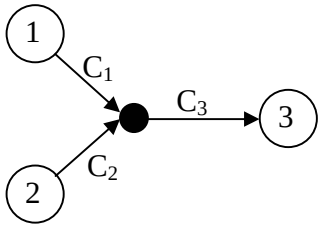
Blocking in Circuit-Switched Networks

- Consider the 3-link network shown below



  - Capacity of link ℓ : C_ℓ = max. number of call that it support.
  - Origin-Destination pairs for Applications:

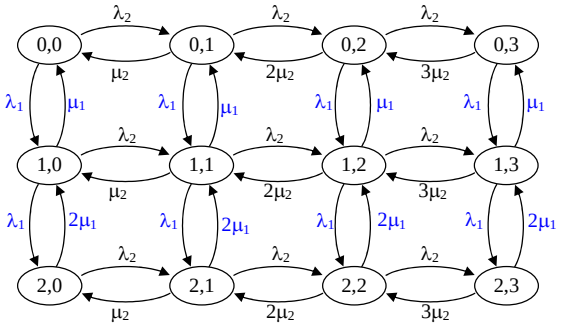
    App-1: $(O_1, D_1) = (1,3)$

    App-2: $(O_2, D_2) = (2,3)$

    With arrival rates Poisson, $\perp\!\!\!\perp$: $r_1(3) = \lambda_1$, $r_2(3) = \lambda_2$
  - Holding times are $\mathcal{E}(\mu_i)$ in application i, $i = 1, 2$; i.i.d. and $\perp\!\!\!\perp$ over applications (independent applications)
  - Blocked calls "lost".
  - Define $N_i(t)$ is # of app-i calls in progress at time t.
  - State: $\underline{N}(t) = (N_1(t), N_2(t))$; continuous-time Markov chain
  - App-i blocked if $N_i(t) = C_i$ or $N_1(t)+N_2(t) = C_3$

  Here, for example, while one of application 1 is in process, it utilizes links 1 and 3. If there are no capacity left in link 1 or link 3, then that application is lost.

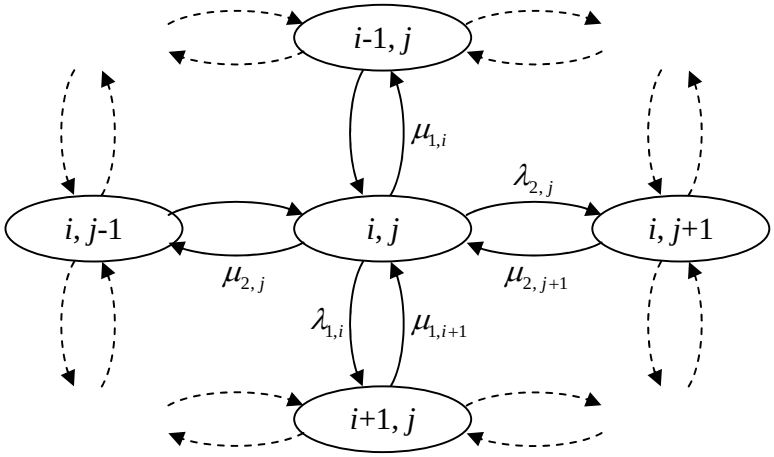
- **Case 1** of 3-link network example: $C_3 \geq C_1+C_2$.



Then the two applications do not interact. The equilibrium distribution is a product of truncated Poissons:

$$P_{ij} = \lim_{t\to\infty} P\left(N_1(t) = i, N_2(t) = j\right) = \frac{\frac{\rho_1^i}{i!}}{\sum_{n=0}^{C_1} \frac{\rho_1^n}{n!}} \frac{\frac{\rho_2^j}{j!}}{\sum_{m=0}^{C_2} \frac{\rho_2^m}{m!}} \;;\; \rho_i = \frac{\lambda_i}{\mu_i}$$

Proof. Here, $\lambda_{1,i} = \lambda_1$, $\lambda_{2,j} = \lambda_2$, $\mu_{1,i} = i\mu_1$, $\mu_{2,j} = j\mu_2$.



