EE 550: Notes on Network Scheduling

Michael J. Neely
University of Southern California
http://www-bcf.usc.edu/~mjneely

I. $N \times N$ PACKET SWITCH

Recall the $N \times N$ packet switch model has N inputs, N outputs, and operates over discrete time with time slots $t \in \{0, 1, 2, \ldots\}$. The *crossbar constraint* requires us to send no more than 1 packet from each input $i \in \{1, \ldots, N\}$ per slot, and no more than 1 packet to each output $j \in \{1, \ldots, N\}$ per slot. Every slot t we must choose a binary *permutation matrix* $(S_{ij}(t))$ that defines which packets we can switch from inputs to outputs.[1] That is, if $S_{ij}(t) = 0$ then we cannot send any packet from input i to output j. If $S_{ij}(t) = 1$ then we can send exactly one packet from input i to output j. Since $(S_{ij}(t))$ is a permutation matrix, on every slot t the switch can deliver at most N packets to their desired outputs.

A. Minimum clearance time

Let $\boldsymbol{Q} = (Q_{ij})$ be a $N \times N$ matrix of nonnegative integers. This shall be called the *backlog matrix* and represents the packets that need to be scheduled for transmission over the switch. Specifically, Q_{ij} is the number of packets waiting at input i that need to be sent to output j. Assume we are given an initial backlog matrix $\boldsymbol{Q} = (Q_{ij})$ and we want to minimize the *clearance time* (assuming no new arrivals take place). Let $T_{min}(\boldsymbol{Q})$ be the minimum time required to clear backlog matrix $\boldsymbol{Q}$.

As an example consider the following 4×4 backlog matrix:

$$\boldsymbol{Q} = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Here is one attempt to clear the backlog. The particular *matchings* that are used on every slot are shown by boxing the nonzero elements that are selected, subject to the constraint that no row or column has more than one selected entry.

$$\begin{bmatrix} \boxed{2} & 0 & 1 & 0 \\ 0 & \boxed{1} & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & \boxed{2} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & \boxed{2} & 0 & 0 \\ 0 & 0 & \boxed{1} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 0 & 0 \\ \boxed{2} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \boxed{1} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \boxed{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow (0)$$

This seems to be a reasonable sequence of choices. It clears all backlog in 5 timeslots. Is 5 timeslots the best we can do? No. It is possible to make a different sequence of choices to clear in only 4 timeslots. (Can you find a way to do this?) Notice that row three has 4 packets at the start and so it is impossible to clear in less than 4 slots. Thus, $T_{min}(\boldsymbol{Q}) = 4$ for this example. Also, any choices that clear in *exactly* 4 slots must choose exactly one packet from row three on each of those slots (this is why the above choices fail to clear the backlog in 4 slots).[2]

[1] A permutation matrix is a square matrix with binary entries that has exactly one "1" entry on each row and column. There are $N!$ permutation matrices of size $N \times N$.

[2] A *matching* is a selection of the nonzero entries of the matrix so that no row or column has more than one selected entry. The problem of finding a matching on a particular slot can also be represented by a *bipartite graph* with N input nodes on the left, N output nodes on the right, and an edge (i, j) between an input node i and an output node j if and only if $Q_{ij} > 0$. A *maximal matching* is a selection of entries that cannot be trivially improved by keeping the selected entries but adding more. It is easy to find a maximal matching by a simple greedy algorithm that sequentially adds new selections until it is impossible to add more. The *size* of a match is the number of entries selected. A *max size match* is a matching that has maximum size when compared to all other possible matchings. The size of a match is always less than or equal to N, and so a match that has size N must be a max size match. A match of size N is also called a *perfect match*. A max size match is always a maximal match, but a maximal match is not always a max size match. A simple example of a matching that is maximal, but not max size, is below:

$$\begin{bmatrix} \boxed{1} & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This matching has size 1 and is *maximal* since we cannot trivially improve it by adding new entries without violating a row or column constraint. On the other hand we can produce a matching of size 2 by erasing the single selection shown above and selecting completely new entries $(1, 2)$ and $(2, 1)$. It is not always easy to find a max size match, but there are polynomial time *max size match* algorithms. The above 4×4 example that clears backlog in 5 slots is interesting because it uses a sequence of max size matches on every slot, yet it does not achieve minimum clearance time. Theorem 1 shows how minimum clearance time can be achieved by first *augmenting* the matrix and then performing a sequence of max size matches on the augmented matrix. Using the augmented matrix ensures that the max size matches used on each step have size N and hence are perfect matches. It can be shown that greedy maximal matching on every slot clears the backlog within a factor of 2 of the minimum clearance time and so such greedy matchings are often preferred in scheduling algorithms [1][2].

Theorem 1. $T_{min}(\boldsymbol{Q}) = \max[r_1, r_2, ..., r_N, c_1, c_2, ..., c_N]$ *where* r_i, c_j *are the row and column sums for indices* i *and* j*, respectively, defined by*

$$r_i = \sum_{j=1}^{n} Q_{ij} \quad \forall i \in \{1, \ldots, N\}$$

$$c_j = \sum_{i=1}^{n} Q_{ij} \quad \forall j \in \{1, \ldots, N\}$$

Proof. Define $T = \max[r_1, r_2, ..., r_N, c_1, c_2, ..., c_N]$.

1) ($T_{min}(\boldsymbol{Q}) \geq T$) Consider any particular algorithm for emptying the backlog matrix $\boldsymbol{Q}$. Fix $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, N\}$. Since this algorithm can deliver at most one packet from row i per slot, and there are a total of r_i packets on row i, it must take *at least* r_i time slots to empty the system. Similarly, since it can deliver at most one packet from column j per slot, it must take *at least* c_j time slots to empty the system. This holds for all $i, j \in \{1, \ldots, N\}$ and so the time to empty is *at least* $T = \max[r_1, ..., r_N, c_1, ..., c_N]$. This holds for all algorithms, so $T_{min}(\boldsymbol{Q}) \geq T$.
2) ($T_{min}(\boldsymbol{Q}) \leq T$) This is not obvious and is sometimes called the *Birkhoff-Von Neumann theorem* [3]. It has two parts, which are sketched below: To avoid triviality, assume $T > 0$.
   - (Von Neumann) *Greedily add packets* to (Q_{ij}) to produce a new backlog matrix $(\tilde{Q}_{ij})$ that has each row and column sum equal to T.[3] A simple theorem (attributed to Von Neumann) shows this can always be done.
   - (Birkhoff) This part is attributed to *Birkhoff*, although it primarily uses a theorem called *Hall's theorem* from graph theory. Hall's theorem can be used to show that for any square matrix $\boldsymbol{Q}$ composed of nonnegative integers with all row and column sums equal to $T > 0$, there exists a *perfect match*, meaning a permutation matrix that schedules exactly N packets (one for each input and one for each output).[4] The matrix $\tilde{Q}$ is such a matrix (all rows and columns sum to $T > 0$), so we find a perfect match, remove one packet from each row and column, so the matrix on the next slot is composed of nonnegative integers with all row and column sums being exactly $T - 1$. If $T - 1 = 0$ then the system is empty and we have finished in $T = 1$ steps. Else, $T - 1 > 0$ and we can again find a perfect match, remove exactly one packet on each row and column, and the the new matrix on the next slot has all row and column sums exactly $T - 2$. Proceeding this way, we strip off perfect matches at each step $t \in \{1, ..., T\}$ to empty the system with exactly T steps.

