Utility Optimal Scheduling in Processing Networks

Longbo Huang, Michael J. Neely*†‡

Abstract

We consider the problem of utility optimal scheduling in general *processing networks* with random arrivals and network conditions. These are generalizations of traditional data networks where commodities in one or more queues can be combined to produce new commodities that are delivered to other parts of the network, and can be used to model problems such as in-network data fusion, stream processing, MapReduce scheduling, and grid computing. Scheduling actions are complicated by the *underflow problem* that arises when some queues with required components go empty. In this paper, we develop a novel methodology for constructing and analyzing *online* algorithms for such processing networks. Specifically, we develop the Perturbed Max-Weight algorithm (PMW) to achieve optimal utility. The idea of PMW is to perturb the weights used by the usual Max-Weight algorithm to "push" queue levels towards non-zero values (avoiding underflows). We then show, using a novel combination of Lyapunov drift analysis and duality theory, that when the perturbations are carefully chosen, PMW is able to achieve a utility that is within $O(1/V)$ of the optimal value for any $V \geq 1$, while ensuring an average network backlog of $O(V)$. The methodology developed here is very general and can also be applied to other problems that involve such underflow constraints.

1 Introduction

Recently, there has been much attention on developing optimal scheduling algorithms for the class of *processing networks* e.g., [1], [2], [3], [4], [5]. These networks are generalizations of traditional data networks. Contents in these networks can represent information, data packets, or certain raw materials, that need to go through multiple processing stages in the network before they can be utilized. One example of such processing networks is the Fork and Join network considered in [4], which models, e.g., stream processing [6] [7], MapReduce scheduling [8], and grid computing [9]. In the stream processing case, the contents in the network represent different types of data, say voice and video, that need to be combined or jointly compressed, and the network topology represents a particular sequence of operations that needs to be conducted

*Longbo Huang and Michael J. Neely (emails: {longbohu, mjneely }@usc.edu) are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA.

†This material is supported in part by one or more of the following: the NSF Career grant CCF-0747525, NSF grant 0540420, the Network Science Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory W911NF-09-2-0053.

‡This paper appears in Proceedings of IFIP Performance 2011.

during processing. Another example of a processing network is a sensor network that performs data fusion [10], in which case sensor data must first be fused before it is delivered. Finally, these processing networks also contain the class of manufacturing networks, where raw materials are assembled into products [3], [5].

In this paper, we develop optimal scheduling algorithms for the following general utility maximization problem in processing networks. We are given a discrete time stochastic processing network. The network state, which describes the network randomness (such as random channel conditions or commodity arrivals), is time varying according to some probability law. A network controller performs some action at every time slot, based on the observed network state, and subject to the constraint that *the network queues must have enough contents to support the action*. The chosen action generates some utility, but also consumes some amount of contents from some queues, and possibly generates new contents for some other queues. These contents cause congestion, and thus lead to backlogs at queues in the network. The goal of the controller is to maximize its time average utility subject to the constraint that the time average total backlog in the network is finite.

Many of the utility maximization problems in data networks fall into this general framework. For instance, [11], [12], [13] [14], [15], can be viewed as special cases of the above framework which allow scheduling actions to be independent of the content level in the queues (see [16] for a survey of problems in data networks). By comparing the processing networks with the data networks, we note that the main difficulty in performing utility optimal scheduling in these processing networks is that *we need to build an optimal scheduling algorithm on top of a mechanism that prevents queue underflows*. Scheduling problems with such *no-underflow* constraints are usually formulated as dynamic programs, e.g., [17], which require substantial statistical knowledge of the network randomness, and are usually very difficult to solve.

In this paper, we develop a novel methodology for constructing and analyzing *online* algorithms for these processing networks. Specifically, we develop the Perturbed Max-Weight algorithm (PMW) for achieving optimal utility in processing networks. PMW is a greedy algorithm that makes decisions every time slot, *without requiring any statistical knowledge of the network randomness*. PMW is based on the Max-Weight algorithm developed in the data network context [18] [19]. There, Max-Weight has been shown to be able to achieve a time average utility that is within $O(1/V)$ of the optimal network utility for any $V \geq 1$, while ensuring that the average network delay is $O(V)$, when the network dynamics are i.i.d. [19]. The idea of PMW is to perturb the weights used in the Max-Weight algorithm so as to "push" the queue sizes towards some nonzero values. Doing so properly, we can ensure that the queues always have enough contents for the scheduling actions. Once this is accomplished, we then do scheduling as in the usual Max-Weight algorithm with the perturbed weights. In this way, we simultaneously avoid queue underflows and achieve good utility performance, and also eliminate the need to solve complex dynamic programs. We show, using a novel

combination of Lyapunov drift analysis and duality theory, that when the perturbations are carefully chosen, PMW achieves a utility that is within $O(1/V)$ of the optimal for any $V \geq 1$, while ensuring that the average network backlog is $O(V)$.

The PMW algorithm is quite different from the approaches used in the processing network literature. [1] analyzes manufacturing networks using Brownian approximations. [2] applies the Max-Weight algorithm to do scheduling in manufacturing networks, assuming all the queues always have enough contents. [3] develops the Deficit Max-Weight algorithm (DMW), by using Max-Weight based on an alternative control process for decision making. [4] formulates the problem as a convex optimization problem to match the input and output rates of the queues, without considering the queueing level dynamics. PMW instead provides a way to explicitly avoid queue underflows, and allows us to compute explicit backlog bounds. Our algorithm is perhaps most similar to the DMW algorithm in [3]. DMW achieves the desired performance by bounding the "deficit" incurred by the algorithm and applies to both stability and utility maximization problems; whereas PMW uses perturbations to avoid deficits entirely and allows for more general time varying system dynamics, e.g., random arrivals and random costs. Our previous work [5] introduces a similar perturbed Lyapunov function. However, [5] treats a 1-hop manufacturing model with a simplified structure, whereas the current work treats a model with general cost functions and possibly multi-hop dynamics. The choice of the perturbation value is not obvious in this general context, and this paper develops a method for choosing an appropriate perturbation. It also uses a combination of Lyapunov drift analysis and duality theory that is different from the approach in [5].

The main contributions of this paper are summarized as follows:

- We present a general processing network model that can be used to model many problems involving the *no-underflow* constraint, including manufacturing networks, stream processing, and energy harvesting networks etc.
- We develop a general methodology for constructing *online* algorithms for such processing networks based on weight perturbation in Max-Weight. Using the methodology, we develop the Perturbed Max-Weight algorithm (PMW), and prove that PMW resolves the no-underflow constraint and achieves an $[O(1/V), O(V)]$ utility-congestion tradeoff.
- We present an explicit construction of approximate PMW algorithms for networks with output reward.
- We present an explanation on why perturbation is needed for processing network problems, which contributes to a better understanding of the interaction between queue engineering and mathematical programming.

The paper is organized as follows: In Section 2 we set up our notations. In Section 3, we present a study on a data processing example to demonstrate the main idea of the paper. In Section 4 we state the general

model and the scheduling problem. In Section 5 we characterize optimality, and in Sections 6 we develop the PMW algorithm and show its utility can approach the optimum. Section 7 constructs a PMW algorithm for a broad class of networks. Simulation results are presented in Section 8. We discuss the role of perturbation in algorithm design for processing networks in Section 9.

2 Notations

Here we first set up the notations used in this paper: $\mathbb{R}$ represents the set of real numbers. $\mathbb{R}_+$ (or $\mathbb{R}_-$) denotes the set of nonnegative (or non-positive) real numbers. $\mathbb{R}^n$ (or $\mathbb{R}^n_+$) is the set of n dimensional *column* vectors, with each element being in $\mathbb{R}$ (or $\mathbb{R}_+$). **Bold** symbols $\boldsymbol{a}$ and $\boldsymbol{a}^T$ represent a *column* vector and its transpose. $\boldsymbol{a} \succeq \boldsymbol{b}$ means vector $\boldsymbol{a}$ is entrywise no less than vector $\boldsymbol{b}$. $||\boldsymbol{a} - \boldsymbol{b}||$ is the Euclidean distance of $\boldsymbol{a}$ and $\boldsymbol{b}$. $\mathbf{0}$ and $\mathbf{1}$ denote column vectors with all elements being 0 and 1. For any two vectors $\boldsymbol{a} = (a_1, ..., a_n)^T$ and $\boldsymbol{b} = (b_1, ..., b_n)^T$, the vector $\boldsymbol{a} \otimes \boldsymbol{b} = (a_1 b_1, ..., a_n b_n)^T$. Finally $[a]^+ = \max[a, 0]$.

3 A data processing example

In this section, we study a data processing example and develop the Perturbed Max-Weight algorithm (PMW) in this case. This example demonstrates the main idea of this paper. We will later present the general model in Section 4.

3.1 Network Settings

We consider a network shown in Fig. 1, where the network performs a 2-stage data processing for the arriving data. In this network, there are two random data streams $R_1(t), R_2(t)$, which represent, e.g., sensed data

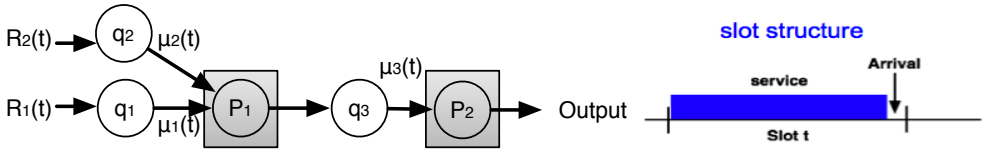

Figure 1: LEFT: An example network consisting of three queues q_1, q_2, q_3 and two processors P_1, P_2. RIGHT: The slot structure.

that arrives, or video and voice data that need to be mixed. We assume that $R_i(t) = 1$ or 0, equally likely, for $i = 1, 2$. At every time slot, the network controller first decides whether or not to admit the new arrivals. The controller then has to decide how to activate the two processors P_1, P_2 for data processing. We assume that both processors can be activated simultaneously. When activated, P_1 consumes one unit of data from both q_1 and q_2, and generates one unit of fused data into q_3. This data needs further processing that is done by P_2. When P_2 is activated, it consumes one unit of data from q_3, and generates one unit of processed data.

We assume that each unit of successfully fused and processed data generates a profit of $p(t)$, where $p(t)$ is i.i.d. and takes value 4 or 1 with equal probabilities. However, every activation of the processor P_i incurs a unit cost. The network controller's objective is to maximize the average utility, i.e., profit minus cost, subject to queue stability.

For the ease of presenting the general model later, we define a *network state* $S(t) = (R_1(t), R_2(t), p(t))$, which describes the current network randomness. [1] We also denote the controller's *action* at time t to be $x(t) = (D_1(t), D_2(t), I_1(t), I_2(t))$, where $D_j(t) = 1$ ($D_j(t) = 0$) means to admit (reject) the new arrivals into queue j, and $I_i(t) = 1$ ($I_i(t) = 0$) means processor P_i is activated (turned off). We note the following *no-underflow constraints* must be met for all time when we activate processors P_1, P_2:

$$I_1(t) \leq q_1(t), I_1(t) \leq q_2(t), I_2(t) \leq q_3(t). \tag{1}$$

That is, $I_1(t) = 1$ only when q_1 and q_2 are both nonempty, and $I_2(t) = 1$ only if q_3 is nonempty. Note that [3] is the first to identify such no-underflow constraints and propose an explicit solution that shows the fraction of time these constraints are violated converges to zero under certain network assumptions. Here we propose a different approach that ensures the constraints are never violated, and holds for a broader class of problems. Subject to (1), we can write the amount of arrivals into q_1, q_2, q_3, and the service rates of the queues at time t as functions of the network state $S(t)$ and the action $x(t)$, i.e.,

$$A_j(t) = A_j(S(t), x(t)) = D_j(t)R_j(t),\ j = 1, 2, \quad A_3(t) = A_3(S(t), x(t)) = I_1(t). \tag{2}$$

$$\mu_j(t) = \mu_j(S(t), x(t)) = I_1(t),\ j = 1, 2, \quad \mu_3(t) = \mu_3(S(t), x(t)) = I_2(t). \tag{3}$$

Then we see that the queues evolve according to the following:

$$q_j(t+1) \quad = \quad q_j(t) - \mu_j(t) + A_j(t),\ j = 1, 2, 3,\ \forall\, t. \tag{4}$$

We assume that in every time slot, each queue first supplies the contents to the associated processors. It then receives new contents at the end of the slot. This slot structure is shown in Fig. 1. The *instantaneous utility* is given by:

$$f(t) = f(S(t), x(t)) = p(t)I_2(t) - I_1(t) - I_2(t). \tag{5}$$

The goal is to maximize the time average value of $f(t)$ subject to network stability.

