http://pubsonline.informs.org/journal/opre/

OPERATIONS RESEARCH
Vol. 65, No. 5, September–October 2017, pp. 1398–1413
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Flexible Queueing Architectures

John N. Tsitsiklis,[a] **Kuang Xu**[b]

[a] LIDS, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139; [b] Graduate School of Business, Stanford University, Stanford, California 94305

Contact: jnt@mit.edu (JNT); kuangxu@stanford.edu, http://orcid.org/0000-0002-2221-1648 (KX)

Received: May 13, 2015
Revised: February 23, 2016; October 23, 2016
Accepted: January 25, 2017
Published Online in Articles in Advance: July 21, 2017

Subject Classifications: probability: stochastic model applications; queues: algorithms
Area of Review: Stochastic Models

https://doi.org/10.1287/opre.2017.1620

Abstract. We study a multiserver model with n *flexible* servers and n queues, connected through a bipartite graph, where the level of flexibility is captured by an upper bound on the graph's average degree, d_n. Applications in content replication in data centers, skill-based routing in call centers, and flexible supply chains are among our main motivations. We focus on the scaling regime where the system size n tends to infinity, while the overall traffic intensity stays fixed. We show that a large capacity region and an asymptotically vanishing queueing delay are simultaneously achievable even under limited flexibility ($d_n \ll n$). Our main results demonstrate that, when $d_n \gg \ln n$, a family of expander-graph-based flexibility architectures has a capacity region that is within a constant factor of the maximum possible, while simultaneously ensuring a diminishing queueing delay for *all* arrival rate vectors in the capacity region. Our analysis is centered around a new class of virtual-queue-based scheduling policies that rely on dynamically constructed job-to-server assignments on the connectivity graph. For comparison, we also analyze a natural family of modular architectures, which is simpler but has provably weaker performance.

Funding: This research was supported in part by the National Science Foundation [Grant CMMI-1234062].
Supplemental Material: The online appendix is available at https://doi.org/10.1287/opre.2017.1620.

Keywords: queueing • flexibility • dynamic matching • resource pooling • expander graph • asymptotics

1. Introduction

At the heart of a number of modern queueing networks lies the problem of allocating processing resources (manufacturing plants, web servers, or call-center staff) to meet multiple types of demands that arrive dynamically over time (orders, data queries, or customer inquiries). It is usually the case that a *fully flexible* or *completely resource-pooled* system, where every unit of processing resource is capable of serving all types of demands, delivers the best possible performance. Our inquiry is, however, motivated by the unfortunate reality that such full flexibility is often infeasible due to overwhelming implementation costs (in the case of a data center) or human skill limitations (in the case of a skill-based call center).

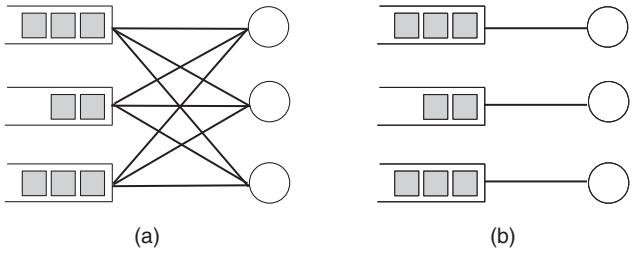
What are the key benefits of flexibility and resource pooling in such queueing networks? Can we harness the same benefits even when the degree of flexibility is *limited*, and how should the network be designed and operated? These are the main questions that we wish to address. While these questions can be approached from a few different angles, we will focus on the metrics of *capacity region* and *expected queueing delay*; the former measures the system's *robustness* against *demand uncertainties* (i.e., when the arrival rates for different demand types are unknown or likely to fluctuate over time), while the latter is a direct reflection of *performance*. Our main message is positive: in the regime where the system size is large, improvements in both the capacity region and delay are *jointly achievable* even under very limited flexibility, given a proper choice of the architecture (interconnection topology) and scheduling policy.

Benefits of Full Flexibility. We begin by illustrating the benefits of flexibility and resource pooling in a very simple setting. Consider a system of n servers, each running at rate one, and n queues, where each queue stores jobs of a particular demand type. For each $i \in \{1, \ldots, n\}$, queue i receives an independent Poisson arrival stream of rate λ_i. The average arrival rate $(1/n)\sum_{i=1}^{n} \lambda_i$ is denoted by ρ, and is referred to as the *traffic intensity*. The sizes of all jobs are independent and exponentially distributed with mean 1.

For the remainder of this paper, we will use a *measure of flexibility* given by the average number of servers that a demand type can receive service from, denoted by d_n. Let us consider the two extreme cases: a *fully flexible* system, with $d_n = n$ (Figure 1(a)), and an *inflexible* system, with $d_n = 1$ (Figure 1(b)). Fixing the traffic intensity $\rho < 1$, and letting the system size, n, tend to infinity, we observe the following qualitative benefits of full flexibility:

1. ***Large Capacity Region.*** In the fully flexible case and under any work-conserving scheduling policy,[1] the *collection of all jobs* in the system evolves as an $M/M/n$

Figure 1. Extreme Cases of Flexibility: $d_n = n$ vs. $d_n = 1$



queue, with arrival rate $\sum_{i=1}^{n} \lambda_i$ and service rate n. It is easy to see that the system is stable for all arrival rates that satisfy $\sum_{i=1}^{n} \lambda_i < n$. In contrast, in the inflexible system, since all $M/M/1$ queues operate independently, we must have $\lambda_i < 1$, for all i, to achieve stability. Comparing the two, we see that the fully flexible system has a much larger capacity region and is hence more robust to uncertainties or changes in the arrival rates.

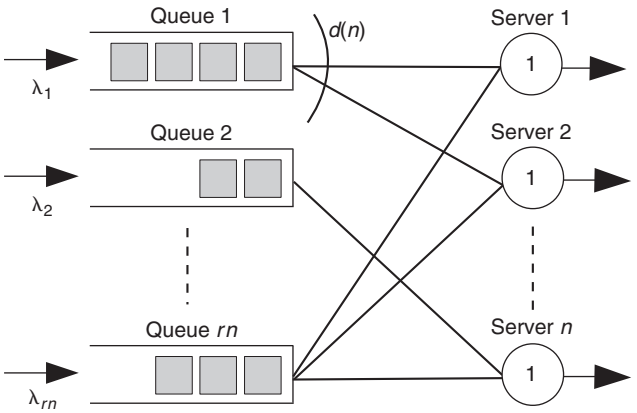
2. Diminishing Delay. Let W be the steady-state expected waiting time in queue (time from entering the queue to the initiation of service). As mentioned earlier, the total number of jobs in the system for the fully flexible case evolves as an $M/M/n$ queue with traffic intensity $\rho < 1$. It is not difficult to verify that for any fixed value of ρ, the expected total number of jobs in the queues is *bounded above* by a constant independent of n, and hence the expected waiting time in queue satisfies $\mathbb{E}(W) \to 0$, as $n \to \infty$.[2] In contrast, the inflexible system is simply a collection of n independent $M/M/1$ queues, and hence the expected waiting time is $\mathbb{E}(W) = \rho/(1-\rho) > 0$, for all n. Thus, the expected delay in the fully flexible system *vanishes asymptotically* as the system size increases but stays bounded away from zero in the inflexible system.

Preview of Main Results. Will the above benefits of fully flexible systems continue to be present if the system only has limited flexibility—that is, if $d_n \ll n$? The main results of this paper show that a large capacity region and an asymptotically vanishing delay can still be *simultaneously achieved*, even when $d_n \ll n$. However, when flexibility is limited, the architecture and scheduling policy need to be chosen with care. We show that, when $d_n \gg \ln n$, a family of expander-graph–based flexibility architectures has the largest possible capacity region, up to a constant factor, while simultaneously ensuring a diminishing queueing delay, of order $\ln n/d_n$ as $n \to \infty$, for *all* arrival rate vectors in the capacity region (Theorem 3.4). For comparison, we also analyze a natural family of modular architectures, which is simpler but has provably weaker performance (Theorems 3.5 and 3.6).

1.1. Motivating Applications

We describe here several motivating applications for our model; Figure 2 illustrates the overall architecture

Figure 2. A Processing Network with rn Queues and n Servers



that they share. *Content replication* is commonly used in data centers for bandwidth intensive operations such as database queries (Soundararajan et al. 2006) or video streaming (Leconte et al. 2012), by hosting the same piece of content on multiple servers. Here, a server corresponds to a physical machine in the data center, and each queue stores incoming demands for a particular piece of content (e.g., a video clip). A server j is connected to queue i if there is a copy of content i on server j, and d_n reflects the average number of replicas per piece of content across the network. Similar structures also arise in *skill-based routing in call centers*, where agents (servers) are assigned to answer calls from different categories (queues) based on their domains of expertise (Wallace and Whitt 2005), and in *process-flexible supply chains* (Jordan and Graves 1995, Simchi-Levi and Wei 2012, Chou et al. 2010, Iravani et al. 2005, Gurumurthi and Benjaafar 2003), where each plant (server) is capable of producing multiple product types (queues). In many of these applications, demand rates can be unpredictable and may change significantly over time; for instance, unexpected "spikes" in demand traffic are common in modern data centers (Kandula et al. 2009). These demand uncertainties make *robustness* an important criterion for system design. These practical concerns have been our primary motivation for studying the interplay between robustness, performance, and the level of flexibility.

1.2. Related Research

Bipartite graphs provide a natural model for capturing the relationships between demand types and service resources. It is well known in the supply chain literature that limited flexibility, corresponding to a sparse bipartite graph, can be surprisingly effective in resource allocation even when compared to a fully flexible system (Jordan and Graves 1995, Gurumurthi and Benjaafar 2003, Iravani et al. 2005, Chou et al. 2010, Simchi-Levi and Wei 2012). The use of sparse random graphs or expanders as flexibility structures to

improve robustness has recently been studied in Chou et al. (2011), Chen et al. (2015) in the context of supply chains, and in Leconte et al. (2012) for content replication. Similar to the robustness results reported in this paper, these works show that random graphs or expanders can accommodate a large set of demand rates. However, in contrast to our work, nearly all analytical results in this literature focus on static allocation problems, where one tries to match supply with demand in a single shot, as opposed to our model, where resource allocation decisions need to be made dynamically over time.

In the queueing theory literature, the models that we consider fall under the umbrella of multiclass multiserver systems, where a set of servers are connected to a set of queues through a bipartite graph. Under these (and similar) settings, complete resource pooling (full flexibility) is known to improve system performance (Mandelbaum and Reiman 1998, Harrison and Lopez 1999, Bell and Williams 2001). However, much less is known when only limited flexibility is available: systems with a nontrivial connectivity graph are extremely difficult to analyze, even under seemingly simple scheduling policies (e.g., first-come, first-served) (Talreja and Whitt 2008, Visschers et al. 2012). Simulations in Wallace and Whitt (2005) show empirically that limited cross-training can be highly effective in a large call center under a skill-based routing algorithm. Using a very different set of modeling assumptions, Bassamboo et al. (2012) proposes a specific chaining structure with limited flexibility, which is shown to perform well under heavy traffic. Closer to the spirit of the current work is Tsitsiklis and Xu (2012), which studies a partially flexible system where a fraction $p > 0$ of all processing resources are fully flexible, while the remaining fraction, $1 - p$, is dedicated to specific demand types, and which shows an exponential improvement in delay scaling under heavy traffic. However, both Bassamboo et al. (2012) and Tsitsiklis and Xu (2012) focus on the heavy-traffic regime, which is different from the current setting where traffic intensity is assumed to be fixed, and the analytical results in both works apply only to uniform demand rates. Furthermore, with a constant fraction of the resources being fully flexible, the average degree in Tsitsiklis and Xu (2012) scales linearly with the system size n, whereas here we are interested in the case of a much slower (sublinear) degree scaling.

