Routing and Scheduling for Energy and Delay Minimization in the Powerdown Model

Matthew Andrews
Bell Labs
Murray Hill, NJ
andrews@research.bell-labs.com

Antonio Fernández Anta
Institute IMDEA Networks
Madrid, Spain
antonio.fernandez@imdea.org

Lisa Zhang
Bell Labs
Murray Hill, NJ
ylz@research.bell-labs.com

Wenbo Zhao
UCSD
La Jolla, CA
w3zhao@ucsd.edu

Abstract

Energy conservation is drawing increasing attention in data networking. As networks are designed for peak traffic, network elements typically operate at full speed and consume maximum power even when carrying low traffic. One school of thought believes that a dominant amount of power saving comes from turning off network elements. The difficulty is that transitioning between the active and sleeping modes consumes considerable energy and time. This results in an obvious trade-off between saving energy and provisioning performance guarantees such as end-to-end delays.

We study the following routing and scheduling problem in a network in which each network element either operates in the full-rate active mode or the zero-rate sleeping mode. For a given network and traffic matrix, routing determines the path that each traffic stream traverses. For frame-based periodic scheduling, a schedule determines the active period per element within each frame and prioritizes packets within each active period. For a line topology, we present a schedule with close-to-minimum delay for a minimum active period per element. For an arbitrary topology, we partition the network into a collection of lines and utilize the near-optimal schedule along each line. Additional delay is incurred only when a path switches from one line to another. By minimizing the number of switchings via routing, we show a logarithmic approximation for both power consumption and end-to-end delays.

If routing is given as input, we present two schedules one of which has active period proportional to the traffic load per network element, and the other has active period proportional to the maximum load over all elements. The end-to-end delay of the latter is much improved compared to the delay for the former. This demonstrates the trade-off between power and delay.

Finally, we provide simulation results to validate our algorithmic approaches.

Keywords: Energy, power, routing, scheduling, networks, powerdown model.

1 Introduction

As computing and communication needs continue to grow at a formidable rate, a parallel growth in power consumption is not sustainable, due to exponentially increasing energy costs, society awareness of carbon emissions and carbon footprint, and government energy regulations [1, 15, 16, 18, 24, 37, 50]. This implies that the business-as-usual mode of the computing and communication industry regarding power consumption is no longer an option, and the whole sector is hence making a big effort to increase power efficiency. Different techniques have been used to reduce consumption, like deploying energy-efficient technologies (such as replacing copper with optics and disks with flash), matching power consumption to work by turning off and dialing down, piggy backing energy events via shared caches, etc. [45].

Regarding specifically the communications sector, the telecommunication infrastructure accounts for about 31% of the total carbon emissions of global ICT emissions [36]. Such a consumption partially results from the fact that most networks are engineered to handle peak traffic. Network elements tend to operate at full speed and consume maximum power, while typical traffic is only a small fraction of the maximum throughput. It is estimated that, if the power consumption of each network element is adapted to be proportional to its traffic load, up to 80% of the energy in the access layer and up to 40% in the network core can be saved [17]. It is even claimed that by "rethinking the way telecom networks are designed in terms of low energy processing" it is possible to cut energy consumption by a factor of $1,000$ [24].

Powering down and *speed scaling* are promising mechanisms for dynamically adapting the power consumption to the actual load at each network element (routers, switches, CPUs, Ethernet links, etc). Powering down saves power by switching off the element when possible, while speed scaling saves by adjusting the operational speed of the element to its load. These two mechanisms have been intensively studied, specially for energy-efficient task scheduling in CPUs [9, 12, 22, 32, 33, 34, 35, 39, 51] and for reducing power consumption at the single network element level [13, 25, 26, 27, 28, 29]. Moreover, they are also available in off-the-shelf processors [31], and are being introduced in the new communication standards [23, 30, 44].

In this paper we focus on the powering down model, and we study speed scaling in a separate effort [4, 7]. In this model each network element either operates in the active mode at the full rate or in the sleep mode at the zero rate. We consider provisioning a set of connections each with a desired connection rate in a network, with two potentially conflicting objectives of minimizing the total power consumption by the network elements and the end-to-end delay experienced by the connections. Routing and scheduling are two integral components of the problem. Routing determines which path each connection follows, and scheduling decides the active periods for each network element and prioritizes packets within each active period. If switching between the two modes were instantaneous, then one plausible approach for scheduling would be activating a network element instantaneously at the arrival of each packet, processing at full rate until the queue drains and then switching to the sleep mode instantaneously. This effort tends to favor both power and delay.

Unfortunately, switching between the active and sleep modes may consume considerable time [46, 48]. For instance, under the recently approved IEEE 802.3az Energy Efficient Ethernet standard [30], a link operating at 10 Gbps requires at least 4.48 microseconds to wake up and 2.88 microseconds to go to sleep. Since a 1500 byte packet has a transmission time of 1.2 microseconds, a 14% load of evenly spaced such packets would prevent the link from going to sleep. In fact, it has been observed with real traces [3] that a link with a load as low as 25% may be in low-power sleeping mode just 5% of time. Moreover, two traces

with a similar load of 15% lead to two different percentages of time in sleeping mode (namely, 9 and 14.5%). The rest of the time the link is switching between modes.

If the energy consumed while switching was very small, then the largest drawback would be the delay introduced (with respect to the always active option) at each link due to the waiting for the link to become active. Unfortunately, with current technologies, switching between modes consumes almost full power [46]. This implies that, without coalescing packets, the traffic pattern has a big impact in the potential energy saved.

Ideally, from the perspective of energy, the fraction of the total active time of a network element should be proportional to the total connection rates that the element carries. In addition, if the active proportion is the same, it consumes less power to have long active periods infrequently than to have short active periods frequently, as this saves transition costs. Hence, coalescing is essential to increase energy savings. On the other hand, from the perspective of delay, it is most convenient to operate in the active mode all the time. Barring this option, for the same active proportion it is advantageous to have short active periods frequently than long active periods infrequently, since the long sleep periods that accompany the long active periods contribute to the queuing delay.

1.1 Previous Work

The study of power consumption minimization in processors was started by Yao et al. [51]. They assumed a speed scaling model in which the power to run a processor at a speed s grows polynomially with s as $P(s) = s^\alpha$, for some $\alpha > 1$, and explored the problem of scheduling a set of tasks with the smallest amount of energy. The powerdown model for similar task scheduling problems was later introduced by Irani et al. [32, 34], both in isolation and combined with speed scaling [35]. The survey of Irani and Pruhs [33] revises results on efficient task scheduling with powering down and speed scaling on a single processor.

