Most Balancing Algorithms for Optimal Packet Scheduling in Multi-Server Wireless Systems

Hussein Al-Zubaidy, Changcheng Huang, James Yan,
Carleton University, Ottawa, ON, Canada, e-mail: {hussein, huang}@sce.carleton.ca, jim.yan@sympatico.ca.

***Abstract*—We present an algorithm to implement "most balancing" packet scheduling policies in a discrete-time multi-server system of parallel queues with independent random queue-server connectivity. The MB policies are characterized by minimizing the total difference in queue lengths at every time slot. The proposed algorithm produces the server allocation strategy that achieves the minimum "imbalance index" at every time slot. This algorithm has a reduced computational complexity compared to a full-search through the set of all feasible policies. We also provide a low-complexity approximate implementation algorithm that performs close to the exact algorithm. Simulation results confirm our claims.**

I. Introduction and Model Description

Emerging wireless networks are IP-based packet access networks that are characterized by their capability of providing high data rate on the last hop [1]. Packet scheduling in emerging wireless networks plays a major role in achieving the desired data rate. In these systems, there are multiple network resources, e.g., CDMA codes, OFDMA channels, etc., to be allocated to users at every time slot. A user might be allocated the minimum portion or the full size of the available network resources at every time slot depending on the implemented packet scheduling policy [2] [3]. The connectivity of users to the base station in any wireless system is varying with time and can be best modelled as a random process.

The wireless system is modelled by a set of L parallel queues with infinite capacity (see Figure 1). The time in this system is slotted into equal length deterministic intervals. Let $X_i(n)$ be the number of packets in the i^{th} queue at the beginning of time slot n. There are K identical servers to be shared between the L queues in the system. The service time required per packet is assumed to have constant length that is equal to one time slot. A server can serve one packet from a connected, non-empty queue, at any given time slot.

The queue-server connectivity between the i^{th} queue and the j^{th} server during the n^{th} time slot is denoted by $G_{i,j}(n)$ and can be either connected ($G_{i,j}(n) = 1$) or not connected ($G_{i,j}(n) = 0$). We assume that, for all $i = 1, 2, \ldots, L$, $j = 1, 2, \ldots, K$ and n, $G_{i,j}(n)$ are independent Bernoulli random variables with parameter p[1]. The number of arrivals to the i^{th} queue during time slot n is denoted by $Z_i(n)$. The arrival processes to different queues ($\{Z_i(n)\}, \forall i$) are assumed to be independent of each other and independent of the processes $\{G_{i,j}(n)\}$ for $i = 1, 2, \ldots, L$, $j = 1, 2, \ldots, K$.

[1]Although, the application of MB policies is not limited to systems with symmetrical arrivals and link statistics, the MB is proven optimal for these systems only [4]. Therefore, we limit our description to such systems.

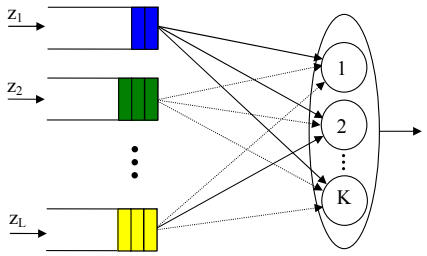

Fig. 1. Scheduler model in emerging wireless network.

A packet scheduling (or server allocation) policy decides, at every time slot, what servers will be allocated to which queue (or alternatively, how many packets will be withdrawn from every queue) during that time slot. In previous work, we have shown that the class of *most balancing* (MB) policies is optimal [4], in that they minimize, in a stochastic ordering sense, a range of cost functions of the system queue sizes, including the total number of queued packets, in the aforementioned system. An MB policy attempts to balance all queue sizes at every time slot, so that the total sum of queue size differences will be minimized. In [4], we used stochastic dominance [6] and coupling arguments [7] to prove the optimality of the MB policies. We also proved that the MB policies are still optimal when retransmission of packets, resulted from previously failed transmission attempts, is considered [5].

A special case of the previous model was investigated in [8]. The authors tackled a simpler problem where a single server (i.e., $K = 1$) can only be allocated to one user at every time slot. They proved that a policy which allocates the available server to its longest connected queue is optimal. In such model, the LCQ policy belong to the set of MB policies. Another relevant result is the one reported in [9]. The authors studied a system of L homogeneous, parallel queues competing for K identical servers. At each time slot, no more than one server is allocated to each scheduled queue. They proved that LCQ, a policy that allocates the K servers to the K longest connected queues at every time slot, is optimal. This policy can also be considered as an MB policy within the model limitations and constraints. A more recent result that has relevance to the optimality of the MB policies is the one reported in [10]. They proved, using dynamic programming, that a maximum-throughput and load-balancing (MTLB) policy minimizes the expected average cost for a two-queue multi-server system.

