Joint Optimization of Scheduling and Congestion Control in Communication Networks

Matthew Andrews
Bell Laboratories, Lucent Technologies
Murray Hill, NJ 07974, USA
Email: andrews at research.bell-labs.com

***Abstract*— In this paper we study congestion control and scheduling in communication networks. In contrast to standard protocol design where there is minimal communication between the scheduling and the congestion control, we argue that there are a number of benefits to jointly optimizing these algorithms, especially in wireless networks. The first benefit of the coordination is that we are able to do effective buffer sizing, even when channel rates are variable. The second benefit is that we can prevent conflict situations where the congestion control and the scheduler both try to assign bandwidth to the flows. The third benefit is that coordination allows us to prove theoretical utility maximization results that are not affected by possible oscillations.**

In the first part of the paper we discuss in detail why joint optimization of scheduling and congestion control can be beneficial. We then describe algorithms from the literature that accomplish this coordination, first in the context of stationary channel rates, second in the context of adversarially defined channel rates and traffic patterns.

I. Introduction

One principle of current internet design is that congestion control is largely separated from packet scheduling. Packet schedulers that are internal to the network make local decisions about which packet to forward, without explicit regard to the congestion control algorithm. In many cases extremely simple scheduling algorithms such as FIFO are used and the standard TCP congestion control algorithm makes its decisions at the source using only minimal information from the network such as packet loss information and round-trip time delay.

In this paper we argue than in many situations, we would benefit greatly by having a tighter interaction between the schedulers in the network and the congestion control algorithms. Through most of the paper we focus on wireless networks. We do this for two reasons. First, of the three arguments that we present as to why scheduling and congestion control should be combined, the first and second reasons only apply to wireless networks. Second, almost all current internet hosts run the TCP algorithm and all routers in the internet run scheduling algorithms that do not directly interact with TCP. Therefore, it would be an essentially impossible task to change the basic structure of congestion control and scheduling in the wide-area internet in such a way that would allow them to work jointly. However, in the case of wireless networks where we may have a group of wireless nodes forming a well-defined, self-contained wireless ad-hoc network, we believe that there are possibilities to change the congestion control and scheduling in the network so that all the algorithms work in concert in order to get the maximum performance.

We now outline three reasons why we believe there are significant benefits to be obtained by performing congestion control jointly with scheduling.

- In a wireless network, if we do not coordinate the congestion control and the scheduling then buffer sizing becomes problematic. The standard rule for sizing buffers states that in order to keep a link fully utilized when the TCP congestion control is used, the buffer at the head of the link should equal the bandwidth-delay product (BDP) for the link. (By "delay" we mean the end-to-end propagation delay for the TCP flows that use the link.) However, in a wireless network the bandwidth can be extremely variable, which means that it is difficult to size the buffer correctly. As we discuss in Section II-A, the normal solution is to size the buffer for the high link rates. However, this can cause delays to be unacceptable large when the bandwidth is low.
- In many wireless networks the link schedulers and TCP work against each other. This is because the design of TCP assumes that link scheduling is done in a flow-oblivious manner (for example using FIFO schedulers). In this setting one of the goals of the congestion control algorithm is to make sure that bandwidth is shared in a fair manner. However, often in a wireless network it is the wireless link scheduler that determines how bandwidth is shared. Hence there are two entities that are both trying to allocated bandwidth for the flows. As we discuss in Section II-B, this conflict can lead to excessively large queues.
- There are benefits to putting the performance of the congestion control algorithm on a sound theoretical footing. In analysis first done by Kelly, Maulloo and Tan in [8] and later continued by many other authors (e.g. [10], [11], [6], [15], [14], [9]), it has been shown that many variants of TCP can be viewed as approximations of primal-dual algorithms that solve an underlying optimization problem. However, in most cases this analysis abstracts away some of the details of the scheduling problem. In Section II-C we describe some results of [3] which show that when the scheduling dynamics are fully modeled many of the primal-dual algorithms presented in the literature

can oscillate in a suboptimal state. We believe that the best method for preventing such oscillations and ensuring system optimality is to jointly coordinate the scheduling and congestion control.

The remainder of the paper is organized as follows. In Section II we elaborate on the benefits of combined scheduling and congestion control in more detail. In Section III we describe some possible methods for carrying out this joint optimization. We first briefly describe a scheme of Stolyar [12] that is appropriate when the channel conditions can be modeled by some stationary stochastic process. We then present methods for the solving the problem under worst-case adversarial channel conditions and traffic patterns.