Note that the constraint (1) greatly complicates the design of an optimal scheduling algorithm. This is because the decision made at time t may affect the queue states in future time slots, which can in turn affect the set of possible actions in the future. In the following, we will develop the Perturbed Max-Weight

[1] The network state here contains just $R_1(t)$, $R_2(t)$ and $p(t)$. More complicated settings, where the amount consumed from queues may also depend on the random link conditions between queues and processors can also be modeled by incorporating the link components into the network state, e.g., [20].

algorithm (PMW) for this example. The idea of PMW is to use the usual Max-Weight algorithm, but perturb the weights so as to push the queue sizes towards certain nonzero values. By carefully designing the perturbation, we can simultaneously ensure that the queues always have enough data for processing and the achieved utility is close to optimal.

3.2 The Perturbed Max-Weight algorithm (PMW)

We now present the construction of the PMW algorithm for this simple example (this is extended to general network models in Section 6). To start, we first define a *perturbation vector* $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^T$ and the Lyapunov function $L(t) = \frac{1}{2}\sum_{j=1}^{3}[q_j(t) - \theta_j]^2$. We then define the one-slot conditional drift as:

$$\Delta(t) = \mathbb{E}\big\{L(t+1) - L(t) \mid \boldsymbol{q}(t)\big\}, \tag{6}$$

where the expectation is taken over the random network state $S(t)$ and the randomness over the actions. Using the queueing dynamics (4), it is easy to obtain that:

$$\Delta(t) \leq B - \sum_{j=1}^{3}\mathbb{E}\big\{(q_j(t) - \theta_j)[\mu_j(t) - A_j(t)] \mid \boldsymbol{q}(t)\big\},$$

where $B = \frac{3}{2}$. Now we use the "drift-plus-penalty" approach in [19] to design our algorithm for this problem. To do so, we define a control parameter $V \geq 1$, which will affect our utility-backlog tradeoff, and add to both sides the term $-V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\}$ to get:

$$\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} - \sum_{j=1}^{3}\mathbb{E}\big\{(q_j(t) - \theta_j)[\mu_j(t) - A_j(t)] \mid \boldsymbol{q}(t)\big\}. \tag{7}$$

Denote $\Delta_V(t) = \Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\}$, and plug (2), (3) and (5) into the above, to get:

$$\begin{aligned}\Delta_V(t) \leq B &+ \mathbb{E}\big\{D_1(t)R_1(t)[q_1(t) - \theta_1] \mid \boldsymbol{q}(t)\big\} + \mathbb{E}\big\{D_2(t)R_2(t)[q_2(t) - \theta_2] \mid \boldsymbol{q}(t)\big\} \\ &- \mathbb{E}\big\{I_2(t)[q_3(t) - \theta_3 + (p(t)-1)V] \mid \boldsymbol{q}(t)\big\} - \mathbb{E}\big\{I_1(t)[q_1(t) - \theta_1 + q_2(t) - \theta_2 - (q_3(t) - \theta_3) - V] \mid \boldsymbol{q}(t)\big\}.\end{aligned} \tag{8}$$

We now develop our PMW algorithm by choosing an action at every time slot to *minimize the right-hand side (RHS) of (8)* subject to (1). The algorithm then works as follows:

PMW: At every time slot, observe $S(t)$ and $\boldsymbol{q}(t)$, and do the following:

1. Data Admission: For each $j = 1, 2$, choose $D_j(t) = 1$, i.e., admit the new arrivals to q_j if:

$$q_j(t) - \theta_j < 0, \tag{9}$$

   else set $D_j(t) = 0$ and reject the arrivals.

2. Processor Activation: Choose $I_1(t) = 1$, i.e., activate processor P_1, if $q_1(t) \geq 1$, $q_2(t) \geq 1$, and that:

$$q_1(t) - \theta_1 + q_2(t) - \theta_2 - (q_3(t) - \theta_3) - V > 0, \tag{10}$$

else choose $I_1(t) = 0$. Similarly, choose $I_2(t) = 1$, i.e., activate processor P_2, if $q_3(t) \geq 1$, and that:

$$q_3(t) - \theta_3 + (p(t) - 1)V > 0, \tag{11}$$

else choose $I_2(t) = 0$.

3. Queueing update: Update $q_j(t)$, $\forall\, j$, according to (4).

One important thing to notice here is that if we use the usual Max-Weight algorithm, i.e., do not use perturbation and set $\theta_j = 0, \forall\, j$, then (9) implies that we admit a new arrival to q_j only when $q_j < 0$, which is impossible because $q_j(t) \geq 0$ for all t. Hence, this example clearly demonstrates the fact that *the usual Max-Weight may not be applicable to processing network problems.* Below, we show that the use of perturbation effectively resolves this problem.

3.3 Performance of PMW

Here we analyze the performance of PMW. We will first prove the following important claim: *under a proper $\boldsymbol{\theta}$ vector, PMW minimizes the RHS of (8) over all possible policies of arrival admission and processor activation, including those that choose actions regardless of the constraint (1).* We then use this claim to prove the performance of PMW, by comparing the value of the RHS of (8) under PMW versus that under an alternate policy.

To prove the claim, we first see that the policy that minimizes the RHS of (8) without the constraint (1) differs from PMW only in the processor activation part, where PMW also considers the constraints $q_1(t) \geq 1$, $q_2(t) \geq 1$ and $q_3(t) \geq 1$. Thus if one can show that these constraints are indeed redundant in the PMW algorithm under a proper $\boldsymbol{\theta}$ vector, i.e., one can activate the processors without considering them but still ensure them, then PMW minimizes the RHS of (8) over all possible policies. In the following, we will use the following θ_j values:

$$\theta_1 = 2V + 1,\ \theta_2 = 2V + 1,\ \theta_3 = 3V + 1. \tag{12}$$

We will also assume that $q_j(0) = 1$ for all $j = 1, 2, 3$. Note that this can easily be satisfied by storing an initial backlog in the queues.

We now look at the queue sizes $q_j(t), j = 1, 2, 3$. From (11), P_2 is activated, i.e., $I_2(t) = 1$ if and only if:

$$q_3(t) \geq \theta_3 - (p(t) - 1)V + 1, \quad \text{and} \quad q_3(t) \geq 1. \tag{13}$$

Since $p(t) - 1 = 3$ or 0, we see that $I_2(t) = 1$ whenever $q_3(t) \geq \theta_3 + 1$, but $I_2(t) = 0$ unless $q_3(t) \geq \theta_3 - 3V + 1$. Since q_3 can receive and deliver at most one unit of data at a time, we get:

$$\theta_3 + 1 \geq q_3(t) \geq \theta_3 - 3V,\ \forall\ t. \tag{14}$$

Using $\theta_3 = 3V + 1$, this implies:

$$3V + 2 \geq q_3(t) \geq 1, \quad \forall\, t. \tag{15}$$

This shows that with $\theta_3 = 3V + 1$, the activations of P_2 are always feasible even if the constraint $q_3(t) \geq 1$ is removed. We now look at $q_1(t)$ and $q_2(t)$. We see from (9) that for $\theta_1, \theta_2 \geq 0$, we have:

$$q_j(t) \leq \theta_j, \ j = 1, 2. \tag{16}$$

Also, using (10) and (14), it is easy to see that when $I_1(t) = 1$, i.e., when P_1 is turned on, we have:

$$q_1(t) - \theta_1 + q_2(t) - \theta_2 > q_3(t) - \theta_3 + V \geq -2V. \tag{17}$$

Combining (17) with (16), we see that if $I_1(t) = 1$, we have:

$$q_j(t) \geq 2, \ j = 1, 2. \tag{18}$$

This is so because, e.g., if $q_1(t) \leq 1$, then $q_1(t) - \theta_1 \leq 1 - \theta_1 = -2V$. Since $q_2(t) - \theta_2 \leq 0$ by (16), we have:

$$q_1(t) - \theta_1 + q_2(t) - \theta_2 \leq -2V - 0 = -2V,$$

which cannot be greater than $-2V$ in (17). Thus, by (15), (18), and the fact that $q_j(0) \geq 1$, $\forall\, j$, we have:

$$q_j(t) \geq 1, \quad j = 1, 2, 3, \forall\, t. \tag{19}$$

This shows that by using the θ_j values in (12), PMW automatically ensures that no queue underflow happens, and hence PMW minimizes the RHS of (8) over all possible policies.

Given the above observation, the utility performance of PMW can now be analyzed using a similar argument as in [5]. Specifically, we can first prove that there exists a stationary and randomized policy which chooses scheduling actions purely as a function of $S(t)$, and achieves $\mathbb{E}\big\{\mu_j(t) - A_j(t) \mid \boldsymbol{q}(t)\big\} = 0$ for all j and $\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} = f^*_{av} = \frac{1}{2}$, where f^*_{av} is the optimal average utility. Then we can compare the drift under PMW with that under this optimal policy. Note that *this analysis approach would not have been possible here without using the perturbation to ensure (19).* Now plugging this policy into (7), we obtain:

$$\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B - V f^*_{av}. \tag{20}$$

Taking expectations over $\boldsymbol{q}(t)$ on both sides and summing it over $t = 0, 1, ..., T-1$, we get:

$$\mathbb{E}\big\{L(T) - L(0)\big\} - V\sum_{t=0}^{T-1}\mathbb{E}\big\{f(t)\big\} \leq TB - VTf^*_{av}. \tag{21}$$

Now rearranging the terms, dividing both sides by VT, and using the fact that $L(t) \geq 0$, we get:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big\{f(t)\big\} \geq f^*_{av} - \frac{B}{V} - \frac{\mathbb{E}\big\{L(0)\big\}}{TV}. \tag{22}$$

Taking a liminf as $T \to \infty$, and using $\mathbb{E}\{L(0)\} < \infty$,

$$f_{av}^{PMW} = \liminf_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{f(t)\} \geq f_{av}^{*} - \frac{B}{V}, \tag{23}$$

where f_{av}^{PMW} denotes the time average utility achieved by PMW. This thus shows that PMW is able to achieve a time average utility that is within $O(1/V)$ of the optimal value, and guarantees $q_j(t) \leq O(V)$ for all time (recall equations (12), (15) and (16)). Note that PMW is similar to the DMW algorithm developed in [3]. However, DMW allows the queues to be empty when activating processors, which may lead to "deficit," whereas PMW effectively avoids this by using a perturbation vector.

In the following, we will present the general processing network utility optimization model, and analyze the performance of the general PMW algorithm under this general model. Our analysis uses a novel combination of Lyapunov drift analysis and duality theory, and will be different from that used above.

4 General System Model

In this section, we present the general network model. We consider a network controller that operates a general network with the goal of maximizing the time average utility, subject to the network stability. The network is assumed to operate in slotted time, i.e., $t \in \{0, 1, 2, ...\}$, and contains $r \geq 1$ queues.

4.1 Network State

In every slot t, we use $S(t)$ to denote the current network state, which indicates the current network parameters, such as a vector of channel conditions for each link, or a collection of other relevant information about the current network links and arrivals. We assume that $S(t)$ is i.i.d. every time slot, with a total of M different random network states denoted by $\mathcal{S} = \{s_1, s_2, \ldots, s_M\}$. [2] We let $\pi_{s_i} = Pr\{S(t) = s_i\}$. The network controller can observe $S(t)$ at the beginning of every slot t, but the π_{s_i} probabilities are not necessarily known.