At a higher level, our work is focused on the interplay between robustness, delay, and the degree of flexibility in a queueing network, which is much less studied in the existing literature, and especially for networks with a nontrivial interconnection topology.

On the technical end, we build on several existing ideas. The techniques of batching (cf. Neely et al. 2007, Shah and Tsitsiklis 2008) and the use of virtual queues (cf. McKeown et al. 1999, Kunniyur and Srikant 2001) have appeared in many contexts in queueing theory, but the specific models considered in the literature bear little resemblance to ours. The study of expander graphs has become a rich field in mathematics (cf. Hoory et al. 2006), but we will refrain from providing a thorough review because only some elementary and standard properties of expander graphs are used in the current paper.

We finally note that preliminary (and weaker) versions of some of the results were included in the conference paper Tsitsiklis and Xu (2013).

Organization of the Paper. We describe the model in Section 2, along with the notation to be used throughout. The main results are provided in Section 3. The construction and the analysis associated with the expander architecture will be presented separately, in Section 4. We conclude the paper in Section 5 with a further discussion of the results and directions for future research.

2. Model and Metrics

2.1. Queueing Model and Interconnection Topologies

The Model. We consider a sequence of systems operating in *continuous time*, indexed by the integer n, where the nth system consists of rn queues and n servers (Figure 2), and where r is a constant that is held fixed as n varies. For simplicity, we will set r to one but note that all results and arguments in this paper can be extended to the case of general r without difficulty.

A *flexible architecture* is represented by an $n \times n$ undirected bipartite graph $g_n = (E, I \cup J)$, where I and J represent the sets of queues and servers, respectively, and E the set of edges between them.[3] We will also refer to I and J as the sets of left and right nodes, respectively. A server $j \in J$ is *capable* of serving a queue $i \in I$, if and only if $(i, j) \in E$. We will use the following notation.

1. Let $\mathcal{G}_n$ be the set of all $n \times n$ bipartite graphs.
2. For $g_n \in \mathcal{G}_n$, let $\deg(g_n)$ be the average degree among the n left nodes, which is the same as the average degree of the right nodes.
3. For a subset of nodes, $M \subset I \cup J$, let $g|_M$ be the graph induced by g on the nodes in M.
4. Denote by $\mathcal{N}(i)$ the set of servers in J connected to queue i, and similarly, by $\mathcal{N}(j)$ the set of queues in I connected to server j.

Each queue i receives a stream of incoming jobs according to a Poisson process of rate $\lambda_{n,i}$, independent of all other streams, and we define $\boldsymbol{\lambda}_n = (\lambda_{n,1}, \lambda_{n,2}, \ldots, \lambda_{n,n})$, which is the *arrival rate vector*. When the value of n is clear from the context, we sometimes suppress the subscript n and write $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$ instead. The sizes of the jobs are exponentially distributed with

mean 1, independent from each other and from the arrival processes. All servers are assumed to be running at a constant rate of 1. The system is assumed to be empty at time $t = 0$.

Jobs arriving at queue i can be assigned (immediately, or in the future) to an idle server $j \in \mathcal{N}(i)$ to receive service. The assignment is *binding*: once the assignment is made, the job cannot be transferred to, or simultaneously receive service from, any other server. Moreover, service is *nonpreemptive*: once service is initiated for a job, the assigned server has to dedicate its full capacity to this job until its completion.[4] Formally, if a server j has just completed the service of a previous job at time t or is idle, its available actions are as follows. (a) *Serve a new job*: Server j can choose to fetch a job from any queue in $\mathcal{N}(j)$ and immediately start service. The server will remain occupied and take no other actions until the processing of the current job is completed, which will take an amount of time that is equal to the size of the job. (b) *Remain idle*: Server j can choose to remain idle. While in the idling state, it will be allowed to initiate a service (action (a)) at any point in time.

Given the limited set of actions available to the server, the performance of the system is fully determined by a *scheduling policy*, π, which specifies for each server $j \in J$, (a) when to remain idle, and when to serve a new job, and (b) from which queue in $\mathcal{N}(j)$ to fetch a job when initiating a new service. We only allow policies that are causal, in the sense that the decision at time t depends only on the history of the system (arrivals and service completions) up to t. We allow the scheduling policy to be *centralized* (i.e., to have full control over all server actions) based on the knowledge of all *queue lengths* and server states. On the other hand, the policy does *not* observe the actual sizes of the jobs before they are served.

2.2. Performance Metrics

Characterization of Arrival Rates. We will restrict ourselves to arrival rate vectors with average *traffic intensity* at most ρ; i.e.,

$$\sum_{i=1}^{n} \lambda_i \leqslant \rho n, \tag{1}$$

where $\rho \in (0,1)$ will be treated throughout the paper as a given absolute constant. To quantify the *level of variability* or *uncertainty* of a set of arrival rate vectors, $\mathbf{\Lambda}$, we introduce a *fluctuation parameter*, denoted by u_n, with the property that $\lambda_i < u_n$, for all i and $\boldsymbol{\lambda} \in \mathbf{\Lambda}$.

Note that, for a graph with maximum degree d_n, the fluctuation parameter should not exceed d_n, because otherwise there could exist some $\lambda \in \mathbf{\Lambda}$ under which at least one queue would be unstable. Therefore, the best we can hope for is a flexible architecture that can accommodate arrival rate vectors with a u_n that is close to d_n. The following condition formally characterizes the range of arrival rate vectors we will be interested in, parameterized by the fluctuation parameter, u_n, and traffic intensity, ρ.

Condition 2.1 (Rate Condition)**.** Fix $n \geqslant 1$ and some $u_n > 0$. We say that a (nonnegative) arrival rate vector $\boldsymbol{\lambda}$ satisfies the rate condition if the following hold:

1. $\max_{1 \leqslant i \leqslant n} \lambda_i < u_n$.
2. $\sum_{i=1}^{n} \lambda_i \leqslant \rho n$.

We denote by $\mathbf{\Lambda}_n(u_n)$ the set of all arrival rate vectors that satisfy the above conditions.

Capacity Region. The capacity region for a given architecture is defined as the set of all arrival rate vectors that it can handle. As mentioned in the Introduction, a larger capacity region indicates that the architecture is more robust against uncertainties or changes in the arrival rates. More formally, we have the following definition.

Definition 2.2 (Feasible Demands and Capacity Region)**.** Let $g = (I \cup J, E)$ be an $n \times n'$ bipartite graph. An arrival rate vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$ is said to be feasible if there exists a flow, $\mathbf{f} = \{f_{ij} : (i,j) \in E\}$, such that

$$\begin{aligned} \lambda_i &= \sum_{j \in \mathcal{N}(i)} f_{ij}, \quad \forall\, i \in I, \\ \sum_{i \in \mathcal{N}(j)} f_{ij} &< 1, \quad \forall\, j \in J, \\ f_{ij} &\geqslant 0, \quad \forall\, (i,j) \in E. \end{aligned} \tag{2}$$

In this case, we say that the flow $\mathbf{f}$ *satisfies* the demand $\boldsymbol{\lambda}$. The *capacity region* of g, denoted by $\mathbf{R}(g)$, is defined as the set of all feasible demand vectors of g.

It is well known that there exists a policy under which the steady-state expected delay is finite if and only if $\boldsymbol{\lambda} \in \mathbf{R}(g_n)$; the strict inequalities in Definition 2.2 are important here. For the remainder of the paper, we will use the fluctuation parameter u_n (cf. Condition 2.1) to gauge the size of the capacity region, $\mathbf{R}(g_n)$, of an architecture. For instance, if $\mathbf{\Lambda}_n(u_n) \subset \mathbf{R}(g_n)$, then the architecture g_n, together with a suitable scheduling policy, allows for finite steady-state expected delay, for any arrival rate vector in $\mathbf{\Lambda}_n(u_n)$.

Vanishing Delay. We define the *expected average delay*, $\mathbb{E}(W \mid \boldsymbol{\lambda}, g, \pi)$ under the arrival rate vector $\boldsymbol{\lambda}$, flexible architecture g, and scheduling policy π, as follows. We denote by $W_{i,m}$ the waiting time in queue experienced by the mth job arriving to queue i, define

$$\mathbb{E}(W_i) = \limsup_{m \to \infty} \mathbb{E}(W_{i,m}),$$

and let

$$\mathbb{E}(W \mid \boldsymbol{\lambda}, g, \pi) = \frac{1}{\sum_{i \in I} \lambda_i} \sum_{i \in I} \lambda_i \mathbb{E}(W_i). \tag{3}$$

In the sequel, we will often omit the mention of π, and sometimes of g, and write $\mathbb{E}(W \mid \boldsymbol{\lambda}, g)$ or $\mathbb{E}(W \mid \boldsymbol{\lambda})$, to place emphasis on the dependencies that we wish to focus on.[5]

The delay performance of the system is measured by the following criteria: (a) For what ranges $\boldsymbol{\Lambda}_n(u_n)$ of arrival rates, $\boldsymbol{\lambda}$, does delay diminish to zero as the system size increases (i.e., $\sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}_n(u_n)} \mathbb{E}(W \mid \boldsymbol{\lambda}) \to 0$ as $n \to \infty$), and (b) at what *speed* does the delay diminish, as a function of n?

2.3. Notation

We will denote by $\mathbb{N}$, $\mathbb{Z}_+$, and $\mathbb{R}_+$ the sets of natural numbers, nonnegative integers, and nonnegative reals, respectively. The following short-hand notation for asymptotic comparisons will be used often, as an alternative to the usual $\mathcal{O}(\cdot)$ notation; here, f and g are positive functions, and L is a certain limiting value of interest, in the set of extended reals, $\mathbb{R}\cup\{-\infty,+\infty\}$:

1. $f(x) \lesssim g(x)$ or $g(x) \gtrsim f(x)$ for $\overline{\lim}_{x\to L} f(x)/g(x) < \infty$;
2. $f(x) \ll g(x)$ or $g(x) \gg f(x)$ for $\overline{\lim}_{x\to L} f(x)/g(x) = 0$;
3. $f(x) \sim g(x)$ for $\lim_{x\to L} f(x)/g(x) = 1$.

We will minimize the use of floors and ceilings, to avoid the cluttering of notation, and thus assume that all values of interest are appropriately rounded up or down to an integer, whenever doing so does not cause ambiguity or confusion. Whenever suitable, we will use uppercase letters for random variables and lowercase letters for deterministic values.

3. Main Results: Capacity Region and Delay of Flexible Architectures

The statements of our main results are given in this section. Below is a high-level summary of our results; a more complete comparison is given in Table 1.