II. Benefits of Combined Scheduling and Congestion Control

A. Buffer sizing

As indicated in the Introduction, the standard rule for buffer sizing under the TCP congestion control algorithm is that the buffer size should equal the bandwidth-delay product (BDP). However, in a wireless network the bandwidth on a link can vary dramatically and so the BDP is not well-defined. This leads to a dilemma as to how big the buffer should be. One solution is to size the buffer for a smaller link bandwidth. However, in this case the buffer is not big enough when the link bandwidth is high and so we do not fully utilize the buffer. The alternative, which is often used in commercial networks, is to size the buffer for a larger link bandwidth. The drawback here is that when the link bandwidth the low, we get excessive queueing and so the data experiences too much delay.

As an example of how the delay can change, let us suppose that the channel can vary between 20kbps and 1Mbps. Figures 1-3 show results for downloading the CNN homepage over these two connection speeds. Figures 1 and 2 are scatter plots of packet latencies for the 1Mbps and 20kbps connections respectively. Figure 3 shows the amount of unacknowledged data (i.e. the amount of data in flight) for the 20kbps connection.

We use a 35KB buffer since for a 1Mbps connection that corresponds to about 280ms of buffering. Figure 1 shows that the packet latencies are never more than that amount. However, if the channel is 20kbps, then a 35KB buffer gives 14 seconds of buffering, far more than any reasonable BDP. Note from Figure 2 that the packet delay does get close to 10seconds. Hence TCP is creating a lot of unnecessary buffering. Note that around time 30s, there is more than 25KB still in flight. An examination of the sequence numbers (not shown) reveals that at that time, only 50KB of data had been transmitted. Hence at that point, half of the transmitted data is still in flight.

We note that excessive buffering is bad for a number of reasons. It causes extra delay for other flows that share the same queues. (In particular, other TCP connections have trouble getting synchronized which leads to unnecessary TCP timeouts.) Moreover, in takes a long time to recover from a

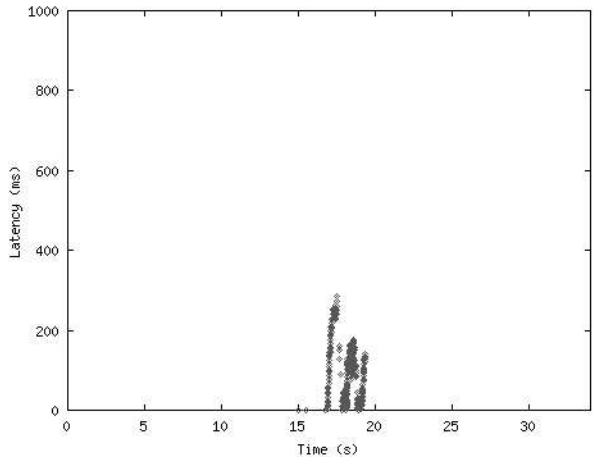

Fig. 1. Packet latencies for a 1Mbps connection.

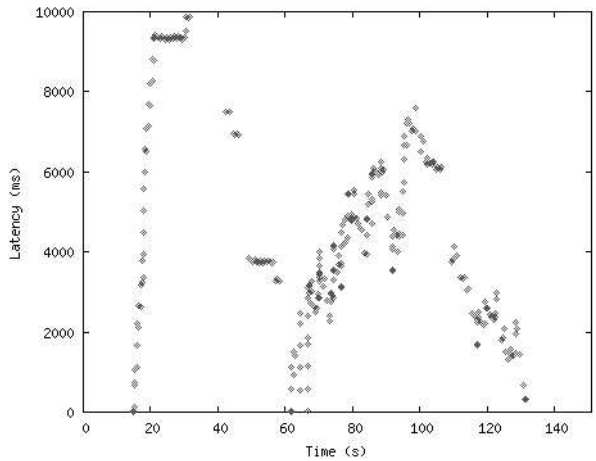

Fig. 2. Packet latencies for a 20kbps connection.

