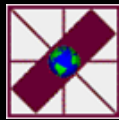


SIGMETRICS – PERFORMANCE
2004



France Telecom R&D



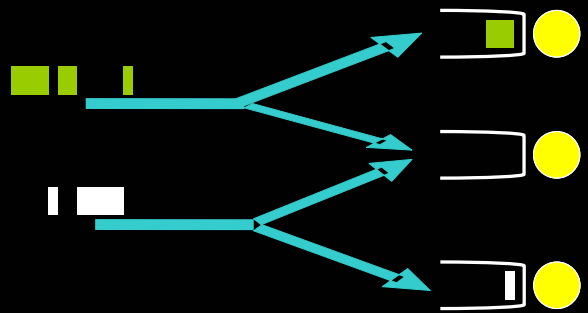
Insensitive Load Balancing

M. Jonckheere

Joint work with T. Bonald and A. Proutière

Load balancing in queuing networks

- K customer classes
- Class-k customers served by a subset of the queues
- Routing independent of job sizes



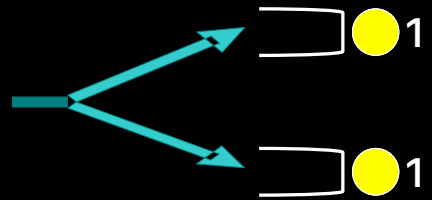
Applications include:

- phone networks
- call centers
- IP networks
- parallel computers

Existing optimality results

Symmetric parallel servers ($K=1$)

- Join the shortest queue (greedy routing)
- Winston (1977) for M/M/1 queues
- Hordijk & Koole (1992) for M/M/1/m queues

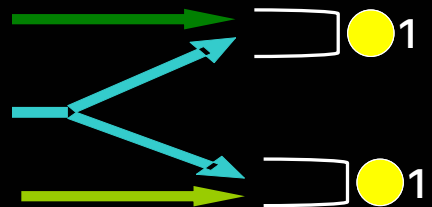


V2-symmetric servers ($K>1$)

- Towsley et. al. (1992) for M/M/1 and M/M/1/m queues

Parallel servers with cross traffic ($K>1$)

- Leeuwaarden et. al. (2001) for M/M/m/m, dynamic programming techniques
- Alanyali & Hajek (1997) for M/M/1 and M/M/m/m in heavy traffic



Towards insensitive load balancing

- Almost all results for M/M queues or in heavy traffic
 - difficult to characterize the optimal policies
 - no explicit performance results
 - what about other service time distributions?
cf. Whitt (1986)
- In practice, we need results for **any** traffic characteristics
 - cf. the Erlang formula
 - stationary distribution insensitive to service distribution
 - the key to simple and robust dimensioning rules...

 **Focus on insensitive policies**

Outline

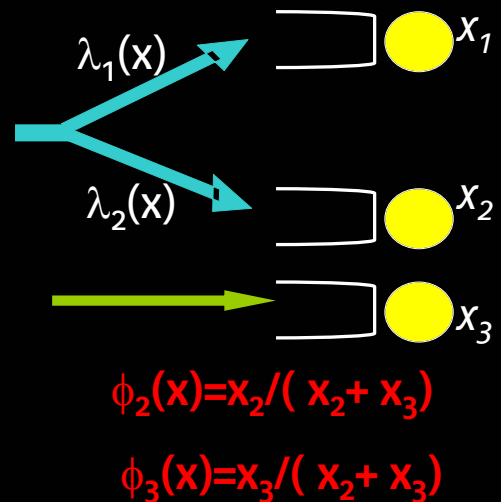
- **Model**
- **Insensitive load balancing**
 - static
 - dynamic
- **Optimality results**
 - single class ($K=1$)
 - multiclass ($K>1$)
- **Examples**
- **Conclusion**

Outline

- **Model**
- **Insensitive load balancing**
 - static
 - dynamic
- **Optimality results**
 - single class ($K=1$)
 - multiclass ($K>1$)
- **Examples**
- **Conclusion**