Not surprisingly, the same techniques used to reduce power consumption in processors, powerdown and speed scaling, are being used to save energy in networks. Gupta and Singh were among the first to identify energy consumption in networks as an important issue [27]. They proposed the use of sleeping modes to reduce power consumption in Ethernet links. In later work [28, 29] they claimed that low power modes are available in most Ethernet interfaces nowadays, but not extensively used to put links into these modes during idle periods, and proposed methods to detect these periods and algorithms to decide when to switch between sleeping and awake modes. Christensen et al. [13] also present the case for power management to reduce power consumption at the edge of the Internet. They propose methods to reduce this consumption in Internet connected PCs, by means of a proxying Ethernet adapter that handles routine networking tasks while the PCs sleeps. In follow up work [25], techniques to reduce power consumption of PCs, Ethernet links, and edge routers were proposed. One of these techniques is to reduce the data transmission rate of Ethernet links when possible, formally proposed as adaptive link rate (ALR) by Nordman and Christensen [43], and studied in [26]. For other kinds of link standards, ADSL2 and ADSL2+ already support a variety of power states and link rates [23].

The recently approved IEEE 802.3az Energy Efficient Ethernet (EEE) standard [30], uses powerdown to save energy at the physical level of an Unshielded Twisted Pair Ethernet link. The energy is saved by switching the link to a low power (sleep) mode when it has no traffic. Different low power modes have been defined for three Ethernet modes, namely, 100BASE-TX, 1000BASE-T, and 10GBASE-T (operating at 100

Mbps, 1Gbps, and 10 Gbps, respectively). Modes have different values for the time T_s to switch from active to sleep mode and the time T_w to switch from sleep to active mode. Interestingly, in 1000BASE-T the two directions of a link have to be in the same state. Reviriego et al. [48, 49] explore the potential energy savings with EEE under Poisson arrivals without coalescing, showing that under low load the savings can be significant. They also observe that savings get drastically reduced as load increases, making advisable grouping packets into bursts. Christensen et al. [14] explore the impact of coalescing, and shows that it can significantly improve energy efficiency.

To our knowledge, integral approaches to globally optimize power consumption at a network level are scarce. One of the papers that studies techniques of energy saving at a network level is due to Nedevschi et al. [42]. In this paper, changing the transmission rate and powering down (parts of) the network equipment are explored as two alternative techniques to globally reduce energy consumption in a network. When using sleep states, much time and power can be used to transit from sleeping to active and vice versa if many such transitions occur. Then, the authors propose that edge routers group packets of the same source-destination pair and transmit them in bursts, in order to reduce the number of transitions and maximize the sleeping time. The authors evaluate their proposals on real topologies and compare the power saved with powerdown and rate scaling under different operational parameters. In general they observe that the use of powerdown provides larger power saving than scaling under low load. Francini and Stiliadis [21] propose an energy model for network elements that combines powerdown and speed scaling. They propose a class of policies to dynamically adapt the operational state of each element, combining powerdown and rate scaling. Their simulations shown that their policies provide better power savings that using only one of the two techniques, if the whole load spectrum is considered. Interestingly, the largest savings occur under low load, where powerdown clearly outperforms speed scaling.

1.2 Model and Results

We are given a network to support a set of connections, each of which needs to transmit packets between a pair of terminal nodes at a specified rate. Connection i is routed along path P_i, where, P_i can be either given as input or be computed as part of the output. We study both situations. If the routes are not part of the input, the routing part of the problem specifies a route P_i for each connection i. The packet arrival is leaky-bucket controled and connection i has a desired rate of ρ_i. The scheduling part of the problem specifies the active period of each link, i.e., when a link operates at the full rate normalized to $R = 1$, and how packets are prioritized within each active period. Transitioning between two modes takes δ units of time, and consumes power at the same rate as the full rate, though it cannot process any packets during the transition.

We focus on frame-based schedules, in which each link is active for a fixed duration within every frame of T steps, $(0, T]$, $(T, 2T], \ldots$. Let $R_e = \sum_{i:e \in P_i} \rho_i$ be the total of rates that go through link e. Let

$$A_e = T \cdot R_e$$

be the minimum length of an active period for link e. We call a schedule *minimal* if the duration of the active period on each link e is exactly A_e. Obviously, for a fixed set of routes and for a fixed T, a minimal schedule is power optimal.

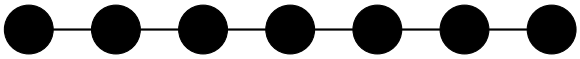

Figure 1: Line topology.

The fact that links are active only once in every frame has clearly an impact on the delay experience by the connections. We assume that the link load is low, and hence when a packet arrives it will most probably have to wait for the next active period. We then assume that T is a lower bound on the delay experienced by any connection. If convenient, we may assume that all packets that arrive in one frame of length T are scheduled in the active period of the next T-frame.

We prove the following results. For simplicity, we assume that transmitting one packet across a link takes one unit of time, or *time step*. Our results below generalize to links with arbitrary link transmission times.

- We begin with a line topology (see Figure 1). Since routing is fixed, we focus on scheduling only. We present a minimal schedule in which the end-to-end delay for each connection i of k_i links is upper bounded by $2T + k_i$. We also present a *quasi-minimal* schedule in which the duration of the active period is $A_{\max} = \max_e A_e$ for all e. For this case we show an end-to-end delay of $T + k_i$ for all i.

  For each fixed T, the energy-delay trade-off between these two schedules is obvious, since the one with smaller delay consumes more energy and vice versa. Within each schedule, we also observe the trade-off since a larger T implies a larger delay bound but fewer active-sleeping transitions, and therefore less power consumption.

- We then show how to apply our techniques for line scheduling to obtain results for networks with arbitrary topology. If routing is not part of the input we first choose routes with the objective of minimizing $\sum_e A_e$ together with the transition cost. This sum can be approximated to within a factor of $\tilde{O}(\log n)$ where n is the size of the network. In addition, we can also guarantee that the routes form a tree. Using caterpillar decomposition we partition the tree into a collection of lines such that each connection path intersects with at most $2 \log n$ lines. Along each line we adopt the minimal schedule. When a packet moves from line to line, a maximum extra delay of T is sufficient. This allows us to prove an $\tilde{O}(\log n)$ guarantee in the end-to-end delay for all connections.