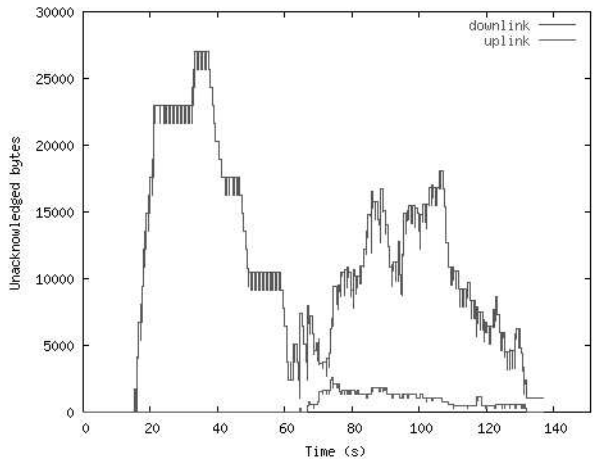

Fig. 3. Unacknowledged data for a 20kbps connection. Most data is sent in the downlink direction which is represented by the top curve.

packet loss because the retransmitted packet has to wait for all the outstanding data to be served before it can be received. Even worse, in a wireless network with changing connectivity, we may reach a situation where the inflight data cannot reach the destination. Hence all that data will be lost and will need retransmitting.

Large queueing delays of the type described above have been observed in commercial networks. In [5] Chakravorty and Pratt report on measurements in a GPRS network in which they observed 30 seconds of queueing delay. This is far above what is typically expected in data networks.

As we shall see in Section III, by combining congestion control and scheduling we are able to keep queue sizes low even for variable rate links. Specifically, by employing a back-pressure mechanism we can directly signal to the congestion control to reduce the sending rate if the link schedulers cannot serve all the data being sent.

B. Preventing conflicts between the scheduler and the congestion control

Another issue in wireless networks is that the congestion control and the link scheduler can sometimes work against each other. In particular, congestion control algorithms such as TCP implicitly make the assumption that there is not a fixed bandwidth assigned to each flow. That is, if the congestion control increases the amount of data being sent then the flow may get more service. This is the case for example with the FIFO scheduling protocol. In that case whenever the source is not seeing packet drops it makes sense to increase the sending rate since there is a potential to gain more bandwidth. Theoretical analyses of TCP show that for a fixed bandwidth link it leads to a fair allocation among flows. In contrast, in a wireless network we have to contend with time-varying and user-dependent channel conditions. Hence in order to obtain a fair allocation among user flows we need a link level scheduler that takes channel conditions into account.

For example, consider a single transmitter transmitting to a set of mobile users in a wireless system such as EV-DO [4]. In this system, each mobile user k tells the transmitter its current instantaneous channel rate $r_k(t)$. Let $R_k(t)$ be a current measure of the average throughput of user k. The aim of the widely used Proportional Fair scheduling algorithm [13], [7] is to try and maximize $\sum_k \log R_k(t)$. This is achieved by in each time step serving the user with the maximum value of $r_k(t)/R_k(t)$. (In other words Proportional Fair tries to serve a user with a good instantaneous channel and low average throughput).

Note that if a scheduling algorithm such as Proportional Fair is used, it is the *scheduler* that determines how much service each flow gets. It is not the job of the congestion control to try and increase the sending rate to try and get more bandwidth. What typically happens in wireless networks therefore is that the congestion control keeps increasing the sending rate but the scheduler does not grant any more bandwidth. The end result is that we get large amounts of queueing which are not desirable for the reasons outlined earlier. A much better solution is for the congestion control to interact with the scheduler to try and discover how much bandwidth is available, and then try and match the sending rate to that available bandwidth. This can be done by a backpressure mechanism that reacts whenever a queue becomes large.

C. Prevention of oscillations

Starting with the seminal work of Kelly, Maulloo and Tan [8] there has recently been a great deal of work aimed at understanding the behavior of TCP as the solution to an underlying optimization problem. Specifically, consider a set of sessions (flows) each of which passes through a set of servers. Let P_k be the path of session k. If server Q lies on this path then we say that $Q \in P_k$. Let x_k be the injection rate into session k and let μ_Q be the capacity of server Q. Suppose that we wish to solve the problem,

$$\begin{aligned} &\max \sum_k \log x_k \\ \text{subject to:} \quad & \\ &\sum_{k:Q\in P_k} x_k \le \mu_Q \qquad \forall Q. \end{aligned} \tag{1}$$