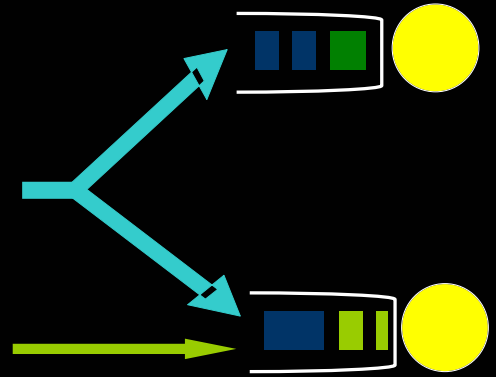
Model

- N processor sharing nodes
- K customer classes
- Each node serves a single class
- Poisson arrivals
- State-dependent arrival rates $\lambda_i(\mathbf{x})$
- State-dependent service rates $\phi_i(\mathbf{x})$



Model

- N processor sharing nodes
- K customer classes
- Each node serves a single class
- Poisson arrivals
- State-dependent arrival rates $\lambda_i(\mathbf{x})$
- State-dependent service rates $\phi_i(\mathbf{x})$
- **Finite** state space



We aim at minimizing the **blocking probability**

Outline

- Model
- Insensitive load balancing
 - static
 - dynamic
- Optimality results
 - single class ($K=1$)
 - multiclass ($K>1$)
- Examples
- Conclusion

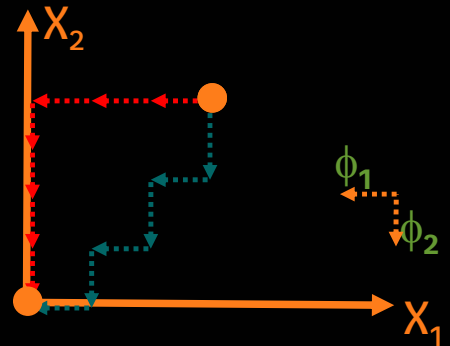
Static load balancing

- The network is insensitive **if and only if** the service rates are balanced:

$$\forall i, \forall x, \phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)}$$

- Balance \Leftrightarrow **reversibility property**
 $1/\Phi(x)$ = product of service rates along any path from x to 0
- **Stationary measure**

$$\pi(x) = \Phi(x) \prod \lambda_i^{x_i}$$



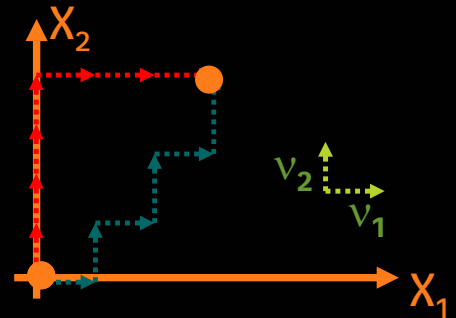
Dynamic load balancing

- We consider balanced service rates
- The network is insensitive **if and only if** the arrival rates are balanced:

$$\forall i, \forall x, \lambda_i(x) = \frac{\Lambda(x + e_i)}{\Lambda(x)}$$

- Balance \Leftrightarrow **reversibility property**
 $\Lambda(x)$ = product of arrival rates along any path from 0 to x
- **Stationary measure**

$$\pi(x) = \Lambda(x)\Phi(x)$$

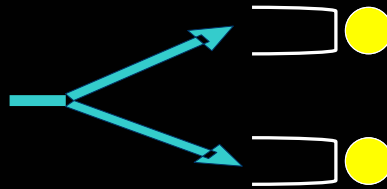


Outline

- Model
- Insensitive load balancing
 - static
 - dynamic

- **Optimality results**

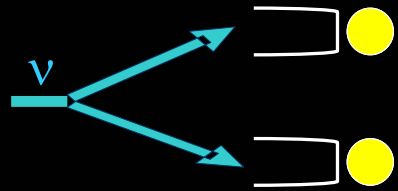
- single class ($K=1$)
- multiclass ($K>1$)



- Examples
- Conclusion

Simple load balancing

- Characterized by a **unique blocking state** y

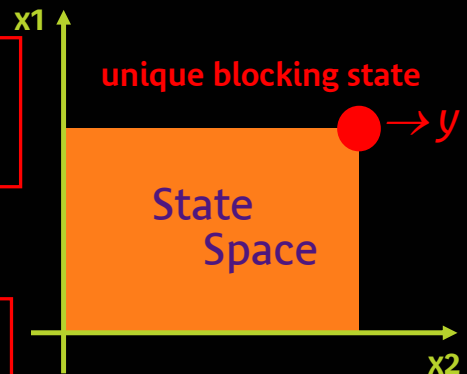


- Balance function

$$\Lambda(x) = \begin{pmatrix} y_1 - x_1 + y_2 - x_2 \\ y_1 - x_1 \end{pmatrix} \nu^{x_1+x_2}$$

