Stochastic methods for measurement-based network control

Ellens, W.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Routing under partial observability and controllability

Outline. Chapters 6 and 7 considered a queuing system for which the system state is only known at some moments in time. In the current chapter we study a queuing system which is observed continuously, but for which only part of the system state is visible. Our motivation stems from (online) navigation systems that can only use information about their own subscribers. With this application in mind, we investigate routing policies for a two-server queueing system with two types of jobs, the first (the subscribers) being observable and controllable, the second (the background traffic) being neither observable nor controllable.

The routing policies — as well as the model and the notation — are introduced in Section 8.1. Section 8.2 presents a simulation study comparing the performance (in terms of the average sojourn time) of these policies and providing insight into the important question of the minimal penetration level (the fraction of controllable jobs) needed for effective control. The simulations show that a simple policy using partial state information performs almost as good as join-the-shortest-queue (JSQ), the optimal policy in fully controllable systems. This remarkable result is analysed analytically in Section 8.3, which is based on fluid approximations and (hence) especially useful for highly loaded systems. The chapter is concluded by Section 8.4, which summarises the results of our study and suggests directions for further research.
Chapter 8 Routing under partial observability and controllability

8.1 Model

This section describes the queueing model and provides background for the discussion of our simulation and fluid approximation results presented in Section 8.2 and Section 8.3 respectively. We start by introducing the modelling assumptions and the notation in Section 8.1.1 which is followed by the introduction of the routing policies in Section 8.1.2. In the latter section we also discuss the goals and our expectations of these policies.

8.1.1 The queueing system

We consider a two-server queueing system, as depicted in Figure 8.1. The queues operate independently in a FIFO-manner with exponential service times of rates \( \mu = (\mu_1, \mu_2) \). Jobs arrive at the queueing system according to a Poisson process with rate \( \lambda \) and are of one of the following two types.

- Jobs of the first type, called X jobs, are controllable and observable. They make up for a fraction \( \alpha \) of the total load. X jobs thus arrive according to a Poisson process of rate \( \alpha \lambda \). The parameter \( \alpha \) is called the penetration level. X jobs are controllable, because upon their arrival a router decides on which queue they should join. A (routing) policy is a set of rules according to which the routing decisions for X jobs are taken. Saying that X jobs are observable, we mean that at all times their numbers in both queues (possibly including a job that is in service) and their individual positions are
8.1 Model

known. The number of X jobs is denoted by \( X(t) = (X_1(t), X_2(t)) \), with \( X_i(t) \) denoting the number of X jobs in queue \( i \). The position of a job in a queue is the number of jobs that must be served before finishing the given job, thus a job currently in service is at position 1. We are especially interested in the position of the last X job in the queue, which will be denoted by \( L^X(t) = (L^X_1(t), L^X_2(t)) \). If there are no X jobs present in one of the two queues, we set the corresponding component of \( L^X \) to 0.

- Jobs of the other type, Y jobs, represent background traffic. They are non-controllable and non-observable. Y jobs arrive according to a Poisson process of rate \((1 - \alpha)\lambda\) and join one of the two queues according to the static probabilities \( p^Y = (p^Y_1, p^Y_2) \), where \( p^Y_2 = 1 - p^Y_1 \). For the router the number of Y jobs in each queue, which will be denoted by \( Y(t) = (Y_1(t), Y_2(t)) \), is unknown, thus leaving the total queue lengths, denoted by \( Q(t) = (Q_1(t), Q_2(t)) \), with \( Q_i(t) := X_i(t) + Y_i(t) \), unknown as well.

The load on the queues depends on the routing policy and the penetration level \( \alpha \). In order to conduct comparisons for \( \alpha \) ranging from 0 to 1, we will measure the load in each queue by the actual load when \( \alpha = 0 \). For queue \( i \) this is given by

\[
\rho^Y_i := \frac{p^Y_i \lambda}{\mu_i}.
\]

When \( \rho^Y_1 = \rho^Y_2 \) we use \( \rho^Y \) to denote either of them. For any choice of parameters for which the load in both queues is below 1, it will thus be possible for the router to maintain the queue stable for all \( \alpha \in [0, 1] \).

8.1.2 Routing policies

In order to route incoming X jobs to a queue, one can think of two types of policies: dynamic policies that use state information and static policies that do not. We consider the four dynamic policies introduced below.

- **Number of X jobs** (denoted by \( \#X \)): this policy sends an arriving X job to the queue with the fewest X jobs (including a job currently in service).
Chapter 8 Routing under partial observability and controllability

- **Last X position** (abbreviated to LXP) compares the queues by the position of the last X job and sends an arriving X job to the queue with the lowest last X position.

- **Weighted last X position** (w-LXP) multiplies the last X position by the mean service time $1/\mu_i$ (giving an estimation of the sojourn time for the queue $i$) and sends an arriving X job to the lowest weighted last X position.

- **Estimated weighted last X position** (ew-LXP) does the same as w-LXP except that it does not assume $\mu$ to be known. The mean service time is estimated by dividing the sojourn times of the X jobs lastly departed from queue $i$ by their arrival position. The average is over the last $w$ number of departed X jobs, with $w$ (the window size) a parameter of the policy.