The MB policies maybe implemented using an algorithm that searches through the full range of the feasible action space at every time slot. Such an algorithm poses significant com-

978-1-61284-254-7/11/$26.00 ©2011 IEEE

putational challenge especially for systems with large L and K. The objective of this work is to devise an implementation algorithm for the MB policies that has a reduced computational complexity compared to the full search algorithm.

In this work, we present an implementation algorithm (Algorithm 1) and prove that the resulted policy of that algorithm is most balancing. In addition, we present a policy (that we call LCSF/LCQ) and its implementation algorithm (Algorithm 2), to approximate the behaviour of MB policies. This policy has low complexity compared to Algorithm 1 and has a performance that is comparable to that of MB policies.

The rest of the paper is organized as follows. In section II, we introduce notation and feasibility conditions. In section III, we introduce the MB policies. In section IV, we present an MB implementation algorithm. In section V, we introduce the LCSF/LCQ policy, a practical approximation for MB policies. In section VI, we present simulation results.

II. Formulation and Policy Definition

In this section we will provide notation that we will use in the rest of this paper. We will also provide a formal definition and feasibility constraints for server allocation policies in the system described earlier. We represent the policy action that corresponds to "idling" a server by introducing a special, "dummy" queue that we denote by queue 0. Allocating a server to this queue is equivalent to idling that server. We assume that queue 0 is always connected to all servers and contains only "dummy" packets We will use the following notation:

- $\mathbf{X}(n) = (X_0(n), X_1(n), X_2(n), \ldots, X_L(n))^T$ is the queue lengths vector (measured in number of packets) at time slot n. We assume $X_0(n) = 0$.
- $\mathbf{Z}(n) = (Z_0(n), Z_1(n), Z_2(n), \ldots, Z_L(n))^T$ is a vector with its elements represent the number of exogenous arrivals to each queue during time slot $n = 1, 2, \ldots$.
- $\mathbf{G}(n)$ is an $(L+1) \times K$ matrix, where $G_{i,j}(n)$ for $i > 0$ is the channel connectivity random variable. We assume that $G_{0,j}(n) = 1$ for all j, n.
- $\mathbf{Q}(n) = (Q_1(n), \ldots, Q_K(n))^T$ is the *server allocation control* vector. $Q_j(n) \in \{0, 1, \ldots, L\}$ denotes the index of the queue that is selected (according to some rule) to be served by server j during time slot n. Note that setting $Q_j(n)=0$ means that server j is idled at time slot n.
- $\mathbf{Y}(n) = (Y_0(n), Y_1(n), Y_2(n), \ldots, Y_L(n))^T$ is the *withdrawal control* For any i, $Y_i(n) \in \{0, 1, \ldots, K\}$ denotes the number of packets withdrawn from queue i during time slot n.
- $\mathbf{V}(n)$ is a $(L+1) \times K$ matrix. $V_{i,j}(n) = \mathbb{1}_{\{i=Q_j(n)\}} \cdot G_{i,j}(n), i = 0, \ldots, L$ and $j = 1, \ldots, K$.
- The tuple $(\mathbf{X}(n), \mathbf{G}(n))$ denotes the "state" of the system at the beginning of time slot n.

Where $\mathbb{1}_{\{A\}}$ denotes the indicator function for condition A.

Using the previous notation and given a scheduling control vector $\mathbf{Q}(n)$, the withdrawal control vector is computed using:

$$Y_i(n) = \sum_{j=1}^{K} \mathbb{1}_{\{i=Q_j(n)\}}, \quad i = 0, 1, 2, \ldots, L. \tag{1}$$

A. Feasibility Conditions

We assume that the state information is available to the controller at any time slot n. Then at n, a vector $\mathbf{Q}(n) \in \{0, 1, \ldots, L\}^K$ is a *feasible* server allocation control if: (a) a server is allocated to one connected queue, and (b) the number of servers allocated to a queue cannot exceed the size of the queue. Mathematically, these conditions may be stated using the following (necessary and sufficient) constraints:

$$\mathbf{V}^T(n) \cdot \mathbf{I}_{L+1} = \mathbf{I}_K \tag{2}$$

$$\mathbf{V}^*(n) \cdot \mathbf{I}_K \leq \mathbf{X}(n) \tag{3}$$

where $\mathbf{I}_l$ is l-dimensional vector with all entries equal 1, and

$$V^*_{i,j}(n) = \begin{cases} 0, & i = 0; \\ V_{i,j}(n), & \text{otherwise.} \end{cases}$$

The K constraints in Equation (2) satisfies condition (a) above. Condition (b) captured by the point-wise inequality in (3). To insure that Inequality (3) is satisfied for the dummy queue, we use $\mathbf{V}^*(n)$ instead of $\mathbf{V}(n)$ in the inequality. Note that allocating more than one server to a queue is feasible.