Our main results focus on an expander architecture, where the interconnection topology is an expander graph with appropriate expansion. We show that, when $d_n \gg \ln n$, the expander architecture has a capacity region that is within a constant factor of the maximum possible among all graphs with average degree d_n, while simultaneously ensuring an asymptotically diminishing queueing delay of order $\ln n/d_n$ for all arrival rate vectors in the capacity region, as $n \to \infty$ (Theorem 3.4). Our analysis involves on a new class of virtual-queue–based scheduling policies that rely on dynamically constructed job-to-server assignments on the connectivity graph.

Our secondary results concern a modular architecture, which has a simpler construction and scheduling rule compared to the expander architecture. The modular architecture consists of a collection of *separate* smaller subnetworks, with complete connectivity between all queues and servers within each subnetwork. Since the subnetworks are disconnected from each other, a modular architecture does *not* admit a large capacity region: there always exists an *infeasible* arrival rate vector even when the fluctuation parameter is of constant order (Theorem 3.5). Nevertheless, we show that with proper randomization in the construction of the subnetworks (randomized modular architecture), a simple greedy scheduling policy is able to deliver asymptotically vanishing delay for "most" arrival rate vectors with nearly optimal fluctuation parameters, with high probability (Theorem 3.6). These findings suggest that, thanks to its simplicity, the randomized modular architecture could be a viable alternative to the expander architecture if the robustness requirement is not as stringent and one is content with probabilistic guarantees on system stability.

Table 1. Summary and Comparison of the Studied Flexibility Architectures, in Terms of Capacity and Delay

Flexible architectures	Rate conditions	Capacity region	Delay
Expander (Theorem 3.4)	$d_n \gg \ln n$, $u_n \lesssim d_n$	Good for all $\boldsymbol{\lambda}$	Good for all $\boldsymbol{\lambda}$, with $\mathbb{E}(W) \lesssim \ln n/d_n$
Modular (Theorems 3.5, 3.7)	$d_n \gg 1$, $u_n > 1$	Bad for some $\boldsymbol{\lambda}$ (even if $u_n \lesssim 1$)	Good for uniform $\boldsymbol{\lambda}$, with $\mathbb{E}(W) \lesssim \exp(-c \cdot d_n)$
Random modular (w.h.p.) (Theorems 3.6, 3.7)	$d_n \gtrsim \ln n$, $u_n \lesssim d_n/\ln n$	Good for most $\boldsymbol{\lambda}$, bad for some $\boldsymbol{\lambda}$	Good for most $\boldsymbol{\lambda}$, with $\mathbb{E}(W) \lesssim \exp(-c \cdot d_n)$; bad for some $\boldsymbol{\lambda}$

Notes. We say that capacity is "good" for $\boldsymbol{\lambda}$ if $\boldsymbol{\lambda}$ falls within the capacity region of the architecture, and that delay is "good" if the expected delay is vanishingly small for large n. When describing the size of the set of $\boldsymbol{\lambda}$ for which a statement applies, we use the following (progressively weaker) quantifiers:

1. "For all" means that the statement holds for all $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_n(u_n)$;
2. "For most" means that the statement holds with high probability when $\boldsymbol{\lambda}$ is drawn from an arbitrary distribution over $\boldsymbol{\Lambda}_n(u_n)$, independently from any randomization in the construction of the flexibility architecture;
3. "For some" means that the statement is true for a nonempty set of values of $\boldsymbol{\lambda}$.

The label "w.h.p." means that all statements in the corresponding row hold "with high probability" with respect to the randomness in generating the flexibility architecture.

3.1. Preliminaries

Before proceeding, we provide some information on expander graphs, which will be used in some of our constructions and proofs.

Definition 3.1. An $n \times n'$ bipartite graph $(I \cup J, E)$ is an (α, β)-expander, if for all $S \subset I$ that satisfy $|S| \leqslant \alpha n$, we have that $|\mathcal{N}(S)| \geqslant \beta |S|$, where $\mathcal{N}(S) = \bigcup_{i \in S} \mathcal{N}(i)$ is the set of nodes in J that are connected to some node in S.

The usefulness of expanders in our context comes from the following lemma, which relates the parameters of an expander to the size of its capacity region, as measured by the fluctuation parameter, u_n. The proof is elementary and is given in the online Appendix A.1.

Lemma 3.2 (Capacity of Expanders). *Fix $n, n' \in \mathbb{N}$, $\rho \in (0,1)$, $\gamma > \rho$. Suppose that an $n \times n'$ bipartite graph, g_n, is a $(\gamma/\beta_n, \beta_n)$-expander, where $\beta_n \geqslant u_n$. Then, $\mathbf{\Lambda}_n(u_n) \subset \mathbf{R}(g_n)$.*

The following lemma ensures that such expander graphs exist for the range of parameters that we are interested in. The lemma is a simple consequence of a standard result on the existence of expander graphs, and its proof is given in the online Appendix A.3.

Lemma 3.3. *Fix $\rho \in (0,1)$. Suppose that $d_n \to \infty$ as $n \to \infty$. Let $\beta_n = \frac{1}{2} \cdot (\ln(1/\rho)/(1+\ln(1/\rho)))d_n$, and $\gamma = \sqrt{\rho}$. There exists $n' > 0$, such that for all $n \geqslant n'$, there exists an $n \times n$ bipartite graph which is a $(\gamma/\beta_n, \beta_n)$-expander with maximum degree d_n.*

Remark. It is well known that random graphs with appropriate average degree are expanders with high probability (cf. Hoory et al. 2006). For instance, it is not difficult to show that if $d_n \gg \ln n$ and $\beta_n = ((1-\gamma)/4)d_n/\ln n$, then an Erdős–Rényi random bipartite graph with average degree d_n is a $(\gamma/\beta_n, \beta_n)$-expander, with high probability, as $n \to \infty$ (cf. Xu 2014, Lemma 3.12). We note, however, that to deterministically construct expanders in a computationally efficient manner can be challenging and is in and of itself an active field of research; the reader is referred to the survey paper Hoory et al. (2006) and the references therein.

3.2. Expander Architecture

Construction of the Architecture. The connectivity graph in the expander architecture is an expander graph with maximum degree d_n and appropriate expansion.

Scheduling Policy. We employ a scheduling policy that organizes the arrivals into batches, stores the batches in a virtual queue, and dynamically assigns the jobs in a batch to appropriate servers. Theorem 3.4, which is the main result of this paper, shows that under this policy the expander architecture achieves an asymptotically *vanishing* delay for *all* arrival rate vectors in the set $\mathbf{\Lambda}_n(u_n)$. Of course, we assume that d_n is sufficiently large so that the corresponding expander graph exists (Lemma 3.3, with ρ replaced with $\hat{\rho}$). At a high level, the strong guarantees stem from the excellent connectivity of an expander graph, and similarly of random subsets of an expander graph, a fact which we will exploit to show that jobs arriving to the system during a small time interval can be quickly assigned to connected idle servers with high probability, which then leads to a small delay. The proof of the theorem, including a detailed description of the scheduling policy, is given in Section 4.

Theorem 3.4 (Capacity and Delay of Expander Architectures). *Let $\hat{\rho} = 1/(1+(1-\rho)/8)$. For every $n \in \mathbb{N}$, define*

$$\beta_n = \frac{1}{2} \cdot \frac{\ln(1/\hat{\rho})}{\ln(1/\hat{\rho}) + 1} d_n,$$

and

$$\gamma = \sqrt{\hat{\rho}}.$$

Suppose that $\ln n \ll d_n \ll n$, and

$$u_n \leqslant \frac{1-\rho}{2} \beta_n.$$

Let g_n be a $(\gamma/\beta_n, \beta_n)$-expander with maximum degree d_n. The following holds.

1. *There exists a scheduling policy, π_n, under which*

$$\sup_{\lambda_n \in \mathbf{\Lambda}_n(u_n)} \mathbb{E}(W \mid \lambda_n, g_n) \leqslant \frac{c \ln n}{d_n}, \tag{4}$$

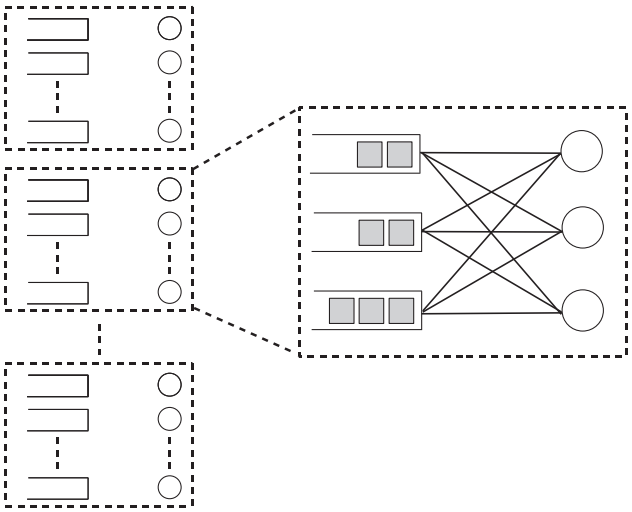
where c is a constant independent of n and g_n.

2. *The scheduling policy, π_n, only depends on g_n and an upper bound on the traffic intensity, ρ. It does not require knowledge of the arrival rate vector λ_n.*

Note that when ρ is viewed as a constant, the upper bound on u_n in the statement of Theorem 3.4 is just a constant multiple of d_n. Since the fluctuation parameter, u_n, should be no more than d_n for stability to be possible, the size of $\mathbf{\Lambda}_n(u_n)$ in Theorem 3.4 is within a constant factor of the *best possible*.

Remark. Compared to our earlier results, in a preliminary version of this paper (theorem 1 in Tsitsiklis and Xu 2013), Theorem 3.4 is stronger in two major aspects: (1) the guarantee for diminishing delay holds deterministically over all arrival rate vectors in $\mathbf{\Lambda}_n(u_n)$, as opposed to "with high probability" over the randomness in the generation of g_n; and (2) the fluctuation parameter, u_n, is allowed to be of order d_n in Theorem 3.4, while Tsitsiklis and Xu (2013) required that $u_n \ll \sqrt{d_n/\ln n}$. The flexible architecture considered in Tsitsiklis and Xu (2013) was based on Erdös–Rényi random graphs. It also employed a scheduling policy based on virtual queues, as in this paper. However, the policy in the present paper is simpler to describe and analyze.

Figure 3. A Modular Architecture Consisting of n/d_n Subnetworks, Each with d_n Queues and Servers



Note. Within each subnetwork, all servers are connected to all queues.

3.3. Modular Architectures

In a modular architecture, the designer partitions the network into n/d_n *separate* subnetworks. Each subnetwork consists of d_n queues and servers that are fully connected (Figure 3) but disconnected from queues and servers in other subnetworks.

Construction of the Architecture. Formally, the construction is as follows.

1. We partition the set J of servers into n/d_n disjoint subsets ("clusters") $B_1, \ldots, B_{n/d_n}$, all having the same cardinality d_n. For concreteness, we assign the first d_n servers to the first cluster, B_1, the next d_n servers to the second cluster, etc.

2. We form a partition $\sigma_n = (A_1, \ldots, A_{n/d_n})$ of the set I of queues into n/d_n disjoint subsets ("clusters") A_k, all having the same cardinality d_n.

