0.1); Class4: Disabled
Features	class switching on 1 link

Table 4.33: Features of `gallery_lukumar_reentrant`.

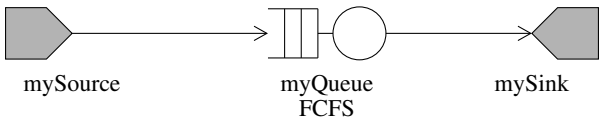


Figure 4.33: Topology of `gallery_mapm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: MAP
Service	myQueue – myClass: Exp(2)

Table 4.34: Features of `gallery_mapm1`.

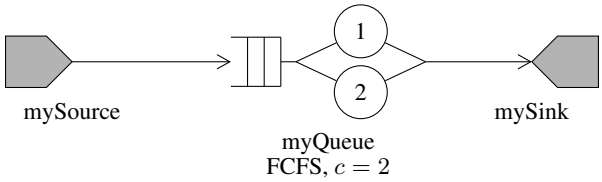


Figure 4.34: Topology of `gallery_mapmk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: MAP
Service	myQueue – myClass: Exp(2)

Table 4.35: Features of `gallery_mapmk`.

4.2.35 gallery_mdk

The M/D/k queue: Poisson arrivals, deterministic service, and k parallel servers.

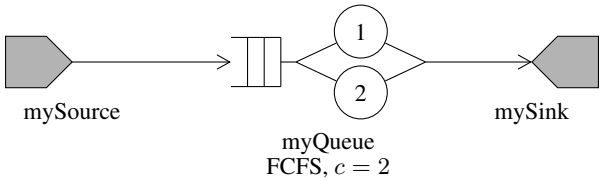


Figure 4.35: Topology of `gallery_mdk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Det(1)

Table 4.36: Features of `gallery_mdk`.

4.2.36 gallery_merl1

The M/Erl/1 queue: Poisson arrivals, Erlang service, and a single server.

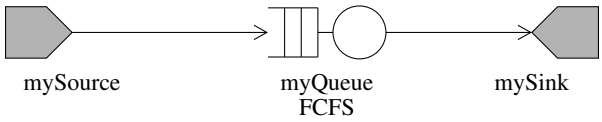


Figure 4.36: Topology of `gallery_merl1`: an open network over 1 queueing station.

4.2.37 gallery_merl1_linear

The M/Erl/1 queue: Poisson arrivals, Erlang service, and a single server. In this variant, the model is a linear chain of such stations.

4.2.38 gallery_merl1_reentrant

The M/Erl/1 queue: Poisson arrivals, Erlang service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

4.2.39 gallery_merl1_tandem

Two M/Erl/1 queues in series.

4.2.40 gallery_merlk

The M/Erl/k queue: Poisson arrivals, Erlang service, and k parallel servers.

4.2.41 gallery_mhyp1

The M/HyperExp/1 queue: Poisson arrivals, hyperexponential service, and a single server.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Erlang(4,2)

Table 4.37: Features of `gallery_merl1`.

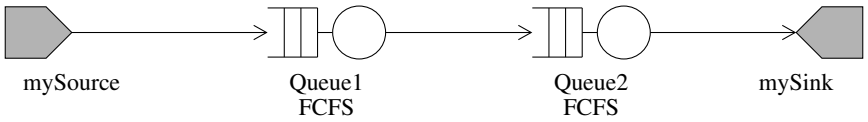


Figure 4.37: Topology of `gallery_merl1_linear`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: Erlang(1.111,1) Queue2 – myClass: Erlang(2.222,2)

Table 4.38: Features of `gallery_merl1_linear`.

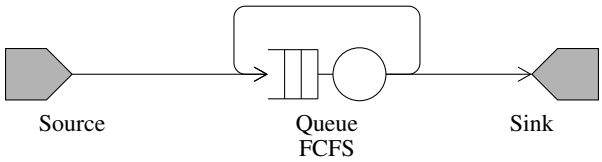


Figure 4.38: Topology of `gallery_merl1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Disabled
Service	Queue – Class1: Erlang(10,5); Class2: Exp(3)
Features	class switching on 1 link

Table 4.39: Features of `gallery_merl1_reentrant`.

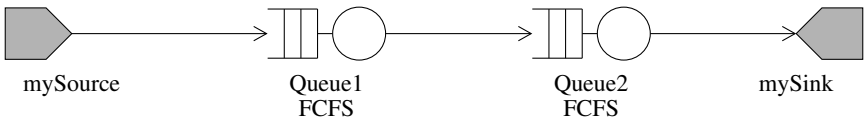


Figure 4.39: Topology of `gallery_merl1_tandem`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: Erlang(1.111,1) Queue2 – myClass: Erlang(2.222,2)

Table 4.40: Features of `gallery_merl1_tandem`.

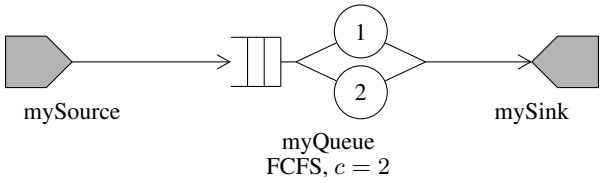


Figure 4.40: Topology of `gallery_merlk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Erlang(4,2)

Table 4.41: Features of `gallery_merlk`.

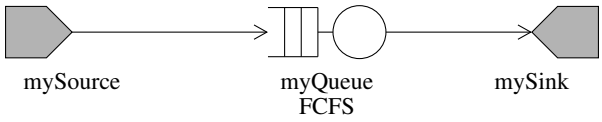


Figure 4.41: Topology of `gallery_mhyp1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: HyperExp(0.99,0.01;2.281,0.1517)

Table 4.42: Features of `gallery_mhyp1`.

4.2.42 gallery_mhyp1_linear

The M/HyperExp/1 queue: Poisson arrivals, hyperexponential service, and a single server. In this variant, the model is a linear chain of such stations.

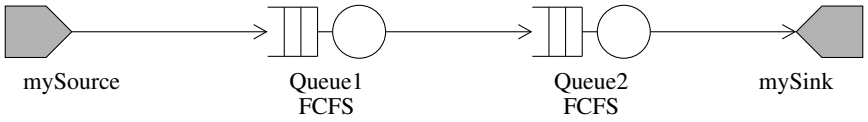


Figure 4.42: Topology of gallery_mhyp1_linear: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: HyperExp(0.99,0.01;1.196,0.1383) Queue2 – myClass: HyperExp(0.99,0.01;1.196,0.1383)

Table 4.43: Features of gallery_mhyp1_linear.

4.2.43 gallery_mhyp1_reentrant

The M/HyperExp/1 queue: Poisson arrivals, hyperexponential service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

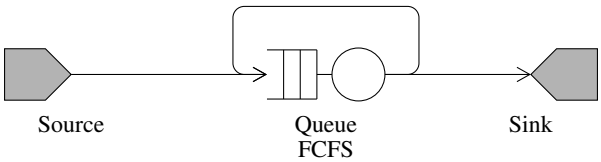


Figure 4.43: Topology of gallery_mhyp1_reentrant: an open network over 1 queueing station.

4.2.44 gallery_mhyp1_tandem

Two M/HyperExp/1 queues in series.

4.2.45 gallery_mhypk

The M/HyperExp/k queue: Poisson arrivals, hyperexponential service, and k parallel servers.

4.2.46 gallery_mm1

The M/M/1 queue: Poisson arrivals, exponential service, and a single server.

4.2.47 gallery_mm1_feedback

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, a completing job returns to the station with fixed probability.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Disabled
Service	Queue – Class1: Coxian; Class2: Exp(3)
Features	class switching on 1 link

Table 4.44: Features of `gallery_mhyp1_reentrant`.

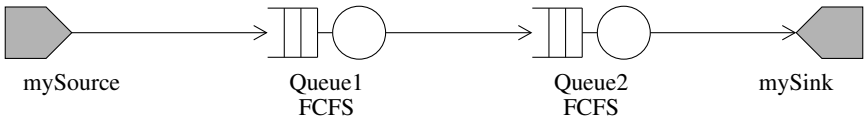


Figure 4.44: Topology of `gallery_mhyp1_tandem`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: HyperExp(0.99,0.01;1.196,0.1383) Queue2 – myClass: HyperExp(0.99,0.01;1.196,0.1383)

Table 4.45: Features of `gallery_mhyp1_tandem`.

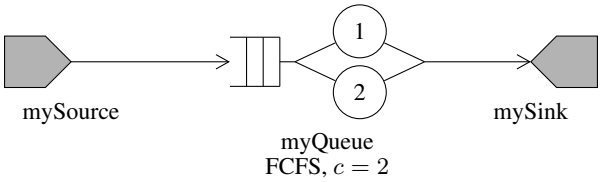


Figure 4.45: Topology of `gallery_mhypk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Coxian

Table 4.46: Features of `gallery_mhypk`.

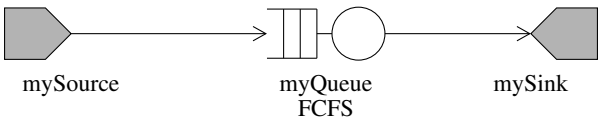


Figure 4.46: Topology of `gallery_mm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Exp(2)

Table 4.47: Features of `gallery_mm1`.

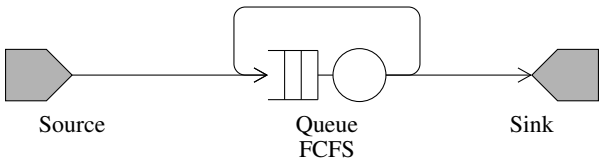


Figure 4.47: Topology of `gallery_mm1_feedback`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Queue – Class1: Exp(2)

Table 4.48: Features of `gallery_mm1_feedback`.

4.2.48 `gallery_mm1_linear`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the model is a linear chain of such stations.

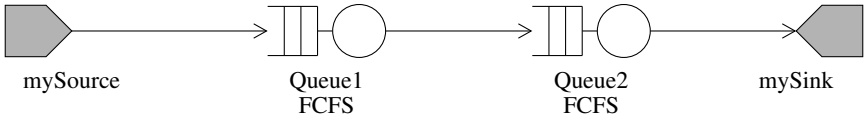


Figure 4.48: Topology of `gallery_mm1_linear`: an open network over 2 queueing stations.

4.2.49 `gallery_mm1_multiclass`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, several open classes share the station.

4.2.50 `gallery_mm1_prio`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the classes carry priorities, served head-of-line.

4.2.51 `gallery_mm1_ps`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS.

4.2.52 `gallery_mm1_ps_feedback`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS and a completing job returns to the station with fixed probability.

4.2.53 `gallery_mm1_ps_multiclass`

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS and several open classes share the station.

Nodes	4: Source, Queue × 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: Exp(1.111) Queue2 – myClass: Exp(1.111)

Table 4.49: Features of `gallery_mm1_linear`.