Similarly, we say that a withdrawal vector $\mathbf{Y}(n) \in \{0, 1, \ldots, K\}^{L+1}$ is *feasible* (during time slot n) if there exist a matrix $\mathbf{V}(n)$ that satisfies the constraints (2) and (3) s.t.

$$\mathbf{Y}(n) = \mathbf{V}(n) \cdot \mathbf{I}_K \tag{4}$$

We denote the set of all feasible withdrawal controls while in state $(\mathbf{x}, \mathbf{g})$ by $\mathcal{Y}(\mathbf{x}, \mathbf{g})$. For any given feasible control $\mathbf{y}(n)$, we refer to $\mathbf{q}(n)$ as its *implementation*.

B. Policies for Server Allocation

For any feasible control ($\mathbf{Y}(n)$), the system described earlier evolves according to

$$\mathbf{X}(n+1) = \mathbf{X}(n) - \mathbf{Y}(n) + \mathbf{Z}(n), \qquad n = 1, 2, \ldots \tag{5}$$

We assume that arrivals during time slot n can only be added after removing served packets.

A *server allocation policy* π (or policy π for simplicity) is a rule that determines feasible withdrawal vectors $\mathbf{Y}(n)$ for all n, as a function of the past history and current state of the system $\mathbf{H}(n)$. The state history is given by the sequence of random variables

$$\begin{aligned} \mathbf{H}(1) &= (\mathbf{X}(1)), \quad \text{and} \\ \mathbf{H}(n) &= (\mathbf{X}(1), \mathbf{G}(1), \mathbf{Z}(1), \ldots, \mathbf{G}(n-1), \mathbf{Z}(n-1), \mathbf{G}(n)), \\ &\qquad n = 2, 3, \ldots \end{aligned} \tag{6}$$

Let $\mathcal{H}_n$ be the set of all histories up to time slot n. Then a policy π can be formally defined as the sequence of measurable functions

$$\begin{aligned} & u_n : \mathcal{H}_n \longmapsto \mathcal{Z}_+^{L+1}, \\ \text{s.t.} \quad & u_n(\mathbf{H}(n)) \in \mathcal{Y}(\mathbf{X}(n), \mathbf{G}(n)), \quad n = 1, 2, \ldots \end{aligned} \tag{7}$$

$\mathcal{Z}_+$ is the set of non-negative integers, $\mathcal{Z}_+^{L+1} = \mathcal{Z}_+ \times \ldots \times \mathcal{Z}_+$, where the Cartesian product is taken $L+1$ times.

At each time slot, the following sequence of events happens: First, $\mathbf{G}(n)$ and $\mathbf{X}(n)$ are observed. Second, $\mathbf{Y}(n)$ is

determined according to the policy in effect. Finally, $\mathbf{Z}(n)$ are added and $\mathbf{X}(n+1)$ is computed.

III. MB Policies for Server Allocation

In this section, we provide a formal description and mathematical characterization of the class of MB policies.

Intuitively, the MB policies "attempt to balance the lengths of all queues in the system *as much as possible*, at every time slot n"; they do so by choosing a control ($\mathbf{y}(n) \in \mathcal{Y}(\mathbf{x}, \mathbf{g})$) that minimizes the number of cases where a server is not allocated to its longest connected queue. For a more formal definition of MB policies, we first define the following:

Given a state $(\mathbf{x}(n), \mathbf{g}(n))$ and a policy π that chooses the feasible control $\mathbf{y}(n)$ at time slot n, define the "updated queue size" $\hat{x}_i(n) = x_i(n) - y_i(n)$ as the size of queue $i, i = 1, \ldots, L$, after applying the control $y_i(n)$ and just before adding the arrivals during time slot n. For notational simplicity we also define $\hat{x}_0(n) = 0$. Furthermore, we define the *"imbalance index"*, $\kappa_n(\pi)$, as the total sum of differences of the $L+1$-dimensional vector $\hat{x}(n)$ under the policy π at n (where π selects $\mathbf{y}(n) \in \mathcal{Y}(\mathbf{x}, \mathbf{g})$ at time slot n), i.e.,

$$\kappa_n(\pi) = \sum_{i=1}^{L+1} \sum_{j=i+1}^{L+1} (\hat{x}_{[i]}(n) - \hat{x}_{[j]}(n)) \tag{8}$$

where the subscript $[k]$ denotes the index of the k^{th} longest component of the vector. By convention, queue '0' (the "dummy queue") will always have order $L+1$ (i.e., the queue with the minimum length). Let Π^{MB} denotes the set of all MB policies.