- If routing is already specified, we present a straightforward minimum schedule with a delay bound of $T \cdot k_i$ for each connection i. However, if links are active for longer in sufficiently large frames, then we give an improved delay bound. This again demonstrates a trade-off between power and delay.

The time to compute the desired schedule in each of the above cases is polynomialy bounded. It is implicitly assumed that the set of connections and their rates ρ_i stay the same for a significant amount of time. Hence, we assume that it is practical to compute the schedules since they can be used for an extended period of time.

2 Scheduling over a Line

We begin with scheduling a set of connections over a line. The terminals of each connection are not necessarily the end points of the line. Although the line topology is restricted, the solution is used as a building block for handling arbitrary topologies. For a line, we can choose any frame size T, as long as $T \geq A_e + 2\delta$ for all e. Since $A_e = T \cdot R_e$, this implies

$$T \geq \max_e \frac{2\delta}{1 - R_e}.$$

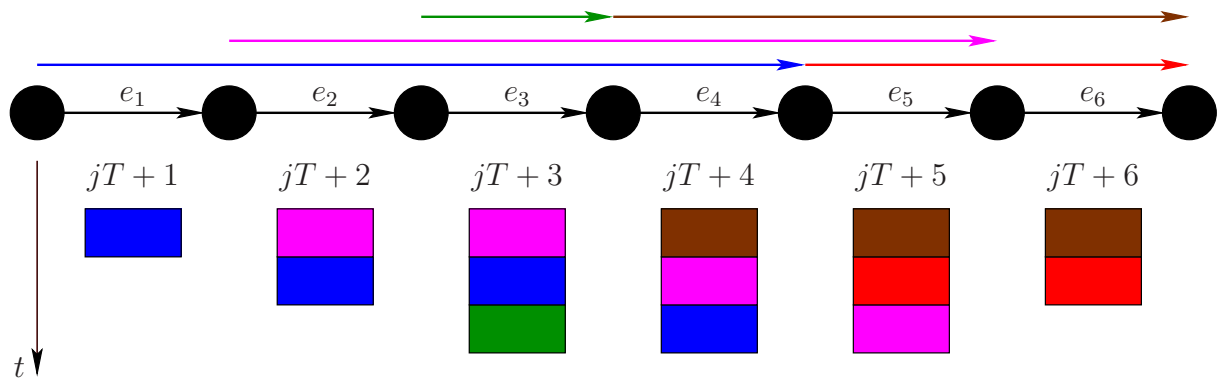

Figure 2: Schedule of minimal energy over a line as described in Lemma 1.

Lemma 1. *For a line topology, after an initial delay of at most T each packet can reach its destination with at most $A_{\max}$ additional queueing delay. The activation period is A_e, which is minimal for each link e.*

Proof. We first define a minimal schedule. Let us label the links along the line as e_1, e_2, etc. For any integer $j \geq 1$, the jth active period for the ith link e_i is $[jT + i, jT + i + A_{e_i})$. From now on we consider a fixed j. Let S_i be the set of packets that have e_i as their first link and that are injected during the period $[(j-1)T + i, jT + i)$. We show in the following that the packets in $\cup_i S_i$ travel to their destinations during the jth active period of each link. (See Figure 2.)

We assign to these packets timeslots in the active periods such that i) for each packet the timeslots over its routing path are non-decreasing; ii) for two packets that share a common link, they are assigned distinct slots over this common link. With these requirements, the timeslots define when each packet gets to advance.

From packets in $\cup_i S_i$, we repeatedly extract packets whose paths form a minimal cover of the entire line, i.e., any proper subset of these packet paths cannot cover the entire line. From this minimal cover C, we remove overlaps and create an exact cover in which each link is covered by exactly one packet path. Since C is minimal, we start with the unique packet path, say p, that contains e_1. We walk along p until it first overlaps with another packet path, say q. We keep $p \setminus (p \cap q)$ for the exact cover and remove $p \cap q$. Note that due to the minimality of C, q again is unique. We proceed to walk along q until it overlaps with another packet path. We remove the overlap from q in a similar manner. When all overlaps are removed, C becomes an exact cover. Now every path in C is assigned the next available timeslot from each of the active periods. Note that the next available timeslot is the kth slot of each active period for some common k. This invariant

holds through a simple induction. Note also that if a packet path is cut, the removed part always follows the unremoved part and is therefore always assigned a larger timeslot later on. This ensures the timeslot assignment is feasible for packet movement. When the union of the remaining packet paths do not cover the entire line, we carry out the above process on each disconnected line. Note also that at most A_e of packets in $\cup_i S_i$ require each link e. The active periods therefore have enough slots for all packets. Hence, in addition to transmission time (reflected by the shifts of the active periods from one link to the next) packets in $\cup_i S_i$ experience a total of at most $A_{\max}$ queueing time.

The above centralized algorithm that iteratively creates covers can easily be replaced by a distributed protocol called *farthest-to-go* (FTG). For each link e, during its active period FTG gives priority to the packet in $\cup_i S_i$ that has the most number of remaining links to traverse. To see that FTG fits the proof above, we note that for each $k \geq 0$, the packets that use the kth timeslot in each active interval have their packet paths form a minimal cover of the line. The invariant is that during an active period of any link e, packets in $\cup_i S_i$ leave e in every timeslot and in the order farthest-to-go first. The statement holds for the first link e_1 trivially since e_1 only processes packets in S_1 and all of them are queueing at e_1 at time $jT+1$. For any subsequent link e_i, the farthest-to-go packet is either in S_i or comes from e_{i-1}. If it is the former, the packet is ready to leave e_i first; if it is the latter, by induction it leaves e_{i-1} first and is therefore ready to leave e_i first as well. This argument trickles down to subsequent packets leaving e_i. □