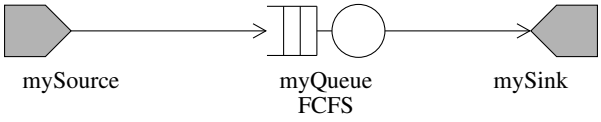


Figure 4.49: Topology of `gallery_mm1_multiclass`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass1: Exp(1); myClass2: Exp(0.5)
Service	myQueue – myClass1: Exp(4); myClass2: Exp(4)

Table 4.50: Features of `gallery_mm1_multiclass`.

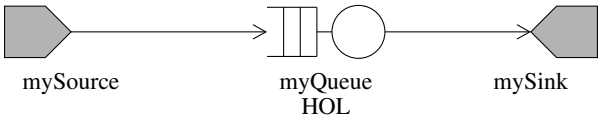


Figure 4.50: Topology of `gallery_mm1_prio`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: HOL
Arrivals	mySource – myClass1: Exp(1); myClass2: Exp(0.5)
Service	myQueue – myClass1: Exp(4); myClass2: Exp(4)

Table 4.51: Features of `gallery_mm1_prio`.

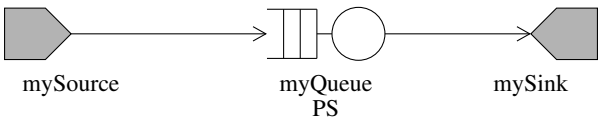


Figure 4.51: Topology of `gallery_mm1_ps`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: PS
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Exp(2)

Table 4.52: Features of `gallery_mm1_ps`.

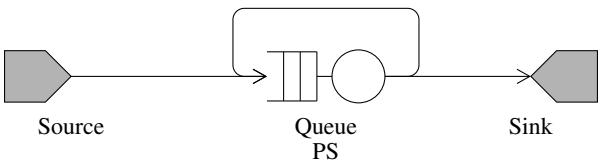


Figure 4.52: Topology of `gallery_mm1_ps_feedback`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: PS
Arrivals	Source – Class1: Exp(1)
Service	Queue – Class1: Exp(2)

Table 4.53: Features of `gallery_mm1_ps_feedback`.

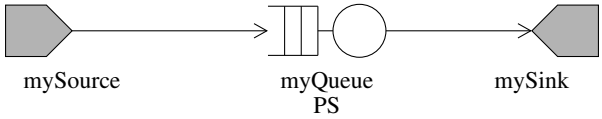


Figure 4.53: Topology of `gallery_mm1_ps_multiclass`: an open network over 1 queueing station.

4.2.54 gallery_mm1_ps_reentrant

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, the station serves under processor sharing rather than FCFS and jobs reenter the station, so a visit is counted more than once.

4.2.55 gallery_mm1_reentrant

The M/M/1 queue: Poisson arrivals, exponential service, and a single server. In this variant, jobs reenter the station, so a visit is counted more than once.

4.2.56 gallery_mm1_tandem

Two M/M/1 queues in series.

4.2.57 gallery_mm1_tandem_multiclass

Two M/M/1 queues in series under a multiclass workload.

4.2.58 gallery_mm1k

The M/M/1/K queue: a single exponential server behind a buffer of K places, so an arrival finding the buffer full is lost.

4.2.59 gallery_mmap1

The M/MAP/1 queue: Poisson arrivals, MAP-modulated service, and a single server.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: PS
Arrivals	mySource – myClass1: Exp(1); myClass2: Exp(0.5)
Service	myQueue – myClass1: Exp(4); myClass2: Exp(4)

Table 4.54: Features of `gallery_mm1_ps_multiclass`.

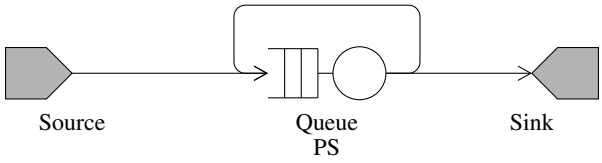


Figure 4.54: Topology of `gallery_mm1_ps_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: PS
Arrivals	Source – Class1: Exp(1); Class2: Disabled
Service	Queue – Class1: Exp(2); Class2: Exp(3)
Features	class switching on 1 link

Table 4.55: Features of `gallery_mm1_ps_reentrant`.

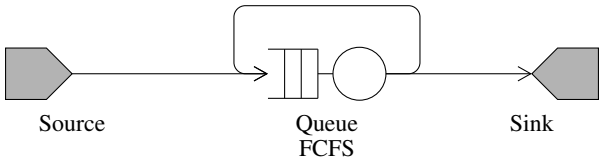


Figure 4.55: Topology of `gallery_mm1_reentrant`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Disabled
Service	Queue – Class1: Exp(2); Class2: Exp(3)
Features	class switching on 1 link

Table 4.56: Features of `gallery_mm1_reentrant`.

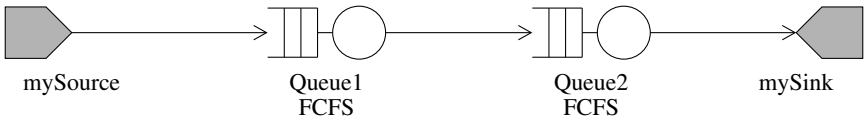


Figure 4.56: Topology of `gallery_mm1_tandem`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: myClass (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	Queue1 – myClass: Exp(1.111) Queue2 – myClass: Exp(1.111)

Table 4.57: Features of `gallery_mm1_tandem`.

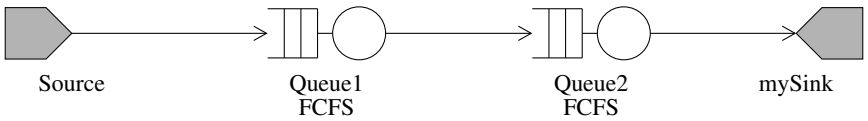


Figure 4.57: Topology of `gallery_mm1_tandem_multiclass`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	Queue1: FCFS; Queue2: FCFS
Arrivals	Source – myClass1: Exp(1); myClass2: Exp(0.5)
Service	Queue1 – myClass1: Exp(4); myClass2: Exp(2) Queue2 – myClass1: Exp(6); myClass2: Exp(6)

Table 4.58: Features of `gallery_mm1_tandem_multiclass`.

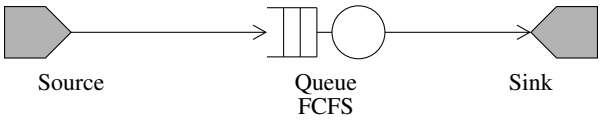


Figure 4.58: Topology of `gallery_mm1k`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(0.8)
Service	Queue – Class1: Exp(1)
Features	finite buffer 3 at Queue

Table 4.59: Features of `gallery_mm1k`.

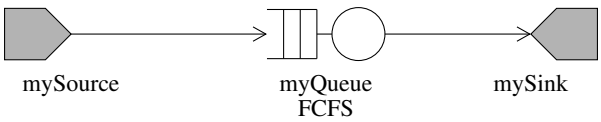


Figure 4.59: Topology of `gallery_mmap1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: MAP

Table 4.60: Features of `gallery_mmap1`.

4.2.60 gallery_mmap1_multiclass

The M/MAP/1 queue: Poisson arrivals, MAP-modulated service, and a single server. In this variant, several open classes share the station.

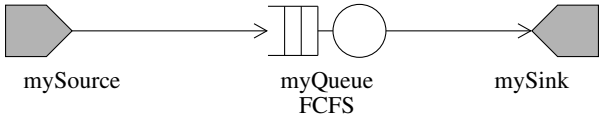


Figure 4.60: Topology of `gallery_mmap1_multiclass`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: myClass1 (open), myClass2 (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass1: Exp(0.7); myClass2: Exp(0.3)
Service	myQueue – myClass1: MAP; myClass2: MAP

Table 4.61: Features of `gallery_mmap1_multiclass`.

4.2.61 gallery_mmapk

The M/MAP/k queue: Poisson arrivals, MAP-modulated service, and k parallel servers.

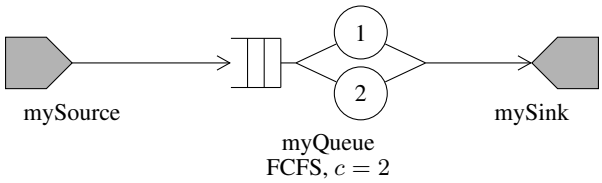


Figure 4.61: Topology of `gallery_mmapk`: an open network over 1 queueing station.

4.2.62 gallery_mmk

The M/M/k queue: Poisson arrivals, exponential service, and k parallel servers.

4.2.63 gallery_mpar1

The M/Pareto/1 queue: Poisson arrivals, Pareto service, and a single server.

4.2.64 gallery_multitier

A multitier application: presentation, application and database tiers as a closed network.

4.2.65 gallery_multitier_storage

The multitier application with an explicit storage tier behind the database.

4.2.66 gallery_parm1

The Pareto/M/1 queue: Pareto inter-arrival times, exponential service, and a single server.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: MAP

Table 4.62: Features of `gallery_mmapk`.

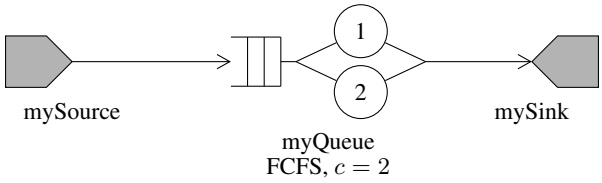


Figure 4.62: Topology of `gallery_mmk`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS ($c = 2$)
Arrivals	mySource – myClass: Exp(1)
Service	myQueue – myClass: Exp(2)

Table 4.63: Features of `gallery_mmk`.

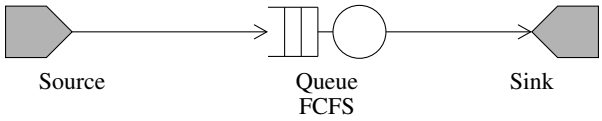


Figure 4.63: Topology of `gallery_mpar1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Queue – Class1: Pareto(2.008,0.251)

Table 4.64: Features of `gallery_mpar1`.

Processors	3: P0 (ps, $m = 1$), P1 (ps, $m = 1$), P2 (ps, $m = 1$)
Tasks	3: T0 (ref, $m = 1$), T1 (fcfs, $m = 1$), T2 (fcfs, $m = 1$)
Entries	9: E0, E10, E11, E12, E13, E20, E21, E22, E23
Activities	22, host demands A0: 1.0, A1: 1.0, A2: 1.0, A3: 1.0, B0: 1.0, B1: 1.0, B2: 1.0, B3: 1.0, ...
Calls	8 (8 synchronous, 0 asynchronous)

Table 4.65: Features of `gallery_multitier`.

Processors	4: P0 (ps, $m = 1$), P1 (ps, $m = 1$), P2 (ps, $m = 1$), P3 (ps, $m = 1$)
Tasks	4: T0 (ref, $m = 1$), T1 (fcfs, $m = 1$), T2 (fcfs, $m = 1$), T3 (fcfs, $m = 1$)
Entries	10: E0, E10, E11, E12, E13, E20, E21, E22, E23, E3
Activities	24, host demands A0: 1.0, A1: 1.0, A2: 1.0, A3: 1.0, B0: 1.0, B1: 1.0, B2: 1.0, B3: 1.0, ...
Calls	10 (10 synchronous, 0 asynchronous)

Table 4.66: Features of `gallery_multitier_storage`.