For most of these policies no system information (e.g. the values of $\alpha$, $\lambda$, $\mu$, $p_Y$) is assumed to be known by the router. Only in w-LXP the values of $\mu$ are used. The first two policies are designed for symmetric queues (equal service rates), the last two for the asymmetric case.

In Section 8.2 we compare the performance of the above-mentioned policies with respect to the expected sojourn time of the jobs (the total time of a job in the system, also called response time). Let $S_i$ denote the expected sojourn time in queue $i$ and $S_i^X$ the expected sojourn time in queue $i$ over X jobs only. We use the same symbols for the average sojourn times in the simulations. The averages over both queues are denoted by $S$ and $S^X$, respectively for all and for X jobs only. In the policy comparison we also consider the following two reference policies:

- **Weighted join-the-shortest-queue** (w-JSQ): a dynamic policy that sends an arriving X job to the queue with the fewest jobs (X and Y jobs together). In case of unequal service times, the number of jobs is weighted by the average service time, as for w-LXP. This reference policy assumes full state information (that is, also Y jobs are observable) and knowledge of $\mu$, but no other system information. Since w-JSQ uses more information than our policies, it will likely outperform our policies. Note that it may not be optimal for our partially controllable system, although it is optimal for fully controllable systems [108].
• A static policy that applies probabilistic routing. It sends arriving X jobs to queue \(i\) with a fixed probability \(p_i^X\). The probabilities \(p_i^X\) are set such that the difference between \(S_1^X\) and \(S_2^X\) is minimised. The policy thus has the same goals as the policies discussed before: it aims at equalising the sojourn times in both queues. It can be verified though, that this policy is not equal to the static policy that minimises \(S\). Although the static policy uses no state information, it may not always be outperformed by the dynamic policies, because the static policy uses system parameters \((\alpha, \lambda, \mu, p_Y)\) that are not (all) available to the dynamic policies. At the end of this section we show how to determine the routing probabilities and the sojourn times for the static policy.

Although all six policies introduced above assume partial controllability (i.e. only X cars can be routed), they differ in terms of the system and state information that is used. See Table 8.1 for a comparison. In some specific cases some of the policies are equivalent, that is, they make the same routing decisions. Obviously, LXP and w-LXP are the same if \(\mu_1 = \mu_2\). Also, for \(\alpha = 0\), all policies are equal, since there are no jobs that can be controlled. For \(\alpha = 1\), \#X and LXP are the same, since the number of jobs and the last position are equal if there are only X jobs. In addition, in this case (for \(\alpha = 1\)), w-LXP and w-JSQ are the same, since there is full observability for all dynamic policies. For the static policy there is no difference between \(S_i\) and \(S_i^X\) (for fixed \(i\)). Note that this is not the case for the dynamic policies (provided that \(\alpha < 1\)), because for these policies the routing of an X job to a specific queue and the length of the queue are not independent.

### Analysis of the static policy

For the static policy (with routing probabilities \(p_i^X\) for the X jobs), the system of Figure 8.1 consists of two independent \(M/M/1\) queues with arrival rate \(\lambda_i := \lambda(\alpha p_i^X + (1 - \alpha)p_Y^i)\) for queue \(i\). As a consequence, the expected sojourn time for queue \(i\) is given by \(S_i = (\mu_i - \lambda_i)^{-1}\). Since the arrivals of X jobs at queue \(i\) for this probabilistic routing policy follow a Poisson process, we have \(S_i^X = S_i\), due to the PASTA property.

In order for each X job to be sent to the queue that minimises its expected sojourn time, \(p_i^X\) is chosen such that \(S_1^X = S_2^X\) (or \(p_i^X \in \{0, 1\}\)
Chapter 8  Routing under partial observability and controllability

<table>
<thead>
<tr>
<th>Policy</th>
<th>#X symmetric</th>
<th>LXP asymmetric</th>
<th>w-LXP</th>
<th>ew-LXP</th>
<th>w-JSQ</th>
<th>static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial controllability</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Partial observability</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>Uses state information</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>number of X jobs</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>position of X jobs</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>state history</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Uses system information</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\mu$</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\lambda, \alpha, p^Y$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 8.1. A comparison of the information used by the routing policies under consideration

if it is impossible to reach the desired equality. This gives

$$p_1^X = (\mu_1 - \mu_2 + \lambda - 2p_1^Y (1 - \alpha)\lambda)/(2\alpha\lambda). \quad (8.2)$$

8.2 Simulations

In this section we investigate whether the dynamic policies with partial observability and controllability described in Section [8.1.2] perform better than the static policy and how close their performance (in terms of average sojourn times) is to that of w-JSQ. Furthermore, we want to know what is the added value of using extra state and/or system information and how the performance depends on the penetration level $\alpha$. To this end, we have performed numerous simulations under various system settings with symmetric and asymmetric service times and background traffic, and different loads and penetration levels $\alpha$.