It is shown in [8] that this problem can be solved using a primal-dual type algorithm. In particular, let $x_k(t)$ be the value of x_k at time t and suppose that the injection rates are updated according to:

$$\begin{aligned} \frac{dx_k(t)}{dt} &= \beta(1 - x_k(t) \sum_{Q\in P_k} (\gamma_Q(t) - \mu_Q + \varepsilon)^+/\varepsilon^2), \\ \gamma_Q(t) &= \sum_{k:Q\in P_k} x_k(t) \end{aligned} \tag{2}$$

for some parameters β, ε. For small values of ε the results of [8] state that this algorithm converges to the solution of the optimization problem (1). This algorithm can be viewed as an approximation of TCP due to the fact that the first term in (2) is an additive increase term and the second term is a multiplicative decrease term based on congestion at server Q. Hence the algorithm may be viewed as an additive-increase multiplicative-decrease algorithm in the spirit of TCP. A large body of subsequent work (e.g. [10], [11], [6], [15], [14], [9]), have extended these ideas to a large variety of contexts, including the performance of the TCP Vegas algorithm, the behavior of power control in wireless networks and the performance of internet routing algorithms.

However, there is one feature of the above framework that means that it is not applicable in all situations. Note that the congestion term $\gamma_Q(t) = \sum_{k:Q\in P_k} x_k(t)$ at server Q is a function of the *injection rates* of the sessions passing through the server. In reality however, the congestion is determined by the *arrival rates* of the sessions passing through the server. In some situations the arrival rate may be different from the injection rate due to the queueing dynamics in the network. Indeed, if the injection rate is suddenly increased it will take some time for this increase to pass through the queues in order to reach downstream servers. More importantly, it may be the

case that the flow injection rates for flows passing through a server Q overload that server. However, due to congestion at an upstream server Q', it is possible that the arrival rates at server Q are less than the session injection rates and so we do not get a buildup of data at server Q.[1]

Since the actual queueing dynamics can affect the performance of a system, in [3] we studied how dramatic this effect can be. In particular, we were able to show that if congestion at a server is modeled as a function of the arrival rates then the optimality of algorithms such as (2) no longer hold. To be more precise, suppose that we treat data as a fluid and at each server we serve the fluid according to some scheduling rule (e.g. FIFO). If too much fluid arrives it starts to queue up and create delays.

We consider congestion control mechanisms of the following form. We say that a session is *blocked* if at least one server on its path has a nonzero amount of fluid queued up. We say that a session *happy* if all the servers on its path have zero fluid queued up. Associated with each session k we have two functions $f_k(\alpha, t)$ and $g_k(t)$.

- $f_k(\alpha, t)$ is the injection rate of session k at time t assuming that at time 0 session k becomes blocked and has injection rate α. As an example $f_k(\alpha, t)$ could represent a multiplicative decrease, e.g. $f_k(\alpha, t) = \alpha e^{-\delta t}$ for some parameter δ.
- $g_k(0, t)$ is the injection rate of session k at time t assuming that at time 0 session k becomes happy and has injection rate 0. As an example $g_k(t)$ could represent an additive increase, e.g. $g_k(t) = \delta t$ for some parameter δ.

The main result of [3] is:

Theorem 2.1: For a wide class of functions $f_k(\cdot, \cdot)$ and $g_k(\cdot)$, there exists a network of servers and a set of sessions such that the system *oscillates* in states that are suboptimal with respect to the optimization problem (1). Moreover, the injection rates are always such that no server is ever overloaded. Hence it is the queueing dynamics that cause the oscillations, not the transient behavior of the injection rates.

Most of the servers in the example of [3] serve data according to FIFO. However, there are a small number of servers that classify sessions into two types and give data from one session type priority over the other. An interesting open problem is to remove these priorities and derive an oscillating example in which all servers are FIFO.

For the full details of the construction, the reader is referred to [3]. However, we give a brief outline here. The example consists of multiple rows of servers. Each row has a single horizontal session H that passes through all the servers in the row. Some of the servers in the row also have a single-hop session S that passes through one server only. We configure the system such that the maximum injection rate for each session gives the optimal solution to the problem (1). However, we can create an initial configuration such that the system persistently oscillates in suboptimal states.

To show how this can happen, suppose that a row starts with some fluid in its leftmost server. We construct the system so that this fluid passes through all the servers in a row. Specifically, for each server that has a single-hop session S, there is a time at which all of the session H fluid resides in that server. (See Figure 4.) At that time the session S is blocked and so its injection rate decreases to zero. This is where the suboptimality arises in that there is always some session S that is not transmitting at its optimal rate.

It remains to say how we get the initial buildup at the beginning of row. We do this by having multiple vertical

Fig. 4. One row of the construction of [3].