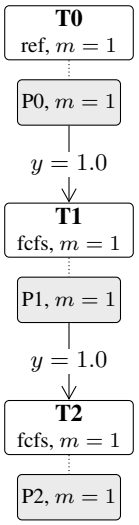


Figure 4.64: Call graph of `gallery_multitier`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

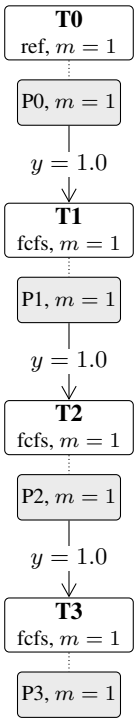


Figure 4.65: Call graph of `gallery_multitier_storage`: 4 tasks over 4 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

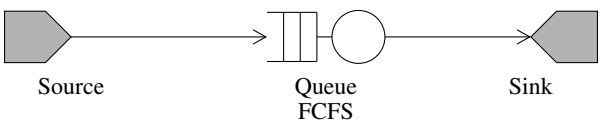


Figure 4.66: Topology of `gallery_parm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Pareto(2.008,0.5019)
Service	Queue – Class1: Exp(2)

Table 4.67: Features of `gallery_parm1`.

4.2.67 `gallery_qn_random`

A randomly generated queueing network, used as a generator-driven parity fixture.

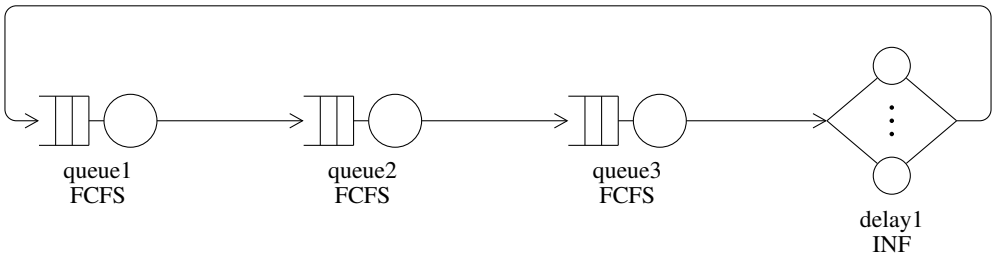


Figure 4.67: Topology of `gallery_qn_random`: a closed network over 4 queueing stations.

Nodes	4: Queue $\times$ 3, Delay
Classes	2: CClass1 (closed, $N = 20$), CClass2 (closed, $N = 17$)
Scheduling	queue1: FCFS; queue2: FCFS; queue3: FCFS; delay1: INF
Service	queue1 – CClass1: Exp(1); CClass2: Exp(1) queue2 – CClass1: Exp(1); CClass2: Exp(1) queue3 – CClass1: Exp(1); CClass2: Exp(1) delay1 – CClass1: Exp(1); CClass2: Exp(1)

Table 4.68: Features of `gallery_qn_random`.

4.2.68 `gallery_renv_breakdown`

A station in a random environment that alternates between an operational and a failed stage.
No topology is drawn for this example: the captured run yielded no `Network` or `LayeredNetwork` to draw one from, either because the script builds a model of another kind (an environment or a workflow) or because it builds it inside a helper it does not expose.

4.2.69 `gallery_repairmen`

The machine repairman model: machines think, fail, and queue for one repair station.

4.2.70 `gallery_replayerm1`

A single queue whose inter-arrival times are replayed from a measurement trace.

4.2.71 `gallery_um1`

The U/M/1 queue: uniform inter-arrival times, exponential service, and a single server.

Each of these is also a parity fixture: the four codebases build the same model and the harness compares the metrics solver by solver, so a divergence in any one of them is caught here before it reaches a larger model.

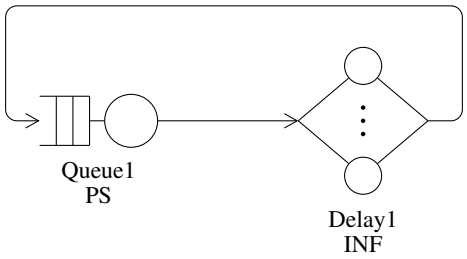


Figure 4.68: Topology of `gallery_repairmen`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 3$)
Scheduling	Queue1: PS; Delay1: INF
Service	Queue1 – Class1: Exp(0.5774) Delay1 – Class1: Exp(0.5)

Table 4.69: Features of `gallery_repairmen`.

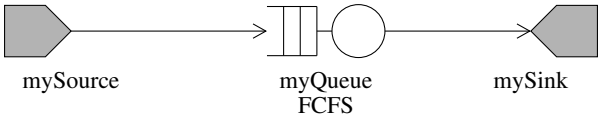


Figure 4.69: Topology of `gallery_replayerm1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: myClass (open)
Scheduling	myQueue: FCFS
Arrivals	mySource – myClass: Replayer(trace)
Service	myQueue – myClass: Exp(29.68)

Table 4.70: Features of `gallery_replayerm1`.

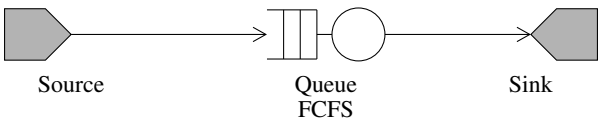


Figure 4.70: Topology of `gallery_um1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Uniform(1,2)
Service	Queue – Class1: Exp(2)

Table 4.71: Features of `gallery_um1`.

Chapter 5

Discrete-time models

In a slotted model time advances in slots of fixed length, and every sampled inter-arrival and service time must fall on the slot lattice. A sample that does not is an error rather than something to round: refusing is what keeps a discrete-time result exact. Within a slot the events are ordered by a fixed phase – completions, internal movement, arrivals, bookkeeping – so that a departure and an arrival in the same slot cannot be confused.

Geometric inter-arrival and service times give the Geo/Geo/1 queue, whose exact closed form is known, and it is against that form that the solvers are checked. The models live under no R twin; load the document and solve it, and slotted mode is requested through the solver options rather than by the model.

5.1 The slotted models

5.1.1 `dt_bernoulli_loaddep`

A load-dependent Bernoulli server, $p(n) = s\min(n, c)$, the discrete-time multiserver approximation.

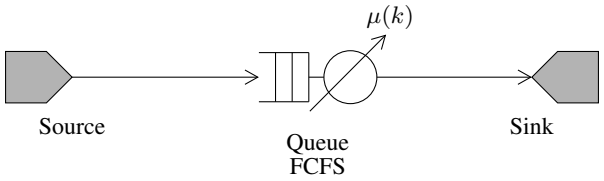


Figure 5.1: Topology of `dt_bernoulli_loaddep`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Geometric(0.6)
Service	Queue – Class1: Geometric(0.3)
Features	finite buffer 20 at Queue; load-dependent rates at Queue
Solvers	NC

Table 5.1: Features of `dt_bernoulli_loaddep`.

5.1.2 `dt_cycle`

A closed cycle of slotted stations.

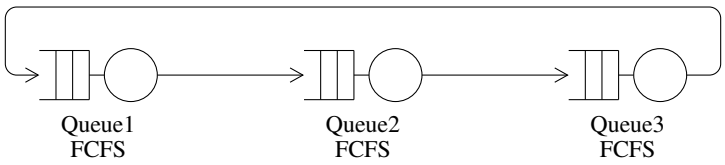


Figure 5.2: Topology of `dt_cycle`: a closed network over 3 queueing stations.

Nodes	3: Queue $\times$ 3
Classes	1: Jobs (closed, $N = 5$)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Queue1 – Jobs: Geometric(0.5) Queue2 – Jobs: Geometric(0.25) Queue3 – Jobs: Geometric(0.7)
Solvers	NC

Table 5.2: Features of `dt_cycle`.

5.1.3 `dt_cycle_loaddep`

The cycle with a load-dependent slotted server.

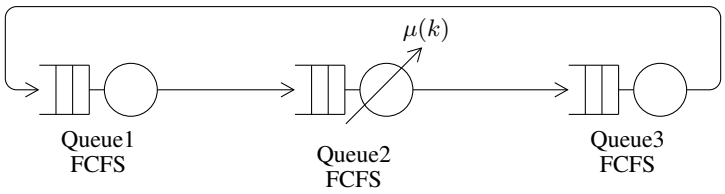


Figure 5.3: Topology of `dt_cycle_loaddep`: a closed network over 3 queueing stations.

5.1.4 `dt_cycle_multiclass`

The multiclass cycle.

5.1.5 `dt_geogeo1`

The Geo/Geo/1 queue: Bernoulli arrivals and Bernoulli service on the slot lattice, against the closed form of Daduna.

5.1.6 `dt_geogeo1_loss`

The Geo/Geo/1/L loss system, and the loss probability.

Superposing two slotted streams produces batches, not a slotted stream of the same kind, so a tandem of slotted queues is the first model in which a discrete-time implementation can go wrong while the single queue still looks right. `dt_cycle` exists for that reason.

Nodes	3: Queue × 3
Classes	1: Jobs (closed, $N = 5$)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Queue1 – Jobs: Geometric(0.5) Queue2 – Jobs: Geometric(0.25) Queue3 – Jobs: Geometric(0.7)
Features	load-dependent rates at Queue2
Solvers	NC

Table 5.3: Features of `dt_cycle_loaddep`.

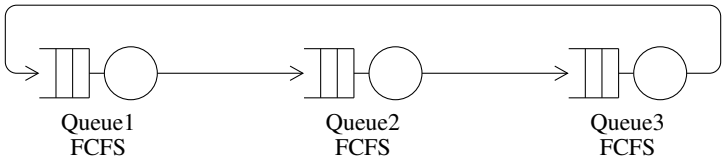


Figure 5.4: Topology of `dt_cycle_multiclass`: a closed network over 3 queueing stations.

Nodes	3: Queue × 3
Classes	2: Chain1 (closed, $N = 3$), Chain2 (closed, $N = 2$)
Scheduling	Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Queue1 – Chain1: Geometric(0.5); Chain2: Geometric(0.5) Queue2 – Chain1: Geometric(0.25); Chain2: Geometric(0.25) Queue3 – Chain1: Geometric(0.7); Chain2: Geometric(0.7)
Solvers	NC

Table 5.4: Features of `dt_cycle_multiclass`.

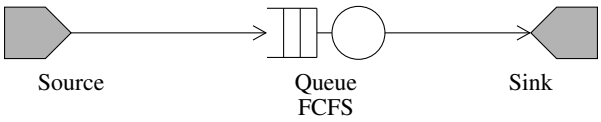


Figure 5.5: Topology of `dt_geogeo1`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Geometric(0.2)
Service	Queue – Class1: Geometric(0.5)
Solvers	NC

Table 5.5: Features of `dt_geogeo1`.