Definition: A *Most Balancing* (MB) policy is a policy $\pi \in \Pi^{MB}$ that, at $n = 1, 2, \ldots$, chooses feasible withdrawal vectors $\mathbf{y}(n) \in \mathcal{Y}(\mathbf{x}, \mathbf{g})$ such that the imbalance index is minimized at every n, i.e.,

$$\Pi^{MB} = \left\{ \pi : \underset{\mathbf{y}(n) \in \mathcal{Y}(\mathbf{x}, \mathbf{g})}{\operatorname{argmin}} \kappa_n(\pi), \quad \forall n \right\} \tag{9}$$

The set Π^{MB} in Equation (9) is well-defined and non-empty, since the minimization is over a finite set. The set of MB policies may have more than one element. This could happen, for example, when at a given time slot n, a server k is connected to two or more queues of equal size, which happen to be the longest queues connected to this server. Then, serving either one of them will satisfy Equation (9).

IV. Implementation Algorithm for MB Policies

The definition of MB policies via Equation (9) is not constructive. Therefore, designing an algorithm to implement an MB policy is required. A determination of an MB policy given $\mathbf{X}(t)$ and $\mathbf{G}(t)$ can be done using a direct search over all possible server allocations. This can be a challenging computational task. We present next an algorithm that searches in a subset of the feasible policies. We prove that it will produce an MB policy. This algorithm has a complexity factor of $O(L \times K!)$.

Recall that there are K servers to be allocated at every time slot. These servers can be ordered in one of $K!$ different server orderings. We consider a given permutation (ordering) of the K servers in the system. For this permutation we define a "sequential LCQ server allocation" a process of allocating the servers to queues in K steps as follows: Starting from the first server, we assign it to its longest connected queue and we update (i.e., reduce by one) its queue size. We continue with the second server following the same principle until we exhaust all servers in K steps. We will show that at least one "sequential LCQ server allocation" corresponding to an ordering among the $K!$ server permutations will result in an MB policy.

Let the set $\mathcal{M}_i^t$ be the set of servers connected to queue i during time slot t. Let $M_i(t) \triangleq |\mathcal{M}_i^t|$ be the number of servers that are connected to queue i during time slot t, that is

$$M_i(t) = \sum_{j=1}^{K} G_{i,j}(t) \tag{10}$$

Let $\mathbb{Q}_k \triangleq \{i : k \in \mathcal{M}_i^t\}$ denote the set of queues that are connected to server k during time slot t; we omit the dependence on t to simplify notation. Let Θ denote the set of all possible permutations of the set $\{1, \ldots, K\}$. We define a server ordering at time n as a permutation $\theta(n) \in \Theta$. There are $|\Theta| = K!$ possible server orderings. We use the subscript $[j]^\theta$ to denote the j^{th} server to be allocated under the ordering rule $\theta(n)$. Algorithm 1 below present the pseudo-code for the approach we described previously.

Algorithm 1 (MB_Implementation 1)**.**

1. $for \quad t = 1, 2, \ldots \quad do \quad \{$
2. *Input:* $\mathbf{X}(t), \mathbf{G}(t)$.*Calculate:* $\mathbb{Q}_{[k]}$, $k = 1, \ldots, K$.
3. *Let:* $\kappa_t^{min} = L \cdot \max_l X_l$ *; maximum possible* κ_t
4. $forall \quad \theta \in \Theta \quad do \quad \{$ *; loop* $|\Theta| = K!$ *times*
5. $\mathbf{X}' \longleftarrow \mathbf{X}(t)$, $\mathbf{Y}' \longleftarrow \mathbf{0}$, $\mathbf{Q}' \longleftarrow \mathbf{0}$
6. $for \quad j = 1 \quad to \quad K \{$ *; allocate servers sequentially*
7. $Q'_{[j]^\theta} = \min \left(k : k \in \left\{ \underset{l : l \in \mathbb{Q}_{[j]^\theta}}{\operatorname{argmax}} (X'_l | X'_l > 0) \right\} \right)$
8. *Let:* $i = Q'_{[j]^\theta}$
9. $Y'_i = Y'_i + 1$
10. $X'_i = X'_i(t) - 1 \quad \}$
11. *Compute:* κ_t^θ *from Equation(8)*
12. $if \quad (\kappa_t^\theta < \kappa_t^{min}) \quad \{$
13. $\kappa_t^{min} = \kappa_t^\theta$
14. $\mathbf{y}(t) \longleftarrow \mathbf{Y}'$, $\mathbf{q}(t) \longleftarrow \mathbf{Q}'$, $\theta(t) \longleftarrow \theta \; \}$
15. $\}$ $\}$ *; End of Algorithm 1.*

Theorem 1 states the main result in this section; Algorithm 1 generates an MB policy for the system in Figure 1.