3. To construct the interconnection topology, for $k = 1, \ldots, n/d_n$, we connect every queue $i \in A_k$ to every server $j \in B_k$. A pair of queue and server clusters with the same index k will be referred to as a subnetwork.

Note that in a modular architecture, the degree of each node is equal to the size, d_n, of the clusters. Note also that different choices of σ_n yield isomorphic architectures. When σ_n is drawn uniformly at random from the set of all possible partitions of I into subsets of size n/d_n, we call the resulting topology a *random modular* architecture.

Scheduling Policy. We use a simple greedy policy, equivalent to running each subnetwork as an $M/M/d_n$ queue. Whenever a server $j \in B_k$ becomes available, it starts serving a job from any nonempty queue in A_k. Similarly, when a job arrives at queue $i \in A_k$, it is immediately assigned to an arbitrary idle server in B_k, if such a server exists, and waits in queue i, otherwise.

Our first result points out that a modular architecture does not have a large capacity region: for any partition σ_n, there always exists an infeasible arrival rate vector, even if u_n is small, of order $\mathcal{O}(1)$. The proof is given in the online Appendix A.3. Note that this is a negative result that applies no matter what scheduling policy is used.

Theorem 3.5 (Capacity Region of Deterministic Modular Architectures). *Fix $n \geqslant 1$ and some $u_n > 1$. Let g_n be a modular architecture with average degree $d_n \leqslant (\rho/2)n$. Then, there exists $\boldsymbol{\lambda}_n \in \boldsymbol{\Lambda}_n(u_n)$ such that $\boldsymbol{\lambda}_n \notin \mathbf{R}(g_n)$.*

However, if we are willing to settle for a weaker result on the capacity region, the next theorem states that with the random modular architecture, any given arrival rate vector $\boldsymbol{\lambda}_n$ has high probability (with respect to the random choice of the partition σ_n) of belonging to the capacity region, if the fluctuation parameter, u_n, is of order $\mathcal{O}(d_n/\ln n)$, but no more than that. Intuitively, this is because the randomization in the connectivity structure makes it unlikely that many large components of $\boldsymbol{\lambda}_n$ reside in the same subnetwork. The proof is given in the online Appendix A.4.

Theorem 3.6 (Capacity Region of Random Modular Architectures). *Let σ_n be drawn uniformly at random from the set of all partitions, and let G_n be the resulting random modular architecture. Let $\mathbb{P}_{G_n}$ be the probability measure that describes the distribution of G_n. Fix a constant $c_1 > 0$, and suppose that $d_n \geqslant c_1 \ln n$. Then, there exist positive constants c_2 and c_3, such that:*

(a) *If $u_n \leqslant c_2 d_n/\ln n$, then*

$$\lim_{n\to\infty} \inf_{\boldsymbol{\lambda}_n \in \boldsymbol{\Lambda}_n(u_n)} \mathbb{P}_{G_n}(\boldsymbol{\lambda}_n \in \mathbf{R}(G_n)) = 1. \tag{5}$$

(b) *Conversely, if $u_n > c_3 d_n/\ln n$ and $d_n \leqslant n^{0.3}$, then*

$$\lim_{n\to\infty} \inf_{\boldsymbol{\lambda}_n \in \boldsymbol{\Lambda}_n(u_n)} \mathbb{P}_{G_n}(\boldsymbol{\lambda}_n \in \mathbf{R}(G_n)) = 0. \tag{6}$$

We can use Theorem 3.6 to obtain a statement about "most" arrival rate vectors in $\boldsymbol{\Lambda}_n(u_n)$, as follows. Suppose that $\boldsymbol{\lambda}_n$ is drawn from an arbitrary distribution μ_n over $\boldsymbol{\Lambda}_n(u_n)$, independently from the randomness in G_n. Let $\mathbb{P}_{G_n} \times \mu_n$ be the product measure that describes the joint distribution of G_n and $\boldsymbol{\lambda}_n$. Using Fubini's theorem, Equation (5) implies that

$$\lim_{n\to\infty} (\mathbb{P}_{G_n} \times \mu_n)(\boldsymbol{\lambda}_n \in \mathbf{R}(G_n)) = 1. \tag{7}$$

A further application of Fubini's theorem and an elementary argument[6] implies that there exists a sequence δ'_n that converges to zero, such that the event

$$\mu_n\left(\boldsymbol{\lambda}_n \in \mathbf{R}(G_n) \,|\, G_n\right) \geqslant 1 - \delta'_n \tag{8}$$

has "high probability," with respect to the measure $\mathbb{P}_{G_n}$. That is, there is high probability that the random modular architecture includes "most" arrival vectors $\boldsymbol{\lambda}_n$ in $\boldsymbol{\Lambda}_n(u_n)$.

We now turn to delay. The next theorem states that in a modular architecture, delay is vanishingly small for all arrival rate vectors in the capacity region that are not too close to its outer boundary. The proof is given in the online Appendix A.5.

We need some notation. For any set S and scalar γ, we let $\gamma S = \{\gamma x : x \in S\}$.

Theorem 3.7 (Delay of Modular Architectures). *Fix some $\gamma \in (0,1)$, and consider a Modular architecture g_n for each n. There exists a constant $c > 0$, independent of n and the sequence $\{g_n\}$, so that*

$$\mathbb{E}(W \mid \boldsymbol{\lambda}_n) \lesssim \exp(-c \cdot d_n), \tag{9}$$

for every $\boldsymbol{\lambda}_n \in \gamma \mathbf{R}(g_n)$.

3.3.1. Expanded Modular Architectures. There is a further variant of the modular architecture that we call the *expanded modular architecture*, which combines the features of a modular architecture and an expander graph via a graph product. By construction, it uses part of the system flexibility to achieve a large capacity region and part to achieve low delay. As a result, the expanded modular architecture admits a smaller capacity region compared to that of an expander architecture. Another drawback is that the available performance guarantees involve policies that require the knowledge of the arrival rates λ_i. On the positive side, it guarantees an asymptotically vanishing delay for *all* arrival rates, uniformly across the capacity region, and can be operated by a scheduling policy that is arguably simpler than in the expander architecture. The construction and a scheduling policy for the expanded modular architecture is given in the online Appendix B, along with a statement of its performance guarantees (Theorem B.1). The technical details can be found in Xu (2014).

4. Analysis of the Expander Architecture

In this section, we introduce a policy for the expander architecture, based on batching and virtual queues, which will then be used to prove Theorem 3.4. We begin by describing the basic idea at a high level.

4.1. The Main Idea

Our policy proceeds by collecting a fair number of arriving jobs to form batches. Batches are thought of as being stored in a virtual queue, with each batch treated as a single entity. By choosing the batch size large enough, one expects to see certain statistical regularities that can be exploited to efficiently handle the jobs within a batch. We now provide an outline of the operation of the policy, for a special case.

Let us fix n and consider the case where $\lambda_i = \lambda < 1$ for all i. Suppose that at time t, all servers are busy serving some job. Let us also fix some γ_n such that $\gamma_n \ll 1$, while $n\gamma_n$ is large. During the time interval $[t, t+\gamma_n)$, "roughly" $\lambda n \gamma_n$ new jobs will arrive and $n\gamma_n$ servers will become available. Let Γ be the set of queues that received any job and let Δ be the set of servers that became available during this interval. Since $\lambda n \gamma_n \ll n$, these incoming jobs are likely to be spread out across different queues, so that most queues receive at most one job. Assuming that this is indeed the case, we focus on $g_n|_{\Gamma\cup\Delta}$, that is, the connectivity graph g_n, restricted to $\Gamma \cup \Delta$. The key observation is that this is a subgraph sampled uniformly at random among all subgraphs of g_n with approximately $\lambda n \gamma_n$ left nodes and $n\gamma_n$ right nodes. When $n\gamma_n$ is sufficiently large, and g_n is well connected (as in an expander with appropriate expansion properties), we expect that, with high probability, $g_n|_{\Gamma\cup\Delta}$ admits a matching that includes the entire set Γ (i.e., a one-to-one mapping from Γ to Δ). In this case, we can ensure that *all* of the roughly $\lambda n \gamma_n$ jobs can start receiving service at the end of the interval, by assigning them to the available servers in Δ according to this particular matching. Note that the resulting queueing delay will be comparable to γ_n, which has been assumed to be small.

The above described scenario corresponds to the normal course of events. However, with a small probability, the above scenario may not materialize, due to statistical fluctuations, such as:

1. Arrivals may be concentrated on a small number of queues.
2. The servers that become available may be located in a subset of g_n that is not well connected to the queues with arrivals.

In such cases, it may be impossible to assign the jobs in Γ to servers in Δ. These exceptional cases will be handled by the policy in a different manner. However, if we can guarantee that the probability of such cases is low, we can then argue that their impact on performance is negligible.

Whether or not the above-mentioned exceptions will have low probability of occurring depends on whether the underlying connectivity graph, g_n, has the following property: with high probability, a randomly sampled sublinear (but still sufficiently large) subgraph of g_n admits a large set of "flows." This property will be used to guarantee that, with high probability, the jobs in Γ can indeed be assigned to distinct servers in the set Δ. We will show that an expander graph with appropriate expansion does possess this property.

4.2. An Additional Assumption

Before proceeding, we introduce an additional assumption on the arrival rates, which will remain in effect throughout this section, and which will simplify some of the arguments. Online Appendix A.6 explains why this assumption can be made without loss of generality.

Assumption 4.1. (Lower Bound on the Total Arrival Rate). *We have that* $\rho \in (1/2, 1)$, *and the total arrival rate satisfies the lower bound*

$$\sum_{i=1}^{n} \lambda_i \geqslant (1-\rho)n. \tag{10}$$

4.3. The Policy

We now describe in detail the scheduling policy. Besides n, the scheduling policy uses the following inputs:

1. ρ, the traffic intensity introduced in Condition 2.1, in Section 2.2,
2. ϵ, a positive constant such that $\rho + \epsilon < 1$.
3. b_n, a batch size parameter,
4. g_n, the connectivity graph.

Notice that the arrival rates, λ_i, and the fluctuation parameter, u_n, are *not* inputs to the scheduling policy.

At this point, it is useful to make a clarification regarding the $\lesssim$ notation. Recall that the relation $f(n) \lesssim g(n)$ means that $f(n) \leqslant cg(n)$, for all n, where c is a positive constant. Whenever we use this notation, we require that the constant c cannot depend on any parameters other than ρ and ϵ. Because we view ρ and ϵ as fixed throughout, this makes c an absolute constant.

4.3.1. Arrivals of Batches. Arriving jobs are organized in *batches* of cardinality ρb_n, where b_n is a design parameter, to be specified later.[7] Let $T_0^B = 0$. For $k \geqslant 1$, let T_k^B be the time of the $(k\rho b_n)$th arrival to the system, which we also view as the *arrival time* of the kth batch. For $k \geqslant 1$, the kth batch consists of the ρb_n jobs that arrive during the time interval $(T_{k-1}^B, T_k^B]$. The length $A_k = T_k^B - T_{k-1}^B$ of this interval will be called the *kth interarrival time*. We record, in the next lemma, some immediate statistical properties of the batch interarrival times.