□

B. IMET algorithm

Now suppose packets randomly arrive to the network over time. We can use the following *Iterative Minimum Emptying Time Algorithm* based on pre-selecting a *frame size* $T > 0$ and implementing the following algorithm over successive frames of size T:

- Frame 1: On the first frame of T slots, do nothing. Wait for new packets to arrive and let $\boldsymbol{Q}[1]$ denote the matrix of accumulated packet arrivals over frame 1. If the max row and column sum of $\boldsymbol{Q}[1]$ is less than or equal to T, keep all these arrivals. Else, drop packets to produce a new matrix $\tilde{\boldsymbol{Q}}[1]$ with all row and column sums less than or equal to T.
- Frame $k \in \{2, 3, 4, ...\}$: On frame $k \in \{2, 3, 4, ...\}$, ignore all new arrivals on that T-slot frame. Schedule to clear all packets $\tilde{\boldsymbol{Q}}[k-1]$ within this frame. This is always possible by Theorem 1. Let $\boldsymbol{Q}[k]$ be the matrix of all packets that arrived on frame 2. As before, if the max row and column sum of $\boldsymbol{Q}[k]$ is less than or equal to T, keep all these arrivals. Else, drop packets to produce a new matrix $\tilde{\boldsymbol{Q}}[k]$ with all row and column sums less than or equal to T. Repeat for frame $k + 1$.

Every arriving packet that is not dropped must arrive on some frame k and is served on the next frame $k + 1$. Hence, it is clear that the worst-case delay of all non-dropped packets is $2T$ (the size of two consecutive frames). We want to choose a frame size T so that packet drops are rare.

C. Bounds for leaky bucket traffic

Suppose packets arrive over time and let $A_{ij}(t)$ be the number of packets that arrive to input $i \in \{1, ..., N\}$ that are destined for output $j \in \{1, ..., N\}$. So far we have used *stochastic* descriptions of traffic (such as Poisson traffic). Let us use a new

[3] What do we mean by "greedily add"? Let's go row-by-row: For entry $(1, 1)$, add packets to Q_{11} until either row 1 or column 1 has a sum of T. For entry $(1, 2)$, add packets to Q_{12} until either row 1 or column 2 has a sum of T. Keep going. Of course, once we know we have completed a particular row i or column j, we can skip those rows/columns from then on. For example, if adding packets to Q_{11} makes row 1 sum to T, we can keep all other entries on row 1 constant and skip directly to row 2.

[4] Such a permutation matrix always exists, but it is not always easy to find: In general we need to use a *max-size match* algorithm to find one.

deterministic description: Assume that each arrival process $\{A_{ij}(t)\}_{t=0}^{\infty}$ is *leaky bucket constrained with rate parameter* $\lambda_{ij} \geq 0$ *and burst parameter* $\sigma_{ij} \geq 0$, meaning that for all slots $t_0 \in \{0, 1, 2, \ldots\}$ and all integers $T > 0$ we have:

$$\sum_{\tau=t_0}^{t_0+T-1} A_{ij}(\tau) \leq \lambda_{ij} T + \sigma_{ij} \tag{1}$$

In particular, the total number of arrivals over any interval of T slots is at most $\lambda_{ij} T + \sigma_{ij}$.

The parameter λ_{ij} is called the long-term *rate parameter* because (1) implies:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{\tau=0}^{T-1} A_{ij}(\tau) \leq \lambda_{ij}$$

and so the long-term rate of packet arrivals of type (i, j) is at most λ_{ij} packets/slot. the parameter σ_{ij} is called the *burst parameter* because it specifies how much the long term rate constraint can be violated in the short term.

Theorem 2. *(IMET under leaky bucket traffic) Suppose that for each* $i, j \in \{1, \ldots, N\}$*, the arrival process* $\{A_{ij}(t)\}_{t=0}^{\infty}$ *satisfies the leaky bucket constraints* (1) *for some parameters* $\lambda_{ij} \geq 0$ *and* $\sigma_{ij} \geq 0$*. Suppose that the rate matrix* (λ_{ij}) *is in the* switch capacity region *so that:*

$$\sum_{j=1}^{N} \lambda_{ij} \leq \rho \quad \forall i \in \{1, \ldots, N\} \tag{2}$$

$$\sum_{i=1}^{N} \lambda_{ij} \leq \rho \quad \forall j \in \{1, \ldots, N\} \tag{3}$$

for some value ρ *that satisfies* $0 \leq \rho < 1$*. Define* σ_{max} *as the largest row or column sum over the matrix* (σ_{ij})*. Then the IMET algorithm with frame size*