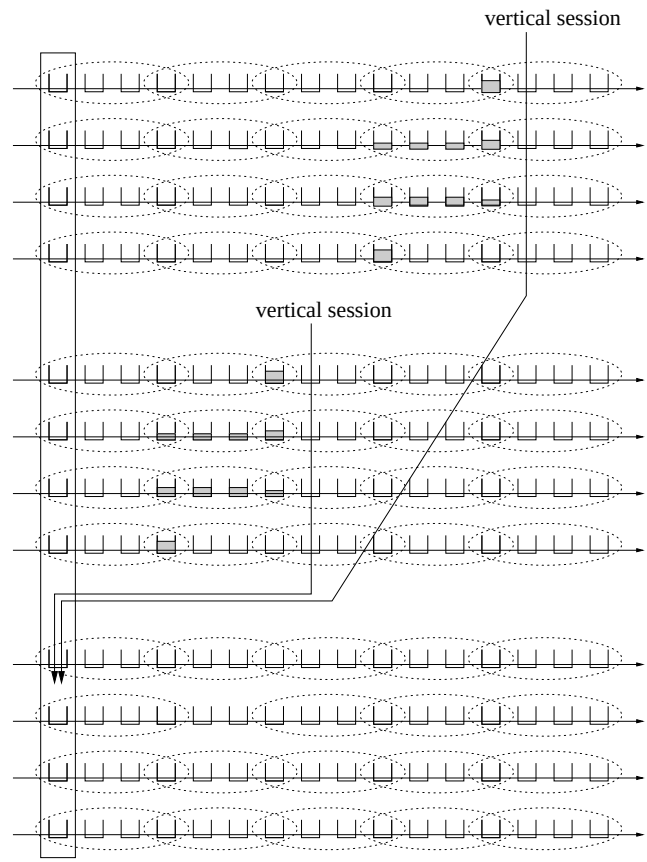

Fig. 5. The first server of a row is replenished by vertical sessions that pass through multiple other rows.

[1]We remark however, that there are some situations where the injection-rate based model is appropriate. For example, suppose that Explicit Congestion Notification (ECN) is used and a congested server starts to "mark" data before there is any queue buildup. In this case the injection rate into a session will be the same as the arrival rate for all the servers on the session path. However, we remark that there are two drawbacks to this approach. First, we declare congestion before the server is fully utilized and so there is a wasting of bandwidth. Second, it might be the case that even though in the limit we can make sure that no server is overloaded, it may still be the case that during the transient behavior, we get buildups in the queues that can affect the arrival rates.

sessions that pass through servers in multiple rows before finally passing through the first server of some row. The system is synchronized so that as the vertical session passes through its initial servers, some of its fluid gets delayed. Due to the fact that it gets delayed at multiple servers in different rows, a large amount of fluid from the vertical sessions is built up. Eventually this fluid is released at a high rate into the first server of some row. We make sure that multiple vertical sessions all release data into the same first server on a row at the same time. This causes the initial buildup of fluid for the row which can then (as described above), pass along the length of the row.

By using servers in a toroidal structure, we are able to repeat the above process indefinitely. This shows that by considering the queueing dynamics, we never converge to the optimal solution of problem (1).

III. Methods for Combined Scheduling and Congestion Control

In this section we briefly outline some methods from the recent literature that allow us to combine scheduling and admission control in such a way as to alleviate the problems described in the previous sections. At a high level, the aim is to decide which data should be injected and how the data should be scheduled so that we obtain high "system utility". The main idea of all of these algorithms is that there is a direct communication between the scheduler and the congestion control as to when a flow is using too much bandwidth. In this case, as we show below, it is possible to obtain algorithms with theoretical performance guarantees. In particular, it allows the congestion control algorithm to stop sending data before there is an excessive queue buildup and hence we get smaller packet delays. One way in which this congestion communication can occur is to utilize backpressure between neighboring nodes so that if there is congestion at some node in the network there is backpressure all the way back to the source and so this prevents more data from being injected.

We begin this section by describing an elegant solution due to to Stolyar [12] for the case where channel conditions are modeled by a stationary stochastic process and the congestion control tries to optimize some utility function of the data served. We then describe some partial solutions for the case where either the traffic or the channel conditions are generated by an adversary.