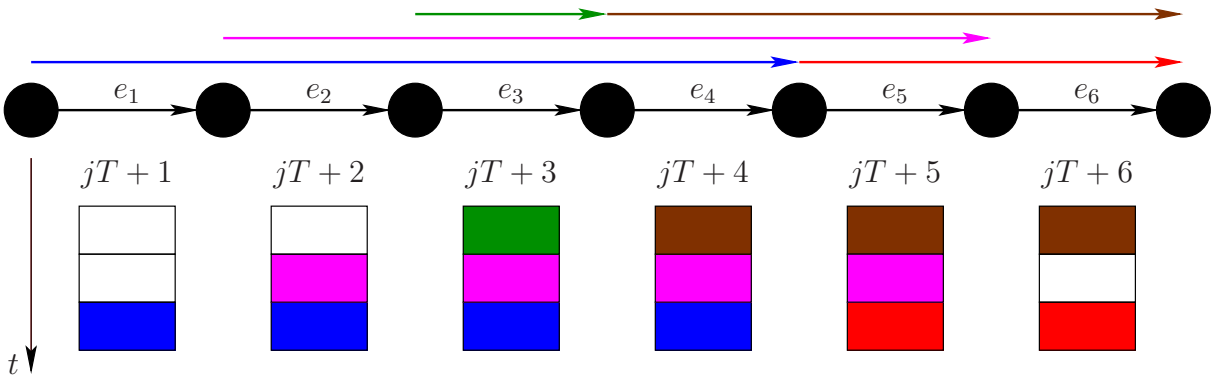

Figure 3: Schedule with small queuing delay over a line as described in Lemma 2. Each row of slots represents a color of the interval graph coloring.

If we relax the active period of each link to be $A_{\max}$, each packet experiences no queueing delay once it starts moving, after an initial delay of up to T.

Lemma 2. *For a line topology, after an initial delay of at most T each packet can reach its destination with no further queueing delay. The activation period is $A_{\max}$ for each link.*

Proof. For any integer $j \geq 1$ the jth active period for the ith link e_i along the line is $[jT+i, jT+i+A_{\max})$. As in the proof of Lemma 1 let S_i be the set of packets that have e_i as their first link and that are injected during the period $[(j-1)T+i, jT+i)$. We show in the following that the packets in $\cup_i S_i$ travel to their destinations during the jth active period of each link.

Note that at most $A_{\max}$ of the packets from $\cup_i S_i$ have e along their paths. Therefore, the well-known interval graph coloring result (e.g., [20]) implies that each packet in S can be assigned a color in $[1, A_{\max}]$ such that two packets are assigned distinct colors if they share any link in common. (See Figure 3.) These colors define the timeslots during the active periods that the packets can advance. In this way, when a packet moves from one link to the next it has a slot immediately for transmission. □

3 Combined Routing and Scheduling

We now turn our attention to the case of an arbitrary network topology. If routing is not given as input, we first choose a path P_i for each connection i. Recall that $R_e = \sum_{i:e\in P_i} \rho_i$ is the total rate on link e as a result of routing. Let

$$f(R_e) \quad = \quad \begin{cases} 0 & \text{for } R_e = 0 \\ 2\delta + R_e T & \text{for } R_e > 0 \end{cases} .$$

By routing connections with the objective of

$$\min \sum_e f(R_e)$$

we minimize the total active periods together with the transition time over all links, and therefore minimize the energy. Note that the above formulation only makes sense when $R_e < 1$. We can reasonably assume $R_e \ll 1$ regardless of routing since this paper is motivated by the scenario in which the full rate $R = 1$ is significantly larger than what is needed.

Note that $f(\cdot)$ is a concave function and the routing problem in fact corresponds to the well-studied buy-at-bulk network design problem. Awerbuch and Azar [8] offer a solution using the concept of *probabilistically approximating* metrics by tree metrics [10, 11, 19]. The idea is to approximate distances in a graph G by a probability distribution over trees. In particular, let $d_{ij}(G)$ be the length of link (i, j) in the graph G. Suppose that tree H is randomly chosen according to the probability distribution and let $d_{ij}(H)$ be the distance between i and j in the tree H. We say that the probability distribution over trees α-*probabilistically approximates* the graph G if, for each link (i, j),

$$E[d_{ij}(H)]/\alpha \le d_{ij}(G) \le d_{ij}(H),$$

where $E[d_{ij}(H)]$ is the expected distance taken over the distribution of trees. The best known result due to Fakcharoenphol, Rao and Talwar has $\alpha = O(\log n)$ [19], where n is the number of nodes in the network. However, the tree H resulting from a metric completion of G may contain edges not in G and any such edge corresponds to a path in G [19]. As will be clear later on, for our purpose we need H to be a subgraph of G. We use the following result from [2].

Theorem 3 (Abraham-Bartal-Neiman [2])**.** *Every network G can be α-probabilistically-approximated by a polynomially computable probability distribution over* spanning *trees, for $\alpha = \tilde{O}(\log n)$.*

The notion of $\tilde{O}(\log n)$ hides terms $\log\log n$ and smaller. In fact the exact value of α is $O(\log n \log\log n$

$\log^3 \log \log n$). Awerbuch and Azar [8] show that an α stretch implies an $O(\alpha)$ approximation for buy-at-bulk if we randomly pick a tree according to the probability distribution and by then route all connections along the unique path specified by the tree. For our purposes this implies

Theorem 4. *There is a randomized algorithm that chooses a routing such that the total active period together with the transition time is $\tilde{O}(\log n)$ times the optimal minimum. In addition, the routes form a tree, and for any connection the expected routing distance over the tree is $\tilde{O}(\log n)$ times the shortest path distance over the original network.*

We further take advantage of the fact that the resulting routes form a tree. We solve the scheduling problem by combining caterpillar decomposition of a tree and scheduling over a line topology. More specifically, we design a minimal schedule on a line under which the queueing delay of a packet is at most T initially plus a total of at most $A_{\max} = T \cdot R_{\max}$ once the packet starts moving. Given that the links that support the routing as a result of Theorem 4 form a tree, we show below that a technique known as caterpillar decomposition allows us to partition this tree into a collection of lines so that the routing path of each connection goes through at most $2 \log n$ lines. Every time a packet switches to a new line, in the worst case, it pays for an initial queueing delay.

Knowing how to schedule on a line, we partition the routing tree into a collection of lines in a fashion similar to the *caterpillar decomposition* [40].