The performance of the policies is compared in Section [8.2.1]. The main (rather surprising) observation is that the performance of the policy w-LXP approaches that of w-JSQ (the optimal policy under full control) when the load becomes large. We continue the discussion of the simulation results by a section (Section [8.2.2]) on the influence of the penetration
level $\alpha$. It turns out that it is often sufficient to have a low penetration level, because the performance as a function of $\alpha$ is steep for low values and flatter for higher values. This suggests that dynamic control (in for example road traffic) is advantageous even if the percentage of controlled jobs (or cars in the example) is small.

8.2.1 Policy comparison

In this section we distinguish two cases, symmetric queues (equal service rates) and asymmetric queues (unequal service rates).

Symmetric queues

We start by considering the case of equal service rates, $\mu_1 = \mu_2$, see Figure 8.2 for an example (there $\mu_1 = \mu_2 = 1$). In Figure 8.2a we plot the load $\rho^Y := \rho_1^Y = \rho_2^Y$ as defined by (8.1) on the x-axis. We compare the two simple dynamic policies based on the number of X jobs (#x) and the positions of the last X jobs (LXP) in the two queues, with the static policy and the full state information policy JSQ. To facilitate the comparison, the sojourn times are depicted relative to those of JSQ (by definition, JSQ thus produces a horizontal line at 1). Every data point in this plot, as well as in the following plots, is the result of a simulation experiment with 10 million events (job arrivals and departures, that is).

The performance of LXP is always slightly better than that of #x. This can be explained by the fact that LXP also takes into account the Y jobs that have arrived before the last X job, while #x only measures the number of X jobs. The relative difference between the two policies is largest for moderate to high loads. Under very high loads ($\rho^Y \uparrow 1$) the three dynamic policies show comparable performance. We explain this fact in Section 8.3 for LXP and JSQ.

Recall that JSQ uses full state information, i.e. also including knowledge of Y jobs. Although for $\alpha < 1$ it is not known to be optimal, we may expect it to perform very well. Figure 8.2a shows that in heavy traffic, the policies #x and LXP are in fact almost as good as JSQ. This is desirable, because the need for good policies is highest for high loads, when delays are largest. A striking observation — for which we have not found an explanation — is that the relative performance of the static policy is
(a) Relative average sojourn time over all X jobs in both queues as a function of $\rho^Y$ for several policies, $\alpha = 0.2$ and $p^Y = (1/2, 1/2)$. The graph displays the relative sojourn times with respect to those of w-JSQ.

(b) Average sojourn time over both queues as a function of $\alpha$ for several policies, $\rho_1^Y = 0.45$, $\rho_2^Y = 0.9$ and $p^Y = (1/3, 2/3)$. We display both the average over all jobs (solid) and over X jobs only (dashed).

Figure 8.2. Simulation results for $\mu = (1, 1)$

practically linear.

The performance of the static policy is clearly inferior to that of the dynamic policies. This conclusion is not true when the Y traffic is not evenly spread over both queues (that is $p_1^Y / \mu_1 \neq p_2^Y / \mu_2$), as can be observed in Figure 8.2b. In that figure we have fixed the load $\rho_1 = \rho_2$ and on the x-axis we vary the penetration level $\alpha$. For low $\alpha$, it is better for the overall average sojourn time to apply the static policy. Note that the dynamic policies are always better for the X jobs, and for all jobs when the penetration level $\alpha$ is relatively large.

Asymmetric queues

We now allow for unequal service rates for the two queues. In this case it seems useful to use a weighted dynamic policy that takes the service rates into account. Figure 8.3a shows a similar set of experiments as in Figure 8.2a, but now for asymmetric service rates. We observe that in this case (non-weighted) LXP also gives better results than $\#X$ for asymmetric queues. We therefore focus on LXP from now on, and consider its weighted versions w-LXP and ew-LXP. In the former the weights are given, in the latter they are estimated using data of the w lastly departed
8.2 Simulations

(a) Relative average sojourn time over all jobs and both queues as a function of $\rho^Y$ for several policies, $\alpha = 0.2$ and $p^Y = (1/3, 2/3)$. The sojourn times are relative to those of w-JSQ.

(b) Average sojourn time over all jobs and both queues as a function of $\alpha$ for several policies, $\rho_1^Y = \rho_2^Y = 0.9$ and $p^Y = (1/3, 2/3)$

Figure 8.3. Simulation results for $\mu = (1, 2)$

jobs for each queue separately. We show results for values $w = 1$ and $w = 10$ in our graphs (denoted by ew-LXP(1) and ew-LXP(10) in the legends).

From the simulations in Figure 8.3a we see that, as expected, the weighted policies (w-LXP and ew-LXP) outperform the unweighted LXP. This turns out to be true across all values of $\alpha$ as can be observed in Figure 8.3b. For practically relevant values of $\alpha$ (that is, relatively small values) this even holds for a small window size of $w = 1$.