$$T = \frac{\sigma_{max}}{1-\rho} \tag{4}$$

never drops any packets and hence all packets have delay at most $2T$*.*

Proof. Let $\boldsymbol{Q}[k]$ be the total arrivals that take place on frame k. It suffices to show that $T_{min}(\boldsymbol{Q}) \leq T$, which means that all of these arrivals can be scheduled and cleared on frame $k+1$. We have

$$\begin{aligned}
T_{min}(\boldsymbol{Q}[k]) &= \max_{i,j\in\{1,\ldots,N\}} \left[\sum_{m=1}^{N} Q_{im}[k], \sum_{n=1}^{N} Q_{nj}[k]\right] \\
&\overset{(a)}{\leq} \max_{i,j\in\{1,\ldots,N\}} \left[\sum_{m=1}^{N} (\lambda_{im} T + \sigma_{im}), \sum_{n=1}^{N} (\lambda_{nj} T + \sigma_{nj})\right] \\
&\overset{(b)}{\leq} \max_{i,j\in\{1,\ldots,N\}} [\rho T + \sigma_{max}, \rho T + \sigma_{max}] \\
&= \rho T + \sigma_{max} \\
&\overset{(c)}{=} \frac{\rho \sigma_{max}}{1-\rho} + \sigma_{max} = T
\end{aligned}$$

where (a) holds by (1); (b) holds by (2)-(3); (c) holds by (4). □

For Poisson arrivals, which are *not* leaky bucket constrained, it can be shown that the rate of packet drops is very small, and this style of frame-based algorithm with min clearance time scheduling can be used to give the best known asymptotic delay for $N \times N$ packet switches [4].

D. IMET with variable size frames

The fixed-frame IMET algorithm may waste time with several idle slots at the end of frame, even when packets are waiting to be served in the next frame. Consider the following redesign. This algorithm never drops packets.

- Frame 1 (slot $t = 0$): Frame 1 consists of one slot, slot $t = 0$. Ignore the new packets that arrive on slot $t = 0$. Call these packets $\boldsymbol{Q}[1]$.
- Frame $k \in \{2, 3, 4, ...\}$: If $\boldsymbol{Q}[k-1] = \boldsymbol{0}$ then frame k has a size of one slot and is idle on that slot. If $\boldsymbol{Q}[k-1] \neq 0$, clear backlog matrix $\boldsymbol{Q}[k-1]$ in minimum time, so frame k has size $T_{min}(\boldsymbol{Q}[k-1])$. Ignore all new arrivals during frame k, and call these packets $\boldsymbol{Q}[k]$. Repeat for frame $k+1$.

Exercise 1. *(Variable frame IMET) Use induction to prove that, under the leaky bucket traffic assumptions of Theorem 2, the maximum size of any frame is $\sigma_{max}/(1-\rho)$ and so, as before, the worst-case delay is $2\sigma_{max}/(1-\rho)$.*

An advantage of this approach is that we do not drop any packets, we do not need to choose the frame size in advance, and we do not need to know the values of ρ or σ_{max}. A danger of this algorithm is that, since no packets are dropped, if the leaky bucket assumptions of Theorem 2 are *not* satisfied, then the system can go unstable and the frame size can go to infinity. A variation on both approaches is to use a variable size frame, but impose a maximum frame size T_{max}. We drop packets to ensure no frames are larger than T_{max}. The value T_{max} can be sized according to σ_{max} and ρ parameters that are reasonable targets. Notice that all of these algorithms require an immediate solution to the min clearance time problem. Indeed, once all packets of frame k have arrived, we assume we can immediately implement the minimum clearance time method on the first slot of frame $k+1$, so we have at most one time slot to solve a very complicated minimum clearance time problem! The next exercise allows more time for computing the solution.