4.2 The Utility, Traffic, and Service

At each time t, after observing $S(t) = s_i$ and the network backlog vector, the controller performs an action $x(t)$. This action represents the aggregate decisions made by the controller at t, which can include, e.g., in the previous example, the set of processors to turn on, or the amount of arriving contents to accept, etc.

We denote $\mathcal{X}^{(s_i)}$ as the set of all possible actions for network state s_i, *assuming all the queues contain enough contents to meet the scheduling requirements.* Note that we always have $x(t) = x^{(s_i)}$ for some

[2] Note that all our results can easily be extended to the case when $S(t)$ evolves according to a finite state aperiodic and irreducible Markov chain, by using the results developed in [21].

$x^{(s_i)} \in \mathcal{X}^{(s_i)}$ whenever $S(t) = s_i$. The set $\mathcal{X}^{(s_i)}$ is assumed to be time-invariant and compact for all $s_i \in \mathcal{S}$. If the chosen action $x(t) = x^{(s_i)}$ at time t can be performed, i.e., it is possible and all the queues have enough contents, then the utility, traffic, and service generated by $x(t)$ are as follows: [3]

(a) The chosen action has an associated utility given by the utility function $f(t) = f(s_i, x^{(s_i)}) : \mathcal{X}^{(s_i)} \mapsto \mathbb{R}$;

(b) The amount of contents generated by the action to queue j is determined by the traffic function $A_j(t) = A_j(s_i, x^{(s_i)}) : \mathcal{X}^{(s_i)} \mapsto \mathbb{R}_+$, in units of contents;

(c) The amount of contents consumed from queue j by the action is given by the rate function $\mu_j(t) = \mu_j(s_i, x^{(s_i)}) : \mathcal{X}^{(s_i)} \mapsto \mathbb{R}_+$, in units of contents.

Note that $A_j(t)$ includes both the exogenous arrivals from outside the network to queue j, and the endogenous arrivals from other queues, i.e., the newly generated contents by processing contents in some other queues, to queue j. We assume the functions $f(s_i, \cdot)$, $\mu_j(s_i, \cdot)$ and $A_j(s_i, \cdot)$ are continuous, time-invariant, their magnitudes are uniformly upper bounded by some constant $\delta_{max} \in (0, \infty)$ for all s_i, j, and they are known to the network operator. Note that these assumptions are not restrictive. In practice, e.g., in a manufacturing network, or in a stream processing system, the amount of contents consumed from or generated for a queue under a particular action, is typically known to the operator and is finite. Thus, the network operator can easily measure the traffic functions $A_j(s_i, \cdot)$ and the service functions $\mu_j(s_i, \cdot)$. In many cases, e.g., the one in Section 7, the operator also knows its utility functions $f(s_i, \cdot)$, or can easily estimate them via measurements over time.

In any actual algorithm implementation, however, we see that not all actions in the set $\mathcal{X}^{(s_i)}$ can be performed when $S(t) = s_i$, due to the fact that some queues may not have enough contents for the action. We say that an action $x^{(s_i)} \in \mathcal{X}^{(s_i)}$ is feasible at time t with $S(t) = s_i$ only when the following general *no-underflow constraint* is satisfied:

$$q_j(t) \geq \mu_j(s_i, x^{(s_i)}), \ \forall \ j. \tag{24}$$

That is, *all the queues must have contents greater than or equal to what will be consumed*. Note that due to this no-underflow constraint, some actions in $\mathcal{X}^{(s_i)}$ may not be possible to execute at time t. Thus, we assume that each set $\mathcal{X}^{(s_i)}$ contains an *idle* action, which always ensures (24) and makes no contribution to the utility. [4] We also assume that there exists a set of actions $\{\hat{x}_k^{(s_i)}\}_{i=1,\ldots,M}^{k=1,2,\ldots,r+2}$ with $\hat{x}_k^{(s_i)} \in \mathcal{X}^{(s_i)}$ and some variables $\hat{a}_k^{(s_i)} \geq 0$ for all s_i and k with $\sum_{k=1}^{r+2} \hat{a}_k^{(s_i)} = 1$ for all s_i, such that:

$$\sum_{s_i} \pi_{s_i} \Big\{ \sum_{k=1}^{r+2} \hat{a}_k^{(s_i)} [A_j(s_i, \hat{x}_k^{(s_i)}) - \mu_j(s_i, \hat{x}_k^{(s_i)})] \Big\} \leq -\eta, \tag{25}$$

[3] Here we implicitly assume that each action takes only one unit time. We will further discuss this issue in Section 4.4.

[4] This assumption is typically satisfied in practical systems. It is made to ensure that the system does not go into a deadlock mode where nothing is possible. For instance, in the example in Section 3, an idle action is to not admit any data and not activate any processor.

for some $\eta > 0$ for all j. That is, the "stability constraints" are feasible with η-slack.

4.3 Queueing, Average Cost, and the Objective

Let $\boldsymbol{q}(t) = (q_1(t), ..., q_r(t))^T \in \mathbb{R}_+^r$, $t = 0, 1, 2, ...$ be the queue backlog vector process of the network, in units of contents. Due to the feasibility condition (24) of the actions, we see that the queues evolve according to the following dynamics:

$$q_j(t+1) = q_j(t) - \mu_j(t) + A_j(t), \quad \forall j, t \geq 0, \tag{26}$$

with some $||\boldsymbol{q}(0)|| < \infty$. Note that using a nonzero $q_j(0)$ can be viewed as placing an "initial stock" in the queues to facilitate algorithm implementation. We assume a slot structure as shown in Fig. 1. In this paper, we adopt the following notion of queue stability:

$$\overline{q} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{j=1}^{r} \mathbb{E}\{q_j(\tau)\} < \infty. \tag{27}$$

We also use f_{av}^{Π} to denote the time average utility induced by an action-choosing policy Π, defined as:

$$f_{av}^{\Pi} \triangleq \liminf_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{f^{\Pi}(\tau)\}, \tag{28}$$

where $f^{\Pi}(\tau)$ is the utility incurred at time τ by policy Π. We call an action-choosing policy *feasible* if at every time slot t it only chooses actions from the possible action set $\mathcal{X}^{(S(t))}$ that satisfy (24). We then call a feasible action-choosing policy under which (27) holds a *stable* policy, and use f_{av}^{*} to denote the optimal time average utility over all stable policies.

In every slot, the network controller observes the current network state and the queue backlog vector, and chooses a feasible control action that ensures (24), with the objective of maximizing the time average utility subject to network stability. Note that if (24) can be ignored, and if any processor only requires contents from a single queue, then this problem falls into the general stochastic network optimization framework considered in [19], in which case it can be solved by using the usual Max-Weight algorithm to achieve a utility that is within $O(1/V)$ of the optimal while ensuring that the average network backlog is $O(V)$.

4.4 Discussion of the Model

We note that the model is very general and can be used to model many problems that involve such no-underflow constraints. For instance, manufacturing networks where parts are assembled into products, or energy harvesting networks that are powered by finite capacity energy storage devices.

Our model assumes a slot structure of the network. This implicitly implies that all the network actions must consume only one unit time. However, in many applications, e.g., stream processing, different processors may take different amounts of time to process different data. Although this problem can easily be incorporated

into our network by using a large enough slot length, such an easy fix may lead to utility loss. Quantifying the utility loss due to slot length selection and designing algorithms that allow for arbitrary heterogeneous action times are interesting problems of our future research.

5 Upper bounding the optimal utility

In this section, we first obtain an upper bound of the optimal utility that the network controller can achieve. This upper bound will later be used to analyze the performance of our algorithm. The result is summarized in the following theorem.

Theorem 1. *Suppose that the initial queue backlog $\boldsymbol{q}(t)$ satisfies $\mathbb{E}\{q_j(0)\} < \infty$ for all $j = 1, ..., r$. Then,*

$$Vf_{av}^* \leq \phi^*, \tag{29}$$

where ϕ^* is the optimal value of the following deterministic optimization *problem:*

$$\max: \quad \phi = \sum_{s_i} \pi_{s_i} V \sum_{k=1}^{r+2} a_k^{(s_i)} f(s_i, x_k^{(s_i)}) \tag{30}$$

$$s.t. \quad \sum_{s_i} \pi_{s_i} \sum_{k=1}^{r+2} a_k^{(s_i)} A_j(s_i, x_k^{(s_i)}) = \sum_{s_i} \pi_{s_i} \sum_{k=1}^{r+2} a_k^{(s_i)} \mu_j(s_i, x_k^{(s_i)}), \tag{31}$$

$$x_k^{(s_i)} \in \mathcal{X}^{(s_i)}, \forall\, s_i, k, \tag{32}$$

$$a_k^{(s_i)} \geq 0, \forall\, s_i, k, \sum_k a_k^{(s_i)} = 1, \forall\, s_i. \tag{33}$$

Proof. See Appendix A. □

Note that the problem (30) only requires that the time average input rate into a queue is equal to its time average output rate. This requirement ignores the action feasibility constraint (24), and makes (30) easier to solve than the actual scheduling problem. We now look at the dual problem of the problem (30). The following lemma shows that the dual problem of (30) does not have to include the variables $\{a_k^{(s_i)}\}_{i=1,...,M}^{k=1,...,r+2}$. This lemma will also be useful for our later analysis.

Lemma 1. *The dual problem of (30) is given by:*

$$\min: \quad g(\boldsymbol{\gamma}), \;\; s.t. \quad \boldsymbol{\gamma} \in \mathbb{R}^r, \tag{34}$$

where the function $g(\boldsymbol{\gamma})$ is defined:

$$g(\boldsymbol{\gamma}) = \sup_{x^{(s_i)} \in \mathcal{X}^{(s_i)}} \sum_{s_i} \pi_{s_i} \Big\{ Vf(s_i, x^{(s_i)}) - \sum_j \gamma_j \big[A_j(s_i, x^{(s_i)}) - \mu_j(s_i, x^{(s_i)}) \big] \Big\}. \tag{35}$$

Moreover, let $\boldsymbol{\gamma}^*$ be any optimal solution of (34), we have: $g(\boldsymbol{\gamma}^*) \geq \phi^*$.

Proof. See Appendix B. □

In the following, it is useful to define the following function:

$$g_{s_i}(\boldsymbol{\gamma}) = \sup_{x^{(s_i)} \in \mathcal{X}^{(s_i)}} \Big\{ Vf(s_i, x^{(s_i)}) - \sum_j \gamma_j \big[A_j(s_i, x^{(s_i)}) - \mu_j(s_i, x^{(s_i)}) \big] \Big\}. \tag{36}$$

That is, $g_{s_i}(\boldsymbol{\gamma})$ is the dual function of (30) when there is a single network state s_i. We can see from (35) and (36) that:

$$g(\boldsymbol{\gamma}) = \sum_{s_i} \pi_{s_i} g_{s_i}(\boldsymbol{\gamma}). \tag{37}$$

In the following, we will use $\boldsymbol{\gamma}^* = (\gamma_1^*, ..., \gamma_r^*)^T$ to denote an optimal solution of the problem (34).

6 The general perturbed max-weight algorithm and its performance

In this section, we develop the general Perturbed Max-Weight algorithm (PMW) to solve our scheduling problem. To start, we first choose a *perturbation vector* $\boldsymbol{\theta} = (\theta_1, ..., \theta_r)^T$. Then we define the following *weighted perturbed* Lyapunov function with some positive constants $\{w_j\}_{j=1}^r$:

$$L(t) = \frac{1}{2} \sum_{j=1}^r w_j \big(q_j(t) - \theta_j \big)^2. \tag{38}$$

We then define the one-slot conditional drift as in (7), i.e., $\Delta(t) = \mathbb{E}\big\{ L(t+1) - L(t) \mid \boldsymbol{q}(t) \big\}$. We will similarly use the "drift-plus-penalty" approach in Section 3 to construct the algorithm. Specifically, we first use the queueing dynamic equation (26), and have the following lemma:

Lemma 2. *Under any feasible control policy that can be implemented at time t, we have:*

$$\Delta(t) - V\mathbb{E}\big\{ f(t) \mid \boldsymbol{q}(t) \big\} \leq B - V\mathbb{E}\big\{ f(t) \mid \boldsymbol{q}(t) \big\} - \sum_{j=1}^r w_j \big(q_j(t) - \theta_j \big) \mathbb{E}\big\{ [\mu_j(t) - A_j(t)] \mid \boldsymbol{q}(t) \big\}, \tag{39}$$

where $B = \delta_{max}^2 \sum_{j=1}^r w_j$.