- Arrival rates

$$\lambda_1(x) = \nu \frac{y_1 - x_1}{y_1 - x_1 + y_2 - x_2}$$



Optimal load balancing

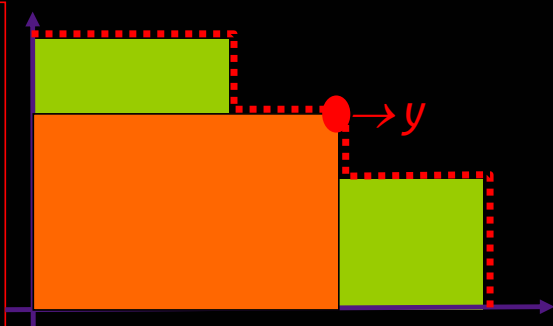
- **Theorem**

Any admissible balance function Λ is a linear combination of "simple" balance functions

$$\Lambda = \sum_{y \in \mathcal{Y}} \alpha(y) \Lambda_y$$

- **Corollary:**
Optimality of "simple" policies

There exists a simple policy that minimizes the blocking probability



Evaluating the blocking probability

- Let $b(\mathbf{y})$ be the blocking probability of the simple policy blocking in state \mathbf{y}
- We have the recursive formula:

$$b(\mathbf{y})^{-1} = 1 + \frac{1}{\nu} \sum_i \phi_i(\mathbf{y} - \mathbf{e}_i) (b(\mathbf{y} - \mathbf{e}_i))^{-1}$$

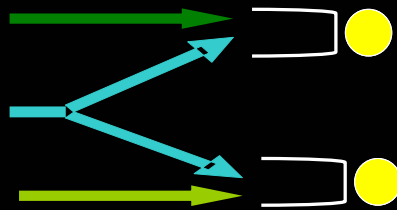
- $O(|\mathbf{y}|)$ complexity with $|\mathbf{y}|$ = number of states

Outline

- Model
- Insensitive load balancing
 - static
 - dynamic

- **Optimality results**

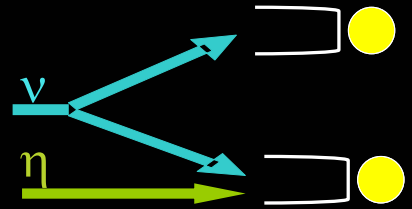
- single class ($K=1$)
- multiclass ($K>1$)



- Examples
- Conclusion

Simple load balancing

- Product of simple per-class load balancing
- Balance function



$$\Lambda(x) = \binom{y_1 - x_1 + y_2 - x_2}{y_1 - x_1} \nu^{x_1 + x_2} \eta^{x_3}$$

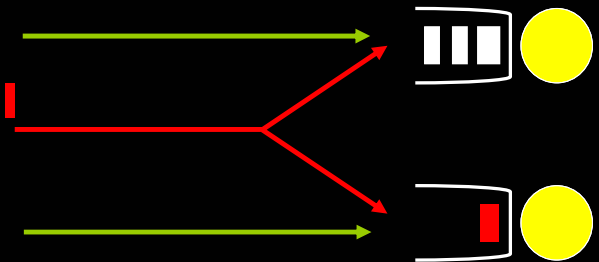
- Arrival rates for admissible states:

$$\lambda_1(x) = \nu \frac{y_1 - x_1}{y_1 - x_1 + y_2 - x_2}$$

$$\lambda_3(x) = \eta$$

Optimal decentralized load balancing:

- No characterization of general optimal policies
- **Per class load balancing**
Routing class-k according to the number customers from class-k
- Simple per-class policy defines a decentralized scheme