Lemma 4.2. *The batch interarrival times,* $\{A_k\}_{k\geqslant 1}$, *are i.i.d., with*

$$\frac{b_n}{n} \leqslant \mathbb{E}(A_k) \leqslant \frac{\rho}{1-\rho} \cdot \frac{b_n}{n},$$

and $\mathrm{Var}(A_k) \lesssim b_n/n^2$.

Proof. The batch interarrival times are i.i.d., due to our independence assumptions on the job arrivals. By definition, A_k is equal in distribution to the time until a Poisson process records ρb_n arrivals. This Poisson process has rate $r = \sum_{i=1}^{n} \lambda_i$, and using also Assumption 4.1 in the first inequality below, we have

$$(1-\rho)n \leqslant \sum_{i=1}^{n} \lambda_i = r \leqslant \rho n.$$

The random variables A_k are Erlang (sum of ρb_n exponentials with rate r). Therefore,

$$\mathbb{E}(A_k) = \rho b_n \cdot \frac{1}{r} \geqslant \rho b_n \cdot \frac{1}{\rho n} = \frac{b_n}{n}.$$

Similarly,

$$\mathbb{E}(A_k) = \rho b_n \cdot \frac{1}{r} \leqslant \rho b_n \cdot \frac{1}{(1-\rho)n}.$$

Finally,

$$\mathrm{Var}(A_k) = \rho b_n \cdot \frac{1}{r^2} \leqslant \rho b_n \cdot \frac{1}{(1-\rho)^2 n^2} \lesssim \frac{b_n}{n^2}. \quad \text{Q.E.D}$$

4.3.2. The Virtual Queue. On arrival, batches are placed in what we refer to as a *virtual queue*. The virtual queue is a GI/G/1 queue, which is operated in FIFO fashion. That is, a batch waits in queue until all previous batches are served and then starts being served by a virtual queueing system. The service of a batch by the virtual queueing system lasts until a certain time by which all jobs in the batch have already been assigned to, and have started receiving service from, one of the physical servers, at which point the service of the batch is completed and the batch departs from the virtual queue. The time elapsed from the initiation of service of batch until its departure is called the *service time* of the batch. As a consequence, the queueing delay of a job in the actual (physical) system is bounded above by the sum of:

(a) the time from the arrival of the job until the arrival time of the batch to which the job belongs;
(b) the time that the batch waits in the virtual queue;
(c) the service time of the batch.

Service slots. The service of the batches at the virtual queue is organized along consecutive time intervals that we refer to as *service slots*. The service slots are intervals of the form $(ls, (l+1)s]$, where l is a nonnegative integer, whose length is[8]

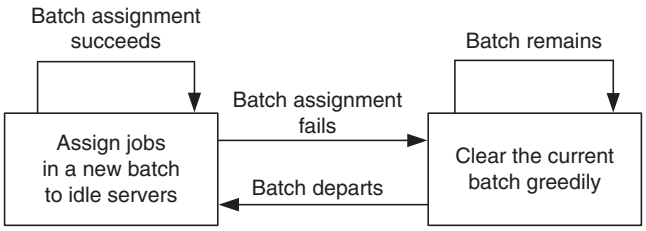
$$s = (\rho + \epsilon) \cdot \frac{b_n}{n}.$$

We will arrange matters so that batches can complete service and depart *only* at the end of a service slot; that is, at times of the form ls. Furthermore, we assume that the physical servers are operated as follows. If either a batch completes service at time ls or if there are no batches present at the virtual queue at that time, we assign to every idle server a dummy job whose duration is an independent exponential random variable, with mean one. This ensures that the state of the n servers is the same (all of them are busy) at certain special times, thus facilitating further analysis, albeit at the cost of some inefficiency.

4.3.3. The Service Time of a Batch. The specification of the service time of a batch depends on whether the batch, on arrival, finds an empty or nonempty virtual queue.

Suppose that a batch arrives during the service slot $(ls, (l+1)s]$ and finds an empty virtual queue; that is,

Figure 4. An Illustration of the Service Slot Dynamics



Note. An arrow indicates the transition from the end of one service slot to the next.

all previous batches have departed by time ls. According to what was mentioned earlier, at time ls, all physical servers are busy, serving either real or dummy jobs. Up until the end of the service slot, any server that completes service is not assigned a new (real or dummy) job and remains idle, available to be assigned a job at the very end of the service slot. Let Δ be the set of servers that are idle at time $(l+1)s$, the end of the service slot. At that time, we focus on the jobs in the batch under consideration. We wish to assign each job i in this batch to a *distinct* server $j \in \Delta$, subject to the constraint that $(i,j) \in E$. We shall refer to such a job-to-server assignment as a *batch assignment*. There are two possibilities (cf. Figure 4):

(1) If a batch assignment can be found, each job in the batch is assigned to a server according to that assignment, and the batch departs at time $(l+1)s$. In this case, we say that the service time of the batch was *short*.

(2) If a batch assignment cannot be found, we start assigning the jobs in the batch to physical servers in some arbitrary greedy manner: Whenever a server j becomes available, we assign to it a job from the batch under consideration, and from some queue i with $(i,j) \in E$, as long as such a job exists. (Ties are broken arbitrarily.) As long as every queue is connected to at least one server, all jobs in the associated batch will be eventually assigned. The last of the jobs in the batch gets assigned during a subsequent service interval $(l's, (l'+1)s]$, where $l' > l$, and we define $(l'+1)s$ as the departure time of the batch.

If the kth batch did indeed find an empty virtual queue on arrival, its service time, denoted by S_k, is the time elapsed from its arrival until its departure.

Suppose now that a batch arrives during a service slot $(ls, (l+1)s]$ and finds a nonempty virtual queue; that is, there are one or more batches that arrived earlier and which have not departed by time ls. In this case, the batch waits in the virtual queue until some time of the form $l's$, with $l' > l$, when the last of the previous batches departs. Recall that, as specified earlier, at time $l's$ all servers are made to be busy (perhaps by giving them dummy jobs), and we are faced with a situation identical to the one considered in the previous case, as if the batch under consideration just arrived at time $l's$; in particular, the same service policy can be applied. For this case, where the kth batch arrives to find a nonempty virtual queue, its service time, S_k, extends from the time of the departure of the $(k-1)$st batch until the departure of the kth batch.

4.4. Bounding the Virtual Queue by a GI/GI/1 Queue

Having defined the interarrival and service times of the batches, the virtual queue is a fully specified, work-conserving, FIFO single-server queueing system.

We note, however, one complication. The service times of the different batches depend on the arrival times. To see this, suppose, for example, that a batch on arrival sees an empty virtual queue and that its service time is "short." Then, its service time will be equal to the remaining time until the end of the current service slot and therefore dependent on the batch's arrival time. Furthermore, the service times of different batches are dependent: if the service time of the previous batch happens to be too long, then the next batch is likely to on arrival see a nonempty virtual queue, which then implies that its own service time will be an integer multiple of s.

To get around these complications, and to be able to use results on GI/GI/1 queues, we define the *modified service time*, S'_k, of the kth service batch to be equal to S_k, rounded above to the nearest integer multiple of s:

$$S'_k = \min\{ls : ls \geqslant S_k,\ l = 1, 2, \ldots\}.$$

Clearly, we have $S_k \leqslant S'_k$.

We now consider a modified (but again FIFO and work-conserving) virtual queueing system in which the arrival times are the same as before but the service times are the S'_k. A simple coupling argument, based on Lindley's recursion, shows that for every sample path, the time that the batch spends waiting in the queue of the original virtual queueing system is less than or equal to the time spent waiting in the queue of the modified virtual queueing system. It therefore suffices to upper bound the expected time spent in the queue of the modified virtual queueing system.

We now argue that the modified virtual queueing system is a GI/GI/1 queue; i.e., that the service times S'_k are i.i.d. and independent from the arrival process. For a batch whose service starts during the service slot $[ls, (l+1)s)$, the modified service time is equal to s, whenever the batch service time is short. Whether the batch service time will be short or not is determined by the composition of the jobs in this batch and by the identities of the servers who complete service during the service slot $[ls, (l+1)s)$. Because the servers start at the same "state" (all busy) at each service slot, it follows that the events that determine whether a batch

service time will be short or not are independent across batches, and with the same associated probabilities.

Similarly, if a batch service time is not short, the additional time to serve the jobs in the batch is affected only by the composition of jobs in the batch and the service completions at the physical servers after time ls, and these are again independent from the interarrival times and the modified service times S'_m of other batches m. Finally, the same considerations show the independence of the S'_k from the batch arrival process.

It should now be clear from the above discussion that the modified service time of a batch is of the form

$$S'_k = s + X_k \cdot \hat{S}_k, \tag{11}$$

where:

(a) X_k is a Bernoulli random variable which is equal to one if and only if the kth batch service time is not short; i.e., it takes more than a single service slot.

(b) $\hat{S}_k$ is a random variable which (assuming that every queue is connected to at least one server) is stochastically dominated by the sum of ρb_n independent exponential random variables with mean one, rounded up to the nearest multiple of s. (This dominating random variable corresponds to the extreme case where all of the ρb_n jobs in the batch are to be served in sequence, by the same physical server.)

(c) The pairs $(X_k, \hat{S}_k)$ are i.i.d.

4.5. Bounds on the Modified Service Times

For the remainder of Section 4, we will assume that

$$g_n \text{ is a } (\gamma/\beta_n, \beta_n)\text{-expander}, \tag{12}$$

where γ and β_n are defined as in the statement of Theorem 3.4.

The main idea behind the rest of the proof is as follows. We will upper bound the expected time spent in the modified virtual queueing system using Kingman's bound (Kingman 1962) for GI/GI/1 queues. Indeed, the combination of a batching policy with Kingman's bound is a fairly standard technique for deriving delay upper bounds (see, e.g., Shah and Tsitsiklis 2008). We already have bounds on the mean and variance of the interarrival times. To apply Kingman's bound, it remains to obtain bounds on the mean and variance of the service times S'_k of the modified virtual queueing system.

We now introduce an important quantity associated with a graph g_n, by defining

$$q(g_n) = \mathbb{P}(X_k = 1 \mid g_n);$$

because of the i.i.d. properties of the batch service times, this quantity does not depend on k. In words, for a given connectivity graph g_n, the quantity $q(g_n)$ stands for the probability that we cannot find a batch assignment, between the jobs in a batch and the servers that become idle during a period of length s.

We begin with the following lemma, which provides bounds on the mean and variance of S'_k.