Proof. See Appendix C. □

The general Perturbed Max-Weight algorithm (PMW) is then obtained by choosing an action $x(t) \in \mathcal{X}^{(S(t))}$ at time t to *minimize the right-hand side (RHS) of (39)* subject to (24). Specifically, define the function $D^{(s_i)}_{\boldsymbol{\theta}, \boldsymbol{q}(t)}(x)$ as:

$$D^{(s_i)}_{\boldsymbol{\theta}, \boldsymbol{q}(t)}(x) \triangleq Vf(s_i, x) + \sum_{j=1}^r w_j \big(q_j(t) - \theta_j \big) \big[\mu_j(s_i, x) - A_j(s_i, x) \big]. \tag{40}$$

We see that the function $D^{(s_i)}_{\boldsymbol{\theta}, \boldsymbol{q}(t)}(x)$ is indeed the term inside the conditional expectation on the RHS of (39)

when $S(t) = s_i$. We now also define $D^{(s_i)*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}$ to be the optimal value of the following problem:

$$\max: \ D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x), \quad \text{s.t.,} \ \ x^{(s_i)} \in \mathcal{X}^{(s_i)}. \tag{41}$$

Hence $D^{(s_i)*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}$ is the maximum value of $D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}$ over all possible policies, including those that may not consider the no-underflow constraint (24). The general Perturbed Max-Weight algorithm (PMW) then works as follows:

PMW: Initialize the perturbation vector $\boldsymbol{\theta}$. At every time slot t, observe the current network state $S(t)$ and the backlog $\boldsymbol{q}(t)$. If $S(t) = s_i$, choose $x^{(s_i)} \in \mathcal{X}^{(s_i)}$ subject to (24) that minimizes $D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$.

Note that depending on the problem structure, the PMW algorithm can usually be implemented easily, e.g., [5], [12]. Now we analyze the performance of the PMW algorithm. We will prove our result under the following condition:

Condition 1. *There exists some finite constant $C \geq 0$, such that at every time slot t with a network state $S(t)$, the value of $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ under PMW is at least $D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)} - C$.*

The immediate consequence of Condition 1 is that PMW also minimizes the RHS of (39), i.e., the conditional expectation, to within C of its minimum value over all possible policies. If $C = 0$, then PMW simultaneously ensures (24) and minimizes the RHS of (39), e.g., as in the example in Section 3. However, we note that Condition 1 does not require the value of $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ to be exactly the same as $D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}$. This allows for more flexibility in constructing the PMW algorithm (See Section 7 for an example). We also note that Condition 1 can be ensured, e.g., by carefully choosing the θ_j values to ensure $q_j(t) \geq \delta_{max}$ for all time [5]. We will show that, under Condition 1, PMW achieves a time average utility that is within $O(1/V)$ of f^*_{av}, while guaranteeing that the time average network queue size is $O(V) + \sum_j w_j\theta_j$, which is $O(V)$ if $\boldsymbol{\theta} = \Theta(V)$ and $w_j = O(1)$, $\forall\, j$. The following theorem summarizes PMW's performance results.

Theorem 2. *Suppose that (25) holds, that Condition 1 holds, and that $\mathbb{E}\{q_j(0)\} < \infty$ for all $j = 1, ..., r$. Then under PMW, we have:* [5]

$$f^{PMW}_{av} \ \geq \ f^*_{av} - \frac{B+C}{V}, \tag{42}$$

$$\overline{q}^{PMW} \ \leq \ \frac{B + C + 2V\delta_{max}}{\eta} + \sum_{j=1}^{r} w_j\theta_j. \tag{43}$$

Here $B = \delta^2_{max}\sum_{j=1}^{r} w_j$, η is the slackness parameter in Section 4.2, f^{PMW}_{av} is defined in (28) to be the time average expected utility of PMW, and $\overline{q}^{PMW}$ is the time average expected weighted network backlog under PMW, defined:

$$\overline{q}^{PMW} \triangleq \limsup_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{j=1}^{r} w_j\mathbb{E}\{q_j(\tau)\}.$$

[5] Easy to see that (43) ensures (27), hence the network is stable under PMW.

Proof. See Appendix D. □

Theorem 2 shows that if Condition 1 holds, then PMW can be used as in previous networking problems, e.g., [12], [13], to obtain explicit utility-backlog tradeoffs. Note that in Theorem 2, $B = \Theta(1)$, i.e., independent of V. Moreover, in many cases, we also have $C = \Theta(1)$ and $\sum_j w_j\theta_j = O(V)$. In this case, Theorem 2 states that PMW indeed achieves an $[O(V), O(1/V)]$ utility-backlog tradeoff for processing network problems. A condition similar to Condition 1 was assumed in [2]. However, [2] only considers the usual Max-Weight algorithm, under which case (24) may not be satisfied for all time. PMW instead resolves this problem by carefully choosing the perturbation vector. One such example of PMW is the recent work [5], which applies PMW to an assembly line scheduling problem and achieves an $[O(1/V), O(V)]$ utility-backlog tradeoff.

We emphasize that though the results in Theorem 2 are similar to those in the data network problems [19], the proof techniques are very different. Our analysis uses a novel combination of Lyapunov drift analysis and duality theory, and allows one to obtain the performance result without proving the existence of an optimal stationary and randomized policy.

7 Constructing PMW for networks with output reward

In this section, we look at a specific yet general processing network model, and explicitly construct a PMW algorithm, including finding the proper $\boldsymbol{\theta}$ vector and choosing actions at each time slot.

7.1 Network Model

We assume that the network is modeled by an acyclic directed graph $\mathcal{G} = (\mathcal{Q}, \mathcal{P}, \mathcal{L})$. Here $\mathcal{Q} = \mathcal{Q}^s \cup \mathcal{Q}^{in}$ is the set of queues, consisting of the set of *source* queues $\mathcal{Q}^s$ where arrivals enter the network, and the set of *internal* queues $\mathcal{Q}^{in}$ where contents are stored for further processing. $\mathcal{P} = \mathcal{P}^{in} \cup \mathcal{P}^o$ is the set of processors, consisting of a set of *internal* processors $\mathcal{P}^{in}$, which generate partially processed contents for further processing at other processors, and *output* processors $\mathcal{P}^o$, which generate fully processed contents and deliver them to the output. $\mathcal{L}$ is the set of directed links that connects $\mathcal{Q}$ and $\mathcal{P}$. Note that a link only exists between a queue in $\mathcal{Q}$ and a processor in $\mathcal{P}$. We denote $N_p^{in} = |\mathcal{P}^{in}|$, $N_p^o = |\mathcal{P}^o|$ and $N_p = N_p^{in} + N_p^o$. We also denote $N_q^s = |\mathcal{Q}^s|$, $N_q^{in} = |\mathcal{Q}^{in}|$ and $N_q = N_q^s + N_q^{in}$.

Each processor P_n, when activated, consumes a certain amount of contents from a set of *supply* queues, denoted by $\mathbb{Q}_n^S$, and generates some amount of new contents. These new contents either go to a set of *demand* queues, denoted by $\mathbb{Q}_n^D$, if $P_n \in \mathcal{P}^{in}$, or are delivered to the output if $P_n \in \mathcal{P}^o$. For any queue $q_j \in \mathcal{Q}$, we use $\mathbb{P}_j^S$ to denote the set of processors that q_j serves as a supply queue, and use $\mathbb{P}_j^D$ to denote the set of processors that q_j serves as a demand queue. An example of such a network is shown in Fig. 2. In the following, we

assume that $|\mathbb{Q}_i^D| = 1, \forall\ P_i \in \mathcal{P}^{in}$, i.e., each processor only generates contents for a single demand queue.

We use β_{nj} to denote the amount processor P_n consumes from a queue q_j in $\mathbb{Q}_n^S$ when it is activated. For each $P_i \in \mathcal{P}^{in}$, we also use α_{ih} to denote the amount P_i generates into the queue q_h if $q_h = \mathbb{Q}_i^D$, when it is activated. For a processor $P_k \in \mathcal{P}^o$, we use α_{ko} to denote the amount of output generated by it when it is turned on. [6] We denote $\beta_{max} = \max_{i,j} \beta_{ij}$, $\beta_{min} = \min_{i,j} \beta_{ij}$ and $\alpha_{max} = \max_{i,j,}[\alpha_{ij}, \alpha_{io}]$. We assume that $\beta_{min}, \beta_{max}, \alpha_{max} > 0$. We also define M_p to be the maximum number of supply queues that any processor can have, define M_q^d to be the maximum number of processors that any queue can serve as a demand queue for, and define M_q^s to be the maximum number of processors that any queue can serve as a supply queue for. We use $R_j(t)$ to denote the amount of contents arriving to a source queue $q_j \in \mathcal{Q}^s$ at time t. We assume $R_j(t)$ is i.i.d. every slot, and that $R_j(t) \leq R_{max}$ for all $q_j \in \mathcal{Q}^s$ and all t. We assume that there are no exogenous arrivals into the queues in $\mathcal{Q}^{in}$.

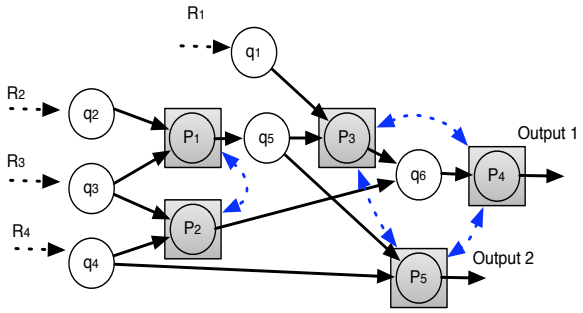

Figure 2: A general processing network. A dotted line between two processors means that the processors share some common resources and thus cannot be activated at the same time.

We assume that in every slot t, admitting any unit amount of $R_j(t)$ arrival incurs a cost of $c_j(t)$, and that activating any internal processor $P_i \in \mathcal{P}^{in}$ incurs a cost of $C_i(t)$, whereas activating any output processor $P_k \in \mathcal{P}^o$ generates a profit of $p_k(t)$ per unit output content. [7] We assume $c_j(t), C_i(t), p_k(t)$ are all i.i.d. every time slot. In the following, we also assume that $p_{min} \leq p_k(t) \leq p_{max}$, and that $c_{min} \leq c_j(t) \leq c_{max}$ and $C_{min} \leq C_i(t) \leq C_{max}$ for all k, j, i and for all time.

Below, we use $I_n(t) = 1$ to denote the activation decision of P_n, i.e., $I_n(t) = 1$ ($I_n(t) = 0$) means that P_n is activated (turned off). We also use $D_j(t) \in [0, 1]$ to denote the portion of arrivals from $R_j(t)$ that are admitted into q_j. We assume there exist some general constraint on how the processors can be activated, which can be due to, e.g., resource sharing among processors. We model this constraint by defining an activation vector $\boldsymbol{I}(t) = (I_1(t), ..., I_{N_p}(t))$, and then assume that $\boldsymbol{I}(t) \in \mathcal{I}$ for all time, where $\mathcal{I}$ denotes the set of all possible processor activation decision vectors, assuming all the queues have enough contents for

[6] Note that here we only consider binary actions of processors, i.e., ON/OFF. Our results can also be generalized into the case when there are multiple operation levels under which different amount of contents will be consumed and generated.