Theorem 1. *The server allocation policy obtained by applying Algorithm 1 is an MB policy.*

Proof: A policy π is an MB policy if it has the MB property at every time slot $n = 1, 2, \ldots$, i.e., it minimizes the total differences between the queues' lengths in the system at every n. To prove Theorem 1 we have to show that Algorithm 1 produces a policy that has the MB property at all time slots.

For the proof of Theorem 1 we first introduce the following properties of a server in a sequential server allocation during time slot n (where the queue lengths are updated after each server allocation) such as that used in Algorithm 1. Given a server allocation policy[2] π^θ and following a server ordering θ, we define the allocation of a server to its longest connected queue (according to π^θ) as "LCQ allocation" (equivalently we may say that the server has the LCQ property). Otherwise, we refer to this allocation as "NLCQ allocation" or we say that the server has NLCQ property. We should note that the LCQ or NLCQ properties of the servers depend on the selected server allocation order θ.

Proceeding with the proof for Theorem 1, we consider a policy π^{θ_1}, that is implemented using sequential server allocation and a server order θ_1 during time slot n, such that π^{θ_1} has the MB property at time slot n. We assume that at least one server has the NLCQ property (i.e., allocated to a queue that is not its longest connected queue) according to this implementation, since otherwise the theorem is trivially true. We will show next that we can construct a server allocation ordering θ_2 under which π^{θ_2} has the MB property at time slot n and all servers have the LCQ property under θ_2.

To complete the proof of Theorem 1, we will need the following lemmas. Their proof can be found in [11].

Lemma 1. *Given a server allocation ordering $\theta \in \Theta$ at time slot n and a policy π^θ that has the MB property at time slot n, if $s_{[K]^\theta}$ (i.e., the last allocated server under θ at time slot n) has the NLCQ property then a policy π^* can be constructed in which: (i) $s^*_{[K]^\theta}$ (i.e., the last server to be allocated under π^*) has the LCQ property at time slot n and (ii) $s^*_{[k]^\theta}$ has the same allocation as $s_{[k]^\theta}, \forall k : 1 \leq k < K$, such that π^* has the MB property at time slot n.*

Lemma 2. *Given a policy π^θ that has the MB property during time slot n, swapping the order of two consecutively allocated servers s_1, s_2 under π^θ, the second of which (s_2) has the LCQ property, will not change the LCQ property of that server under the new ordering.*

The construction of the new ordering θ_2 during time slot n (as described earlier) is summarized in the following steps:

(1) We identify the last NLCQ allocated server (s_i) under π^{θ_1}. Denote the order (in θ_1) of this server by i^* (i.e., $s_i = s_{[i^*]^{\theta_1}}$). Now if this is the last server to be allocated (i.e., $i^* = K$), then go to step (4).

(2) Using Lemma 2 we can swap server s_i with the server next in order, i.e., $s_{[i^*+1]^{\theta_1}}$, (which has the LCQ property according to step (1)) and create a new server ordering θ'_1, in which the swapped server has order $[i^*]^{\theta'_1}$ and retain it's LCQ property.

(3) Repeat step (2) until s_i is the last server to be allocated under θ'_1.

(4) Allocate server s_i to its longest connected queue. According to Lemma 1 the resulting policy will have the MB property at time slot n (with the last server has LCQ property)

(5) Repeat steps (1) to (4) until all servers have the LCQ property. This will result in a new ordering θ_2 and new policy π^{θ_2} that has the MB property at time slot n with all servers having LCQ property under θ_2 at the corresponding slot. ■

[2]By π^θ we denote a policy that is implemented using a sequential server allocation following the order $\theta \in \Theta$

V. LCSF/LCQ to Approximate MB Policies

We present a policy that approximate the behaviour of the MB policies and a feasible implementation algorithm for this policy. Compared to exact implementation algorithms, this algorithm has significantly lower computational complexity which facilitates its use in real systems (as well as simulated systems) with large values of L and K.

We introduce the *Least Connected Server First/Longest Connected Queue* (LCSF/LCQ) policy, a low-overhead approximation of MB policy, with $O(L \times K)$ computational complexity. We show that it results in a feasible withdrawal vector. The policy is stationary and depends only on the current state $(\mathbf{X}(n), \mathbf{G}(n))$ during time slot n.