Lemma 4.3. *There exists a sequence, $\{c_n\}_{n\in\mathbb{N}}$, with $c_n \lesssim b_n$, such that for all $n \geqslant 1$,*

$$s \leqslant \mathbb{E}(S'_k \mid g_n) \leqslant s + q(g_n)c_n$$

and

$$\mathrm{Var}(S'_k \mid g_n) \lesssim q(g_n)c_n^2.$$

Proof. The fact that $\mathbb{E}(S'_k \mid g_n) \geqslant s$ follows from the definition of S'_k in Equation (11) and the nonnegativity of $X_k\hat{S}_k$. The definition of an expander ensures that every queue is connected to at least one server through g_n. Recall that $\hat{S}_k$ is zero if $X_k = 0$; on the other hand, if $X_k = 1$, and as long as every queue is connected to some server, then $\hat{S}_k$ is upper bounded by the sum of ρb_n exponential random variables with mean one, rounded up to an integer multiple of s. Therefore,

$$\begin{aligned}\mathbb{E}(S'_k \mid g_n) &= s + \mathbb{E}(X_k\hat{S}_k \mid g_n) = s + \mathbb{P}(X_k = 1 \mid g_n)\\ &\quad \cdot \mathbb{E}(\hat{S}_k \mid X_k = 1, g_n) \leqslant s + q(g_n)(\rho b_n + s),\end{aligned}$$

which leads to the first bound in the statement of the lemma, with $c_n = b_n + s$. Since s is proportional to b_n/n, we also have $c_n \lesssim b_n$, as claimed. Furthermore,

$$\begin{aligned}\mathrm{Var}(S'_k \mid g_n) &= \mathrm{Var}(X_k\hat{S}_k \mid g_n) \leqslant \mathbb{E}(X_k^2\hat{S}_k^2 \mid g_n)\\ &\lesssim q(g_n)(b_n + s)^2 = q(g_n)c_n^2. \quad \text{Q.E.D}\end{aligned}$$

We now need to obtain bounds on $q(g_n)$. This is nontrivial and forms the core of the proof of the theorem. In what follows, we will show that with appropriate assumptions on the various parameters, and for any $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}(u_n)$, an Erdős–Rényi random graph has a very small $q(g_n)$, with high probability.

4.6. Assumptions on the Various Parameters

From now on, we focus on a specific batch size parameter of the form

$$b_n = \frac{320}{(1-\rho)^2} \cdot \frac{n \ln n}{\beta_n}. \tag{13}$$

We shall also set

$$\epsilon = \frac{1-\rho}{2}. \tag{14}$$

We assume, as in the statement of Theorem 3.4, that $d_n \ll n$, and that

$$\beta_n \gtrsim d_n \gg \ln n. \tag{15}$$

Under these choices of b_n and d_n, we have

$$b_n \lesssim \frac{n}{d_n/\ln n} \ll n; \tag{16}$$

that is, the batch size is *vanishingly small* compared to n. Finally, we will only consider arrival rate vectors that belong to the set $\boldsymbol{\Lambda}_n(u_n)$ (cf. Condition 2.1), where, as in the statement of Theorem 3.4,

$$u_n \leqslant \frac{1-\rho}{2}\beta_n. \tag{17}$$

4.7. The Probability of a Short Batch Service Time

We now come to the core of the proof, aiming to show that if the connectivity graph g_n is an expander graph with a sufficiently large expansion factor, then $q(g_n)$ is small. More precisely, we aim to show that a typical batch will have high probability of having a short service time. A concrete statement is given in the result that follows, and the rest of this subsection will be devoted to its proof.

Proposition 4.4. *Fix $n \geqslant 1$. We have that*

$$q(g_n) \leqslant \frac{1}{n^2}. \tag{18}$$

Let us focus on a particular batch, and let us examine what it takes for its service time to be short. There are two sources of randomness:

1. A total of ρb_n jobs arrive to the queues. Let A_i be the number of jobs that arrive at the ith queue, let $\mathbf{A} = (A_1, \ldots, A_n)$, and let Γ be the set of queues that receive at least one job. In particular, we have

$$\sum_{i=1}^{n} A_i = \sum_{i \in \Gamma} A_i = \rho b_n.$$

2. During the time slot at which the service of the batch starts, each server starts busy (with a real or dummy job). With some probability, and independently from other servers or from the arrival process, a server becomes idle by the end of the service time slot. Let Δ be the set of servers that become idle.

Recalling the definition of X_k as the indicator random variable of the event that the service time of the kth batch is not short, we see that X_k is completely determined by the graph g_n together with $\mathbf{A}$ and Δ. For the remainder of this subsection, we suppress the subscript k, since we are focusing on a particular batch. We therefore have a dependence of the form

$$X = f(g_n, \mathbf{A}, \Delta),$$

for some function f, and we emphasize the fact that $\mathbf{A}$ and Δ are independent.

Recall that $\epsilon = (1-\rho)/2$, and from the statement of Theorem 3.4 that

$$\hat{\rho} = \frac{1}{1+(1-\rho)/8} = \frac{1}{1+\epsilon/4}. \tag{19}$$

Clearly, $\hat{\rho} < 1$, and with some elementary algebra, it is not difficult to show that, for any given $\rho \in (0,1)$,

$$\hat{\rho} > \rho.$$

Let

$$m_n = \frac{\rho}{\hat{\rho}} b_n,$$

so that

$$\hat{\rho} m_n = \rho b_n. \tag{20}$$

Finally, let

$$\hat{u}_n = \beta_n \frac{m_n}{n}.$$

We will say that $\mathbf{A}$ is *nice* if there exists a set $\hat{\Gamma} \supset \Gamma$ of cardinality m_n, such that $A_i = 0$ whenever $i \notin \hat{\Gamma}$, and

$$A_i < \hat{u}_n, \quad \forall\, i \in \hat{\Gamma}.$$

We now establish that $\mathbf{A}$ is nice, with high probability. The main idea is simple: $\mathbf{A}$ is not nice only if one out of a finite collection of binomial variables with large means takes a value which is away from its mean by a certain multiplicative factor. Using the Chernoff bound, this probability can be shown to decay at least as fast as $1/n^3$. The details of this argument are given in the proof of Lemma 4.5, in the online Appendix A.7.

Lemma 4.5. *For all sufficiently large n, we have that*

$$\mathbb{P}(\mathbf{A} \text{ is not nice}) \leqslant \frac{1}{n^3}.$$

We now wish to establish that when $\mathbf{A}$ is nice, there is high probability (with respect to Δ) that the batch service time will be short. Having a short batch service time is, by definition, equivalent to the existence of a batch assignment, which in turn is equivalent to the existence of a certain flow in a subgraph of g_n. The lemma that follows deals with the latter existence problem for the original graph but will be later applied to subgraphs. Let $\bar{\mathbf{R}}(g)$ be the closure of the capacity region, $\mathbf{R}(g)$, of g.

Lemma 4.6. *Fix $n, n' \in \mathbb{N}$, $\rho \in (0,1)$, and $\gamma > \rho$. Suppose that an $n \times n'$ bipartite graph, g_n, is a $(\gamma/\beta_n, \beta_n)$-expander, where $\beta_n \geqslant u_n$. Then $\mathbf{\Lambda}_n(u_n) \subset \bar{\mathbf{R}}(g_n)$.*

Proof. The claim follows directly from Lemma 3.2, by noting that $\bar{\mathbf{R}}(g_n) \supset \mathbf{R}(g_n)$. Q.E.D

The next lemma is the key technical result of this subsection. It states that if g_n is an expander, then, for any given $\hat{\Gamma}$, the random subgraph $g_n|_{\hat{\Gamma} \cup \Delta}$ will be an expander graph with high probability (with respect to Δ). The lemma is stated as a stand-alone result, though we will use a notation that is consistent with the rest of the section. The proof relies on a delicate application of the Chernoff bound, and is given in the online Appendix A.8.

Lemma 4.7. *Fix $n \geqslant 1$, $\gamma \in (0,1)$, and $\rho \in [1/2, 1)$. Let $g_n = (I \cup J, E)$ be an $n \times n$ bipartite graph that is a $(\gamma/\beta_n, \beta_n)$-expander, where $\beta_n \gg \ln n$. Define the following quantities*:

$$\epsilon = \frac{1-\rho}{2},$$
$$\hat{\rho} = \frac{1}{1+\epsilon/4},$$
$$b_n = \frac{320}{(1-\rho)^2} \cdot \frac{n \ln n}{\beta_n} = \frac{80}{\epsilon^2} \cdot \frac{n \ln n}{\beta_n}, \tag{21}$$

$$m_n = \frac{\rho}{\hat{\rho}} b_n,$$
$$\hat{u}_n = \beta_n \frac{m_n}{n}.$$

Let $\hat{\Gamma}$ be an arbitrary subset of the left vertices, I, such that

$$|\hat{\Gamma}| = m_n, \tag{22}$$

and let Δ be a random subset of the right vertices, J, where each vertex belongs to Δ independently and with the same probability, where

$$\mathbb{P}(j \in \Delta) \geqslant (\rho + 3\epsilon/4)\frac{b_n}{n}, \quad \forall\, j \in J, \tag{23}$$

for all n sufficiently large. Denote by $\hat{G}$ the random subgraph $g_n|_{\hat{\Gamma}\cup\Delta}$. Then

$$\mathbb{P}(\hat{G} \text{ is not a } (\gamma/\hat{u}_n, \hat{u}_n)\text{-expander}) \leqslant \frac{1}{n^3}, \tag{24}$$

for all n sufficiently large, where the probability is measured with respect to the randomness in Δ.

To invoke Lemma 4.7, note that the conditions in Equation (21) are identical to the definitions for the corresponding quantities in this section. We next verify that Equation (23) is satisfied by the random subset, Δ, consisting of the idle servers at the end of a service slot. Recall that the length of a service slot is $(b_n/n)(\rho+\epsilon)$, and hence the probability that a given server, j, becomes idle by the end of a service slot is

$$\mathbb{P}(j \in \Delta) = 1 - \exp\left(-\frac{b_n}{n}(\rho+\epsilon)\right) \sim (\rho+\epsilon)\frac{b_n}{n}, \tag{25}$$

as $n \to \infty$. Therefore, for all n sufficiently large, we have that $\mathbb{P}(j \in \Delta) \geqslant (\rho + 3\epsilon/4)(b_n/n)$. We will now apply Lemmas 4.6 and 4.7 to the random subgraph with left (respectively, right) nodes $\hat{\Gamma}$ (respectively Δ), and with the demands A_i, for $i \in \hat{\Gamma}$, playing the role of $\boldsymbol{\lambda}$.

Lemma 4.8. *If n is large enough, and if the value $\mathbf{a}$ of $\mathbf{A}$ is nice, then*

$$\mathbb{P}(X = 1 \mid \mathbf{A} = \mathbf{a}) \leqslant \frac{1}{n^3},$$

where the probability is with respect to the randomness in Δ.