[7] This can be viewed as the difference between profit and cost associated with these processors.

processing. We assume that if a vector $\boldsymbol{I} \in \mathcal{I}$, then by changing one element of $\boldsymbol{I}$ from one to zero, the newly obtained vector $\boldsymbol{I}'$ satisfies $\boldsymbol{I}' \in \mathcal{I}$. Note that the chosen vector $\boldsymbol{I}(t)$ must always ensure the constraint (24), which in this case implies that $\boldsymbol{I}(t)$ has to satisfy the following constraint:

$$q_j(t) \geq \sum_{n \in \mathbb{P}_j^S} I_n(t)\beta_{nj}, \quad \forall\ j = 1, ..., r. \tag{44}$$

Under this constraint, we see that the queues evolve according to the following queueing dynamics:

$$q_j(t+1) = q_j(t) - \sum_{n \in \mathbb{P}_j^S} I_n(t)\beta_{nj} + D_j(t)R_j(t),\ \forall j \in \mathcal{Q}^s,$$
$$q_j(t+1) = q_j(t) - \sum_{n \in \mathbb{P}_j^S} I_n(t)\beta_{nj} + \sum_{n \in \mathbb{P}_j^D} I_n(t)\alpha_{nj},\ \forall j \in \mathcal{Q}^{in}.$$

Note that we have used $j \in \mathcal{Q}$ to represent $q_j \in \mathcal{Q}$, and use $n \in \mathcal{P}$ to represent $P_n \in \mathcal{P}$ in the above for notation simplicity. The objective is to maximize the time average of the following utility function:

$$f(t) \triangleq \sum_{k \in \mathcal{P}^o} I_k(t)p_k(t)\alpha_{ko} - \sum_{j \in \mathcal{Q}^s} D_j(t)R_j(t)c_j(t) - \sum_{i \in \mathcal{P}^{in}} I_i(t)C_i(t). \tag{45}$$

Our model with the objective function (45) can be used to model applications where generating completely processed contents is the primary target, e.g., [5].

7.2 Relation to the general model

We see that in this network, the network state, the action, and the traffic and service functions are as follows:

- The network state is given by: $S(t) = (c_j(t), j \in \mathcal{Q}^s, C_i(t), i \in \mathcal{P}^{in}, p_k(t), k \in \mathcal{P}^o)$.
- The action $x(t) = (D_j(t), j \in \mathcal{Q}^s, I_n(t), n \in \mathcal{P})$.
- The arrival functions are given by: $A_j(t) = A_j(S(t), x(t)) = D_j(t)R_j(t),\ \forall\, q_j \in \mathcal{Q}^s$, and $A_j(t) = A_j(S(t), x(t)) = \sum_{n \in \mathbb{P}_j^D} I_n(t)\alpha_{nj},\ \forall\, q_j \in \mathcal{Q}^{in}$.
- The service functions are given by: $\mu_j(t) = \mu_j(S(t), x(t)) = \sum_{n \in \mathbb{P}_j^S} I_n(t)\beta_{nj}, \quad \forall\ j$.

Thus, we see that this network model falls into the general processing network framework in Section 4, and Theorem 2 will apply in this case. Therefore, to ensure the algorithm performance, we only have to construct our PMW algorithm to ensure that Condition 1 holds. Also note that in this case, we have:

$$\delta_{max} = \max\left[\nu_{max}, N_p^o p_{max}\alpha_{max}, N_q^s R_{max}c_{max} + N_p^{in}C_{max}\right]. \tag{46}$$

Here ν_{max} is defined:

$$\nu_{max} \triangleq \max\left[M_q^d\alpha_{max}, R_{max}, M_q^s\beta_{max}\right]. \tag{47}$$

7.3 The PMW algorithm

We now obtain the PMW algorithm for this general network. In this case, we will look for a perturbation vector that is the same in all entries, i.e., $\boldsymbol{\theta} = \theta\mathbf{1}$. We first compute the "drift-plus-penalty" expression using the weighted perturbed Lyapunov function defined in (38) under some given positive constants $\{w_j\}_{j=1}^r$ and some nonzero constant θ:

$$\begin{aligned}\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B &- \sum_{j\in\mathcal{Q}^s} \mathbb{E}\big\{w_j\big[q_j(t)-\theta\big]\big[\sum_{n\in\mathbb{P}_j^S} I_n(t)\beta_{nj} - R_j(t)D_j(t)\big] \mid \boldsymbol{q}(t)\big\} \\ &- \sum_{j\in\mathcal{Q}^{in}} \mathbb{E}\big\{w_j\big[q_j(t)-\theta\big]\big[\sum_{n\in\mathbb{P}_j^S} I_n(t)\beta_{nj} - \sum_{n\in\mathbb{P}_j^D} I_n(t)\alpha_{nj}\big] \mid \boldsymbol{q}(t)\big\} \\ &- V\mathbb{E}\big\{\sum_{k\in\mathcal{P}^o} I_k(t)p_k(t)\alpha_{ko} - \sum_{j\in\mathcal{Q}^s} D_j(t)R_j(t)c_j(t) - \sum_{i\in\mathcal{P}^{in}} I_i(t)C_i(t) \mid \boldsymbol{q}(t)\big\}.\end{aligned} \tag{48}$$

Here $B = \delta_{max}^2 \sum_j w_j$ with δ_{max} defined in (46). Rearranging the terms in (48), we get the following:

$$\begin{aligned}\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B &+ \sum_{j\in\mathcal{Q}^s} \mathbb{E}\big\{\big[Vc_j(t) + w_j(q_j(t)-\theta)\big]D_j(t)R_j(t) \mid \boldsymbol{q}(t)\big\} \\ &- \sum_{k\in\mathcal{P}^o} \mathbb{E}\big\{I_k(t)\big[\sum_{j\in\mathbb{Q}_k^S} w_j(q_j(t)-\theta)\beta_{kj} + Vp_k(t)\alpha_{ko}\big] \mid \boldsymbol{q}(t)\big\} \\ &- \sum_{i\in\mathcal{P}^{in}} \mathbb{E}\big\{I_i(t)\big[\sum_{j\in\mathbb{Q}_i^S} w_j(q_j(t)-\theta)\beta_{ij} - w_h(q_h(t)-\theta)\alpha_{ih} - VC_i(t)\big] \mid \boldsymbol{q}(t)\big\}.\end{aligned} \tag{49}$$

Here in the last term $q_h = \mathbb{Q}_i^D$. We now present the PMW algorithm. We see that in this case the $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x)$ function is given by:

$$\begin{aligned}D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x) = &- \sum_{j\in\mathcal{Q}^s} \big[Vc_j(t) + w_j(q_j(t)-\theta)\big]D_j(t)R_j(t) + \sum_{k\in\mathcal{P}^o} I_k(t)\big[\sum_{j\in\mathbb{Q}_k^S} w_j(q_j(t)-\theta)\beta_{kj} + Vp_k(t)\alpha_{ko}\big] \\ &+ \sum_{i\in\mathcal{P}^{in}} I_i(t)\big[\sum_{j\in\mathbb{Q}_i^S} w_j(q_j(t)-\theta)\beta_{ij} - w_h(q_h(t)-\theta)\alpha_{ih} - VC_i(t)\big].\end{aligned} \tag{50}$$

Our goal is to design PMW in a way such that under any network state $S(t)$, the value of $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x)$ is close to $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))*}(x)$, which is the maximum value of $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x)$ without the underflow constraint (44), i.e.,

$$D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))*}(x) = \max_{D_j(t)\in[0,1],\boldsymbol{I}(t)\in\mathcal{I}} D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x).$$

Specifically, PMW works as follows:

PMW: Initialize $\boldsymbol{\theta}$. At every time slot t, observe $S(t)$ and $\boldsymbol{q}(t)$, and do the following:

1. Content Admission: Choose $D_j(t) = 1$, i.e., admit all new arrivals to $q_j \in \mathcal{Q}^s$ if:

$$Vc_j(t) + w_j(q_j(t)-\theta) < 0, \tag{51}$$

else set $D_j(t) = 0$.

2. Processor Activation: For each $P_i \in \mathcal{P}^{in}$, define its weight $W_i^{(in)}(t)$ as:

$$W_i^{(in)}(t) = \big[\sum_{q_j \in \mathbb{Q}_i^S} w_j[q_j(t) - \theta]\beta_{ij} - w_h[q_h(t) - \theta]\alpha_{ih} - VC_i(t)\big]^+, \tag{52}$$

where $q_h = \mathbb{Q}_i^D$. Similarly, for each $P_k \in \mathcal{P}^o$, define its weight $W_k^{(o)}(t)$ as:

$$W_k^{(o)}(t) = \big[\sum_{q_j \in \mathbb{Q}_k^S} w_j[q_j(t) - \theta]\beta_{kj} + Vp_k(t)\alpha_{ko}\big]^+. \tag{53}$$

Then, choose an activation vector $\boldsymbol{I}(t) \in \mathcal{I}$ to maximize:

$$W(t) \triangleq \sum_{i \in \mathcal{P}^{in}} I_i(t) W_i^{(in)}(t) + \sum_{k \in \mathcal{P}^o} I_k(t) W_k^{(o)}(t), \tag{54}$$

subject to the following *queue edge constraints:*

(a) For each $P_i \in \mathcal{P}^{in}$, set $I_i(t) = 1$, i.e., activate processor P_i, only if:

- $q_j(t) \geq M_q^s \beta_{max}$ for all $q_j \in \mathbb{Q}_i^S$ & $q_h(t) \leq \theta$, where $q_h = \mathbb{Q}_i^D$.

(b) For each $P_k \in \mathcal{P}^o$, choose $I_k(t) = 1$ only if:

- $q_j(t) \geq M_q^s \beta_{max}$ for all $q_j \in \mathbb{Q}_k^S$.

The approach of imposing the queue edge constraints was inspired by the work [22], where similar constraints are imposed for routing problems. Note that without these queue edge constraints, then PMW will be the same as the action that maximizes $D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ without the underflow constraint (44). Maximizing the quantity (54) in general can be NP-hard and requires centralized control [19]. However, if the processors do not conflict with each other, then we can easily maximize (54) by activating all the processors that have positive weights. Moreover, one can look for constant factor approximation solutions of (54), e.g., [23]. Such approximation results can usually be found in a distributed manner in polynomial time, and one can show that PMW achieves a constant factor of the optimal utility in this case [19].

7.4 Performance

Here we show that PMW indeed ensures that the value of $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ is within some additive constant of $D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$. In the following, we denote $w_{max} = \max_j w_j$ and $w_{min} = \min_j w_j$. We also assume that:

$$\theta \geq \max\big[\frac{V\alpha_{max}p_{max}}{w_{min}\beta_{min}}, \frac{Vc_{min}}{w_{min}} + M_q^s\beta_{max}\big]. \tag{55}$$

We also assume that the $\{w_j\}_{j=1}^r$ values are chosen such that for any processor $P_i \in \mathcal{P}^{in}$ with the demand queue q_h, we have for any supply queue $q_j \in \mathbb{Q}_i^S$ that:

$$w_j\beta_{ij} \geq w_h\alpha_{ih}. \tag{56}$$

We note that (55) can easily be satisfied and only requires $\theta = \Theta(V)$. A way of choosing the $\{w_j\}_{j=1}^r$ values to satisfy (56) is given in Appendix E. Note that in the special case when $\beta_{ij} = \alpha_{ij} = 1$ for all i, j, simply

using $w_j = 1, \forall\, j$ meets the condition (56).

We first look at the queueing bounds. By (51), q_j admits new arrivals only when $q_j(t) < \theta - Vc_{min}/w_j$. Thus:

$$q_j(t) \leq \theta - Vc_{min}/w_j + R_{max}, \quad \forall\ q_j \in \mathcal{Q}^s, t. \tag{57}$$

Now by the processor activation rule, we also see that:

$$0 \leq q_j(t) \leq \theta + M_q^d \alpha_{max}, \quad \forall\ q_j \in \mathcal{Q}^{in}, t. \tag{58}$$

This is because under the PMW algorithm, a processor is activated only when all its supply queues have at least $M_q^s \beta_{max}$ units of contents, and when its demand queue has at most θ units of contents. The first requirement ensures that when we activate a processor, all its supply queues have enough contents, while the second requirement ensures that $q_j(t) \leq \theta + M_q^d \alpha_{max}$. Using the definition of ν_{max} in (47), we can compactly write (57) and (58) as:

$$0 \leq q_j(t) \leq \theta + \nu_{max}, \quad \forall\ q_j \in \mathcal{Q}, t. \tag{59}$$

Recall that by the discussion in Section 7.2, the problem described in this section falls into the general framework presented in Section 4. Hence, to prove the performance of the PMW algorithm, it suffices to prove the following lemma, which shows that Condition 1 holds for some finite constant C under PMW.