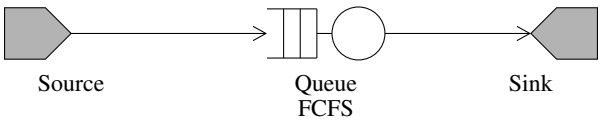


Figure 5.6: Topology of `dt_geogeo1_loss`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Class1 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Geometric(0.2)
Service	Queue – Class1: Geometric(0.5)
Features	finite buffer 4 at Queue
Solvers	NC

Table 5.6: Features of `dt_geogeo1_loss`.

Chapter 6

Solver-specific examples

Most examples in this manual are written around a model and let the solver vary. The scripts of this chapter do the opposite: they are written around one solver, and print what it builds rather than only what it concludes. They live under .

The CTMC solver is the natural subject, because its intermediate objects are the model: the state space, the infinitesimal generator and the stationary distribution are all printable, and the averages are read off them. Printing them is how one checks that a model means what it was intended to mean.

6.0.1 `ctmc_spn_mm1`

The same M/M/1 system written as a queueing model and as a Petri net, and the two generators compared.

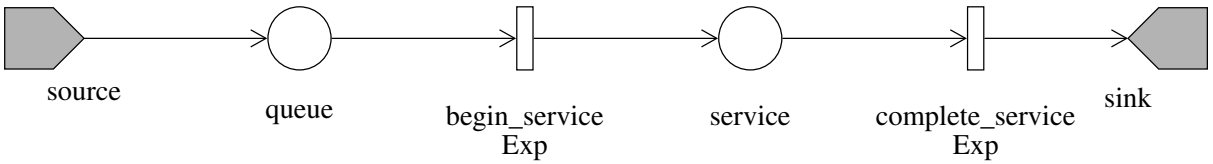


Figure 6.1: Topology of `ctmc_spn_mm1`: a stochastic Petri net over 2 places and 2 transitions; a place is a circle holding its initial marking and a transition a bar, hollow when its firing time is random and solid when it fires at once; the net is open, fed by a Source and drained into a Sink.

Nodes	6: Source, Place × 2, Transition × 2, Sink
Classes	1: jobs (open)
Arrivals	source – jobs: Exp(0.8)
Features	2 transitions with 1 firing mode each
Solvers	CTMC

Table 6.1: Features of `ctmc_spn_mm1`.

6.0.2 `ctmc_spn_vs_nc_closed_qn`

A closed network solved exactly by CTMC over its Petri net form and by the normalizing constant method.

6.0.3 `ctmc_tandem_mm1`

The generator of a two-station tandem, its state space and its stationary distribution, printed in full.

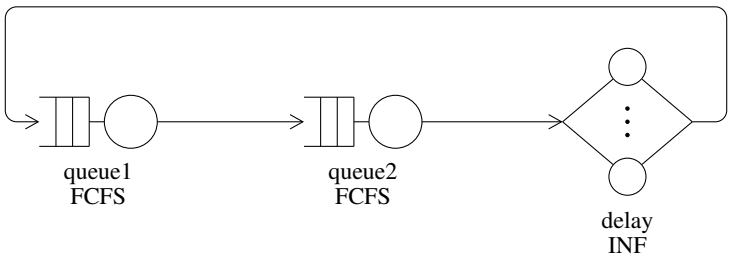


Figure 6.2: Topology of `ctmc_spn_vs_nc_closed_qn`: a closed network over 3 queueing stations.

Nodes	3: Queue $\times$ 2, Delay
Classes	1: jobs (closed, $N = 3$)
Scheduling	queue1: FCFS; queue2: FCFS; delay: INF
Service	queue1 – jobs: Exp(1) queue2 – jobs: Exp(0.8) delay – jobs: Exp(0.5)

Table 6.2: Features of `ctmc_spn_vs_nc_closed_qn`.

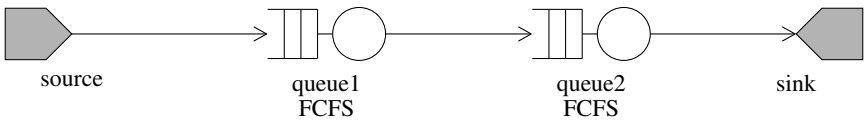


Figure 6.3: Topology of `ctmc_tandem_mm1`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	1: jobs (open)
Scheduling	queue1: FCFS; queue2: FCFS
Arrivals	source – jobs: Exp(0.6)
Service	queue1 – jobs: Exp(1) queue2 – jobs: Exp(1.2)
Solvers	CTMC

Table 6.3: Features of `ctmc_tandem_mm1`.

Two of the three scripts build the same system twice, once as a queueing model and once as a stochastic Petri net, and compare the generators. That is the strongest available check on either notation: the two descriptions are independent, and if they agree state by state then neither has been misunderstood.

Chapter 7

Model interchange

A model can be written to, and read back from, a JSON document. The format is the interchange between the codebases and the wire protocol of the LDES engine, which is driven entirely by it: the client serializes `model.json`, the engine solves it and returns `result.json`.

The examples under are round-trip fixtures: each builds a model that uses one group of wire keys, writes it, reads it back and checks that the reconstructed model has the same structure. What a round trip must preserve is the declared state of the model; what it must not carry is the derived state, which is recomputed on load.

7.1 The fixtures

7.1.1 `json_balking_retrial`

Balking thresholds and retrial orbits over the wire.

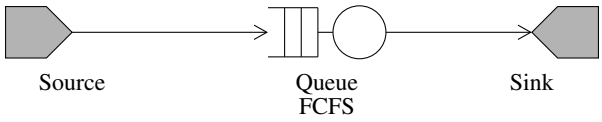


Figure 7.1: Topology of `json_balking_retrial`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Exp(0.5)
Service	Queue – Class1: Exp(2); Class2: Exp(3)
Features	finite buffer 15 at Queue; drop rule {'Class1': 'retrialWithLimit', 'Class2': 'retrialWithLimit'} at Queue

Table 7.1: Features of `json_balking_retrial`.

7.1.2 `json_classcap_droprule`

Per-class capacities and drop rules.

7.1.3 `json_fcr_details`

The finite capacity region block, with its members, limits and drop rules.

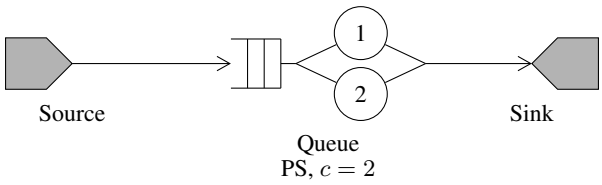


Figure 7.2: Topology of `json_classcap_droprule`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue: PS ($c = 2$)
Arrivals	Source – Class1: Exp(1); Class2: Exp(0.5)
Service	Queue – Class1: Exp(3); Class2: Exp(2)
Features	finite buffer 20 at Queue; load-dependent rates at Queue; drop rule {'Class1': 'drop', 'Class2': 'blockingAfterService'} at Queue

Table 7.2: Features of `json_classcap_droprule`.

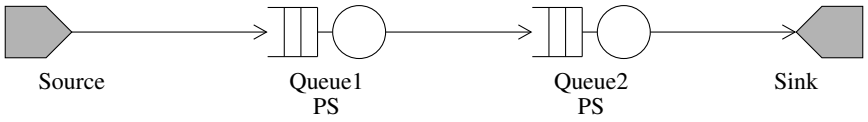


Figure 7.3: Topology of `json_fcr_details`: an open network over 2 queueing stations.

Nodes	4: Source, Queue $\times$ 2, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Queue1: PS; Queue2: PS
Arrivals	Source – Class1: Exp(1); Class2: Exp(0.5)
Service	Queue1 – Class1: Exp(3); Class2: Exp(2) Queue2 – Class1: Exp(4); Class2: Exp(3)

Table 7.3: Features of `json_fcr_details`.

7.1.4 `json_hetero_servers`

Heterogeneous server types at one station, their per-type service rates and the policy that orders them.

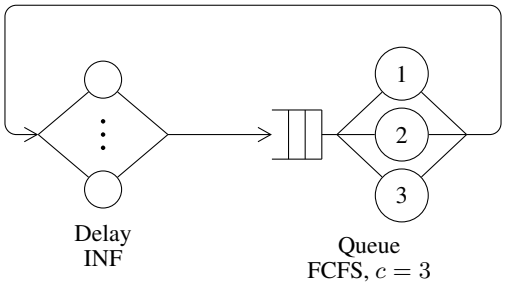


Figure 7.4: Topology of `json_hetero_servers`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 3$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue: FCFS ($c = 3$)
Service	Delay – Class1: Exp(1); Class2: Exp(2) Queue – Class1: Exp(3); Class2: Exp(2)

Table 7.4: Features of `json_hetero_servers`.

7.1.5 `json_join_deadline`

Join deadlines and the quorum they imply.

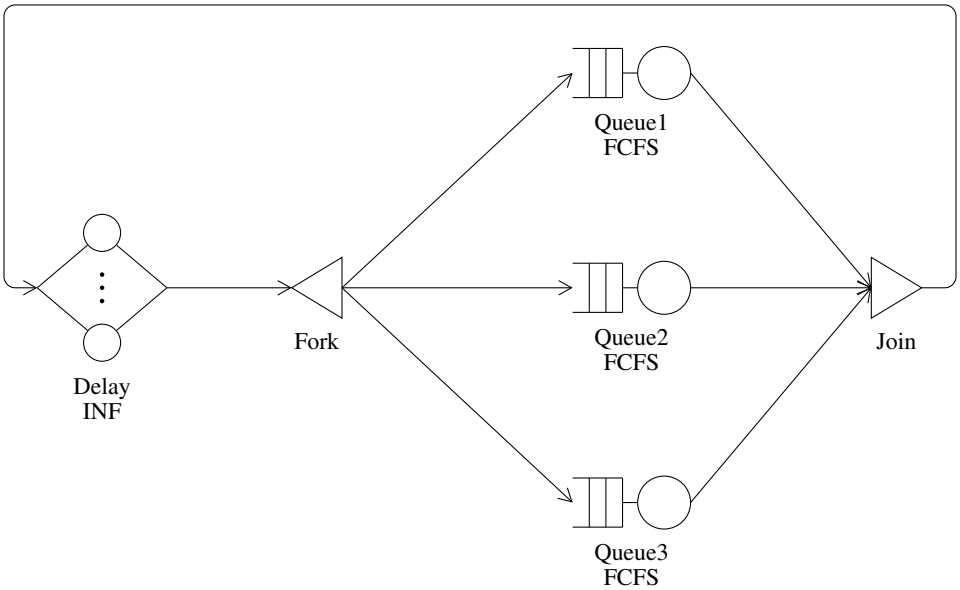


Figure 7.5: Topology of `json_join_deadline`: a closed network over 4 queueing stations with a fork-join block.

7.1.6 `json_signal_classes`

G-network signal classes.

Nodes	6: Queue × 3, Delay, Fork, Join
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: FCFS; Queue2: FCFS; Queue3: FCFS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(3) Queue2 – Class1: Exp(4) Queue3 – Class1: Exp(2)

Table 7.5: Features of `json_join_deadline`.