In practice the mean service times may vary over time. In this case the policy ew-LXP is particularly relevant. The recommended window size depends on the desired precision and on how fast $\mu$ is changing; the memory of the policy should refresh itself on a smaller time scale than the one on which $\mu$ changes. For a static $\mu$, a window size $w = 10$ gives relatively accurate results as can be seen in Figure 8.3a. In general, in order to get the same precision, $w$ needs to be larger for lower loads (see Figure 8.3a), because the queue lengths are smaller and therefore the average is taken over a smaller number of jobs. In addition, we observe in Figure 8.3b that $w$ needs to be larger for higher penetration levels (because the estimates of the average service time are using more overlapping data when there are fewer Y jobs in the system).
Figure 8.3 refers to the case in which both queues are equally loaded by the Y jobs (that is, $p_Y^1 = p_Y^2$). Note that this does not imply that the queues are symmetric, as the difference between the $\mu_i$ can be compensated by the $p_Y^i$. The observations turned out (in simulations not presented here) to be valid also for unequally loaded queues (e.g. if $p_Y$ is symmetric, while $\mu$ is not). We did notice, however, that then the static policy outperforms our dynamic policies for low values of $\alpha$ (as we also observed from Figure 8.2b). So in that case the dynamic policies are only recommended if it is not possible to implement the static policy (because of a lack of system information). On the other hand, for the equally loaded scenario the weighted policies perform much better than the static policy across all $\alpha$ and, moreover, the w-LXP policy is close to the ‘optimal’ w-JSQ when the load $\rho_Y$ approaches 1 (see Figure 8.3a). As mentioned before, we explain this fact in Section 8.3.

8.2.2 The penetration level

An interesting question from an application perspective is what penetration level $\alpha$ is needed for effective control. To shed light on this question, we now study the performance of the various policies as a function of $\alpha$. From the figures presented in the preceding sections (Figures 8.2b and 8.3b) we can conclude that the sensitivity of the performance to the penetration level is highest for small values of $\alpha$ (i.e. changes in $\alpha$ cause a larger change in the performance when $\alpha$ is small than when it is large), which can be understood as a ‘law of diminishing marginal returns’. In most of the scenarios, the average sojourn time for w-LXP and ew-LXP is already rather close to its minimum if $\alpha$ is approximately 25%. The required penetration level depends on the system load; for lower loads a higher $\alpha$ is required and if the load caused by the Y jobs is symmetric, a lower $\alpha$ suffices than if it is asymmetric (because more X jobs are needed to compensate the ‘wrong’ choices of the Y jobs). In the remainder of this section, we analyse in detail the behaviour of the policies as a function of $\alpha$ by considering the two queues separately. Figure 8.4 illustrates the analysis.

First, let us inspect the average sojourn times $S_1$, $S_2$ for all jobs (X and Y together). From Figure 8.4a, which is representative for a large collection of graphs for a variety of scenarios, we see that for the
8.2 Simulations

![Graphs showing simulated average sojourn time as a function of α for both queues separately.](image)

(a) All jobs

(b) X jobs only

**Figure 8.4.** Simulated average sojourn time as a function of α for both queues separately, \( \rho_1^Y = \rho_2^Y = 0.9 \), \( \mu = (1, 2) \) and \( p^Y = (1/3, 2/3) \)

dynamic policies the average sojourn time in both queues decreases when \( \alpha \) increases. This means that increasing the amount of control decreases the average time a job spends in the system, independently of the queue a job is sent to. In contrast, for the static policy the sojourn time in one queue increases when more jobs are being controlled (while the sojourn time in the other decreases as a function of \( \alpha \)). As one can notice from Figure 8.3b this may result in non-monotonous average sojourn times over both queues. However, for the dynamic policies it is always beneficial to the overall average sojourn time to increase the amount of control.

Let us finally look at the X jobs only; Figure 8.4b shows an example. First note that the performance for X jobs is always better than for an average (X or Y) job. This is essential if jobs can choose themselves whether to be in the X class (as in the road traffic example). For the static policy, the behaviour as a function of \( \alpha \) is the same as before (when looking at both classes together). However, for w-JSQ the quantities \( S_1^X \) and \( S_2^X \) are now both increasing in \( \alpha \). For this policy, increasing the percentage of controlled jobs improves the overall performance, but deteriorates the performance for the controlled jobs, because the advantage of being directed to the ‘right’ queue becomes smaller if more jobs are being directed to it. This reasoning does not hold for the other dynamic
Chapter 8 Routing under partial observability and controllability

policies, because for those policies increasing $\alpha$ does not only mean more control (which is disadvantageous for the controlled jobs), but also more information (which is advantageous). Consequently, the dynamic policies may show non-monotonous behaviour for one of the queues. In fact, in Figure [8.4b] in one queue the average sojourn time of $X$ jobs for $w$-LXP and $ew$-LXP first decreases when $\alpha$ is increased, but mildly increases for larger values of $\alpha$. In the other queue it decreases as a function of $\alpha$.

8.3 Fluid model approximation

The policies that we have proposed and studied through simulation do not allow exact (analytical) performance analysis. In this section we approximate the behaviour of some of the policies under high loads using deterministic fluid models. For the deterministic fluid models we show that the partial information exploited by $w$-LXP is sufficient to give dynamics that yield just as good a performance as the fluid approximation of the ‘optimal’ $w$-JSQ policy. This result confirms the (in Section 8.2.1 observed) convergence of $w$-LXP to $w$-JSQ for loads approaching 1. We also show that the fluid model for static routing does not generate the same dynamics, making it less efficient than dynamic routing. A fluid approximation can often be shown to be the limiting process (fluid limit) of the original stochastic system under an appropriate scaling [86]. Here we do not aim at a technical proof of this limiting procedure, but rather at the analysis of the fluid approximations themselves. Instead of a technical proof, we provide a numerical validation study.