A. The model

As in Section II we consider a fixed set of sessions in the network. Each session k follows a fixed path P_k. We suppose that time is divided into slots and at each time slot, each node can only send data to one neighbor. If node i decides to send data to node j at time t then the amount of data that can be sent is denoted $r_{ij}(t)$. At each node i on the path P_k we maintain a queue Q_i^k that consists of data for session k.[2] (In an abuse of notation we also let Q_i^k be the size of this queue.) We let n_i^k be the next node after node i on the path P_k. The job of the scheduling algorithm is to decide which data should be transmitted by node i at each time step. We shall consider two situations, the static graph setting in which $r_{ij}(t) = 1$ for all i, j, t and the dynamic graph setting in which $r_{ij}(t)$ can be arbitrary. For dynamic graphs there are two further options. In the first we can assume that there is some stationary Markov chain for the system such that if $m(t)$ is the state of the chain at time t, the $r_{ij}(t)$ is completely determined by $m(t)$. The second option is to assume that $r_{ij}(t)$ is given by an adversary.

The job of the congestion control algorithm is to decide how much data is introduced into the system at each time step. More formally let Q_{begin}^k be the first queue on the path of session k. At each time slot t the congestion control algorithm determines how much session k data is injected into the system. We denote this quantity by $b_k(t)$. This data then gets placed into the queue Q_{begin}^k.

The overall goal of the combined scheduling and congestion control is to maximize some system utility subject to the condition that *all queues remain bounded*. For each session k we assume that we have a utility function $U_k(\cdot)$. (The function $U_k(x) = \log(x)$ is a common choice.) We also have a measure $x_k(t)$ of the amount of service that session k has received at time t. The simplest choice for this measure is a smoothed average which is updated by $x_k(t) = (1-\beta)x_k(t-1)+\beta b_k(t)$. The optimization problem that we wish to solve then becomes:

$$\begin{aligned} &\text{maximize} && \sum_k U_k(x_k) \\ &\text{subject to} \\ & && \text{all queues remain stable} \end{aligned} \tag{3}$$

B. Stationary model

We begin by considering a model in which the channel conditions are governed by a stationary Markov Chain. For simplicity we also assume that at each time step the amount of data that is injected into each session is either b or 0, for some parameter b. In this case, the optimization problem (3) was solved by Stolyar in [12] using a *Greedy Primal Dual* algorithm. In our context, the algorithm operates as follows. For each node i and for each session k we have an urgency weight defined by $g_i^k(t) = (Q_i^k(t) - Q_{n_i^k}^k(t))r_{i,n_i^k}(t)$. Note that the urgency is a function of the difference between neighboring queue heights weighted by the channel conditions. Suppose that at time t, session k^* is the session with the maximum urgency at node i. In this case node i serves data from session k^* at time t. The amount of data that is served is $\min\{r_{i,n_i^{k^*}}(t), Q_i^{k^*}(t)\}$.

The above rule describes the scheduling part of the algorithm. The congestion control algorithm is defined as follows. For the parameter β used in the smoothing calculation, we inject data of size b into session k if $bU_k'(t) - \beta Q_{begin}^k(t)$ is positive and we do not inject data otherwise. It is shown in [12] that for small values of β, this algorithm solves the problem (3). (We remark that the paper [12] actually solves

[2] If however the session routes are such that all the paths to a single destination form a tree then at node i we only require a queue Q_i^d for each destination node d rather than a queue Q_i^k for each session k.

a far wider class of problems. For example, we could impose constraints on the set of nodes that are allowed to transmit simultaneously.)

We now briefly describe why in this context the algorithm solves all of the potential problems described earlier. First, note that the scheduling algorithm and the congestion control work in concert to determine the bandwidth for each session and to ensure that we do not get excessive delays. Whenever there is congestion at a server we get a small queue buildup. By the scheduling rule this causes the queues all the way back to the source to also build up. This in turn means that according to the congestion control rule we are less likely to inject new data. Hence we do not get large queues due to persistent injections into the session. Moreover, we do not get the oscillation problems that occur in the example of Section II-C because the scheduling algorithm tends to even out the queues along the row session H. This means that the vertical session data suffers less delay which implies that it cannot create a queue buildup at the first server of a row.