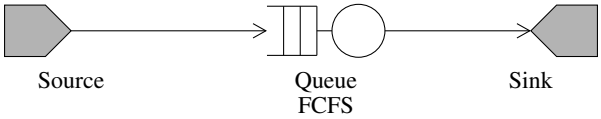


Figure 7.6: Topology of `json_signal_classes`: an open network over 1 queueing station.

The fixtures are grouped by wire key rather than by model family on purpose: a key that no fixture exercises is a key that no round trip tests, and grouping them this way makes the gap visible.

Nodes	3: Source, Queue, Sink
Classes	2: Class1 (open), Signal1 (signal)
Scheduling	Queue: FCFS
Arrivals	Source – Class1: Exp(2); Signal1: Exp(0.5)
Service	Queue – Class1: Exp(5); Signal1: Immediate

Table 7.6: Features of `json_signal_classes`.

Chapter 8

Inference and estimation

Inference runs the modeling process backwards: given measurements of a system – arrival rates, utilizations, response times, or a raw trace – it estimates the model parameters that would produce them. The unknown parameters are marked in the model by a distribution with a `NaN` parameter, and the estimator fills them in.

The estimators divide into three families: likelihood-based (MLE and its quasi-likelihood variant), recursive (the extended Kalman filter and its smoother, which track a parameter that drifts), and Bayesian (sampling and variational, which return a posterior rather than a point estimate). The changepoint scripts test all three against a workload that changes abruptly, which is where a recursive estimator earns its cost. The reference material is in the Inference chapter of the LINE user manual.

The scripts live under , and each one generates a synthetic dataset from known parameters, so the estimate can be compared against the truth.

8.1 The estimators

8.1.1 `est_ekf_changepoint`

The filter under a changepoint in the workload.

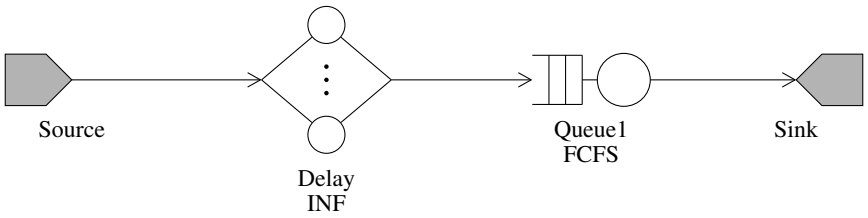


Figure 8.1: Topology of `est_ekf_changepoint`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	1: Class1 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(3.616)

Table 8.1: Features of `est_ekf_changepoint`.

8.1.2 est_ekf_closed

The closed-network filter.

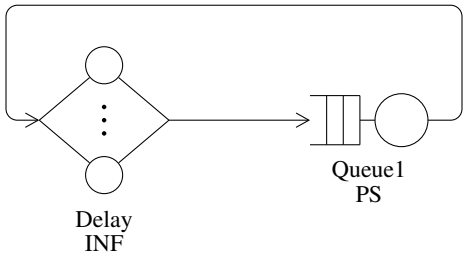


Figure 8.2: Topology of `est_ekf_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(2.5)
Solvers	MVA

Table 8.2: Features of `est_ekf_closed`.

8.1.3 est_ekf_open

An extended Kalman filter tracking the demands of an open network online.

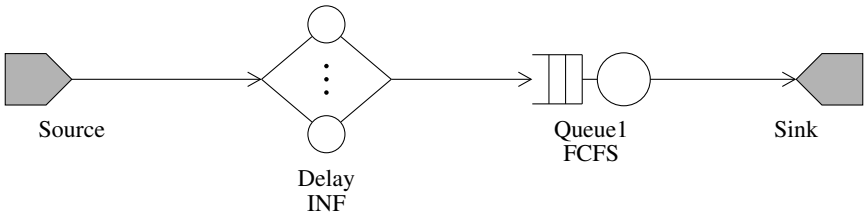


Figure 8.3: Topology of `est_ekf_open`: an open network over 2 queueing stations.

8.1.4 est_erps_closed

The closed-network smoother.

8.1.5 est_erps_open

The Rauch-Tung-Striebel smoother on an open network.

8.1.6 est_mcmc_closed

Markov chain Monte Carlo sampling of the posterior over the demands.

8.1.7 est_mle_open

Maximum likelihood estimation of the service demands of an open network from arrival rate, utilization and response time samples.

Nodes	4: Source, Queue, Delay, Sink
Classes	1: Class1 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(4.123)
Solvers	MVA

Table 8.3: Features of `est_ekf_open`.

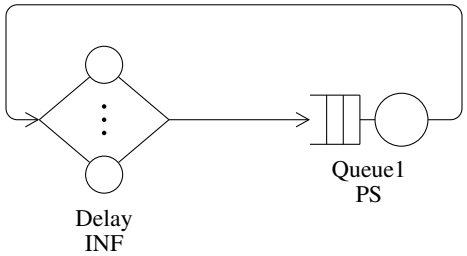


Figure 8.4: Topology of `est_erps_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 3$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(10); Class2: Exp(3.333)
Solvers	MVA

Table 8.4: Features of `est_erps_closed`.

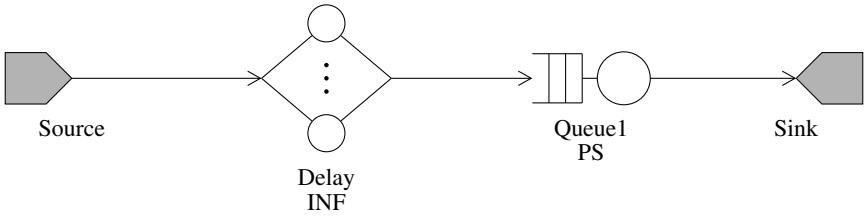


Figure 8.5: Topology of `est_erps_open`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF; Queue1: PS
Arrivals	Source – Class1: Exp(0.1); Class2: Exp(0.05)
Service	Delay – Class1: HyperExp(0.5,0.5;3,10); Class2: HyperExp(0.5,0.5;2,8) Queue1 – Class1: Exp(10); Class2: Exp(3.333)
Solvers	MVA

Table 8.5: Features of `est_erps_open`.

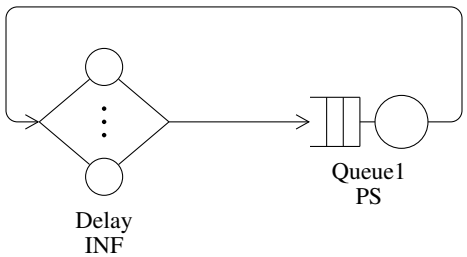


Figure 8.6: Topology of `est_mcmc_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 3$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(0.5703); Class2: Exp(0.5703)
Solvers	MVA

Table 8.6: Features of `est_mcmc_closed`.

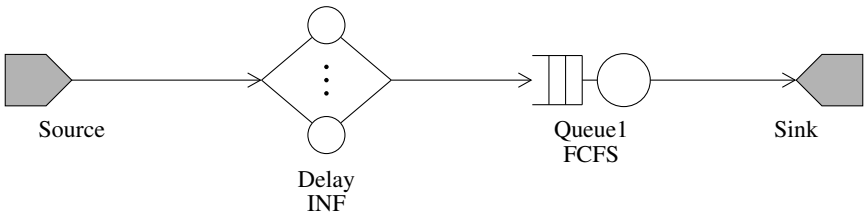


Figure 8.7: Topology of `est_mle_open`: an open network over 2 queueing stations.

8.1.8 `est_qmle_closed`

Quasi-likelihood estimation on a closed network.

8.1.9 `est_trace_closed`

Estimation from a raw arrival and completion trace rather than from aggregate samples.

8.1.10 `est_ubo_changepoint`

The variational estimator under a changepoint.

8.1.11 `est_ubo_closed`

The closed-network version.

8.1.12 `est_ubo_open`

Variational Bayesian estimation on an open network.

8.1.13 `est_ubo_variants_closed`

The variants of the variational estimator, side by side.

8.1.14 `est_ubr_changepoint`

The regression estimator under a changepoint.

Nodes	4: Source, Queue, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Exp(0.5)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(7.962); Class2: Exp(4.77)
Solvers	MVA

Table 8.7: Features of `est_mle_open`.

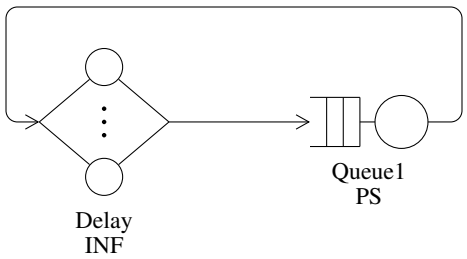


Figure 8.8: Topology of `est_qmle_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 2$), Class2 (closed, $N = 3$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(9.553); Class2: Exp(4.837)
Solvers	MVA

Table 8.8: Features of `est_qmle_closed`.

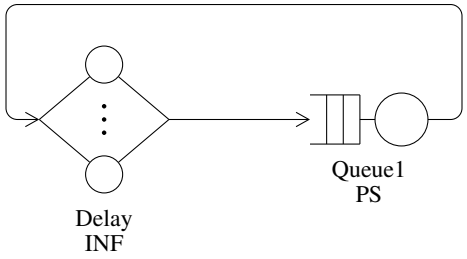


Figure 8.9: Topology of `est_trace_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 5$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(5.43)
Solvers	MVA, CTMC

Table 8.9: Features of `est_trace_closed`.

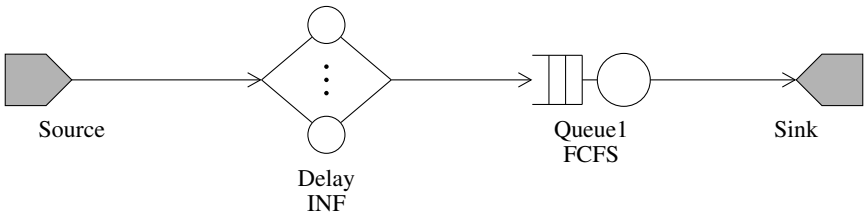


Figure 8.10: Topology of `est_ubo_changepoint`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(1); Class2: Exp(0.5)
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(3.338); Class2: Exp(5.012)

Table 8.10: Features of `est_ubo_changepoint`.

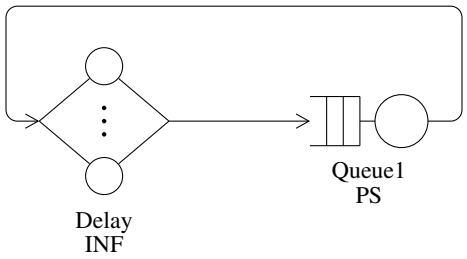


Figure 8.11: Topology of `est_ubo_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(10); Class2: Exp(3.333)
Solvers	MVA

Table 8.11: Features of `est_ubo_closed`.