Exercise 2. *(Low complexity IMET) Fix T as an integer frame size. Compute the size of T for which no packets are dropped under the leaky bucket assumptions of Theorem 2, considering the following variation that uses 3 phases. Assume $k > 2$: (i) Ignore new arrivals during frame k. Call these $\boldsymbol{Q}[k]$. (ii) During frame k, use the T-slot time duration of this frame to perform the following operations and computations: Drop packets in matrix $\boldsymbol{Q}[k-1]$ to get a new matrix $\tilde{\boldsymbol{Q}}[k-1]$ that can be cleared within time T. Compute the min clearance time for backlog $\tilde{\boldsymbol{Q}}[k-1]$. (iii) During frame k, implement the min clearance time solution for the backlog $\tilde{\boldsymbol{Q}}[k-2]$.*

Compute the size of T for which no packets are dropped under the leaky bucket assumptions of Theorem 2. Show that worst-case delay is at most $3T$.

The methods of the above exercises can be combined to allow variable sized frames, max frame sizes T_{max}, and computation time intervals that are different from the frame sizes. This IMET technique of having back-to-back intervals over which backlog is cleared in minimum time can be used for other networks, as shown in [5].

II. Max Weight Scheduling

A different type of *max-weight* scheduling can be used [6]. The following model is from [2] where lower complexity *maximal scheduling* is considered. Here we derive a special case of max-weight scheduling from the drift-plus-penalty framework.

A. Model

Suppose we have L links that are activated in a time slotted system $t \in \{0, 1, 2, ...\}$. Let $\boldsymbol{b}(t) = (b_1(t), \ldots, b_L(t))$ be the (binary) activation vector on slot t, and assume this is chosen from some fixed set $\mathcal{B}$ of vector options:

$$\mathcal{B} = \{\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, ..., \boldsymbol{b}^{(M)}\}$$

where M is the (finite) number of options. For the special case of an $N \times N$ packet switch, we can write the N^2 matrix components as a vector so that there are $L = N^2$ links, and there are $M = N!$ options (each option being a permutation matrix). The set $\mathcal{B}$ can also represent the options available in a wireless network due to interference constraints.

B. Problem

Suppose we select each option $\boldsymbol{b}^{(m)} \in \mathcal{B}$ *randomly* with probability $p^{(m)}$. We want to select probabilities that ensure the expected transmission rate over each link $l \in \{1, \ldots, L\}$ is at least λ_l, where λ_l are given target rates for each link. Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_L)$. We want to solve the *linear program* of finding decision variables $p^{(m)}$ to satisfy:

$$\sum_{m=1}^{M} p^{(m)} \boldsymbol{b}^{(m)} \geq \boldsymbol{\lambda}$$
$$\sum_{m=1}^{M} p^{(m)} = 1$$
$$p^{(m)} \geq 0 \quad \forall m \in \{1, \ldots, M\}$$

Notice that there is no objective function to minimize. We can view the objective function as the all-zero function. The above reduces to

$$\sum_{m=1}^{M} p^{(m)} b_l^{(m)} \geq \lambda_l \quad \forall l \in \{1, \ldots, L\} \tag{5}$$
$$\sum_{m=1}^{M} p^{(m)} = 1 \tag{6}$$
$$p^{(m)} \geq 0 \quad \forall m \in \{1, \ldots, M\} \tag{7}$$

C. Drift-plus-penalty (DPP) implementation

Recall the drift-plus-penalty (DPP) rule for solving convex programs (as specified in the network optimization notes):

http://ee.usc.edu/stochastic-nets/docs/network-optimization-notes.pdf

For a DPP implementation we have L virtual queues, one for each link:

$$Q_l(t+1) = \max\left[Q_l(t) + \lambda_l - \sum_{m=1}^{M} p^{(m)}(t)b_l^{(m)}, 0\right] \quad \forall l \in \{1, \ldots, L\}$$

Since the objective function is zero, the decision every slot t is to choose probabilities $(p^{(m)}(t))$ that satisfy (6)-(7) to minimize:

$$V \cdot 0 + \sum_{l=1}^{L} Q_l(t)\left[\lambda_l - \sum_{m=1}^{M} p^{(m)}(t)b_l^{(m)}\right]$$