At a high level of abstraction, the fluid approximation may be seen to mimic the dynamics of the stochastic process at large system states; i.e. when the queueing processes move far away from the origin. Consequently, when the system load is low the applicability is limited to transient performance analyses, because in such a case large system states are rarely visited and therefore the fluid approximation does not reflect the typical behaviour of the process. When the system load is high, however, the stochastic process does typically move far away from the origin and the analysis of the fluid approximation provides valuable insight into the stationary behaviour of the stochastic process as well.

Our main goal in this section is thus to provide an approximative
analysis of the performance of different policies under high-load conditions. Section 8.3.1 is devoted to a preliminary discussion describing 1) the Markov processes under consideration, 2) the conditions for and consequences of high traffic loads, and 3) the definition of a fluid limit. In Section 8.3.2 we propose and analyse fluid approximations for these Markov processes. The last section, Section 8.3.3, provides numerical evidence that the proposed deterministic approximations indeed provide valuable insight into the dynamics and performance of several policies in the stochastic setting.

8.3.1 Description of the setting

Since Little’s Law applies to the queue lengths and the sojourn times irrespective of the policy, it is sufficient to concentrate on queue length dynamics to obtain the expectations of the sojourn time (system delay). Clearly under all of the policies presented in Section 8.1.2 it is possible to describe the joint queue length processes as a Markov process if the state descriptor contains both queue lengths, and the positions of all jobs, of both types, plus some information about the past. Note that indeed, for some policies, there is a dependence of the dynamics on the positions of X jobs and/or the current value of a parameter, which is an estimate based on past system states. Therefore, the state space can be rather complicated in general. We focus our analysis on the w-JSQ, w-LXP and static routing policies. For these policies the random vector

\[ M(t) := (Q_1(t), Q_2(t), L_{X1}^1(t), L_{X2}^2(t)) \]

is Markovian. Let the Markov processes for the three policies be denoted by \( M^J \), \( M^L \) and \( M^{stat} \) respectively.

The condition

\[ \lambda/(\mu_1 + \mu_2) < 1, \tag{8.3} \]

is necessary for these processes to be stable. In addition, we assume that \( \alpha \) to be sufficiently large, such that with appropriate routing, it is possible to keep both queues from getting overloaded. Besides (8.3), we therefore require the load of Y jobs on each queue to be below 1, i.e.

\[ p_1^Y (1 - \alpha) \lambda < \mu_i, \tag{8.4} \]
for both $i = 1, 2$. The system is said to be in heavy traffic if the boundaries of condition (8.3) are approached, i.e. if $\lambda \uparrow \mu_1 + \mu_2$. If we impose that the system reaches heavy traffic, while satisfying (8.4) for both queues, we need

$$\alpha \geq 1 - \frac{1}{p_i^Y} \frac{\mu_i}{\mu_1 + \mu_2},$$

for both $i = 1, 2$. In particular, if $\alpha$ is 0, it must be that $p_i^Y \lambda \uparrow \mu_i$ for both queues simultaneously, so that $\mu_2 p_1^Y = \mu_1 p_2^Y$.

Although we do not consider heavy-traffic limits, we assume that the system load is high, that is, $\lambda$ is close to $\mu_1 + \mu_2$. Under such high-load conditions, the number of jobs present in either queue will typically be very large and any substantial change in the queue lengths requires such a long time that the arrivals of both types virtually occur in a continuous fashion. This implies that in at least one of the two queues, there must be an X job near the end of the queue. More precisely, for at least one of the two queues, the number of Y jobs standing behind the last X job in line is negligible compared to the total number of jobs in the queue. This intuitive reasoning forms the rationale behind the fluid approximations proposed in Section 8.3.2. We conjecture that these (deterministic) fluid processes are the so-called fluid limits of the original (stochastic) processes.

To facilitate the discussion, we end this section with the definition of a fluid limit, which formally describes the chosen scaling. If $\{M(c)(t), t \geq 0\}_{c \in \mathbb{N}}$ is a sequence of Markov processes with $|M(c)(0)|_1 = c$, we define

$$\bar{M}(c)(t) := \frac{M(c)(ct)}{c}.$$

The fluid limit of this sequence is obtained by letting $c \to \infty$. Under suitable conditions, the limit often turns out to be a deterministic fluid process [86]. We believe (and prove numerically) that this is also the case for the three processes ($M^J$, $M^L$ and $M^{stat}$) of our interest.