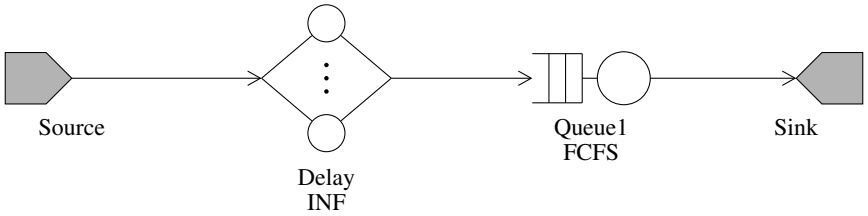


Figure 8.12: Topology of `est_ubo_open`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	2: Class1 (open), Class2 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(0.1); Class2: Exp(0.05)
Service	Delay – Class1: HyperExp(0.5,0.5;3,10); Class2: HyperExp(0.5,0.5;2,8) Queue1 – Class1: Exp(10); Class2: Exp(3.333)
Solvers	MVA

Table 8.12: Features of `est_ubo_open`.

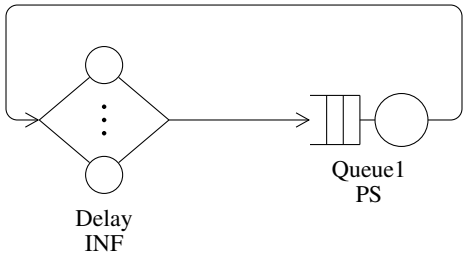


Figure 8.13: Topology of `est_ubo_variants_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 3$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(11.21); Class2: Exp(4.547)
Solvers	MVA

Table 8.13: Features of `est_ubo_variants_closed`.

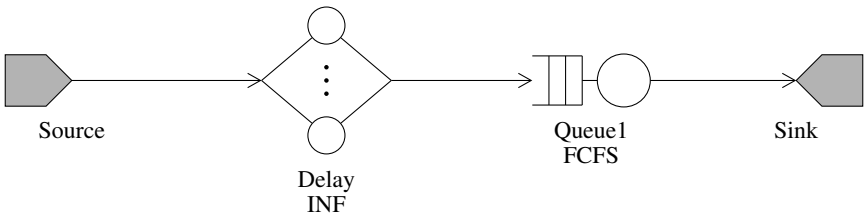


Figure 8.14: Topology of `est_ubr_changepoint`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	1: Class1 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(1)
Service	Delay – Class1: Exp(1) Queue1 – Class1: Exp(1.665)

Table 8.14: Features of `est_ubr_changepoint`.

8.1.15 `est_ubr_closed`

The closed-network version.

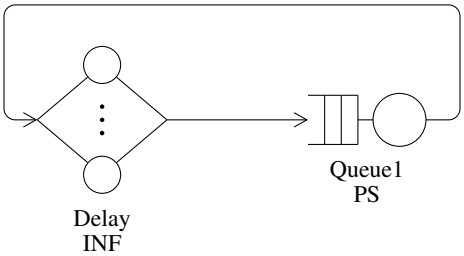


Figure 8.15: Topology of `est_ubr_closed`: a closed network over 2 queueing stations.

8.1.16 `est_ubr_open`

The regression-based estimator on an open network.

8.1.17 `est_vi_closed`

Variational inference over the transition counts of a closed loop, from noisy queue-length readings, returning a Gamma posterior per rate rather than a point estimate.

Nodes	2: Queue, Delay
Classes	2: Class1 (closed, $N = 1$), Class2 (closed, $N = 2$)
Scheduling	Delay: INF; Queue1: PS
Service	Delay – Class1: Exp(1); Class2: Exp(1) Queue1 – Class1: Exp(2.502); Class2: Exp(13.93)
Solvers	MVA

Table 8.15: Features of `est_ubr_closed`.

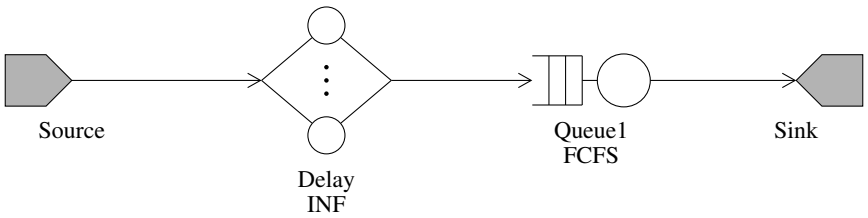


Figure 8.16: Topology of `est_ubr_open`: an open network over 2 queueing stations.

Nodes	4: Source, Queue, Delay, Sink
Classes	1: Class1 (open)
Scheduling	Delay: INF; Queue1: FCFS
Arrivals	Source – Class1: Exp(0.1)
Service	Delay – Class1: HyperExp(0.5,0.5;3,10) Queue1 – Class1: Exp(1.976)
Solvers	MVA

Table 8.16: Features of `est_ubr_open`.

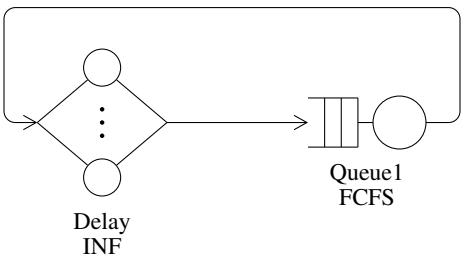


Figure 8.17: Topology of `est_vi_closed`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Class1 (closed, $N = 10$)
Scheduling	Delay: INF; Queue1: FCFS
Service	Delay – Class1: Exp(0.5) Queue1 – Class1: Exp(1.994)
Solvers	MVA

Table 8.17: Features of `est_vi_closed`.

Chapter 9

Optimization

Besides evaluating a given model, LINE can solve design problems on queueing networks. While the solvers answer evaluation questions (given a model, what are its performance metrics?), the optimization layer answers design questions: how many servers, what service rates, which routing probabilities, or what job populations optimize a cost or performance objective subject to service-level constraints. The reference material on decision variables, objectives, constraints and solvers is in the Optimization chapter of the LINE user manual; this chapter collects the worked examples.

The relevant scripts are included under the and each runs in seconds. The general structure of an optimization script consists of four blocks:

1. Definition of the LINE network model
2. Declaration of the decision variables
3. Specification of the objective and constraints
4. Solution

9.1 Sizing an M/M/c queue

The M/M/c queue is a classic model of a service center with c parallel servers fed by a common infinite-capacity buffer. In this example, we wish to find the minimum number of servers such that the utilization does not exceed 50%. Jobs arrive at rate $\lambda = 3$ job/s and each server works at rate $\mu = 1$ job/s, hence the utilization with c servers is $\rho = \lambda/(c\mu) = 3/c$. It is known from theory that the optimum is therefore $c^* = 6$; we wish to recover this value. The following script () solves the problem with the default differential evolution solver, at a cost of 10 units per server: Obtaining the following output:

```
Optimal servers : 6
Cost            : 60.0
Feasible        : True
LINE evaluations: 10
```

Note that only 10 distinct configurations required a LINE solve: the evaluator caches results at the level of the decoded configuration, so the hundreds of candidate vectors explored by differential evolution collapse onto the 10 possible server counts.

9.2 Exact sizing by bisection

The sizing problem of Tutorial 1 has a special structure: feasibility is monotone in the decision variable, since adding servers can only decrease utilization. In such cases `BisectionSolver` finds the exact

optimum in $O(\log n)$ LINE evaluations. Model and problem definition are identical to Tutorial 1 (); only the solution block changes: Obtaining the following output:

```
Optimal servers : 6
Cost            : 60.0
LINE evaluations: 4 (vs hundreds for DE)
```

9.3 Continuous service rate tuning

We now choose the cheapest service rate, a continuous decision, meeting a response time SLA on an M/M/1 queue. With $\lambda = 3$, the response time is $R = 1/(\mu - 3)$, so $R \leq 0.5$ requires $\mu \geq 5$; the rate is billed at 20 cost units per unit, hence the optimum is $\mu^* = 5$ at cost 100. In the script (), the model is an M/M/1 queue built as in Tutorial 1 with initial rate $\mu = 4$, and the problem is declared as: Obtaining the following output:

```
Optimal rate    : 5.001 (theory: 5.0)
Cost            : 100.0
Feasible        : True
```

9.4 Load balancing across heterogeneous servers

An arrival stream of rate $\lambda = 2$ must be split between a fast queue ($\mu_1 = 4$) and a slow queue ($\mu_2 = 2.5$) so that the mean end-to-end response time is minimized. Modeling each queue as M/M/1, the mean number of jobs in the system is $N(p) = \frac{\rho_1}{1-\rho_1} + \frac{\rho_2}{1-\rho_2}$ with $\rho_1 = p\lambda/\mu_1$ and $\rho_2 = (1-p)\lambda/\mu_2$, and by Little's law the system response time is $N(p)/\lambda$. Setting $dN/dp = 0$ yields $p^* \approx 0.743$ with response time 0.4250: the optimal policy does not saturate the fast server. The script () declares the routing split as the decision variable: Obtaining the following output:

```
Fraction to Fast: 0.736 (theory: 0.743)
Fraction to Slow: 0.264
System RT       : 0.4248 (theory: 0.4250)
```

The system response time is computed by Little's law from the chain queue lengths and throughput, which remains correct on parallel topologies.

9.5 Population sizing in a closed network

An interactive system is modeled as a closed network with a Delay station (mean think time 1) and a processor sharing server (rate 2). We seek the largest number of concurrent users the system can sustain while the server response time stays within an SLA of 2 time units. Feasibility is monotone nonincreasing in the population, so bisection applies with the `max_feasible` direction (): Obtaining the following output:

```
Max users       : 5
Feasible        : True
LINE evaluations: 6
```

9.6 Robust sizing under workload scenarios

Capacity decisions should often hold under load uncertainty. Here the server pool of Tutorial 1 is sized so that the utilization SLA holds both under the average load ($\lambda = 3$) and under a peak-load scenario

($\lambda = 4.5$). Scenario models share the node and class names of the base model and are registered with ; the solver enforces every constraint on all scenarios (): Obtaining the following output:

```
Robust servers  : 9 (6 for average load only)
Cost            : 90.0
Feasible        : True
```

The average load alone needs 6 servers; the peak scenario pushes the answer to $\lceil 4.5/0.5 \rceil = 9$.

9.7 Cost-performance tradeoff frontier

How much does tightening the utilization SLA cost? `ParetoSweep` re-solves the sizing problem for a sweep of utilization bounds (epsilon-constraint method) and reports the nondominated cost frontier, here with each point solved exactly by bisection (): Obtaining the following output:

```
Utilization bound -> optimal cost (servers)
  util <= 0.3  ->  100.0  (10)
  util <= 0.4  ->   80.0  (8)
  util <= 0.5  ->   60.0  (6)
  util <= 0.6  ->   50.0  (5)
  util <= 0.75 ->   40.0  (4)
```

The bound 0.9 is absent from the frontier: it requires the same 4 servers as 0.75, so its point is dominated.

9.8 Decomposing a multivariable problem

Jointly choosing the number of servers and the service rate couples two variable types. The decomposition workflow splits them into blocks (servers first, then rates), solves the blocks in sequence with the other block's values held fixed, and cycles until the full penalized objective stabilizes, i.e., block coordinate descent (): Obtaining the following output:

```
Converged       : True after 2 cycle(s)
Servers         : 6
Rate            : 2.003
Objective       : 100.05
```

As with any coordinate descent scheme on a nonconvex problem, the workflow converges to a blockwise optimum that may differ from the joint optimum; a joint solve on this instance finds cost 74.5 (3 servers, rate 2.225). The blockwise optimum reached depends on each codebase's search path, so the printed values (from Python) can differ in the other languages. Decomposition remains preferable when the joint dimensionality is intractable.