Optimality results

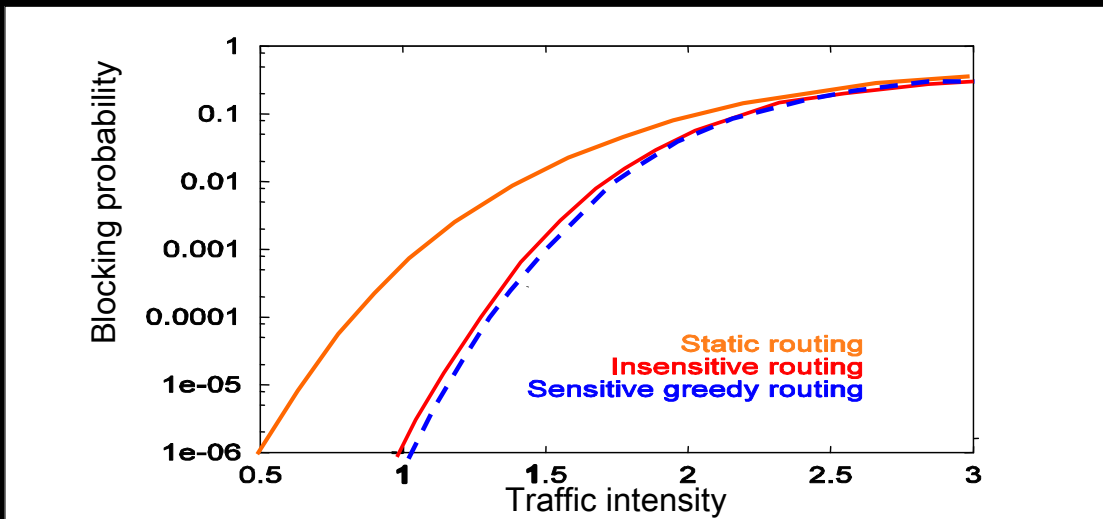
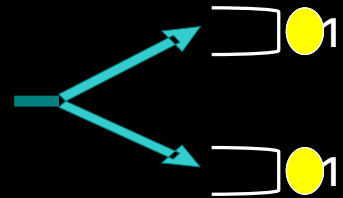
*Optimal decentralized policy =
Simple per class load balancing*

Outline

- Model
- Insensitive load balancing
 - static
 - dynamic
- Optimality results
 - single class ($K=1$)
 - multiclass ($K>1$)
- **Examples**
- Conclusion

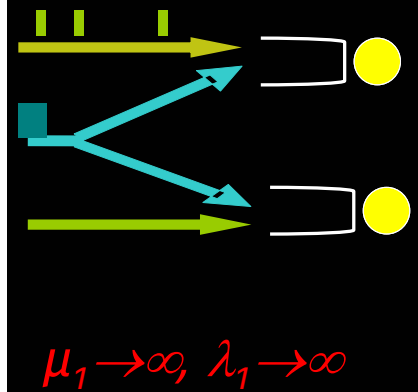
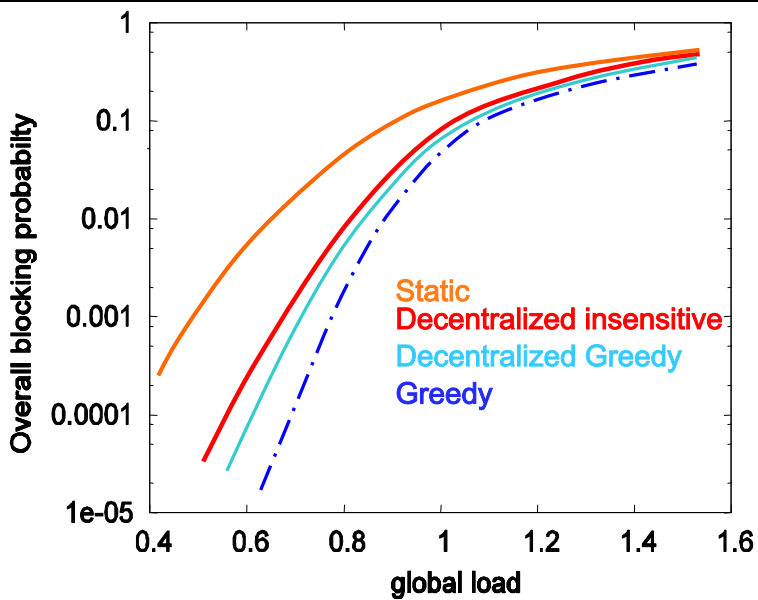
Parallel servers

- The greedy policy is known to be optimal for exponential service times
- The insensitive load balancing gives an accurate approximation of the optimal policy



Parallel servers with cross traffic

- The greedy policy is known to be optimal for exponential service times of same mean
- The performance of greedy policies depends on traffic characteristics



Conclusion

- **Insensitive** load balancing leads to **robust, explicit** results
- **Main results**
 - A single class: there exists a **simple** optimal policy
 - Extension to multiclass networks with a **decentralized** policy
- **Open issues:** more general multiclass results?