Lemma 5 ([40])**.** *Any tree can be partitioned into lines, such that the unique path between any two nodes traverses at most $2 \log n$ lines, where n is the number of nodes in the tree.*

For completeness we describe the decomposition procedure. We root the routing tree at an arbitrary node r. Consider a node y, and let z be its parent node and $C(y)$ be the set of child nodes. For every child node $x \in C(y)$, let $N(x)$ be the size of the subtree rooted at x. If x' is the child node that has the largest subtree then the only line through node y is along $x'yz$. All other lines terminate at xy where $x \in C(y)$ and $x \neq x'$. We now bound the number of lines any connection has to traverse from its source node to the root r. Let $L(x)$ be the maximum number of lines from any node in the subtree rooted at x to the parent node y of x. Then

$$L(y) = \max_{x \in C(y), x \neq x'} \{L(x'), L(x) + 1\}.$$

Note that $N(y) = 1 + \sum_{x \in C(y)} N(x)$. Since $N(x') \geq N(x)$ for any $x \in C(y) - \{x'\}$, we have $N(x) < N(y)/2$. This means that, whenever the number of lines increases by 1, the size of the subtree is cut by at least half. Hence, the maximum number of lines from a node in the tree to the root r is at most $\log n$ where n is the number of nodes in the tree. For any connection to go from its source node to its destination node the number of lines is at most $2 \log n$.

Once the tree is decomposed into lines as in Lemma 5, the minimal schedule of Lemma 1 is used in each line.

Lemma 6. *The expected end-to-end delay of connection i is $4T \log n + \tilde{O}(k_i \log n)$, where k_i is the shortest path distance over the original network. Further, the schedule is minimal.*

Proof. The stretch of the spanning tree implies the length of the routing path in the selected tree is $\tilde{O}(\log n)$ times the shortest path distance in the original graph. Therefore, the transmission time is $\tilde{O}(\log n)$ times k_i.

From caterpillar decomposition, a routing path in the tree may be partitioned into $2 \log n$ lines. Therefore, the queueing delay is at most $2 \log n$ times the maximum delay in one line, $T + A_{\max} \leq 2T$. □

Observe that as $R_e \ll 1$, the length of the non-active period in each frame is $T - A_e = \Omega(T)$. Then, a packet that arrives at the beginning of this period has to wait $\Omega(T)$ before moving. Hence, the packet delay for connection i is lower bounded as $\Omega(T + k_i)$. Combining this with Theorem 4 and Lemma 6 we have

Theorem 7. *The combined routing and scheduling scheme ensures an $\tilde{O}(\log n)$ approximation for delay minimization and $\tilde{O}(\log n)$ for energy minimization.*

4 Scheduling with Given Routes

In the previous section on the combined routing and scheduling scheme, we took advantage of the fact that the routes form a tree. In this section we focus on scheduling assuming routes are given as input, and assume these routes form an arbitrary topology.

4.1 Energy-Optimal Algorithm

We begin with a simple algorithm to demonstrate that a minimal schedule is always possible in a network of arbitrary topology, but a packet may experience a delay of T per link. Let k_i be the hop count of route P_i for connection i.

Theorem 8. *For a network with an arbitrary topology, the end-to-end delay of connection i is bounded by $T \cdot k_i$ under a minimal schedule.*

Proof. We first define a minimal schedule. For each time frame, link e is activated for the first $T \cdot R_e$ time steps. The schedule works as follows. At (the start of) time step jT, let S_e be the set of packets queueing at link e. During the time frame $[jT, (j+1)T)$, only packets from S_e advance along link e. That is, packets that arrive during $[jT, (j+1)T)$ have to wait until the next frame $[(j+1)T, (j+2)T)$ even if there is room during the active period $[jT, jT + TR_e)$. In the following we show that $|S_e| \leq TR_e$. We assume inductively that so far it takes one frame for each packet to advance one link. Therefore, S_e consists of packets from connections that start at link e and are injected during $[(j-1)T, jT)$, and from connections that have e as the ith link and are injected during $[(j-i)T, (j-i+1)T)$. The total rate of these connections is R_e. Therefore, $|S_e| \leq T \cdot R_e$. Since all packets from S_e are queueing at link e at the beginning of a time frame, they can be transmitted along link e during the active period of the frame which is the first $T \cdot R_e$ time steps. □

This minimal schedule can be seen as a direct application of Lemma 1 on the trivial decomposition of connection routes into lines in which each link is a different line. For that reason a packet experiences a line switch (and hence a delay of at most T) at each edge. Clearly, any other decomposition of the connection paths into lines would improve the end-to-end delay by reducing line switches. Unfortunately, there is not a lot of room for improvement in the worst case, since it is possible to find networks and connection sets such that, independent of how the decomposition is done, almost every connection i will experience $\Omega(k_i)$ line switches. However, experimental results presented in Section 5 show that at a practical level this could be a promising approach.

4.2 Delay-Improved Algorithm

We show now that, by allowing the active period to be non-minimal, the maximum delay suffered by the packets can be reduced to $2T$, for large enough T.

4.2.1 Overview and algorithm

The overview of the approach is as follows. For each link, time is partitioned into frames of length T, $[0, T), [T, 2T)$, etc. The first A time steps during each frame is the active period. We create a schedule so that packets that are injected during the jth frame $[jT, (j+1)T)$ queue up at their source during the frame and traverse to their destinations during the active period of the following frame, i.e., during $[(j+1)T, (j+1)T+A)$. In the following we describe a *successful* schedule for one frame, and the same schedule is repeated for every subsequent frame.

A schedule for a frame is formed by a schedule for each link of the network. In the schedule of link e, one time step is assigned to each of the packets that have to cross e in the frame[1]. Observe that, for each connection i, there is a set S_i of $\rho_i T$ packets at the beginning of each frame, waiting at the first queue of the path P_i. Then, the schedule will assign to each packet $p \in S_i$ a sequence of time steps, one in the schedule of each of the links in P_i. These time steps must satisfy that, if link e precedes link e' in the path P_i then the time step in the schedule of e must occur before the time step in the schedule of e'. Then, p will simply cross each edge $e \in P_i$ in the corresponding time step in e's schedule.

To build the link schedules with the desired properties, we use a randomized scheduling algorithm (described below) that finds such schedules with high probability. Observe that all that is needed is one set of link schedules. Once these are found, they will be used over and over. Hence this algorithm has to be run only until one such set has been found. Since this happens with high probability in every run, the algorithm will be run very few times (most possibly only once).