9.9 The models behind the tutorials

The tutorials above are design problems, and each one optimizes a queueing network that the narrative describes but does not draw. This section gives that network for every script of : its topology, its features, and the output the script prints once the optimizer has run.

9.9.1 `opt_bisection_sizing`

The same sizing problem solved exactly by bisection, which is admissible because the constraint is monotone in the number of servers.

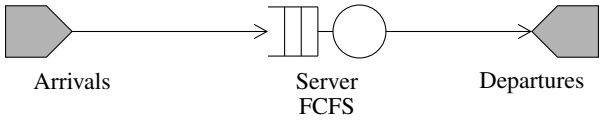


Figure 9.1: Topology of `opt_bisection_sizing`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(1)
Features	class switching on 3 links
Solvers	MVA

Table 9.1: Features of `opt_bisection_sizing`.

9.9.2 `opt_decomposition`

Decomposing a multivariable problem into blocks solved in turn, the coordinate-descent workflow and its blockwise optimum.

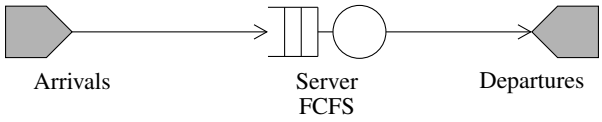


Figure 9.2: Topology of `opt_decomposition`: an open network over 1 queueing station.

9.9.3 `opt_load_balancing`

Load balancing across heterogeneous servers: the decision variables are the routing probabilities, which must form a simplex.

9.9.4 `opt_lqn_host_demand`

Host demand tuning on a layered model, with the gradient optimizer and layer freezing.

9.9.5 `opt_pareto_frontier`

The cost-performance tradeoff frontier, swept rather than solved at one weighting, through `ParetoSweep`.

9.9.6 `opt_population_sizing`

Population sizing in a closed network: how many jobs to admit so that throughput is maximized without breaching a residence-time bound.

9.9.7 `opt_robust_sizing`

Robust sizing, in which the constraint must hold across a set of workload scenarios rather than at one nominal load.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(1)
Features	class switching on 3 links
Solvers	MVA

Table 9.2: Features of `opt_decomposition`.

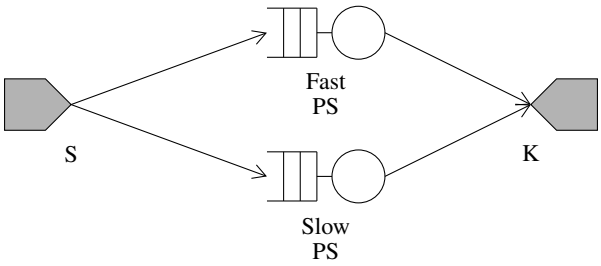


Figure 9.3: Topology of `opt_load_balancing`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: Jobs (open)
Scheduling	Fast: PS; Slow: PS
Arrivals	S – Jobs: Exp(2)
Service	Fast – Jobs: Exp(4) Slow – Jobs: Exp(2.5)
Features	class switching on 4 links

Table 9.3: Features of `opt_load_balancing`.

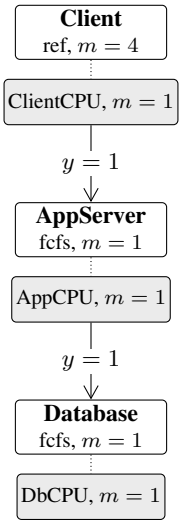


Figure 9.4: Call graph of `opt_lqn_host_demand`: 3 tasks over 3 processors, drawn with the reference task on top and a called task below its caller; the shaded box under each task is the processor it runs on.

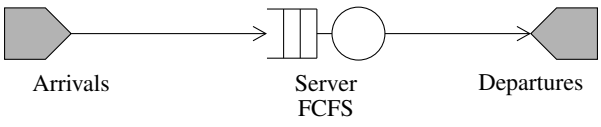


Figure 9.5: Topology of `opt_pareto_frontier`: an open network over 1 queueing station.

Processors	3: ClientCPU (inf, $m = 1$), AppCPU (ps, $m = 1$), DbCPU (ps, $m = 1$)
Tasks	3: Client (ref, $m = 4$), AppServer (fcfs, $m = 1$), Database (fcfs, $m = 1$)
Entries	3: Browse, Render, Query
Activities	3, host demands BrowseAct: 0.5, RenderAct: 0.25, QueryAct: 0.2
Calls	2 (2 synchronous, 0 asynchronous)
Think times	Client: 1.0

Table 9.4: Features of `opt_lqn_host_demand`.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(1)
Features	class switching on 3 links

Table 9.5: Features of `opt_pareto_frontier`.

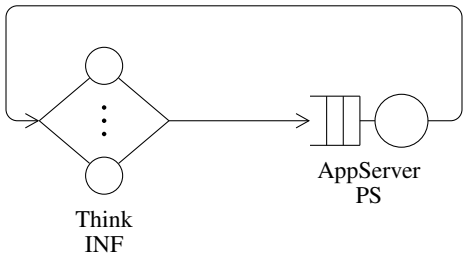


Figure 9.6: Topology of `opt_population_sizing`: a closed network over 2 queueing stations.

Nodes	2: Queue, Delay
Classes	1: Users (closed, $N = 1$)
Scheduling	Think: INF; AppServer: PS
Service	Think – Users: Exp(1) AppServer – Users: Exp(2)
Features	class switching on 2 links
Solvers	MVA

Table 9.6: Features of `opt_population_sizing`.

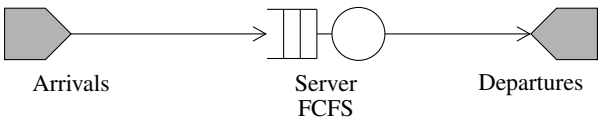


Figure 9.7: Topology of `opt_robust_sizing`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(1)
Features	class switching on 3 links
Solvers	MVA

Table 9.7: Features of `opt_robust_sizing`.

9.9.8 opt_sensitivity_report

The sensitivity of the optimum to each parameter, reported alongside the solution.

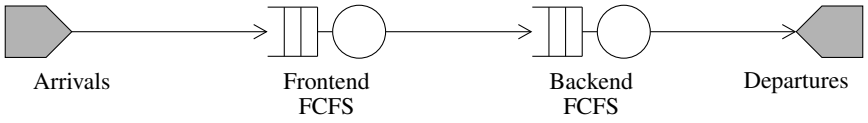


Figure 9.8: Topology of `opt_sensitivity_report`: an open network over 2 queueing stations.

Nodes	4: Source, Queue × 2, Sink
Classes	1: Jobs (open)
Scheduling	Frontend: FCFS; Backend: FCFS
Arrivals	Arrivals – Jobs: Exp(1)
Service	Frontend – Jobs: Exp(2) Backend – Jobs: Exp(1.5)

Table 9.8: Features of `opt_sensitivity_report`.

9.9.9 opt_server_sizing

Sizing an M/M/c queue: choose the number of servers that minimizes cost subject to a response-time bound, by differential evolution over an integer decision variable.

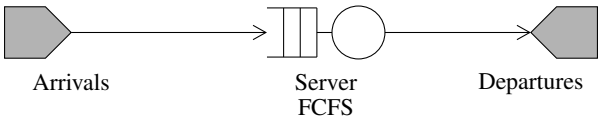


Figure 9.9: Topology of `opt_server_sizing`: an open network over 1 queueing station.

9.9.10 opt_service_rate

Continuous tuning of a service rate, the archetype of a continuous decision variable under a service-level constraint.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(1)
Features	class switching on 3 links
Solvers	MVA

Table 9.9: Features of `opt_server_sizing`.

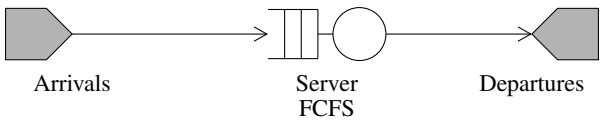


Figure 9.10: Topology of `opt_service_rate`: an open network over 1 queueing station.

Nodes	3: Source, Queue, Sink
Classes	1: Jobs (open)
Scheduling	Server: FCFS
Arrivals	Arrivals – Jobs: Exp(3)
Service	Server – Jobs: Exp(4)
Features	class switching on 3 links
Solvers	MVA

Table 9.10: Features of `opt_service_rate`.

Index

ag_closed_network, 110
ag_gnetwork, 110
ag_jackson_network, 110
ag_multiclass_closed, 110
ag_tandem_open, 110
ag_tandem_phasetype, 110
agent models, 108

bisection, 165
block coordinate descent, 167
busy period, 97
busyp_subnetwork, 97

Cache node, 51
cache task, 103
cache_compare_replc, 52
cache_itemsize_costcap, 52
cache_mmap_rr_env, 52
cache_replc_climb, 52
cache_replc_fifo, 52
cache_replc_hlru, 52
cache_replc_lru, 52
cache_replc_qlru, 55
cache_replc_routing, 55
cache_replc_rr, 55
cache_rmf_transient, 56
cachenet_tandem, 56
caching, 51, 115
capacity planning, 166
cdf_respt_closed, 95
cdf_respt_closed_threeclasses, 95
cdf_respt_distrib, 95
cdf_respt_open_twoclasses, 95
cdf_respt_populations, 95
cl_basic, 60
cl_closed, 60
cl_compare, 62
cl_multiclass, 62
cl_scheduling, 62
cl_stations, 63
cl_sweep, 63
class priorities, 23
class switching, 21
ClassSwitch node, 21
closed queueing network, 8
cluster network, 60
constraint, 165
cqn_bas_blocking, 10
cqn_bcmp_theorem, 10
cqn_lcfs_multiclass, 10
cqn_mmpp2_service, 10
cqn_multichain_cs, 10
cqn_multiserver, 12
cqn_multiserver_nc, 12
cqn_mvablo, 12
cqn_oneline, 12
cqn_repairmen, 14
cqn_repairmen_multi, 14
cqn_scheduling_dps, 14
cqn_threeclass_hyperl, 14
cqn_twoclass_erl, 14
cqn_twoclass_hyperl, 14
cqn_twoqueues_multi, 17
cross-codebase parity, 1
cs_implicit, 21
cs_multi_diamond, 21
cs_single_diamond, 21
cs_transient_class, 23
CTMC solver, 150
ctmc_spn_mm1, 150
ctmc_spn_vs_nc_closed_qn, 150
ctmc_tandem_mm1, 150
custom solver, 112
cyclic polling, 91

decision variable, 165
design problem, 165, 167
differential evolution, 165
discrete time, 147
dispatch_closed, 63
dispatch_mixed, 63
dispatch_open, 63
distributions
    Geometric, 147