This is equivalent to the following *max-weight rule* of choosing (nonnegative) probabilities $(p^{(m)}(t))$ that sum to 1 to maximize:

$$\sum_{m=1}^{M} p^{(m)} \underbrace{\left[\sum_{l=1}^{L} Q_l(t)b_l^{(m)}\right]}_{\boldsymbol{Q}(t)^T\boldsymbol{b}^{(m)}}$$

The answer is to place the full probability 1 on the option m that maximizes $\boldsymbol{Q}(t)^T\boldsymbol{b}^{(m)}$, which is the inner product of the current queue backlog matrix $\boldsymbol{Q}(t) = (Q_1(t), \ldots, Q_L(t))$ and the transmission vector $\boldsymbol{b}^{(m)} = (b_1^{(m)}, \ldots, b_L^{(m)})$ for option m. That is, we select the option m that maximizes this inner product. This is called the *max-weight rule* [6]. Since we place full probability on one option, we can call $\boldsymbol{b}(t) = (b_1(t), \ldots, b_L(t))$ the transmission option in $\mathcal{B}$ that is chosen on slot t, so that $\boldsymbol{b}(t) = \sum_{m=1}^{M} p^{(m)}(t)b_l^{(m)}$ and the virtual queue equation reduces to:

$$Q_l(t+1) = \max\left[Q_l(t) + \lambda_l - b_l(t), 0\right] \quad \forall l \in \{1, \ldots, L\}$$

It can be shown that this *max-weight rule* stabilizes the network whenever possible, even when the virtual queues are replaced by *actual queues* with random arrivals $A_l(t)$, *even if the arrival rates* $\lambda_l = \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} A_l(t)$ *are unknown* [6]:

$$Q_l(t+1) = \max\left[Q_l(t) + A_l(t) - b_l(t), 0\right] \quad \forall l \in \{1, \ldots, L\}$$

where $\boldsymbol{b}(t) = (b_1(t), \ldots, b_L(t))$ is the vector $\boldsymbol{b}^{(m)} \in \mathcal{B}$ that maximizes the inner product $\boldsymbol{Q}(t)^T\boldsymbol{b}^{(m)}$ (breaking ties arbitrarily).

Notice that for the special case of $N \times N$ packet switches, the *max-weight rule* uses *max weight matches* rather than the *max size matches* of IMET. A much simpler *maximal scheduling* rule is used in [2] to support all traffic *within a factor of 2* of the network capacity region, and to produce very low delay.

Exercise 3. *Consider the same problem but now assume there is an energy cost $\theta^{(m)}$ for using mode $m \in \{1, \ldots, M\}$ on a slot. Now the linear program has an objective function to minimize! Reformulate the problem* (5)-(7) *to support the rate vector $\boldsymbol{\lambda}$ while minimizing average energy cost. What does the DPP algorithm (with parameter $V > 0$) do in this case? Show that if $\theta^{(m)} = 0$ for all m then the algorithm reduces to the max-weight rule specified above.*

REFERENCES

[1] T. Weller and B. Hajek. Scheduling nonuniform traffic in a packet switching system with small propagation delay. *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 813-823, Dec. 1997.

[2] M. J. Neely. Delay analysis for maximal scheduling with flow control in wireless networks with bursty traffic. *IEEE Transactions on Networking*, vol. 17, no. 4, pp. 1146-1159, August 2009.

[3] C-S Chang, W-J Chen, and H-Y Huang. Birkhoff-von neumann input buffered crossbar switches. *Proc. IEEE INFOCOM*, 2000.

[4] M. J. Neely, E. Modiano, and Y.-S. Cheng. Logarithmic delay for $n \times n$ packet switches under the crossbar constraint. *IEEE Transactions on Networking*, vol. 15, no. 3, pp. 657-668, June 2007.

[5] M. J. Neely, J. Sun, and E. Modiano. Delay and complexity tradeoffs for dynamic routing and power allocation in a wireless network. *Proc. of Allerton Conf. on Communication, Control, and Computing*, Oct. 2002.

[6] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936-1948, Dec. 1992.