The LCSF/LCQ implementation during a given time slot is described as follows: The least connected server is identified and is allocated to its longest connected queue. The queue length is updated (i.e., decremented). We proceed accordingly to the next least connected server until all servers are assigned. In algorithmic terms, the LCSF/LCQ policy can be described/implemented as follows:

Algorithm 2 (LCSF/LCQ Implementation)**.**

1. $for \quad t = 1, 2, \ldots \quad do \quad \{$
2. *Input:* $\mathbf{X}(t), \mathbf{G}(t)$. *Calculate* $\mathbb{Q}_{[l]}, l = 1, \ldots, K.$
3. $\mathbf{X}' \longleftarrow \mathbf{X}(t),\ \mathbf{Y} \longleftarrow \mathbf{0},\ \mathbf{Q} \longleftarrow \mathbf{0}$
4. $for\ j = 1 \quad to \quad K \quad \{$ *; allocate servers sequentially*
5. $Q_{[j]} = \min \left(l : l \in \left\{ \underset{k:k \in \mathbb{Q}_{[j]}}{\operatorname{argmax}} (X'_k | X'_k > 0) \right\} \right)$
6. $for\ i = 1 \quad to \quad L \quad \{$
7. $Y_i = Y_i + \mathbb{1}_{\{i = Q_{[j]}\}}$
8. $X'_i = X_i(t) - Y_i \quad \} \quad \}$
9. *Output:* $\mathbf{y}(t) \longleftarrow \mathbf{Y}, \mathbf{q}(t) \longleftarrow \mathbf{Q}$ *; report outputs*
10. $\}$ *; End of Algorithm 2.*

Recall that $\mathbb{Q}_j$ denotes the set of all queues that are connected to server j at time slot t. Let $\mathbb{Q}_{[i]}$ be the i^{th} element in the sequence $(\mathbb{Q}_1, \ldots, \mathbb{Q}_K)$, when ordered in ascending manner according to their size (set cardinality),

i.e., $|\mathbb{Q}_{[l]}| \geq |\mathbb{Q}_{[m]}|$ if $l > m$. Ties are broken arbitrarily. Then under the LCSF/LCQ policy, the K servers are allocated according to Algorithm 2. Note that in line 5 of Algorithm 2, if the set $\mathbb{Q}_{[j]}$ is empty, then the argmax returns the empty set. In this case, the j^{th} order server will not be allocated (i.e., will be idle during time slot t). Algorithm 2 produces two outputs, when ran at $t = n$: $\mathbf{y}(n)$ and $\mathbf{q}(n)$ as shown in line 9 of the algorithm. Although allocating the available servers to their longest connected queues in the order specified by Algorithm 2 may not be "most balancing", the LCSF/LCQ is expected to perform very close to MB policy.

In accordance to the definition of a policy in Equation (7), the LCSF/LCQ policy can be formally defined as the sequence of time-independent mappings $u(\mathbf{x}(n), \mathbf{g}(n))$ that produce the withdrawal vector $\mathbf{y}(n)$ described in line 9. Lemma 3 asserts that the mapping defines feasible controls.

Lemma 3. *The policy obtained from applying Algorithm 2 results in a feasible withdrawal vector at every time slot n and any state $(\mathbf{x}(n), \mathbf{g}(n))$.*

Proof: Let $\mathbf{y}(n)$ and $\mathbf{q}(n)$ denote the outputs of Algorithm 2 Let $\mathbf{V}(n)$ be as defined previously. We must show that the output $\mathbf{y}(n)$ can be written as

$$\mathbf{y}(n) = \mathbf{v}(n) \cdot \mathbf{I}_K \tag{11}$$

and that $\mathbf{v}(n)$ satisfies the feasibility constraints (2) and (3).

From Algorithm 2, line 5, it can be seen that for every server $[j]$, only the set of queues that are connected to server $[j]$ are considered as candidates for allocating this server. Therefore, $v_{i,[j]}(n) = 1$ is true only when $g_{i,[j]}(n) = 1$ and $\mathbb{1}_{\{i=q_{[j]}(n)\}} = 1$ are true.From Equations (11) and (3) we can easily see that

$$\mathbf{y}(n) \leq \mathbf{x}(n) \tag{12}$$

is a sufficient condition for Inequality (3) to hold. Note that queue i will be selected in Algorithm 2, line 5 (to be served by server $[j]$) only if its current size x'_i is strictly positive. This will ensure that the number of servers allocated to any queue is no larger than its queue size. Therefore, $y_i(n) \leq x_i(n), i = 1, \ldots, L$, proving Inequality (12).

Constraints (2) are satisfied. To prove that, fix a server $[j]$; the initialization step 3 assigns this server to the dummy queue. Observe that even though the inner for-loop in Algorithm 2 is executed $L+1$ times, the indicator function $\mathbb{1}_{\{i=Q_{[j]}\}}$ in line 7 is non-zero for only one value of $i \in \{0, 1, \ldots, L\}$; each server is allocated to one queue only, either the dummy queue or the queue with the minimum index out of the outcome of the argmax function in line 5 of Algorithm 2. Therefore

$$\sum_{i=0}^{L} \mathbb{1}_{\{i=q_{[j]}(t)\}} = 1$$

is true for all j, proving equality (2). ∎

Lemma 4. *LCSF/LCQ is not an MB policy.*

Proof: To prove lemma 4 we present the following counter example. Consider a system with $L = 4$ and $K = 7$. At time slot n the system has the following configuration:

The queue state at time slot n is $\mathbf{x}(n) = (5, 5, 5, 4)$. Servers 1 to 6 are connected to queues 1, 2 and 3 and server 7 is connected to queues 1 and 4 only.