Lemma 3. *Suppose (55) and (56) hold. Then under PMW,* $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x) \geq D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x) - C$, *where* $C = N_p w_{max} M_p \nu_{max} \beta_{max}$.

Proof. See Appendix F. □

We now use Theorem 2 to have the following corollary concerning the performance of PMW in this case:

Corollary 1. *Suppose (25), (55) and (56) hold. Then under PMW, (59) holds, and that:*

$$f_{av}^{PMW} \quad \geq \quad f_{av}^* - \frac{B+C}{V}, \tag{60}$$

$$\overline{q}^{PMW} \quad \leq \quad \frac{B + C + 2V\delta_{max}}{\eta} + \theta \sum_{j=1}^{r} w_j, \tag{61}$$

where $C = N_p w_{max} M_p \nu_{max} \beta_{max}$, f_{av}^{PMW} *and* $\overline{q}^{PMW}$ *are the time average expected utility and time average expected weighted backlog under PMW, respectively, and* δ_{max} *is given in (46).* ■

Also, since (55) only requires $\theta = \Theta(V)$, and $w_j = \Theta(1)$ for all j, we see that PMW achieves an $[O(1/V), O(V)]$ utility-backlog tradeoff in this case.

8 Simulation

In this section, we simulate the example given in Fig. 2. In this example, we assume each $R_j(t)$ is Bernoulli being 0 or 2 with equal probabilities. For each $P_i \in \mathcal{P}^{in}$, i.e., P_1, P_2, P_3, $C_i(t)$ is assumed to be 1 or 10 with probabilities 0.3 and 0.7, respectively. For the output processors $P_k \in \mathcal{P}^o$, i.e., P_4 and P_5, we assume that $p_k(t) = 1$ or 5 with probabilities 0.8 and 0.2, respectively. We assume that each processor, when activated, takes one unit of content from each of its supply queues and generates two units of contents into its demand queue (or to the output if it is an output processor). Due to the activation constraints, P_1 and P_2 can not be activated at the same time. Also only one among P_3, P_4, P_5 can be turned on at any time. Note that in this case, we have $c_j(t) = 0$ for all source queues q_j. It is easy to see that in this case $M_p = M_q^s = M_q^d = 2$, $\beta_{max} = \beta_{min} = 1$, and $\alpha_{max} = 2$. Using the results in the above, we choose $w_6 = 1$, $w_1 = w_4 = w_5 = 2$, $w_2 = w_3 = 4$. We also use $\theta = 10V$ according to (55). We simulate the PMW algorithm for $V \in \{5, 7, 10, 15, 20, 50, 100\}$. Each simulation is run over 5×10^6 slots.

Fig. 3 shows the utility and backlog performance of the PMW algorithm. We see that as V increases, the average utility performance quickly converges to the optimal value. The average backlog size also only grows linear in V. Fig. 3 also shows three sample path queue processes in the first 10^4 slots under $V = 100$.

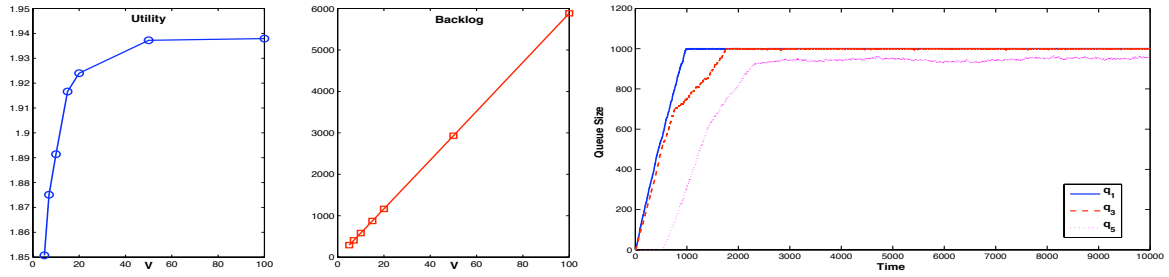

Figure 3: LFET: Utility and backlog performance of PMW. RIGHT: Sample path backlog processes with $V = 100$.

We see that no queue has an underflow. This shows that all the activation decisions of PMW are feasible. It is also easy to verify that the queueing bounds (57) and (58) hold. We observe in Fig. 3 that the queue sizes usually fluctuate around certain fixed values. Similar "exponential attraction" phenomenon has been observed in prior work [20]. Hence our results can also be extended, using the results developed in [20], to achieve an average utility that is within $O(1/V)$ of the optimal with only $\Theta([\log(V)]^2)$ average backlog size.

9 Perturbation and Tracking the Lagrange Multiplier

Note that although the main difference between the PMW algorithm and the usual Max-Weight algorithm is the use of the perturbation vector $\boldsymbol{\theta}$, this seemingly small modification indeed has a large effect on algorithm

design for processing networks. To see this, first recall that we have shown in Section 3.2 that *the usual Max-Weight algorithm cannot be applied to this problem.*

Now the reason perturbation resolves this issue is as follows. It has been shown in [20] that Max-Weight applied to a queueing problem is equivalent to solving a corresponding deterministic optimization problem using the randomized incremental subgradient method (RISM) [24], with the backlog vector being the Lagrange multiplier. Thus *Max-Weight relies on using the backlog vector to track the optimal Lagrange multiplier value* for achieving the desired utility performance, and the backlog vector under Max-Weight will move towards the optimal Lagrange multiplier. However, in processing network problems, due to the equality constraints in (32), the optimal Lagrange multiplier value may be *negative*. In this case, we can no longer use only the queue size to represent the Lagrange multiplier as the queue will be stuck at zero and the Max-Weight algorithm will not run properly. This is shown in Fig. 4.

The PMW algorithm instead resolves this problem by using the *queue size minus the perturbation* to represent the Lagrange multiplier. This is equivalent to "lifting" the Lagrange multiplier by the perturbation value, and making it trackable by the queue size. If we choose the perturbation carefully enough to ensure that no further deviation from the optimal Lagrange multiplier will happen, we can guarantee that no underflow ever happens. Then under the PMW algorithm, the queue size minus the perturbation will move towards the optimal Lagrange multiplier and we can thus achieve the close-to-optimal utility performance, as in the data network case.

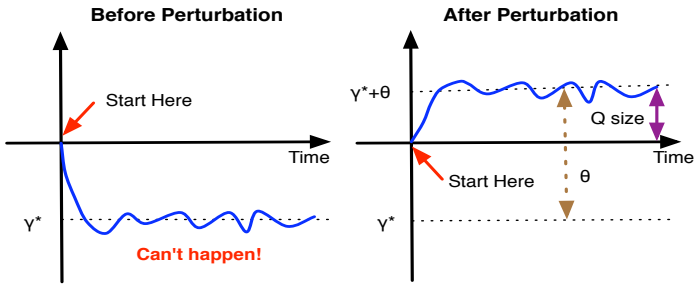

Figure 4: An explanation on why perturbation is needed and effective.

10 Conclusion and Future Work

In this paper, we present a general processing network model that can be used to model many problems involving the no-underflow constraint. We develop a general methodology for constructing online algorithms for such processing networks based on weight perturbation in Max-Weight. Using this approach, we develop the Perturbed Max-Weight algorithm (PMW) and show that PMW achieves an $[O(V), O(1/V)]$ utility-

backlog tradeoff. The results developed in this paper can find applications in areas such as manufacturing networks, stream processing and energy harvesting networks.

The current work can be further extended in the following two directions. First, as we discussed in Section 4.4, the use of a common slot size may lead to efficiency loss in networks where different components consume different times to process contents. It is thus desirable to quantify the effect. Second, designing optimal control algorithms that incorporate such heterogeneous processing times is another interesting future research direction.

References

[1] J. M. Harison. A broader view of brownian networks. *Ann. Appl. Probab.*, 2003.

[2] J. G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research, Vol 53, 197-218*, 2005.

[3] L. Jiang and J. Walrand. Stable and utility-maximizing scheduling for stochastic processing networks. *Allerton Conference on Communication, Control, and Computing*, 2009.

[4] H. Zhao, C. H. Xia, Z. Liu, and D. Towsley. A unified modeling framework for distributed resource allocation of general fork and join processing networks. *Proc. of ACM Sigmetrics*, 2010.

[5] M. J. Neely and L. Huang. Dynamic product assembly and inventory control for maximum profit. *IEEE Conference on Decision and Control (CDC), Atlanta, Georgia*, Dec. 2010.

[6] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure. Adaptive control of extreme-scale stream processing systems. *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[7] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. *Proceedings of the international conference on Management of data*, 2010.

[8] A.D. Joseph R. Katz M. Zaharia, A. Konwinski and I. Stoica. Improving mapreduce performance in heterogeneous environments. *OSDI*, December 2008.

[9] J. Cao, S. A. Jarvis, S.Saini, and G. R. Nudd. Gridflow: workflow management for grid computing. *Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, 2003.

[10] S. Eswaran, M. P. Johnson, A. Misra, and T. La Porta. Adaptive in-network processing for bandwidth and energy constrained mission-oriented multi-hop wireless networks. *Proc. of DCOSS*, 2009.

[11] A. Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. *IEEE/ACM Trans. Netw.*, 15(6):1333–1344, 2007.

[12] M. J. Neely. Energy optimal control for time-varying wireless networks. *IEEE Transactions on Information Theory 52(7): 2915-2934*, July 2006.

[13] L. Huang and M. J. Neely. The optimality of two prices: Maximizing revenue in a stochastic network. *IEEE/ACM Transactions on Networking, Vol. 18, No.2*, April 2010.

[14] R. Urgaonkar and M. J. Neely. Opportunistic scheduling with reliability guarantees in cognitive radio networks. *IEEE Transactions on Mobile Computing, vol. 8, no. 6, pp. 766-777*, June 2009.

[15] M. J. Neely. Super-fast delay tradeoffs for utility optimal fair scheduling in wireless networks. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Nonlinear Optimization of Communication Systems, vol. 24, no. 8, pp. 1489-1501*, Aug. 2006.

[16] Y. Yi and M. Chiang. Stochastic network utility maximization: A tribute to Kelly's paper published in this journal a decade ago. *European Transactions on Telecommunications*, vol. 19, no. 4, pp. 421-442, June 2008.

[17] D. I. Shuman and M. Liu. Energy-efficient transmission scheduling for wireless media streaming with strict underflow constraints. *WiOpt*, 2008.

[18] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. on Automatic Control, vol. 37, no. 12, pp. 1936-1949*, Dec. 1992.

[19] L. Georgiadis, M. J. Neely, and L. Tassiulas. *Resource Allocation and Cross-Layer Control in Wireless Networks*. Foundations and Trends in Networking Vol. 1, no. 1, pp. 1-144, 2006.

[20] L. Huang and M. J. Neely. Delay reduction via Lagrange multipliers in stochastic network optimization. *IEEE Transactions on Automatic Control, Volume 56, Issue 4, pp. 842-857*, April 2011.

[21] L. Huang and M. J. Neely. Max-weight achieves the exact $[O(1/V), O(V)]$ utility-delay tradeoff under Markov dynamics. *arXiv:1008.0200v1*, 2010.

[22] M. J. Neely. Universal scheduling for networks with arbitrary traffic, channels, and mobility. *Proc. IEEE Conf. on Decision and Control (CDC)*, Atlanta, GA, Dec 2010.