### 8.3.2 Introduction and analysis of the model

We now propose deterministic fluid processes $m^J$, $m^L$ and $m^{stat}$ as approximations to the stochastic processes $M^J$, $M^L$ and $M^{stat}$ respectively,
under high-load conditions. Each of these processes consists of four components and, similarly to before, we generically write

\[ m(t) := (q_1(t), q_2(t), l_1(t), l_2(t)) . \]

For w-JSQ the evolution satisfies the following system of ordinary differential equations (ODEs). For conciseness we only report the derivatives for positive \( q_i \) and \( l_i \); at level zero the negative term in the derivative is to be removed, since there are no departures in this case.

\[
\frac{d}{dt} q_1^J = \begin{cases} 
\alpha \lambda + p_1^Y (1 - \alpha) \lambda - \mu_1, & \text{if } q_1^J / \mu_1 < q_2^J / \mu_2, \\
\alpha \lambda / (\mu_1 + \mu_2) + p_1^Y (1 - \alpha) \lambda - \mu_1, & \text{if } q_1^J / \mu_1 = q_2^J / \mu_2, \\
p_1^Y (1 - \alpha) \lambda - \mu_1, & \text{if } q_1^J / \mu_1 > q_2^J / \mu_2, 
\end{cases}
\]

\[
\frac{d}{dt} q_2^J = \begin{cases} 
\alpha \lambda / (\mu_1 + \mu_2) + p_2^Y (1 - \alpha) \lambda - \mu_2, & \text{if } q_1^J / \mu_1 = q_2^J / \mu_2, \\
\alpha \lambda + p_2^Y (1 - \alpha) \lambda - \mu_2, & \text{if } q_1^J / \mu_1 > q_2^J / \mu_2, 
\end{cases}
\]

\[
\frac{d}{dt} l_1^J = \begin{cases} 
\alpha \lambda + p_1^Y (1 - \alpha) \lambda - \mu_1, & \text{if } q_1^J / \mu_1 < q_2^J / \mu_2, \\
- \mu_1, & \text{if } q_1^J / \mu_1 > q_2^J / \mu_2, 
\end{cases}
\]

\[
\frac{d}{dt} l_2^J = \begin{cases} 
- \mu_2, & \text{if } q_1^J / \mu_1 < q_2^J / \mu_2, \\
\alpha \lambda + p_2^Y (1 - \alpha) \lambda - \mu_2, & \text{if } q_1^J / \mu_1 > q_2^J / \mu_2. 
\end{cases}
\]

These equations reflect that newly arriving X jobs are routed to the weighted shortest queue. Note that the policy’s rule in case the weighted queue lengths are equal is irrelevant, because any tie breaking rule forces the process to move along states with \( q_1^J / \mu_1 = q_2^J / \mu_2 \). It is worth emphasising that the last-X component of the weighted shortest queue also contains the effect of Y arrivals: both types of arrivals occur in a continuous fashion and are interleaved at the weighted shortest queue. Y jobs thus push the last X position further back.

The above system lacks a description for the behaviour of the components \( l_i^J \) when the process hits the switching (hyper)plane \( q_1^J / \mu_1 = q_2^J / \mu_2 \). We argued in Section 8.3.1 that for one of the two queues the last X position must be (almost) equal to the queue length. For w-JSQ this is the case for the weighted shortest queue. Indeed, new X jobs are routed
to the weighted shortest queue and the number of Y jobs arriving in-between two arriving X jobs is negligible compared to the queue lengths. As a consequence, whenever the switching plane \( q_1/\mu_1 = q_2/\mu_2 \) is hit, the last X position of the largest weighted queue also moves to the level of the corresponding queue length. Therefore we have to supplement the above ODEs with the following jumps:

\[
I_i(t^+) = q_i(t), \quad \text{for both } i, \quad \text{if } q_i/\mu_i(t) = q_j/\mu_j(t).
\]

Here, by \( t^+ \) we mean immediately after time \( t \).

We now turn our attention to an approximating deterministic fluid system for \( w_{\text{LXP}} \). Much of the discussion holds in this case as well. The difference is due to the fact that the router now chooses the queue with the lowest weighted last X position and only switches to the other queue when the weighted last X positions are equal. Thus the ODEs characterising the evolution of \( m_L \) are — similarly to (8.6) — as follows:

\[
\frac{d}{dt} q_1 = \begin{cases} 
\alpha \lambda + p_Y^1 (1 - \alpha) \lambda - \mu_1, & \text{if } l_1/\mu_1 < l_2/\mu_2, \\
\alpha \lambda \mu_1/\mu_2 + p_Y^1 (1 - \alpha) \lambda - \mu_1, & \text{if } l_1/\mu_1 = l_2/\mu_2, \\
p_Y^1 (1 - \alpha) \lambda - \mu_1, & \text{if } l_1/\mu_1 > l_2/\mu_2,
\end{cases}
\]

\[
\frac{d}{dt} q_2 = \begin{cases} 
\alpha \lambda \mu_2/\mu_2 + p_Y^2 (1 - \alpha) \lambda - \mu_2, & \text{if } l_1/\mu_1 < l_2/\mu_2, \\
\alpha \lambda p_Y^2 (1 - \alpha) \lambda - \mu_2, & \text{if } l_1/\mu_1 = l_2/\mu_2, \\
p_Y^2 (1 - \alpha) \lambda - \mu_2, & \text{if } l_1/\mu_1 > l_2/\mu_2.
\end{cases}
\]

\[
\frac{d}{dt} I_1 = \begin{cases} 
\alpha \lambda + p_Y^1 (1 - \alpha) \lambda - \mu_1, & \text{if } l_1/\mu_1 < l_2/\mu_2, \\
- \mu_1, & \text{if } l_1/\mu_1 > l_2/\mu_2,
\end{cases}
\]

\[
\frac{d}{dt} I_2 = \begin{cases} 
- \mu_2, & \text{if } l_1/\mu_1 < l_2/\mu_2, \\
\alpha \lambda + p_Y^2 (1 - \alpha) \lambda - \mu_2, & \text{if } l_1/\mu_1 > l_2/\mu_2.
\end{cases}
\]