Proof. We fix some $\mathbf{a}$, assumed to be nice. Recall that

$$\hat{u}_n = \beta_n \frac{m_n}{n},$$

and from the statement of Theorem 3.4 that

$$\gamma = \sqrt{\hat{\rho}} > \hat{\rho}.$$

We apply Lemma 4.6 to the randomly sampled subgraph $\hat{G}$, with left nodes $\hat{\Gamma}$, $|\hat{\Gamma}| = m_n$ and right nodes Δ. We have the following correspondence: the parameters n and ρ, in Lemma 4.6 become, in the current context, m_n and $\hat{\rho}$, respectively, and the parameters β_n and u_n both become $\hat{u}_n$. Thus, by Lemma 4.6,

$$\text{if } \hat{G} \text{ is a } (\gamma/\hat{u}_n, \hat{u}_n)\text{-expander, then } \boldsymbol{\Lambda}_{m_n}(\hat{u}_n) \subset \bar{\mathbf{R}}(\hat{G}). \tag{26}$$

Let $\hat{\mathbf{A}}$ be the vector of job arrival numbers $\mathbf{A}$, restricted to the set of nodes in $\hat{\Gamma}$, and let $\hat{\mathbf{a}}$ be the realization of $\hat{\mathbf{A}}$. Note that we have

$$\sum_{i\in\hat{\Gamma}} \hat{a}_i = \sum_{i=1}^{n} A_i = \rho b_n = \hat{\rho} m_n,$$

because of Equation (20). Furthermore, for any $i \in \hat{\Gamma}$, the fact that $\mathbf{a}$ is nice implies that $\hat{a}_i < \hat{u}_n$. Thus, $\hat{\mathbf{a}} \in \boldsymbol{\Lambda}_{m_n}(\hat{u}_n)$. By Equation (26), this further implies that

$$\text{if } \hat{G} \text{ is a } (\gamma/\hat{u}_n, \hat{u}_n)\text{-expander, then } \hat{\mathbf{a}} \in \bar{\mathbf{R}}(\hat{G}). \tag{27}$$

By Lemma 4.7, the graph $\hat{G}$ is a $(\gamma/\hat{u}_n, \hat{u}_n)$-expander with probability at least $1 - n^{-3}$. Combining this fact with Equation (27), we have thus verified that $\hat{\mathbf{a}}$ belongs to $\bar{\mathbf{R}}(\hat{G})$, with probability at least

$$1 - \frac{1}{n^3}.$$

With $\bar{\mathbf{R}}(\hat{G})$ having been defined as the closure of the capacity region, $\mathbf{R}(\hat{G})$ (cf. Definition 2.2), the fact that the vector $\hat{\mathbf{a}}$ belongs to $\bar{\mathbf{R}}(\hat{G})$ is a statement about the existence of a feasible flow, $\{\hat{f}_{ij}\colon (i,j) \in \hat{E}\}$ (where $\hat{E}$ is the set of edges in $\hat{G}$), in a linear network flow model of the form

$$\hat{a}_i = \sum_{j:(i,j)\in\hat{E}} f_{ij}, \quad \forall\, i \in \hat{\Gamma},$$
$$\sum_{i:(i,j)\in\hat{E}} f_{ij} \leqslant 1, \quad \forall\, j \in \Delta,$$
$$f_{ij} \geqslant 0, \quad \forall\, (i,j) \in \hat{E}.$$

Because the "supplies" $\hat{a}_i$ in this network flow model, as well as the unit capacities of the right nodes are integer, it is well known that there also exists an integer flow. That is, we can find $\hat{f}_{ij} \in \{0,1\}$ such that $\sum_j \hat{f}_{ij} = \hat{a}_i$, for all i, and $\sum_i \hat{f}_{ij} \leqslant 1$, for all j. But this is the same as the statement that there exists a feasible batch assignment over $\hat{G}$. Thus, for large enough n and for any given nice $\mathbf{a}$, the conditional probability that a batch assignment does not exist is upper bounded by n^{-3}, as claimed. Q.E.D

We can now complete the proof of Proposition 4.4. By considering unconditional probabilities where $\mathbf{A}$ is random, and for n large enough, we have that

$$\begin{aligned}
\mathbb{P}(X=1) &\leqslant \mathbb{P}(\mathbf{A} \text{ is not nice}) \\
&\quad + \sum_{\mathbf{a}\text{ nice}} \mathbb{P}(X=1 \mid \mathbf{A}=\mathbf{a}) \cdot \mathbb{P}(\mathbf{A}=\mathbf{a}) \\
&\overset{(a)}{\leqslant} \frac{1}{n^3} + \sum_{\mathbf{a}\text{ nice}} \mathbb{P}(X=1 \mid \mathbf{A}=\mathbf{a}) \cdot \mathbb{P}(\mathbf{A}=\mathbf{a}) \\
&\overset{(b)}{\leqslant} \frac{1}{n^3} + \sum_{\mathbf{a}\text{ nice}} \frac{1}{n^3} \cdot \mathbb{P}(\mathbf{A}=\mathbf{a}) \\
&\leqslant \frac{2}{n^3} \\
&\leqslant \frac{1}{n^2},
\end{aligned} \tag{28}$$

where steps (a) and (b) follow from Lemmas 4.5 and 4.8, respectively. This concludes the proof of Proposition 4.4.

4.8. Service and Waiting Time Bounds for the Virtual Queue

4.8.1. Service Time Bounds. We will now use Lemma 4.3 and Proposition 4.4 to bound the mean and variance of the service times in the modified virtual queue.

Lemma 4.9. *The modified batch service times, S'_k, are i.i.d., with*

$$\mathbb{E}(S'_k \mid g_n) \sim (\rho+\epsilon)\cdot\frac{b_n}{n}, \quad \text{and} \quad \mathrm{Var}(S'_k \mid g_n) \lesssim \frac{b_n^2}{n^2}.$$

Proof. We use the fact from Lemma 4.3 that $s \leqslant \mathbb{E}(S'_k \mid g_n) \leqslant s + q(g_n)c_n$, where $c_n \lesssim b_n$. We recall that $s = (\rho+\epsilon)b_n/n$ and use the fact $q(g_n)\leqslant n^{-2}$, as guaranteed by Proposition 4.4. The term $q(g_n)c_n$ satisfies $q(g_n)c_n \lesssim b_n/n^2$, which is of lower order than b_n/n and hence negligible compared to s. This proves the first part of the lemma.

For the second part, we use Lemma 4.3 in the first inequality below, and the fact that $q(g_n)\leqslant n^{-2}$ in the second, to obtain

$$\mathrm{Var}(S'_k \mid g_n) \lesssim q(g_n)c_n^2 \lesssim \frac{b_n^2}{n^2}. \quad \text{Q.E.D}$$

4.8.2. Waiting Time Bounds. Fix n and the graph g_n. Let W^B be a random variable whose distribution is the same as the steady-state distribution of the time that a batch spends waiting in the queue of the virtual queueing system introduced in Section 4.3.2.

Proposition 4.10. *We have that*

$$\mathbb{E}(W^B \mid g_n) \lesssim \frac{b_n}{n}. \tag{29}$$

Proof. As discussed in Section 4.4, the waiting time of a batch, in the virtual queueing system, is dominated by the waiting time in a modified virtual queueing system, which is a GI/GI/1 queue, with independent interarrival times A_k (defined in Section 4.3.1) and independent service times S'_k. Let W' be a random variable whose distribution is the same as the steady-state distribution of the time that a batch spends waiting in the queue of the modified virtual queueing system.

According to Kingman's bound Kingman (1962), W' satisfies

$$\mathbb{E}(W' \mid g_n) \leqslant \tilde{\lambda}\frac{\sigma_a^2+\sigma_s^2}{2(1-\tilde{\rho})},$$

where $\tilde{\lambda}$ is the arrival rate, $\tilde{\rho}$ is the traffic intensity, and σ_a^2 and σ_s^2 are the variances of the interarrival times and service times, respectively, that are associated with the modified virtual queueing system.

From Lemma 4.2, we have

$$\tilde{\lambda} = \frac{1}{\mathbb{E}(A_k)} \leqslant \frac{n}{b_n}$$

and

$$\sigma_a^2 = \mathrm{Var}(A_k) \lesssim \frac{b_n}{n^2}.$$

We now bound

$$\tilde{\rho} = \frac{\mathbb{E}(S'_k \mid g_n)}{\mathbb{E}(A_k)}.$$

From the first part of Lemma 4.9, we have $\mathbb{E}(S'_k \mid g_n) \sim (\rho+\epsilon)b_n/n$. Together with the bound $1/\mathbb{E}(A_k) \leqslant n/b_n$, we obtain that as $n \to \infty$, $\tilde{\rho}$ is upper bounded by a number strictly less than one. We also have, from the second part of Lemma 4.9,

$$\sigma_s^2 = \mathrm{Var}(S'_k \mid g_n) \lesssim \frac{b_n^2}{n^2}.$$

Using these inequalities in Kingman's bound, we obtain

$$\mathbb{E}(W^B \mid g_n) \leqslant \mathbb{E}(W' \mid g_n) \lesssim \frac{n}{b_n}\cdot\frac{b_n^2}{n^2} = \frac{b_n}{n}. \quad \text{Q.E.D}$$

4.9. Completing the Proof of Theorem 3.4

Proof. As discussed in Section 4.3.2, the expected waiting time of a job is upper bounded by the sum of three quantities:

(1) The expected time from the arrival of the job until the arrival time of the batch to which the job belongs. This is bounded above by the expected time until there are ρb_n subsequent arrivals, which is equal to $\mathbb{E}(A_1)$. By Lemma 4.2, this is bounded above by $c_1 b_n/n$, for some constant c_1.

(2) The expected time that the batch waits in the virtual queue. This is also upper bounded by $c_2 b_n/n$, by Proposition 4.10, for some constant c_2.

(3) The service time of the batch, which (by Lemma 4.9) again admits an upper bound of the form $c_3 b_n/n$, for some constant c_3.

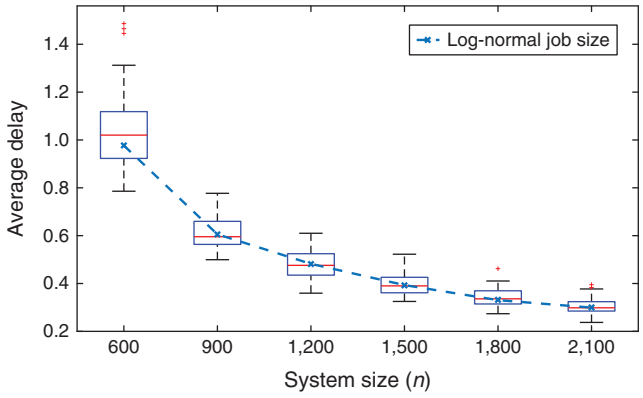
Furthermore, in the results that give these upper bounds, c_1, c_2, and c_3, are absolute constants that do not depend on $\boldsymbol{\lambda}_n$ or g_n.

By our assumptions on the choice of b_n in Section 4.6, we have $b_n = (320/(1-\rho)^2)\cdot(n\ln n/\beta_n)$, and β_n is proportional to d_n. We conclude that there exists a constant c such that for large enough n, we have $\mathbb{E}(W \mid g_n, \boldsymbol{\lambda}_n) \leqslant c\ln n/d_n$, for any given $\boldsymbol{\lambda}_n \in \boldsymbol{\Lambda}_n(u_n)$, which is an upper bound of the desired form. This establishes part 1 of the theorem. Finally, part 2 follows from the way that the policy was constructed. Q.E.D

4.10. On Practical Policies

Figure 5 provides simulation results for the average delay under the virtual-queue–based scheduling policy

Figure 5. (Color online) Simulations of the Virtual-Queue–Based Policy Given in Section 4.3, with $d_n = n^{2/3}$, $b_n = n\ln(n)/d_n$, and $\lambda_i = 0.5$ for all $i = 1, \ldots, n$



Notes. The boxplot contains the average delay from 50 runs of simulations where the job size distribution is assumed to be exponential with mean one. Each run is performed on a random d_n-regular graph over 10^4 service slots and a 1,000-slot burn-in period. The center line of a box represents the median, and the upper and lower edges of the box represent the 25th and 75th percentiles, respectively. The dashed line depicts the median average waiting times when the job sizes are distributed according to a log-normal distribution with mean one and variance 10.

used in proving Theorem 3.4. The main role of the policy is to demonstrate the fundamental potential of the expander architecture in jointly achieving a small delay and large capacity region when the system size is large. In smaller systems, however, there could be other policies that yield better performance. For instance, simulations suggest that a seemingly naive greedy heuristic can achieve a smaller delay in moderately sized systems, which is practically zero in the range of parameters in Figure 5. Under the greedy heuristic, an available server simply fetches a job from a longest connected queue, and a job is immediately sent to a connected idle server on arrival if possible. Intuitively, the greedy policy can provide a better delay because it avoids the overhead of holding jobs in queues while forming a batch. Unfortunately, it appears challenging to establish rigorous delay or capacity guarantees for the greedy heuristic and other similar policies.