[23] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer congestion control in wireless networks. *IEEE/ACM Transactions on Networking, vol. 14, no. 2, pp302-315*, April 2006.

[24] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Boston: Athena Scientific, 2003.

[25] M. J. Neely. *Dynamic Power Allocation and Routing for Satellite and Wireless Networks with Time Varying Channels*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems (LIDS), 2003.

Appendix A – Proof of Theorem 1

We prove Theorem 1 in this section, using an argument similar to the one used in [13].

Proof. (Theorem 1) Consider any stable scheduling policy Π, i.e., the conditions (24) and (27) are satisfied under Π. We let $\{(f(0), \boldsymbol{A}(0), \boldsymbol{\mu}(0)), (f(1), \boldsymbol{A}(1), \boldsymbol{\mu}(1)), ...\}$ be a sequence of (utility, arrival, service) triple generated by Π. Then there exists a subsequence of times $\{T_i\}_{i=1,2,...}$ such that $T_i \to \infty$ and that the limiting time average utility over times T_i is equal to the liminf average utility under Π (defined by (28)). Now define the conditional average of utility, and arrival minus service over T slots to be:

$$(\phi^{(s_i)}(T); \epsilon_1^{(s_i)}(T); ...; \epsilon_r^{(s_i)}(T)) \triangleq \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\big\{f(t); \epsilon_1(t); ...; \epsilon_r(t) \mid S(t) = s_i\big\},$$

where $\epsilon_j(t) = A_j(t) - \mu_j(t)$. Using Caratheodory's theorem, it can be shown, as in [13] that, there exists a set of variables $\{a_k^{(s_i)}(T)\}_{k=1}^{r+2}$ and a set of actions $\{x_k^{(s_i)}(T)\}_{k=1}^{r+2}$ such that $\phi^{(s_i)}(T) = \sum_{k=1}^{r+2} a_k^{(s_i)}(T) f(s_i, x_k^{(s_i)}(T))$,

and for all $j = 1, ..., r$ that $\epsilon_j^{(s_i)}(T) = \sum_{k=1}^{r+2} a_k^{(s_i)}(T)[A_j(s_i, x_k^{(s_i)}(T)) - \mu_j(s_i, x_k^{(s_i)}(T))]$. Now using the continuity of $f(s_i, \cdot), A_j(s_i, \cdot), \mu_j(s_i, \cdot)$, and the compactness of all the actions sets $\mathcal{X}^{(s_i)}$, we can thus find a sub-subsequence $\tilde{T}_i \to \infty$ of $\{T_i\}_{i=1,2,...}$ that:

$$a_k^{(s_i)}(\tilde{T}_i) \to a_k^{(s_i)}, x_k^{(s_i)}(\tilde{T}_i)) \to x_k^{(s_i)}, \phi^{(s_i)}(\tilde{T}_i) \to \phi^{(s_i)}, \epsilon_j^{(s_i)}(\tilde{T}_i) \to \epsilon_j^{(s_i)}, \forall\, j = 1, ..., r. \tag{62}$$

Therefore the time average utility under the policy Π can be expressed as:

$$f_{av}^{\Pi} = \sum_{s_i} \pi_{s_i} \phi^{(s_i)} = \sum_{s_i} \pi_{s_i} \sum_{k=1}^{r+2} a_k^{(s_i)} f(s_i, x_k^{(s_i)}). \tag{63}$$

Similarly, the average arrival rate minus the average service rate under Π can be written as:

$$\epsilon_j = \sum_{s_i} \pi_{s_i} \epsilon_j^{(s_i)} = \sum_{s_i} \pi_{s_i} \sum_{k=1}^{r+2} a_k^{(s_i)} [A_j(s_i, x_k^{(s_i)}) - \mu_j(s_i, x_k^{(s_i)})] \leq 0. \tag{64}$$

The last inequality is due to the fact that Π is a stable policy and that $\mathbb{E}\{q_j(0)\} < \infty$, hence the average arrival rate to any q_j must be no more than the average service rate of the queue [25]. However, by (24) we see that what is consumed from a queue is always no more that what is generated into the queue. This implies that the input rate into a queue is always no less than its output rate. Thus, $\epsilon_j \geq 0$ for all j. Therefore we conclude that $\epsilon_j = 0$ for all j. Using this fact and (63), we see that $V f_{av}^{\Pi} \leq \phi^*$, where ϕ^* is given in (30). This proves Theorem 1. □

Appendix B – Proof of Lemma 1

We prove Lemma 1 here.

Proof. (Lemma 1) It is easy to see from (30) that the dual function is given by:

$$\hat{g}(\boldsymbol{\gamma}) = \sup_{x_k^{(s_i)}, a_k^{(s_i)}} \sum_{s_i} \pi_{s_i} \left\{ \sum_{k=1}^{r+2} a_k^{(s_i)} V f(s_i, x_k^{(s_i)}) - \sum_j \gamma_j \sum_{k=1}^{r+2} a_k^{(s_i)} \left[A_j(s_i, x_k^{(s_i)}) - \mu_j(s_i, x_k^{(s_i)}) \right] \right\}. \tag{65}$$

Due to the use of the $\{a_k^{(s_i)}\}_{i=1,...,M}^{k=1,...,r+2}$ variables, it is easy to see that $\hat{g}(\boldsymbol{\gamma}) \geq g(\boldsymbol{\gamma})$. However, if $\{x^{(s_i)}\}_{i=1}^{M}$ is a set of maximizers of $g(\boldsymbol{\gamma})$, then the set of variables $\{x_k^{(s_i)}, a_k^{(s_i)}\}_{i=1,...,M}^{k=1,...,r+2}$ where for each s_i, $x_k^{(s_i)} = x^{(s_i)}$ for all k, and $a_1^{(s_i)} = 1$ with $a_k^{(s_i)} = 0$ for all $k \geq 2$, will also be maximizers of $\hat{g}(\boldsymbol{\gamma})$. Thus $g(\boldsymbol{\gamma}) \geq \hat{g}(\boldsymbol{\gamma})$. This shows that $g(\boldsymbol{\gamma}) = \hat{g}(\boldsymbol{\gamma})$, and hence $g(\boldsymbol{\gamma})$ is the dual function of (30). $g(\boldsymbol{\gamma}^*) \geq \phi^*$ follows from weak duality [24]. □

Appendix C – Proof of Lemma 2

Here we prove Lemma 2.

Proof. Using the queueing equation (26), we have:

$$\begin{aligned}[q_j(t+1)-\theta_j]^2 &= [(q_j(t)-\mu_j(t)+A_j(t))-\theta_j]^2 \\ &= [q_j(t)-\theta_j]^2+(\mu_j(t)-A_j(t))^2-2\big(q_j(t)-\theta_j\big)[\mu_j(t)-A_j(t)] \\ &\leq [q_j(t)-\theta_j]^2+2\delta_{max}^2-2\big(q_j(t)-\theta_j\big)[\mu_j(t)-A_j(t)].\end{aligned}$$

Multiplying both sides with $\frac{w_j}{2}$ and summing the above over $j=1,...,r$, we see that:

$$L(t+1)-L(t)\leq B-\sum_{j=1}^{r}w_j\big(q_j(t)-\theta_j\big)[\mu_j(t)-A_j(t)],$$

where $B=\delta_{max}^2\sum_{j=1}^{r}w_j$. Now add to both sides the term $-Vf(t)$, we get:

$$L(t+1)-L(t)-Vf(t)\leq B-Vf(t)-\sum_{j=1}^{r}w_j\big(q_j(t)-\theta_j\big)[\mu_j(t)-A_j(t)]. \tag{66}$$

Taking expectations over $S(t)$ on both sides conditioning on $\boldsymbol{q}(t)$ proves the lemma. □

Appendix D – Proof of Theorem 2

Here we prove Theorem 2. We first have the following simple lemma.

Lemma 4. *For any network state s_i, we have:*

$$D^{(s_i)*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}=g_{s_i}((\boldsymbol{q}(t)-\boldsymbol{\theta})\otimes\boldsymbol{w}), \tag{67}$$

where $\boldsymbol{w}=(w_1,...,w_r)^T$ and $\boldsymbol{a}\otimes\boldsymbol{b}=(a_1b_1,...,a_nb_n)^T$.

Proof. By comparing (41) with (36), we see that the lemma follows. □

Proof. (Theorem 2) We first recall the equation (66) as follows:

$$L(t+1)-L(t)-Vf(t)\leq B-Vf(t)-\sum_{j=1}^{r}w_j\big(q_j(t)-\theta_j\big)[\mu_j(t)-A_j(t)]. \tag{68}$$

Using $D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ defined in (40), this can be written as:

$$L(t+1)-L(t)-Vf(t)\leq B-D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x(t)).$$

Here $x(t)$ is PMW's action at time t. According to Condition 1, we see that for any network state $S(t)=s_i$, PMW ensures (24), and that:

$$D^{(s_i)}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)\geq D^{(s_i)*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}-C.$$

Using (67), this implies that under PMW,

$$L(t+1)-L(t)-Vf(t)\leq B-g_{s_i}((\boldsymbol{q}(t)-\boldsymbol{\theta})\otimes\boldsymbol{w})+C.$$

Taking expectations over the random network state on both sides conditioning on $\boldsymbol{q}(t)$, and using (37), i.e., $g(\boldsymbol{\gamma}) = \sum_{s_i} \pi_{s_i} g_{s_i}(\boldsymbol{\gamma})$, we get:

$$\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B + C - g((\boldsymbol{q}(t) - \boldsymbol{\theta}) \otimes \boldsymbol{w}). \tag{69}$$

Now using Theorem 1 and Lemma 1, we have: $Vf^*_{av} \leq \phi^* \leq g(\boldsymbol{\gamma}^*) \leq g((\boldsymbol{q}(t) - \boldsymbol{\theta}) \otimes \boldsymbol{w})$. Therefore,

$$\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B + C - Vf^*_{av}. \tag{70}$$

Taking expectations over $\boldsymbol{q}(t)$ on both sides and summing the above over $t = 0, ..., T-1$, we get:

$$\mathbb{E}\big\{L(T) - L(0)\big\} - \sum_{t=0}^{T-1} V\mathbb{E}\big\{f(t)\big\} \leq T(B+C) - TVf^*_{av}.$$

Rearranging terms, dividing both sides by VT, using the facts that $L(t) \geq 0$ and $\mathbb{E}\big\{L(0)\big\} < \infty$, and taking the liminf as $T \to \infty$, we get:

$$f^{PMW}_{av} \geq f^*_{av} - (B+C)/V. \tag{71}$$

This proves (42). Now we prove (43). First, by using the definition of $\hat{g}(\boldsymbol{\gamma})$ in (65), and plugging in the $\{\hat{x}^{(s_i)}_k, \hat{a}^{(s_i)}_k\}^{k=1,...,r+2}_{i=1,...,M}$ variables in the η-slackness assumption (25) in Section 4.2, we see that:

$$\hat{g}((\boldsymbol{q}(t) - \boldsymbol{\theta}) \otimes \boldsymbol{w}) \geq \eta \sum_{j=1}^{r} w_j[q_j(t) - \theta_j] - V\delta_{max}. \tag{72}$$

This by Lemma 1 implies that:

$$g((\boldsymbol{q}(t) - \boldsymbol{\theta}) \otimes \boldsymbol{w}) \geq \eta \sum_{j=1}^{r} w_j[q_j(t) - \theta_j] - V\delta_{max}.$$

Using this in (69), we get:

$$\Delta(t) - V\mathbb{E}\big\{f(t) \mid \boldsymbol{q}(t)\big\} \leq B + C + V\delta_{max} - \eta \sum_{j=1}^{r} w_j[q_j(t) - \theta_j].$$

We can now use a similar argument as above to get:

$$\eta \sum_{t=0}^{T-1} \sum_{j=1}^{r} w_j \mathbb{E}\big\{[q_j(t) - \theta_j]\big\} \leq T(B+C) + 2TV\delta_{max} + \mathbb{E}\big\{L(0)\big\}.$$

Dividing both sides by ηT and taking the limsup as $T \to \infty$, we get:

$$\overline{q}^{PMW} \quad \leq \quad \frac{B + C + 2V\delta_{max}}{\eta} + \sum_{j=1}^{r} w_j \theta_j.$$

This completes the proof the theorem. □

Appendix E – Choosing the $\{w_j\}^r_{j=1}$ values

Here we describe how to choose the $\{w_j\}^r_{j=1}$ values to satisfy (56). We first let K be the maximum number of processors that any path going from a queue to an output processor can have. It is easy to see that $K \leq |N_p|$

since there is no cycle in the network. The following algorithm terminates in K iterations. We use $w_j(k)$ to denote the value of w_j at the k^{th} iteration. In the following, we use q_{h_n} to denote the demand queue of a processor P_n.