As before, if queue \( j \) has the lowest weighted last X position at time \( t \), we must have \( I_j(t) = q_j(t) \). The jumps on the switching plane are now given by

\[
I_i(t^+) = q_i(t), \quad \text{for both } i, \quad \text{if } I_i/\mu_i(t) = l_j/\mu_j(t).
\]

After a transient period the deterministic fluid approximations for both
policies w-JSQ and w-LXP live on their switching planes, where respectively \( q_1/\mu_1 = q_2/\mu_2 \) and \( l_1/\mu_1 = l_2/\mu_2 \). On the switching planes, jobs are sent to both queues, so that the last X positions are updated continuously and are equal to the queue sizes. It follows that (after a transient period) both fluid processes live on the same line \( (q_1/\mu_1 = l_1/\mu_1 = q_2/\mu_2 = l_2/\mu_2) \) and take the same decisions, both equalising the load of the two queues.

For comparison, we also discuss the corresponding fluid approximation for the static policy. In this case the dynamics are very different; since there are no routing decisions the ODE system is much simpler:

\[
\begin{align*}
\frac{dq_1^{\text{stat}}}{dt} &= p_1^X \alpha \lambda + p_1^Y (1 - \alpha) \lambda - \mu_1, \\
\frac{dq_2^{\text{stat}}}{dt} &= p_2^X \alpha \lambda + p_2^Y (1 - \alpha) \lambda - \mu_2, \\
\frac{dl_1^{\text{stat}}}{dt} &= p_1^X \alpha \lambda + p_1^Y (1 - \alpha) \lambda - \mu_1, \\
\frac{dl_2^{\text{stat}}}{dt} &= p_2^X \alpha \lambda + p_2^Y (1 - \alpha) \lambda - \mu_2.
\end{align*}
\]  

Here the static routing probabilities \( p_1^X \) and \( p_2^X = 1 - p_1^X \) are given by (8.2). Since X jobs are routed to both queues regardless of the state of the system, the last X positions clearly follow the queue lengths. Filling in the routing probabilities of (8.2), shows us that all derivatives are equal to \((\lambda - \mu_1 - \mu_2)/2\). As a consequence, under the static policy, the trajectories of the fluid process for different initial states are parallel to each other and most of the trajectories hit the axes away from the origin. This is reflected by the fact that one of the queues empties before the other, thus leaving one of the servers idling, which indicates an inefficient use of capacity.

### 8.3.3 Numerical verification

In this section we describe the simulations conducted to verify the suitability of the deterministic fluid approximations for large system loads. When the service rates are different, we observe the same behaviour as with equal \( \mu \)'s (after applying a correction for the different service rates). Here we therefore limit ourselves to the symmetric case \( \mu_1 = \mu_2 \). We set the initial queue lengths and last X positions intentionally away from the
Chapter 8 Routing under partial observability and controllability

Figure 8.5. Simulated evolution for a load $\rho^Y$ of 0.95, $\alpha = 0.8$, $\mu = (1, 1)$ and $p^Y = (1/2, 1/2)$. In [a] and [b] the (red) curve for the number of (X plus Y) jobs in queue 2 and the (purple) curve for the position of the last X job in this queue coincide. After an initial period all four curves coincide. In [c] the curves for the number of jobs in a queue and the position of the last X job in the same queue coincide.

switching curve, whilst choosing $\alpha$ such that it satisfies (8.5).

In our first set of experiments (see Figure 8.5) we illustrate the appropriateness of the linear deterministic approximations by plotting the trajectories over a very long time scale and at very large system states. We only plot the simulation curves, as the deterministic approximations exactly match those. In these experiments we fixed $\alpha = 0.8$, $\mu = (1, 1)$, $p^Y = (1/2, 1/2)$ and $\rho^Y = \rho^Y_1 = \rho^Y_2 = 0.95$.

In Figure 8.5a we plot the trajectories of the queueing processes under
8.3 Fluid model approximation

As long as the two queues have different sizes, all jobs are routed to the shortest queue (queue 2 in this experiment); the last X position in queue 2 is therefore equal to the queue length in queue 2. For queue 1 we see that the last X position decreases faster than the queue length, since new Y jobs are all placed after the last X job. The position of the last X job may hit zero and remain at zero until the two queues meet. At that point the last X position in queue 1 increases instantly to become equal to the queue length and from then on both queue lengths and the last X positions remain coupled indefinitely.