Under this configuration, we can show that the LCSF/LCQ algorithm will result in $\hat{\mathbf{x}}(n) = (0, 2, 3, 3, 4)$ (where the first element represents the dummy queue that by assumption holds no real packets) and $\kappa_n(LCSF/LCQ) = 18$. A policy π can be constructed that selects the feasible server allocation $\mathbf{q} = (1, 2, 3, 1, 2, 3, 4)$ which yields the state $\hat{\mathbf{x}}(n) = (0, 3, 3, 3, 3)$ and $\kappa_n(\pi) = 12 < \kappa_n(LCSF/LCQ)$. Hence, LCSF/LCQ does not belong to the class of MB policies. ∎

The LCSF/LCQ policy is of particular interest for the following reasons: (a) It follows a particular server allocation ordering (LCSF) to their longest connected queues (LCQ) and thus it is closely related to Algorithm 1, (b) the selected server ordering (LCSF) and allocation (LCQ) intuitively tries to maximize the opportunity to target and reduce the longest connected queue in the system thus minimizing the imbalance among queues, and (c) as we will see in Section VI, the LCSF/LCQ performance is statistically indistinguishable from that of an MB policy (implying that the counterexamples similar to the one in Lemma 4 proof have low probability of occurrence under LCSF/LCQ system operation).

VI. Performance Evaluation and Simulation Results

We used simulation to study the performance of the system under MB policies and to compare against the system performance under several other policies. The metric we used in this study is $EQ \triangleq E(\sum_{i=1}^{L} X_i)$, the average of the total number of packets in the system. This metric reflects the average queuing delay for the system under investigation and the corresponding policy.

The policies used in this simulation are: LCSF/LCQ, as an approximation of an MB policy; MCSF/SCQ, as an approximation of a least balancing (LB) policy, a policy that maximizes the imbalance index. An MB policy is implemented following Algorithm 1 and its performance was indistinguishable from that of the LCSF/LCQ. Therefore, in the simulation graphs the MB and LCSF/LCQ are represented by the same curves. For larger K, we expect small deterioration in the performance of LCSF/LCQ compared to MB policy.

Other policies that were simulated include the randomized, Most Connected Server First/Longest Connected Queue (MCSF/LCQ), and Least Connected Server First/Shortest Connected Queue (LCSF/SCQ) policies. The randomized policy is the one that at each time slot allocates each server, randomly and with equal probability, to one of its connected queues. The MCSF/LCQ policy differs from the LCSF/LCQ policies in the order that it allocates the servers. It uses the exact reverse order, starting the allocation with the most connected server and ending it with the least connected one. However, it resembles MB policies in that it allocates each server to its longest

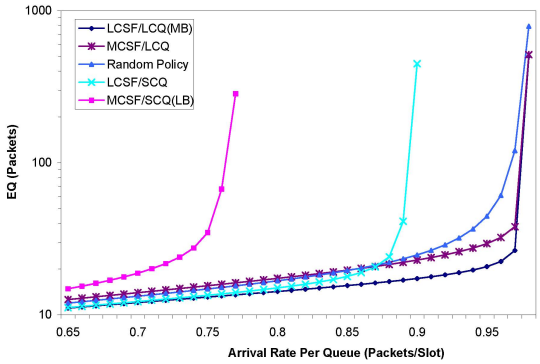

Fig. 2. EQ vs. load under different policies, $L = 16$, $K = 16$, $p = 0.2$.

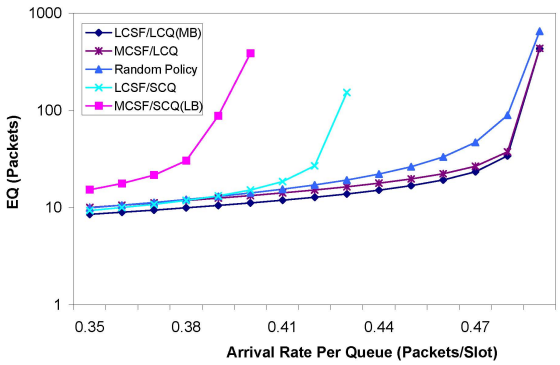

Fig. 3. EQ versus load, $L = 16$, $K = 8$ and $p = 0.2$.

connected queue. The LCSF/SCQ policy allocates each server, starting from the one with the least number of connected queues, to its shortest connected queue. The difference from an MB policy is obviously the allocation to the shortest connected queue. This policy will result in an unbalanced queue lengths and hence a performance that is closer to the LB policies. These policies are implemented using an algorithm similar to Algorithm 2 with slight modification to the order of servers and/or the selection of the queue to be served at every step.