We now describe how to schedule a set of $\cup_i S_i$ packets in A time steps where S_i consists of $\rho_i T$ packets at the source of connection i. All required parameters are defined in the subsequent section. Each active period consists of $k_{\max} + M - 1$ intervals of L steps, where the ℓth such interval in the active period of link e is denoted $I_{e,\ell}$. The algorithm has two phases.

1. In the first phase, each packet p chooses an integer λ_p uniformly at random from the range $[1, M]$. For each link in its path $e_1, e_2, \ldots$, packet p is mapped to the $(\lambda_p + i - 1)$th interval of its ith link e_i, namely $I_{e_i, \lambda_p + i - 1}$.

2. In a second phase, for each interval $I_{e,\ell}$, all the packets mapped to it are spread within the interval, so that each packet is assigned a different time step in the interval. Observe that the random integers λ_p chosen in the first phase may map more than L packets to some interval $I_{e,\ell}$. If this happens, then the schedule fails.

We make a few remarks on a successful schedule.

- In Lemma 9 below we show that a schedule is successful with high probability. Once we find a successful schedule, possibly after multiple attempts, we repeat this schedule for each frame. We do not create a schedule on the fly for each frame.

[1] We assume these packets are uniquely identified, e.g., by connection and arrival order.

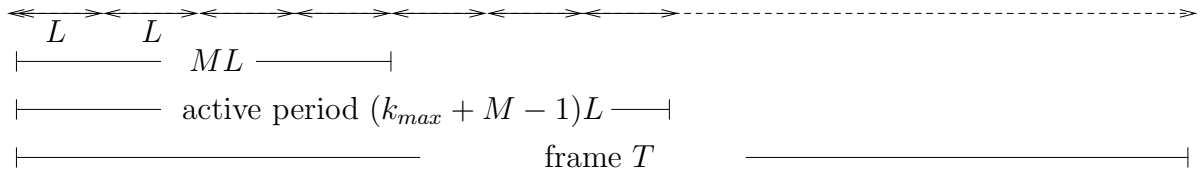

Figure 4: One frame of the schedule over one link.

- Whether a schedule is successful is completely determined by the initial choices of λ_p for $p \in \cup_i S_i$. The spreading within each interval can be done via any distributed contention resolution scheme local to the link.
- A successful schedule is created for $\rho_i T$ packets per connection i. However, if fewer packets have arrived during the previous frame it is easy to see that the schedule continues to be successful.

The randomized scheduling algorithm described inherits from earlier studies on static packet scheduling. For example, Leighton, Maggs and Rao [38] showed that if we have a static scheduling problem in which at most C packets need to be routed through any edge and the length of each path is at most D, then all of the packets may be routed to their destination in time $O(C + D)$. We achieve here a delay for the same static problem of $(1-\epsilon)C + O(D \log(mC))$ with an adapted version of an algorithm presented in [5]. These results were generalized to a dynamic routing problem in [6], where it was shown that if connection i has injection rate ρ_i and path length k_i then all connections can be scheduled in such a way that the end-to-end delay is $O(\frac{1}{\rho_i} + k_i)$.

4.2.2 Parameters

$R_{\max} = \max_e R_e$	max total rate over a link
$k_{\max} = \max_i k_i$	max path length
m	number of links in the network
δ	transition time between modes
$\varepsilon = 1 - R_{\max}$	$\varepsilon \in (0,1)$: "load factor"
$\beta = 1 + \varepsilon$	
$\alpha = 2\varepsilon^{-2}$	
$\varphi = 2\alpha(\beta\alpha + 2\delta)$	
$T \geq \varphi k_{\max} \ln(\varphi m)$	frame size
$Q = mR_{\max}Tk_{\max}$	$1/Q$: failure probability
$M = \frac{R_{\max}T}{\alpha \ln Q}$	$[1, M]$: range of first interval
$L = \alpha\beta \ln Q$	duration of each interval
$A = (k_{\max} + M - 1)L$	duration of each active period

4.2.3 Analysis

To show the correctness of the algorithm, we need to prove a couple of facts. First, that the algorithm has a high success probability. Second, $T \geq A + 2\delta$. Note that under a successful schedule, it is straightforward to see that every packet in $\cup_i S_i$ reaches its destination during the active period by the definition of A and the definition of the algorithm. Therefore, every packet injected during one frame reaches its destination during the next frame. The correctness is complete.

Lemma 9. *The probability that the algorithm fails is at most* $1/Q$.

Proof. Consider the jth interval $I_{e,j}$ of an edge e. Let $N_{e,j}$ be the number of packets mapped to interval $I_{e,j}$ in the first phase of the algorithm. The expected value of $N_{e,j}$ is $R_{\max}T/M = \alpha \ln Q$. From a Chernoff bound [41, Theorem 4.4(1)], we have

$$\begin{aligned} Pr[N_{e,j} > L] &\leq \exp((\beta - 1 - \beta \ln \beta)\alpha \ln Q) \\ &= Q^{-2}, \end{aligned}$$

where the last equality follows from the definitions of α, β and the fact that $\ln(1+\varepsilon) \approx \varepsilon$. We now use a union bound to bound the overall failure probability by multiplying the expression above by the m possible choices of e and the at most $k_{\max} - 1 + M$ choices of j. Assuming, w.l.o.g., that $k_{\max} - 1 + M = k_{\max} - 1 + \frac{R_{\max}T}{\alpha \ln Q} \leq$

$k_{\max} R_{\max} T$, and since $Q = m R_{\max} T k_{\max}$, we have

$$\begin{aligned} Pr[\exists e, j : N_{e,j} > L] &\leq m\,(k_{\max} - 1 + M)\,Q^{-2} \\ &\leq Q^{-2+1}, \end{aligned}$$

and the failure probability is at most $1/Q$. □

Given the high success probability of the scheduling algorithm, with very few executions a suitable schedule will be found. Then, that same schedule will be used subsequently.