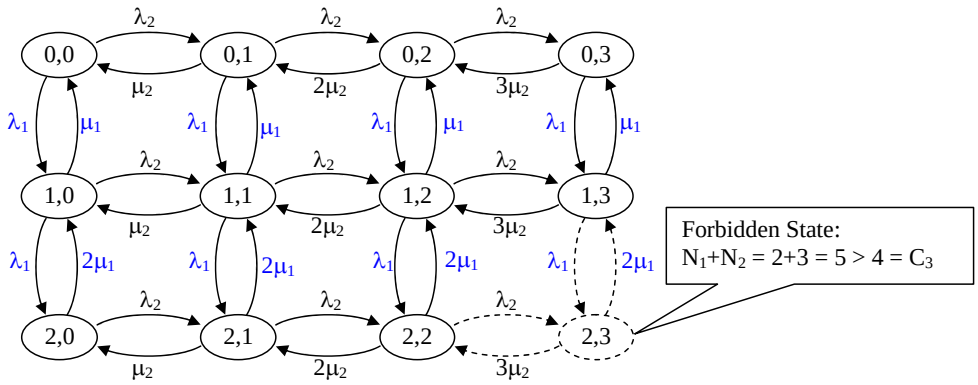
Using local balance equations, we have 1) $p_{i,j+1} = \frac{\lambda_{2,j}}{\mu_{2,j+1}} p_{i,j} = \rho_{2,j+1} p_{i,j}$, 2) $p_{i+1,j} = \frac{\lambda_{1,i}}{\mu_{1,i+1}} p_{i,j} = \rho_{1,i+1} p_{i,j}$. Let $R_{1,i} = \rho_{1,i}\rho_{1,i-1}\cdots\rho_{1,1}$, $R_{2,j} = \rho_{2,j}\rho_{2,j-1}\cdots\rho_{2,1}$, then $p_{i,j} = R_{1,i}R_{2,j}p_{0,0}$. Requiring the sum of $p_{i,j}$ to 1, we have $p_{0,0} = \frac{1}{\sum_i \sum_j R_{1,i}R_{2,j}} = \frac{1}{\left(\sum_i R_{1,i}\right)\left(\sum_j R_{2,j}\right)}$. Hence,

$$p_{i,j} = \frac{R_{1,i}R_{2,j}}{\left(\sum_i R_{1,i}\right)\left(\sum_j R_{2,j}\right)} = \left(\frac{R_{1,i}}{\sum_i R_{1,i}}\right)\left(\frac{R_{2,j}}{\sum_j R_{2,j}}\right).$$

This reduces the problem to two independent truncated birth-and-death processes. The first one gives distribution $\{p_{1,i}\}$, and the second one gives distribution $\{p_{2,j}\}$. The combined distribution is $p_{i,j} = p_{1,i} \times p_{2,j}$.

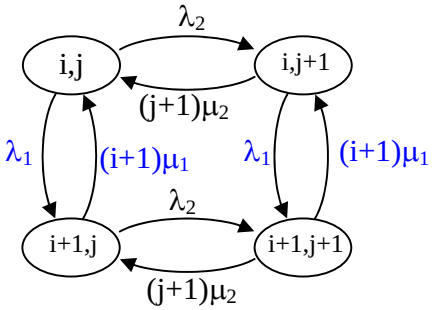
Note that this is possible because the domains are independent. We can write $\sum_{i\in A}\sum_{j\in B} f(i) f(j) = \left(\sum_{j\in A} f(i)\right)\left(\sum_{j\in B} f(j)\right)$.

- **Case 2** of 3-link network example: $C_3 < C_1+C_2$.
  For example, $C_1 = 2$, $C_2 = 3$, $C_3 = 4$



If $\{p_{i,j}\}$ satisfies detailed balance, then $\{p_{i,j}\}$ satisfies global balance and process is time-reversible.

If we temporary neglect the requirement $N_1+N_2 < C_3$, the solution would be $p_{ij} = \dfrac{1}{\text{constant}}\left(\dfrac{\rho_1^i}{i!}\right)\left(\dfrac{\rho_2^j}{j!}\right)$ and this product solution would satisfy detailed balance, (including the dotted one).