Figure 2 shows the average total queue occupancy versus arrival rate under the five different policies. The system in this simulation is a symmetrical system with 16 parallel queues ($L = 16$), 16 identical servers ($K = 16$) and i.i.d. Bernoulli queue-to-server (channel) connectivity with parameter $p = P[G_{i,j}(t) = 1] = 0.2$.

The graphs show that LCSF/LCQ outperforms all other policies. It minimizes, EQ and hence the queuing delay. We also noticed that it maximizes the system stability region and hence the system throughput as well. As expected, the performance of the other three policies lies within the performance of the MB and LB policies.

The MCSF/LCQ and LCSF/SCQ policies are variations of the MB and LB policies. The performance of MCSF/LCQ policy is close to that of the MB policy. The difference in performance is due to the order of server allocation. On the other hand, the LCSF/SCQ policy shows a large performance improvement compared to LB policy. This improvement is a result of the reordering of allocations of servers.

The two figures also show that the randomized policy performs reasonably well. Moreover, its performance improves as the number of servers in the system decreases. The performance advantage of the LCSF/LCQ over the other policies increases as the number of servers in the system increases. The presence of more servers implies that the server allocation action space is larger. Selecting the optimal (i.e., MB) allocation, over any arbitrary policy, out of a large number of options will produce better performance as compared to the case when the number of server allocation options is reduced. We also noticed that the stability region of the system becomes narrower when less servers are used. This is true because fewer resources (servers) are available to be allocated by the working policy in this case.

VII. Conclusion

In this work, we presented an implementation algorithm for the most balancing packet scheduling policies in emerging wireless systems. The system under investigation was modeled using symmetric queues and multiple servers with random server connectivities. The proposed algorithm has reduced complexity ($O(L \times K)$) compared to full-search algorithm. The LCSF/LCQ policy was proposed as an efficient low-overhead approximation of MB policies. Simulation results verified that the LCSF/LCQ performs (in terms of the expected value) very closely to the MB policy. In addition, the results showed that it outperforms all other policies that we investigated. Finally, we observed that a randomized policy can perform very close to the optimal one in many cases, especially for $K \ll L$.

The described algorithms may also be applied to non-symmetrical systems. Because of space constraints, we left the investigation of such systems for future work.

References

[1] J. P. Castro, *All IP in 3G CDMA Networks*. USA: John wiley & Sons Inc., 2004.

[2] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, and A. Viterbi, "CDMA-HDR: A bandwidth-efficient high-speed wireless data service for nomadic users", *IEEE Communication Magazine*, pages 70-77, July 2000

[3] H. R. Shao, C. Shen, D. Gu, J. Z and P. Orlik, Dynamic Resource Control for High-Speed Downlink Packet Access Wireless Channel, *IEEE Transaction on Vehicular Technology*, vol. 44, no. 1, Feb. 1995.

[4] H. Al-Zubaidy, I. Lambadaris and I. Viniotis, "Optimal Resource Scheduling in Wireless Multi-service Systems With Random Channel Connectivity, IEEE Globecom09, Honolulu, HI, USA, Dec. 2009.

[5] H. Al-Zubaidy, I. Lambadaris, I. Viniotis, F.R. Yu, "Optimal Multi-Server Allocation to Parallel Queues With Random Connectivity and Retransmissions," ICC'2010, Cape town, South Africa, May 2010.

[6] D. Stoyan, *Comparison Methods for Queues and other Stochastic Models*, J. Wiley and Sons, Chichester, 1983.

[7] T. Lindvall, *Lectures on the coupling method,* New York: Wiley(1992).

[8] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory*, 39(2): 466 - 478, March 1993.

[9] A. Ganti, E. Modiano and J. N. Tsitsiklis, "Optimal transmission scheduling in symmetric communication models with intermittent connectivity," *IEEE Transactions on Information Theory*, 53(3): 998-1008, March 2007.

[10] S. Kittipiyakul, T. Javidi, "Delay-optimal server allocation in multiqueue multi-server systems with time-varying connectivities," *IEEE Transactions on Information Theory*, 55(5): 2319-2333, May 2009.

[11] H. Al-Zubaidy, *Optimal Dynamic Multi-Server Allocation to Parallel Queues With Independent Random Connectivity*, Technical report: SCE-09-02, SCE department, Carleton University, Feb. 2009.