Observation 10.

$$A = (1 - \varepsilon^2)T + \beta\alpha(k_{\max} - 1)\ln(m R_{\max} T k_{\max}).$$

Proof.

$$\begin{aligned} A &= L(k_{\max} - 1 + M) \\ &= \alpha\beta \ln Q(k_{\max} - 1 + \frac{R_{\max} T}{\alpha \ln Q}) \\ &= \beta R_{\max} T + \beta\alpha(k_{\max} - 1)\ln(m R_{\max} T k_{\max}) \\ &= (1 - \varepsilon^2)T + \beta\alpha(k_{\max} - 1)\ln(m R_{\max} T k_{\max}) \end{aligned}$$

□

Lemma 11. *For any $T \geq \varphi k_{\max} \ln(\varphi m)$, $T \geq A + 2\delta$.*

Proof. We first prove the result for $T = \varphi k_{\max} \ln(\varphi m)$. As we shall see, a larger T only improves the matter. Note that for $\varepsilon \in (0, 1)$, $\varphi \geq 1$. Since $k_{\max} \leq m$ and $R_{\max} < 1$, we have

$$\begin{aligned} T &= \varphi k_{\max} \ln(\varphi m) \\ &\geq \frac{\varphi}{4} k_{\max}\,(2 \ln m + 2 \ln(\varphi m)) \\ &\geq \frac{\varphi}{4} k_{\max}\,(\ln(m k_{\max}) + \ln(\varphi k_{\max}) + \ln\ln(\varphi m)) \\ &= \frac{\varphi}{4} k_{\max} \ln(m T k_{\max}) \\ &= \frac{\beta\alpha + 2\delta}{\varepsilon^2} k_{\max} \ln(m T k_{\max}) \\ &\geq \frac{\beta\alpha(k_{\max} - 1)\ln(m R_{\max} T k_{\max}) + 2\delta}{\varepsilon^2} \end{aligned} \tag{1}$$

On the other hand, for $T \geq A + 2\delta$ to hold, we obtain the following inequality from Observation 10,

$$T \geq (1 - \varepsilon^2)T + \beta\alpha(k_{\max} - 1)\ln(m R_{\max} T k_{\max}) + 2\delta.$$

Observe that this inequality is equivalent to (1). Hence, if $T = \varphi k_{\max} \ln(\varphi m)$ then $T \geq A + 2\delta$. When $T > \varphi k_{\max} \ln(\varphi m)$, inequality (1) continues to hold since T on the left-hand side of (1) grows faster than $\ln T$ on the right-hand side. □

Theorem 12. *If the frame length T satisfies the conditions of Lemma 11, under a successful schedule, all packets injected during one frame can reach their destinations during the active period of the next frame, and the length of the activation period is at most* $(1-\varepsilon^2)T + O(\frac{k_{\max}}{\varepsilon^2}\ln(mT))$.

Proof. Note that a packet p that arrives during the jth frame of length T moves to its destination during that $(j+1)$st frame. In a successful schedule, packet p is assigned an exclusive time step to cross the ith link in its path in the $(\lambda_p + 1 - 1)$th interval of length L of the frame. Then, p reaches its destination at most $T + A$ time units after its injection. From Lemma 11, $A < T$ and hence the first claim of the theorem holds. Using the facts that $\beta\alpha = O(1/\varepsilon^2)$, $R_{\max} < 1$, and $k_{\max} < mT$, from Observation 10 we obtain that the active period is

$$A = (1-\varepsilon^2)T + \beta\alpha(k_{\max}-1)\ln(mR_{\max}Tk_{\max}) = (1-\varepsilon^2)T + O\left(\frac{k_{\max}}{\varepsilon^2}\ln(mT)\right).$$

□

5 Experimental Results

In this section, we compare via simulation two frame-based schedules, a coordinated schedule and a simple schedule without coordination. Both schedules are minimal, and hence are optimal in terms of energy. Therefore, we use the (average and maximum) delay of packets when using each of them as the goodness metric. The coordinated schedule, called here *schedule with coordination* (SWC), is defined for lines. It activates each edge e for a time A_e in each frame (with the active periods shifted along the line), and applies the farthest-to-go (FTG) scheduling protocol in each of the edges of the line, as was described in Lemma 1. The *schedule without coordination* (SWOC) simply activates each edge e for a time A_e at the beginning of the frame, and uses FIFO to schedule packets. This schedule can be seen as a greedy implementation of the schedule described in Section 4.1.

Two sets of simulations are performed. The first uses a network which is a line, while the second uses a general network. In each scenario, the number of connections is fixed, with the terminals of each connection chosen uniformly at random among all the nodes in the network. The connection rate is fixed at $\rho = 1/50$, so that ρT packets arrive for each connection in each T-frame. Every packet arrives at a uniformly chosen time step within the frame. However, following the lines of Lemma 1, SWC does not schedule packets to move until the next frame (even if the link is active and there are available time slots). For any given set of parameters, the experiment corresponding to these parameters is performed 10 times, and the observed results averaged.

5.1 Lines

We first present the simulation results on a network which is a line of 40 links. In this network 50 connections with path length 10 are randomly chosen. Then, simulations with time frame size T in the range 50 to 250 are performed. Figure 5 shows the average and the maximum end-to-end delay observed in these simulations.

From Figure 5, it can be observed that the SWC provides significantly smaller end-to-end delay than the SWOC. Both the average and maximum end-to-end delays increase with T under SWC and SWOC.

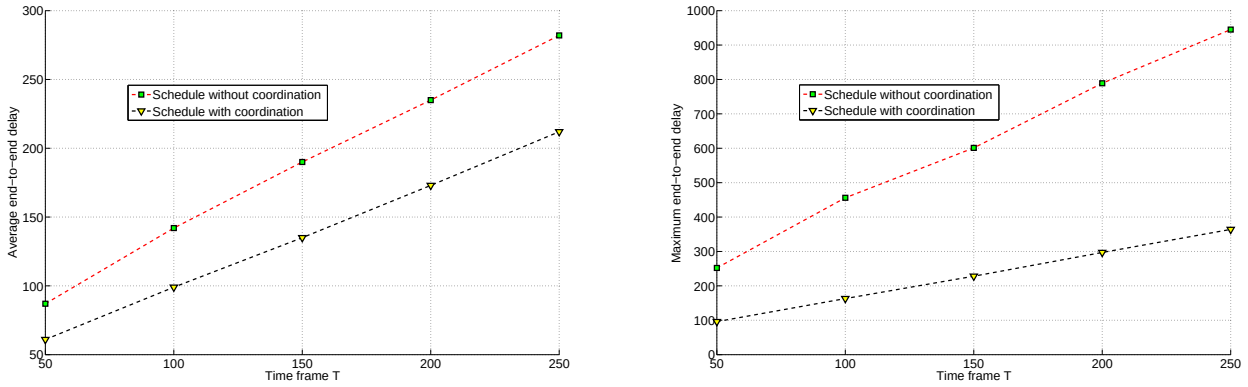

Figure 5: Average (left) and maximum (right) end-to-end delay for a line network of length 40, with 50 connections of path length 10.