1. At Iteration 1, denote the set of queues that serve as supply queues for any output processor as $\mathbb{Q}_1^l$, i.e.,

$$\mathbb{Q}_1^l = \{q_j : \mathbb{P}_j^S \cap \mathcal{P}^o \neq \phi\}.$$

   Then set $w_j(1) = 1$ for each $q_j \in \mathbb{Q}_1^l$. Also, set $w_j(1) = 0$ for all other $q_j \notin \mathbb{Q}_1^l$.

2. At Iteration $k = 2, ..., K$, denote $\mathbb{Q}_k^l$ to be the set of queues that serve as supply queues for any processor whose demand queue is in $\mathbb{Q}_{k-1}^l$, i.e.,

$$\mathbb{Q}_k^l = \{q_j : \exists\, P_n \in \mathbb{P}_j^S \text{ s.t. } \mathbb{Q}_n^D \in \mathbb{Q}_{k-1}^l\}.$$

   Then set:

$$w_j(k) = \max\big[w_j(k-1), \max_{n \in \mathbb{P}_j^S} \frac{w_{h_n}(k-1)\alpha_{nh_n}}{\beta_{nj}}\big], \tag{73}$$

   where α_{nh_n} is the amount P_n generates into q_{h_n}, which is the demand queue of P_n. Also, set $w_j(k) = w_j(k-1)$ for all $q_j \notin \mathbb{Q}_k^l$.

3. Output the $\{w_j\}_{j=1}^r$ values.

The following lemma shows that the above algorithm outputs a set of $\{w_j\}_{j=1}^r$ values that satisfy (56).

Lemma 5. *The $\{w_j\}_{j=1}^r$ values generated by the above algorithm satisfy (56).*

Proof. (Proof of Lemma 5) The proof consists of two main steps. In the first step, we show that the algorithm updates each w_j value at least once. This shows that all the w_j values for all the queues that serve as demand queues are updated at least once. In the second step, we show that if q_h is the demand queue of a processor $P_i \in \mathcal{P}^{in}$, then every time after w_h is updated, the algorithm will also update w_j for any $q_j \in \mathbb{Q}_i^S$ before it terminates. This ensures that (56) holds for any $P_i \in \mathcal{P}^{in}$ and hence proves the lemma.

First we see that after K iterations, we must have $\mathcal{Q} \subset \cup_{\tau=1}^K \mathbb{Q}_\tau^l$. This is because at Iteration k, we include in $\cup_{\tau=1}^k \mathbb{Q}_\tau^l$ all the queues starting from which there exists a path to an output processor that contains k processors. Thus all the w_j values are updated at least once.

Now consider a queue q_h. Suppose q_h is the demand queue of a processor $P_i \in \mathcal{P}^{in}$. We see that there exists a time $\hat{k} \leq K$ at which w_h is last modified. Suppose w_h is last modified at Iteration $\hat{k} < K$, in which case $q_h \in \mathbb{Q}_{\hat{k}}^l$. Then all the queues $q_j \in \mathbb{Q}_i^S$ will be in $\mathbb{Q}_{\hat{k}+1}^l$. Thus their w_j values will be modified at Iteration $\hat{k}+1 \leq K$. This implies that at Iteration $\hat{k}+1$, we will have $w_j(\hat{k}+1)\beta_{ij} \geq w_h(\hat{k})\alpha_{ih}$. Since $q_h \notin \mathbb{Q}_k^l$ for

$k \geq \hat{k}+1$, we have $w_h(k) = w_h(\hat{k})$ for all $k \geq \hat{k}+1$. Therefore $w_j(k)\beta_{ij} \geq w_h(k)\alpha_{ih}\ \forall\ \hat{k}+1 \leq k \leq K$, because $w_j(k)$ is not decreasing.

Therefore the only case when the algorithm can fail is when w_h is updated at Iteration $k = K$, in which case w_h may increase but the w_j values for $q_j \in \mathbb{Q}_i^S$ are not modified accordingly. However, since w_h is updated at Iteration $k = K$, this implies that there exists a path from q_h to an output processor that has K processors. This in turn implies that starting from any $q_j \in \mathbb{Q}_i^S$, there exists a path to an output processor that contains $K+1$ processors. This contradicts the definition of K. Thus the lemma follows. □

As a concrete example, we consider the example in Fig. 2, with the assumption that each processor, when activated, consumes one unit of content from each of its supply queues and generates two units of contents into its demand queue. In this example, we see that $K = 3$. Thus the algorithm works as follows:

1. Iteration 1, denote $\mathbb{Q}_1^l = \{q_4, q_5, q_6\}$, set $w_4(1) = w_5(1) = w_6(1) = 1$. For all other queues, set $w_j(1) = 0$.
2. Iteration 2, denote $\mathbb{Q}_2^l = \{q_1, q_2, q_3, q_4, q_5\}$, set $w_1(2) = w_2(2) = w_3(2) = w_4(2) = w_5(2) = 2$. Set $w_6(2) = 1$.
3. Iteration 3, denote $\mathbb{Q}_3^l = \{q_2, q_3\}$, set $w_2(3) = w_3(3) = 4$. Set $w_1(3) = w_4(3) = w_5(3) = 2$, $w_6(3) = 1$.
4. Terminate and output $w_1 = w_4 = w_5 = 2$, $w_2 = w_3 = 4$, $w_6 = 1$.

Appendix F – Proof of Lemma 3

Here we prove Lemma 3 by comparing the values of the three terms in $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ in (50) under PMW versus their values under the action that maximizes $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ in (50) subject to only the constraints $D_j(t) \in [0,1]$, $\forall\, j \in \mathcal{Q}^s$ and $\boldsymbol{I}(t) \in \mathcal{I}$, called the max-action. That is, under the max-action, $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x) = D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$. Note that the max-action differs from PMW only in that it does not consider the queue edge constraint.

Proof. (A) We see that the first term, i.e., $-\sum_{j\in\mathcal{Q}^s}\big[Vc_j(t) + w_j(q_j(t)-\theta)\big]D_j(t)R_j(t)$ is maximized under PMW. Thus its value is the same as that under the max-action.

(B) We now show that for any processor $P_n \in \mathcal{P}$, if it violates the queue edge constraint, then its weight is bounded by $M_p w_{max}\nu_{max}\beta_{max}$. This will then be used in Part (C) below to show that the value of $D^{(S(t))}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$ under PMW is within a constant of $D^{(S(t))*}_{\boldsymbol{\theta},\boldsymbol{q}(t)}(x)$.

(B-I) For any $P_i \in \mathcal{P}^{in}$, the following are the only two cases under which P_i violates the queue edge constraint.

1. Its demand queue $q_h(t) \geq \theta$. In this case, it is easy to see from (52) and (59) that:

$$W_i^{(in)}(t) \leq \sum_{j\in\mathbb{Q}_i^S} w_j\nu_{max}\beta_{ij} \leq M_p w_{max}\nu_{max}\beta_{max}. \tag{74}$$

2. At least one of P_i's supply queue has a queue size less than $M_q^s\beta_{max}$. In this case, we denote $\hat{\mathbb{Q}}_i^S = \{q_j \in \mathbb{Q}_i^S : q_j(t) \geq M_q^s\beta_{max}\}$. Then we see that:

$$\begin{aligned} W_i^{(in)}(t) &= \sum_{j\in\hat{\mathbb{Q}}_i^S} w_j[q_j(t)-\theta]\beta_{ij} - w_h[q_h(t)-\theta]\alpha_{ih} + \sum_{j\in\mathbb{Q}_i^S/\hat{\mathbb{Q}}_i^S} w_j[q_j(t)-\theta]\beta_{ij} - VC_i(t) \\ &\leq \sum_{j\in\hat{\mathbb{Q}}_i^S} w_j\nu_{max}\beta_{ij} + w_h\theta\alpha_{ih} + \sum_{j\in\mathbb{Q}_i^S/\hat{\mathbb{Q}}_i^S} w_j[M_q^s\beta_{max}-\theta]\beta_{ij}. \end{aligned}$$

Here $q_h = \mathbb{Q}_i^D$. Now by our selection of $\{w_j\}_{j=1}^r$, $w_j\beta_{ij} \geq w_h\alpha_{ih}$ for any $q_j \in \mathbb{Q}_i^S$. Also using $\nu_{max} \geq M_q^s\beta_{max}$, we have:

$$W_i^{(in)}(t) \leq M_p w_{max}\nu_{max}\beta_{max}. \tag{75}$$

(B - II) For any $P_k \in \mathcal{P}^o$, we see that it violates the queue edge constraint only when at least one of its supply queues has size less than $M_q^s\beta_{max}$. In this case, we see that:

$$\begin{aligned} W_k^{(o)}(t) &\leq \sum_{j\in\hat{\mathbb{Q}}_k^S} w_j[q_j(t)-\theta]\beta_{kj} + Vp_k(t)\alpha_{ko} + \sum_{j\in\mathbb{Q}_k^S/\hat{\mathbb{Q}}_k^S} w_j(M_q^s\beta_{max}-\theta)\beta_{ij} \\ &\leq M_p w_{max}\nu_{max}\beta_{max} + V\alpha_{max}p_{max} - w_{min}\theta\beta_{min}. \end{aligned}$$

This by (55) implies that:

$$W_k^{(o)}(t) \leq M_p w_{max}\nu_{max}\beta_{max}. \tag{76}$$

Using (74), (75) and (76), we see that whenever a processor violates the queue edge constraint, its weight is at most $M_p w_{max}\nu_{max}\beta_{max}$.

(C) We now show that the value of $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x)$ under PMW satisfies $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x) \geq D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))*}(x) - C$, where $C = N_p M_p w_{max}\nu_{max}\beta_{max}$.

To see this, let $\boldsymbol{I}^*(t)$ be the activation vector obtained by the max-action, and let $W^*(t)$ be the value of (54) under $\boldsymbol{I}^*(t)$. We also use $\boldsymbol{I}^{PMW}(t)$ and $W^{PMW}(t)$ to denote the activation vector chosen by the PMW algorithm and the value of (54) under $\boldsymbol{I}^{PMW}(t)$. We now construct an alternate activation vector $\hat{\boldsymbol{I}}(t)$ by changing all elements in $\boldsymbol{I}^*(t)$ corresponding to the processors that violate the queue edge constraints to zero. Note then $\hat{\boldsymbol{I}}(t) \in \mathcal{I}$ is a feasible activation vector at time t, under which no processor violates the queue edge constraint. By Part (B) above, we see that the value of (54) under $\hat{\boldsymbol{I}}(t)$, denoted by $\hat{W}(t)$, satisfies:

$$\hat{W}(t) \geq W^*(t) - N_p M_p w_{max}\nu_{max}\beta_{max}.$$

Now since $\boldsymbol{I}^{PMW}(t)$ maximizes the value of (54) subject to the queue edge constraints, we have:

$$W^{PMW}(t) \geq \hat{W}(t) \geq W^*(t) - N_p w_{max} M_p \nu_{max}\beta_{max}.$$

Thus, by combining the above and Part (A), we see that PMW maximizes the $D_{\boldsymbol{\theta},\boldsymbol{q}(t)}^{(S(t))}(x)$ to within $C = N_p M_p w_{max}\nu_{max}\beta_{max}$ of the maximum. □