dt_bernoulli_loaddep, 147
dt_cycle, 147
dt_cycle_loaddep, 148
dt_cycle_multiclass, 148
dt_geogeo1, 148
dt_geogeo1_loss, 148

epsilon-constraint method, 167
est_ekf_changepoint, 157
est_ekf_closed, 158
est_ekf_open, 158
est_erps_closed, 158
est_erps_open, 158
est_mcmc_closed, 158
est_mle_open, 158
est_qmle_closed, 160
est_trace_closed, 160
est_ubo_changepoint, 160
est_ubo_closed, 160
est_ubo_open, 160
est_ubo_variants_closed, 160
est_ubr_changepoint, 160
est_ubr_closed, 163
est_ubr_open, 163
est_vi_closed, 163
example_mapqn2renv, 98
Extended Kalman Filter, 157

fcr_constraints, 81
fcr_lincon, 81
fcr_lossn, 81
fcr_mm1kdrop, 81
fcr_mm1waitq, 81
fcr_oqndrop, 83
fcr_oqnwaitq, 83
feature table, 2
fes_aggregation, 75
fes_single_class, 75
file formats
    JSON, 153
finite capacity region, 79, 115
fj_asymm, 26
fj_basic_closed, 26
fj_basic_nesting, 26
fj_basic_open, 28
fj_complex_serial, 28
fj_cs_multi_visits, 28
fj_cs_postfork, 28
fj_cs_prefork, 30
fj_deep_nesting, 30
fj_delays, 30
fj_mixed_openclosed, 32
fj_nojoin, 32
fj_route_overlap, 32
fj_serialfjs_closed, 34
fj_serialfjs_open, 34
fj_threebranches, 34
fj_tiny_closed, 36
fj_twoclasses_forked, 36
fj_variable_fanout, 36
flow-equivalent server, 75
Fork node, 26
fork-join, 26, 115

gallery_aphm1, 116
gallery_cache_lru, 116
gallery_cache_routing, 116
gallery_coxm1, 118
gallery_cqn, 118
gallery_cqn_multiclass, 118
gallery_detm1, 118
gallery_dm1, 118
gallery_erldk, 118
gallery_erlerl1, 118
gallery_erlerl1_reentrant, 121
gallery_erlm1, 121
gallery_erlm1_ps, 121
gallery_erlm1_reentrant, 121
gallery_erlm1ps, 121
gallery_fcr, 121
gallery_fj_closed, 123
gallery_fj_open, 123
gallery_fj_quorum, 123
gallery_gamm1, 123
gallery_hyperl1_feedback, 123
gallery_hyperl1_reentrant, 126
gallery_hyperlk, 126
gallery_hyphyp1_linear, 126
gallery_hyphyp1_reentrant, 126
gallery_hyphyp1_tandem, 126
gallery_hypm1, 126
gallery_hypm1_reentrant, 128
gallery_lqn_basic, 128
gallery_lqn_random, 128
gallery_lqn_workflows, 128
gallery_lukumar_reentrant, 128
gallery_mapm1, 128
gallery_mapmk, 128
gallery_mdk, 132
gallery_merl1, 132

gallery_merl1_linear, 132
gallery_merl1_reentrant, 132
gallery_merl1_tandem, 132
gallery_merlk, 132
gallery_mhyp1, 132
gallery_mhyp1_linear, 135
gallery_mhyp1_reentrant, 135
gallery_mhyp1_tandem, 135
gallery_mhypk, 135
gallery_mm1, 135
gallery_mm1_feedback, 135
gallery_mm1_linear, 137
gallery_mm1_multiclass, 137
gallery_mm1_prio, 137
gallery_mm1_ps, 137
gallery_mm1_ps_feedback, 137
gallery_mm1_ps_multiclass, 137
gallery_mm1_ps_reentrant, 139
gallery_mm1_reentrant, 139
gallery_mm1_tandem, 139
gallery_mm1_tandem_multiclass, 139
gallery_mm1k, 139
gallery_mmap1, 139
gallery_mmap1_multiclass, 142
gallery_mmapk, 142
gallery_mmk, 142
gallery_mpar1, 142
gallery_multitier, 142
gallery_multitier_storage, 142
gallery_parm1, 142
gallery_qn_random, 145
gallery_renv_breakdown, 145
gallery_repairmen, 145
gallery_replayerm1, 145
gallery_um1, 145
Geo/Geo/1 queue, 147
Geometric distribution, 147

HOL scheduling, 23

infinitesimal generator, 150
init_state_fcfs_exp, 87
init_state_fcfs_nonexp, 87
init_state_ps, 87

Join node, 26
JSON format, 153
json_balking_retrial, 153
json_classcap_droprule, 153
json_fcr_details, 153
json_hetero_servers, 155
json_join_deadline, 155
json_signal_classes, 155

Kendall notation, 115

large-scale models, 112
largescale_tbi, 112
layered queueing network, 38, 115
lcq_async_cache, 103
lcq_singlehost, 103
lcq_threehosts, 103
ld_class_dependence, 76
ld_fes_multiclass, 76
ld_fes_singleclass, 76
ld_global_dependence, 76
ld_joint_dependence, 76
ld_multiserver_fcfs, 79
ld_multiserver_ps, 79
ld_multiserver_ps_twoclasses, 79
ld_whittle_bandwidth, 79
ldes_warmstart, 89
load balancing, 166, 167
load dependence, 75
lqn_basic, 38
lqn_bpmn, 38
lqn_bpmn_trace, 38
lqn_flatph, 41
lqn_fork_open_arrival, 41
lqn_init, 41
lqn_jsq, 43
lqn_moment3, 43
lqn_multi_solvers, 43
lqn_ofbiz, 43
lqn_open_arrival, 46
lqn_paramident, 46
lqn_rrobin, 47
lqn_serial, 47
lqn_server_pools, 47
lqn_setup, 48
lqn_sockshop, 48
lqn_srvnph, 48
lqn_transient, 48
lqn_twotasks, 48
lqn_workflows, 51
Lu-Kumar reentrant line, 115

mam_transient_mapmap1, 3
maximum likelihood, 157
MCMC, 157
mixed network, 17
model features

caching, 51, 115
class priorities, 23
class switching, 21
cyclic polling, 91
finite capacity region, 79, 115
fork-join, 26, 115
load balancing, 166, 167
load dependence, 75
random environment, 98, 115
reward models, 108
slotted time, 147
state-dependent routing, 89
switchover times, 93
warm start, 87
model gallery, 115
model interchange, 153
mqn_basic, 17
mqn_multichain_cs, 17
mqn_multiserver_fcfs, 19
mqn_multiserver_ps, 19
mqn_singleserver_fcfs, 19
mqn_singleserver_ps, 19

node types
Cache, 51
ClassSwitch, 21
Fork, 26
Join, 26
Place, 65
Transition, 65

objective function, 165
open queueing network, 3
opt_bisection_sizing, 167
opt_decomposition, 168
opt_load_balancing, 168
opt_lqn_host_demand, 168
opt_pareto_frontier, 168
opt_population_sizing, 168
opt_robust_sizing, 168
opt_sensitivity_report, 171
opt_server_sizing, 171
opt_service_rate, 171
oqn_basic, 4
oqn_bmmapt, 4
oqn_cs_routing, 4
oqn_fourqueues, 4
oqn_mapt, 4
oqn_mapt_service, 6
oqn_mmapt, 6
oqn_multichain_cs, 6
oqn_nhpp, 6
oqn_nnpp, 6
oqn_oneline, 8
oqn_trace_driven, 8
oqn_vsinks, 8

parameter inference, 157
Pareto frontier, 167
pas_closed_tandem_fig6, 105
pas_compatibility_5class, 105
pas_cyclic_ctmc_vs_bruteforce, 105
pas_cyclic_normconst_bruteforce, 106
pas_mmk, 106
pas_nc_sampling, 106
pas_saturation, 106
pas_selfloop, 106
pass and swap, 105
Place node, 65
polling_exhaustive_det, 91
polling_exhaustive_exp, 91
polling_gated, 93
polling_klimited, 93
prio_hol_closed, 24
prio_hol_open, 24
prio_identical, 24
prio_psprio, 24

QMLE, 157

random environment, 98, 115
renv_basic, 98
renv_container_terminal, 100
renv_fourstages_repairmen, 100
renv_genqn, 100
renv_lqn_attackgraph, 100
renv_lqn_twostages, 100
renv_map_fallback, 101
renv_node_breakdown, 101
renv_rotterdam_blending, 101
renv_threestages_repairmen, 101
renv_twostages_repairmen, 101
response time distribution, 93
retrieval_chain, 56
retrieval_default, 56
retrieval_ps, 56
retrieval_routing, 56
retrieval_simple, 60
reward models, 108
rewardModel_aggregation, 108

rewardModel_mm1k, 108
rewardModel_multiclass, 108
rewardModel_templates, 108
round-trip fixture, 153
routing matrix, 2
runnable entry, 1

scheduling strategies
  HOL, 23
script structure, 1
sdroute_closed, 89
sdroute_jsq, 90
sdroute_krzesinski, 90
sdroute_multibranch, 90
sdroute_open, 91
sdroute_twoclasses_closed, 91
serial routing, 2
server sizing, 165, 167
slot, 147
slotted time, 147
snc_delay_quantile, 98
snc_tandem_multiclass, 98
solution methods
  bisection, 165
  block coordinate descent, 167
  differential evolution, 165
  epsilon-constraint method, 167
  Extended Kalman Filter, 157
  flow-equivalent server, 75
  maximum likelihood, 157
  MCMC, 157
  QMLE, 157
  stochastic network calculus, 97
  variational inference, 157
solver_custom, 112
solver_custom_analyzer, 112
solvers
  CTMC, 150
spn_basic_closed, 65
spn_basic_open, 66
spn_closed_fourplaces, 66
spn_closed_twoplaces, 66
spn_colored_gspn, 66
spn_fluid_dae, 66
spn_fluid_onoff, 68
spn_fourmodes, 68
spn_inhibiting, 68
spn_lpbounds, 68
spn_open_sevenplaces, 70
spn_pareto_service, 70
spn_productform_nc, 70
spn_queueing_place, 71
spn_twomodes, 71
state probability, 83
state-dependent routing, 89
statepr_aggr, 83
statepr_aggr_large, 83
statepr_allprobs_fcfs, 85
statepr_allprobs_ps, 85
statepr_sys_aggr, 85
statepr_sys_aggr_large, 85
statepr_sys_marg, 85
stochastic network calculus, 97
stochastic Petri net, 65, 152
switchover times, 93
switchover_basic, 93

test_spn_nrm_open, 71
topology figure, 2
Transition node, 65

variational inference, 157

warm start, 87
wf_aph, 71
wf_branch, 73
wf_complex, 73
wf_erlang, 73
wf_loop, 73
wf_parallel, 73
wf_serial, 73
wfnet_and_xor, 74
wfnet_closed, 74
wfnet_nonsp_sound, 74
wfnet_pnml, 74
wfnet_resource, 74
wfnet_serial, 74
wfnet_unsound, 74
wire key, 156
worked example, 2
workflow model, 71
workflow net, 73
workload scenario, 167