C. Adversarial model

In the remainder of the paper we briefly mention some results from [2] that looked at worst-case adversarially-generated channels and traffic patterns. This paper is divided into two sections. The first looks at a scenario consisting of a single wireless transmitter and multiple mobile receivers. In this case the congestion control algorithm is extremely simple. It only has to at the transmitter whatever data is served. The main problem is to generate a schedule such that the total utility is maximized. As mentioned in Section II-B, in the case of stationary channels and logarithmic utility functions the way to maximize utility is to always schedule the user that maximizes $r_k(t)/R_k(t)$ where $r_k(t)$ is the current instantaneous channel rate of user k and $R_k(t)$ is user k's average throughput. In contrast, it is shown in [2] that in the case of adversarial channels, there is no online scheduling algorithm that gives us the maximum utility possible with an offline scheduling algorithm. The best we can hope for in an n-user system is an algorithm that comes with an additive error of $\Omega(n \log n)$. Moreover, simply serving a random user will achieve that bound. Hence there is not in general any benefit to knowledge of the channel conditions in the adversarial context.

The second section of [2] looks the problem of utility maximization in a static graph where the traffic pattern is governed by an adversary. In this case, we define utility differently from before. We assume that at each time step the adversary injects packets into the sessions and specifies a utility for each packet. The problem is to maximize the utility of packets that are routed. The first result is that if we can solve a linear program that determines the maximum fractional utility of packets that can be routed without overloading any link, then we can assign deadlines to the packets such that if the Earliest-Deadline-First scheduling algorithm is applied that the maximum utility is achieved. Moreover, a distributed algorithm for solving this linear program was presented that is a primal-dual algorithm that operates by exchanging congestion information between the sources and the links.

IV. Discussion

In this paper we have presented three benefits for combining scheduling and congestion control in communication networks, with a particular emphasis on wireless networks. We then surveyed some results from the literature that describe how this joint optimization can be carried out. A number of open problems remain, in particular in relation to the adversarial model. First, is there any congestion control algorithm for adversarial traffic patterns that does not involve solving a linear program as in [2]? In particular, how well does the Greedy Primal-Dual algorithm of [12] work in an adversarial setting. It is known [1] that in static networks in which all the injected packets can be scheduled, then an algorithm similar to Greedy Primal-Dual can guarantee queue stability. It would be interesting to know if this result can be extended to the case of dynamic graphs in which not all packets can be routed and we wish to maximize the total utility of packets served.

References

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Adaptive packet routing for bursty adversarial traffic. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 359 – 368, Dallas, TX, May 1998.

[2] M. Andrews. Maximizing profit in overloaded networks. In *Proceedings of IEEE INFOCOM '05*, Miami, FL, March 2005.

[3] M. Andrews and A. Slivkins. Oscillations with TCP-like flow control in networks of queues. In *Proceedings of IEEE INFOCOM '06*, 2006.

[4] P. Bender, P. Black, M. Grob, R. Padovani, and N. Sindhushayana A. Viterbi. CDMA/HDR: A bandwidth efficient high speed data service for nomadic users. *IEEE Communications Magazine*, July 2000.

[5] R. Chakravorty and I. Pratt. WWW Performance over GPRS. In *Proceedings of IEEE MWCN 2002*, August 2002.

[6] M. Chiang. Balancing transport and physical layers in wireless multihop networks: Jointly optimal congestion control and power control. *IEEE Journal on Selected Areas in Communications*, 23(1):104–116, 2005.

[7] A. Jalali, R. Padovani, and R. Pankaj. Data throughput of CDMA-HDR a high efficiency-high data rate personal communication wireless system. In *Proceedings of the IEEE Semiannual Vehicular Technology Conference, VTC2000-Spring*, Tokyo, Japan, May 2000.

[8] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operations Research Society*, 49(3):237–252, 1998.

[9] X. Lin and N. Shroff. The impact of imperfect scheduling on cross-layer rate control in multihop wireless networks. In *Proceedings of IEEE INFOCOM '05*, 2005.

[10] S. Low and D. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861 – 874, 1999.

[11] S. Low, L. Peterson, and L. Wang. Understanding Vegas: a duality model. *Journal of the ACM*, 49(2):207–235, 2002.

[12] A. Stolyar. Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm. *Queueing Systems*, 50(4):401–457, 2005.

[13] D. Tse. Multiuser diversity in wireless networks. http://www.eecs.berkeley.edu/~dtse/stanford416.ps.

[14] T. Voice. A global stability result for primal-dual congestion control algorithms with routing. *Computer Communication Review*, 34(3):35–41, 2004.

[15] L. Xiao, M. Johansson, and S. Boyd. Simultaneous routing and resource allocation in wireless networks. *IEEE Transactions on Communications*, 52(7):1136–1144, 2004.