In some applications, such as call centers, the service times or job sizes may not be exponentially distributed (Brown et al. 2005). In Figure 5, we also include the scenario where the job sizes are drawn from a log-normal distribution (Brown et al. 2005) with an increased variance. Interestingly, the average delay appears to be somewhat insensitive to the change in job size distribution.

5. Summary and Future Research

The main message of this paper is that the two objectives of a large capacity region and an asymptotically vanishing delay can be simultaneously achieved even if the level of processing flexibility of each server is small compared to the system size. Our main results show that, as far as these objectives are concerned, the family of expander architectures is essentially optimal: it admits a capacity region whose size is within a constant factor of the maximum possible, while ensuring an asymptotically vanishing queueing delay for all arrival rate vectors in the capacity region.

An alternative design, the random modular architecture, guarantees small delays for "many" arrival rates, by means of a simple greedy scheduling policy. However, for any given modular architecture, there are always many arrival rate vectors in $\mathbf{\Lambda}_n(u_n)$ that result in an unstable system, even if the maximum arrival rate across the queues is of constant order. Nevertheless, the simplicity of the modular architectures can still be appealing in some practical settings.

Our result for the expander architecture leaves open three questions:

1. Is it possible to lower the requirement on the average degree from $d_n \gg \ln n$ to $d_n \gg 1$?
2. Without sacrificing the size of the capacity region, is it possible to achieve a queueing delay which approaches zero exponentially fast as a function of d_n? The delay scaling in Theorem 3.4 is $\mathcal{O}(\ln n/d_n)$.
3. Is it possible to obtain delay and stability guarantees under simpler policies, such as the greedy heuristic mentioned in Section 4.10? The techniques developed in Visschers et al. (2012) for analyzing first-come, first-served scheduling rules in a multiclass queueing network similar to ours could be a useful starting point.

Finally, the scaling regime considered in this paper assumes that the traffic intensity is fixed as n increases, which fails to capture system performance in the heavy-traffic regime ($\rho \approx 1$). It would be interesting to consider a scaling regime in which ρ and n scale simultaneously (e.g., as in the celebrated Halfin–Whitt regime Halfin and Whitt 1981), but it is unclear at this stage what the most appropriate formulations and analytical techniques are.

Acknowledgments

A preliminary version of this paper appeared at SIGMETRICS 2013 (Tsitsiklis and Xu 2013); the performance of the architectures proposed in the current paper is significantly better than the one in Tsitsiklis and Xu (2013).

Endnotes

[1] A work-conserving policy mandates that a server be always busy whenever there is at least one job in some queue to which it is connected.

[2] The fact that the expected waiting time vanishes asymptotically follows from the bounded expected total number of jobs in steady state, the assumption that the total arrival rate is ρn, which goes to infinity as $n \to \infty$, and Little's law.

[3] For simplicity of notation, we omit the dependence of E, I, and J on n.

[4] While we restrict to binding and nonpreemptive scheduling policies, other common architectures where (a) a server can serve multiple jobs concurrently (processor sharing); (b) a job can be served

by multiple servers concurrently; or (c) job sizes are revealed upon entering the system, are clearly more powerful than the current setting, and are therefore capable of implementing the scheduling policies considered in this paper. As a result, the performance upper bounds developed in this paper also apply to these more powerful variations.

[5] Note that $\mathbb{E}(W \mid \lambda)$ captures a *worst-case* expected waiting time across all jobs in the long run and is always well defined, even under scheduling policies that do not induce a steady-state distribution.

[6] We are using here the following elementary lemma. Let A be an event with $\mathbb{P}(A) \geqslant 1-\epsilon$, and let X be a random variable. Then, there exists a set B with $\mathbb{P}(B) \geqslant 1-\sqrt{\epsilon}$ such that $\mathbb{P}(A \mid X) \geqslant 1-\sqrt{\epsilon}$, whenever $X \in B$. The lemma is applied by letting A be the event $\{\lambda_n \in \mathbf{R}(G_n)\}$ and letting $X = G_n$.

[7] In a slight departure from the earlier informal description, we define batches by keeping track of the number of arriving jobs as opposed to keeping track of time.

[8] To see how the length of the service slot was chosen, recall that the size of each batch is equal to ρb_n. The length of the service slot hence ensures that $(\rho+\epsilon)b_n$, the expected number of servers that will become available (and can therefore be assigned to jobs) during a single service slot, is greater than the size of a batch, so that there is hope of assigning all of these jobs to available servers within a single service slot. At the same time, since $\rho+\epsilon<1$, service slots are shorter than the expected batch interarrival time, which is needed for the stability of the virtual queue.

References

Bassamboo A, Randhawa RS, Mieghem JAV (2012) A little flexibility is all you need: On the asymptotic value of flexible capacity in parallel queuing systems. *Oper. Res.* 60(6):1423–1435.

Bell SL, Williams RJ (2001) Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: Asymptotic optimality of a threshold policy. *Ann. Appl. Probab.* 11(3):608–649.

Brown L, Gans N, Mandelbaum A, Sakov A, Shen H, Zeltyn S, Zhao L (2005) Statistical analysis of a telephone call center: A queueing-science perspective. *J. Amer. Statist. Assoc.* 100(469):36–50.

Chen X, Zhang J, Zhou Y (2015) Optimal sparse designs for process flexibility via probabilistic expanders. *Oper. Res.* 63(5): 1159–1176.

Chou M, Teo C-P, Zheng H (2011) Process flexibility revisited: The graph expander and its applications. *Oper. Res.* 59(5):1090–1105.

Chou M, Chua GA, Teo C-P, Zheng H (2010) Design for process flexibility: Efficiency of the long chain and sparse structure. *Oper. Res.* 58(1):43–58.

Gurumurthi S, Benjaafar S (2003) Modeling and analysis of flexible queueing systems. *Naval Res. Logist.* 51(5):755–782.

Halfin S, Whitt W (1981) Heavy-traffic limits for queues with many exponential servers. *Oper. Res.* 29(3):567–588.

Harrison JM, Lopez MJ (1999) Heavy traffic resource pooling in parallel-server systems. *Queueing Systems* 33(4):339–368.

Hoeffding W (1963) Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.* 58(301):13–30.

Hoory S, Linial N, Wigderson A (2006) Expander graphs and their applications. *Bull. Amer. Math. Soc.* 43(4):439–561.

Iravani SM, Oyen MPV, Sims KT (2005) Structural flexibility: A new perspective on the design of manufacturing and service operations. *Management Sci.* 51(2):151–166.

Jordan W, Graves SC (1995) Principles on the benefits of manufacturing process flexibility. *Management Sci.* 41(4):577–594.

Kandula S, Sengupta S, Greenberg A, Patel P, Chaiken R (2009) The nature of data center traffic: Measurements and analysis. Felmann A, Mathy L, eds. *Proc. 9th ACM SIGCOMM Conf. Internet Measurement, IMC '09* (ACM, New York), 202–208.

Kingman J (1962) Some inequalities for the queue GI/G/1. *Biometrika* 49(3/4):315–324.

Kunniyur S, Srikant R (2001) Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. Cruz RL, Varghese G, eds. *Proc. ACM SIGCOMM Conf., SIGCOMM '01* (ACM, New York), 123–134.

Leconte M, Lelarge M, Massoulie L (2012) Bipartite graph structures for efficient balancing of heterogeneous loads. *ACM SIGMETRICS Performance Evaluation Rev.* 40(1):41–52.

Mandelbaum A, Reiman MI (1998) On pooling in queueing networks. *Management Sci.* 44(7):971–981.

McKeown N, Mekkittikul A, Anantharam V, Walrand J (1999) Achieving 100% throughput in an input-queued switch. *IEEE Trans. Comm.* 47(8):1260–1267.

Neely M, Modiano E, Cheng Y (2007) Logarithmic delay for $n \times n$ packet switches under the crossbar constraint. *IEEE/ACM Trans. Networking* 15(3):657–668.

Shah D, Tsitsiklis JN (2008) Bin packing with queues. *J. Appl. Probab.* 45(4):922–939.

Simchi-Levi D, Wei Y (2012) Understanding the performance of the long chain and sparse designs in process flexibility. *Oper. Res.* 60(5):1125–1141.

Soundararajan G, Amza C, Goel A (2006) Database replication policies for dynamic content applications. *ACM SIGOPS Operating Systems Rev.*, 40(4):89–102.

Talreja R, Whitt W (2008) Fluid models for overloaded multiclass many-server queueing systems with first-come, first-served routing. *Management Sci.* 54(8):1513–1527.

Tsitsiklis JN, Xu K (2012) On the power of (even a little) resource pooling. *Stochastic Systems* 2(1):1–66.

Tsitsiklis JN, Xu K (2013) Queueing system topologies with limited flexibility. *ACM SIGMETRICS Performance Evaluation Rev.* 41(1):167–178.

Visschers J, Adan I, Weiss G (2012) A product form solution to a system with multi-type jobs and multi-type servers. *Queueing Systems* 70(3):269–298.

Wallace R, Whitt W (2005) A staffing algorithm for call centers with skill-based routing. *Manufacturing Service Oper. Management* 7(4):276–294.

Xu K (2014) On the power of (even a little) flexibility in dynamic resource allocation. PhD thesis, Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/91101.

John N. Tsitsiklis is currently a Clarence J. LeBel Professor in the Department of Electrical Engineering and Computer Science (EECS) at MIT.His research interests are in the fields of systems, optimization, control, and operations research. He is a coauthor of four textbooks and research monographs, and is also a coinventor in seven awarded U.S. patents. He has been a recipient of several awards, including, most recently, the ACM SIGMETRICS Achievement Award (2016). He is a Fellow of the IEEE and of INFORMS, and a member of the National Academy of Engineering. In 2008, he was conferred the title of Doctor Honoris Causa from the Université catholique de Louvain (Belgium).

Kuang Xu is an assistant professor of operations, information, and technology at the Graduate School of Business at Stanford University. His research interests are in the analysis and design of large-scale stochastic systems, and applications drawn from engineering, operations, and management. He has received several awards including a first place of the INFORMS George E. Nicholson Student Paper Competition, and Best Paper and Kenneth C. Sevcik Outstanding Student Paper Awards from ACM SIGMETRICS.