In Figure 8.5b we do the same for LXP. Again we start off at a very large state for all components, with queue 2 having the lowest queue length and the lowest last X-position (these become instantly equal), i.e. the router is sending jobs to queue 2. As soon as the last X position of queue 1 has dropped to that of queue 2, it is increased to the size of queue 1. After a single X job has been sent to queue 1, the next X jobs are being sent to queue 2 again. This pattern repeats until the sizes of the two queues meet, from that point on all four components will have one single value. Note that, since the curve for the last X position in queue 1 can only bounce at a countable number of points, basically all X jobs have been routed to queue 2, as was the case for JSQ in Figure 8.5a. The point where the two queues meet is therefore the same for both policies.

For comparison, we also plot the trajectories for the static policy in Figure 8.5c. In this case the last X positions and the corresponding queue lengths are always identical because X jobs are continuously routed to both queues. We see that the trajectories are very different from those of JSQ and LXP. The queue lengths deplete in a parallel fashion (since they have constant and equal derivatives), causing one of the queues to empty before the other one.

Figure 8.5 considers extremely large system states, which will only rarely be visited. In Figure 8.6 we have plotted the trajectories for JSQ and LXP under a higher load of 0.995 and for system states that are more likely to be reached. We see that the trajectories meet quickly, after which the joint trajectories are well approximated by a linear trend.

Our numerical experiments demonstrate that indeed the proposed fluid limit approximations closely follow the trajectories of w-JSQ, w-LXP and the static policy at large system states under high loads. The simulations also corroborate our observations in Section 8.2, where we noticed that
Figure 8.6. Simulated evolution for a load $\rho^Y$ of 0.995, $\alpha = 0.6$, $\mu = (1, 1)$ and $p^Y = (3/5, 2/5)$. All curves coincide after a very brief initial period.

in heavy traffic w-LXP performs similarly to w-JSQ and outperforms the static policy.

8.4 Conclusion

In this chapter we have investigated routing policies for a two-server queueing system in which part of the jobs (the X jobs) can be observed and controlled (routed to one of the queues), while the other jobs (the Y jobs) act as background traffic which is neither observable nor controllable. The choice of the system is motivated by dynamic road traffic control. In fact, routing traffic based on partial observability and controllability is an important challenge in traffic control by smartphone applications and online navigation systems, because these applications only have access to their own users.

An extensive simulation study revealed that the sojourn time as a function of the penetration level (i.e. the percentage of X jobs) declines fast for small penetration levels, but slowly for large levels. As a consequence, with only a small percentage of controllable jobs (say about 25%) it is possible to obtain an average sojourn time which is close to the minimum value.

We also observed in the simulations that a simple policy that sends an arriving X job to the queue with the fewest X jobs performs quite well (in terms of the average sojourn time). If both queues are at least
8.4 Conclusion

moderately loaded, it is much better than a static policy that routes according to fixed routing probabilities using no state information, but full information about the system parameters. A policy that does not base its decisions on the number of X jobs in each of the queues, but on the position of the last X job (that is, its distance to the server in number of jobs) performs even slightly better. Obviously, if the service rates of the two queues are unequal, the last X positions of the queues need to be weighted by their average service times. If this average is unknown, or varying over time, an estimate of it based on the measured sojourn time of only a few recently departed X jobs yields almost as good a performance.

When the load of both queues grows, the performance of the weighted last X position policy (w-LXP) becomes identical to that of weighted join-the-shortest-queue (w-JSQ), the optimal policy for fully controllable systems. We have explained this remarkable result analytically by means of deterministic dynamical systems approximating the stochastic processes under high loads. Indeed, after a transitory phase, the dynamical systems corresponding to w-LXP and w-JSQ have identical trajectories for the queue lengths, whatever the initial conditions are. A similar analysis also explains the deficiencies of the static policy, as it fails to balance the load in the two queues, leading to considerable idling of one server while the other server is busy. We conjecture the analysed dynamical systems to be the fluid limits of the original processes and have provided numerical evidence for the convergence.

Note that the routing policies proposed and investigated in this chapter for the special case of two partially observable queues can be extended to similar systems with more than two queues. This can serve as a model of a road network with multiple possible paths, for instance.

Further research on the topic could take various directions. One possibility is to relax the modelling assumptions, by investigating the performance of the proposed routing policies under non-exponential service times at the queues. Other relaxations or changes in the model could concern non-stationary inputs, state-dependent service rates or service disciplines that are different from FIFO. These are certainly interesting from an application standpoint, for example in an urban road setting there can be substantial differences between the load at rush hour and during the night, and the throughput of the network depends on the
Chapter 8 Routing under partial observability and controllability

density of the vehicles.

Other routing policies could also be considered, for instance policies that take into account more system information, such as the arrival rate of Y-jobs, or estimate the corresponding parameters. However, given the results of the w-LXP and w-JSQ policies, it is to be expected that, especially under high loads, these policies would not significantly improve the performance. An obvious last direction for further research is to formally prove that the proposed deterministic dynamical systems are in fact the limits of the original stochastic processes under a so-called fluid scaling, possibly applying the fluid analysis to other routing policies as well.