Proof

Column: $\lambda_1 p_{i,j} = (i+1)\mu_1 p_{i+1,j} \Rightarrow$

$$\lambda_1 \frac{1}{\text{constant}}\left(\frac{\rho_1^i}{i!}\right)\left(\frac{\rho_2^j}{j!}\right) = (i+1)\mu_1 \frac{1}{\text{constant}}\left(\frac{\rho_1^{i+1}}{(i+1)!}\right)\left(\frac{\rho_2^j}{j!}\right) \text{ : checked.}$$

Row: $\lambda_2 p_{i,j} = (i+1)\mu_2 p_{i,j+1} \Rightarrow$

$$\lambda_2 \frac{1}{\text{constant}}\left(\frac{\rho_1^i}{i!}\right)\left(\frac{\rho_2^j}{j!}\right) = (i+1)\mu_2 \frac{1}{\text{constant}}\left(\frac{\rho_1^i}{i!}\right)\left(\frac{\rho_2^{j+1}}{(j+1)!}\right) \text{ : checked.}$$

Therefore, detailed balance is satisfied for every $(i,j) - (i',j')$ state pair:

Let $\alpha\big((i,j)\to(i',j')\big)$ = weight on arrow. If no arrow, this is 0.

Then, $p_{i,j}\alpha\big((i,j)\to(i',j')\big) = p_{i',j'}\alpha\big((i',j')\to(i,j)\big)$

Hence, a solution of this form satisfies detailed balance even after we impose $N_1+N_2 \le C_3$ so that state (2,3) becomes forbidden.

Then, all we have to do is change the constant in the denominator.

Thus, the steady-state solution is $p_{ij} = \dfrac{\frac{\rho_1^i}{i!}\frac{\rho_2^j}{j!}}{\sum_{(i',j')\in S}\frac{\rho_1^{i'}}{i'!}\frac{\rho_2^{j'}}{j'!}}$, $(i,j) \in \boldsymbol{S}$

where $\boldsymbol{S} = \{(i,j): i \le C_1, j \le C_2, i+j \le C_3\}$

- Description:

  Suppose traffic for application *i* arrives independently of that for all other applications in a Poisson λ_i fashion.

  Assume fixed routes. (If traffic from a given application actually is routed along several routes randomly in fixed proportions, view each as separate applications and the result below will hold.)

  State: $\underline{N}(t) = (N_1(t), N_2(t), \ldots, N_a(t))$, where $N_i(t)$ is # of app-*i* calls in progress at time *t*, and *a* = number of applications

  Routes: for each link, $R_\ell = \{i : \text{application i uses link } \ell\}$

  Capacity constraint: $\sum_{i\in R_\ell} N_i(t) \le C_\ell$, $\ell \in \boldsymbol{\mathcal{L}}$

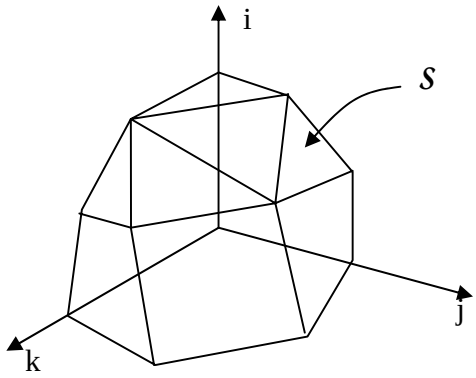
  Parameters for application *i* : λ_i and μ_i, $\rho_i = \dfrac{\lambda_i}{\mu_i}$, (O_i, D_i), a route from O_i to D_i over directed (and predefined) link.

  Define $P_{\underline{n}} = \lim_{t\to\infty} P(\underline{N}(t) = \underline{n})$.

- Solution $P_{\underline{n}} = const. \prod_{i=1}^{a} \dfrac{\rho_i^{n_i}}{n_i!}$, $\underline{n} \in \boldsymbol{S}$. $\boldsymbol{S} = \left\{\underline{n} : \forall \ell \sum_{i\in R_\ell} n_i \le C_\ell\right\}$.

  (Basically, sum only (i,j) that is permitted.)

- When one application is being process, it uses all the links from its origin to its destination. If any of the links in its path is full, then it is lost.
- $S \Rightarrow$ multidimensional parallelepiped with planes cut away.



- Formula for calculating value in the rage of this function is a product of $g_i(n_i)$'s, but domain is no longer the cross product of domain of $g_i(\cdot)$'s. Hence, N_i's are dependent.
- For large networks, the constant is hard to find numerically
- Also, it is desirable to use adaptive routing so that when a boundary is reached (approached), traffic is re-routed. This reduces blocking, is still Markov, but is no longer solved by a truncated product-form solution.