However, the growth is faster under SWOC, which means that the larger the frame, the more convenient it is to use SWC instead of SWOC. We have also observed that changing the length of the connection path does not change the fact that SWC performs better than SWOC. For instance, for path length of 30, SWC gives at least 26.8% better average end-to-end delay.

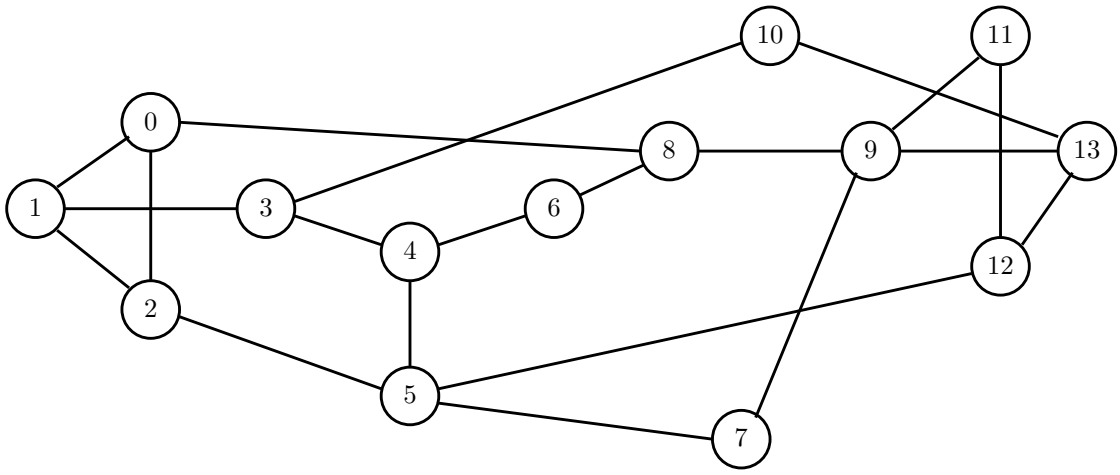

Figure 6: The NSF network.

5.2 NSF Network

The second network we consider is the NSF network (shown in Figure 6), which consists of 14 nodes and 20 links. In this (and any arbitrary) network, SWOC works as it did in the line, simply using shortest path routing. However, we need to define how to make SWC work in the NSF network (and in any arbitrary network in general). The solution is to do line decomposition: decompose the network into edge-disjoint lines,

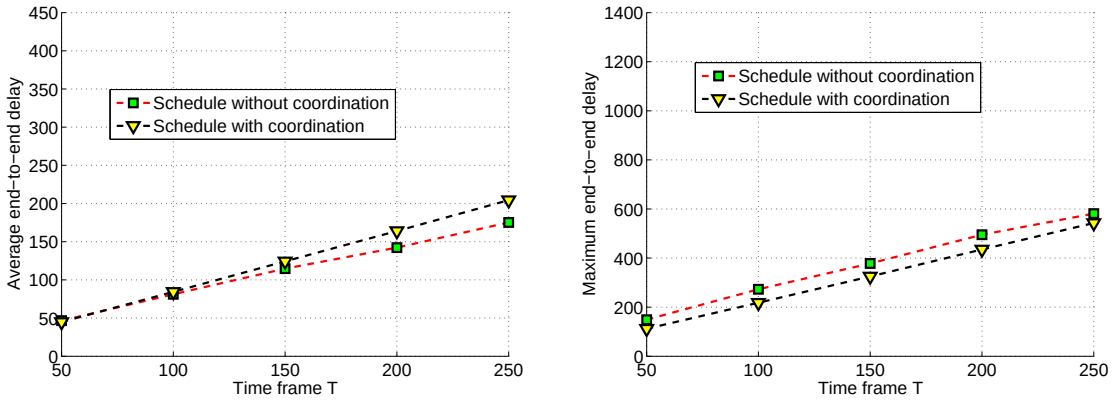

Figure 7: Average (left) and maximum (right) end-to-end delay for $l = 1$.

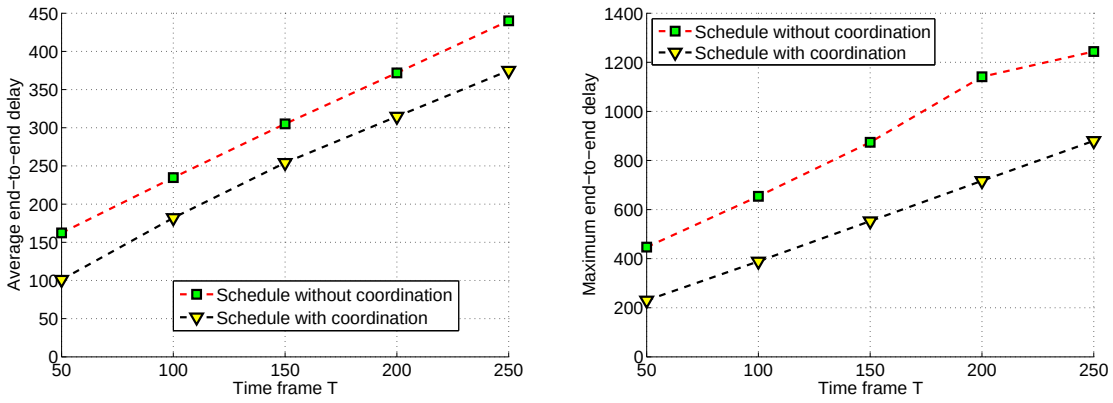

Figure 8: Average (left) and maximum (right) end-to-end delay for $l = 10$.

route connections along the lines, and use SWC in each line. In this case, we have manually decomposed the NSF network into four link-disjoint lines, shown in Table 1.

When needed, a packet has to switch lines. In our simulations, when this occurs, the switching packet p is treated in the new line as any new arriving packet (and, in particular, p will not move until the next frame). This highly penalizes SWC at each line switch. We are interested in the impact of line switches on the end-to-end delay with respect to the path length, captured by a parameter κ, which is the ratio of the length of a connection path to the number of line switches. Since the original NSF network is small, we extend the NSF network by replacing each link with a path of length l. This allows increasing the ratio κ just defined without changing the number of line switches. Hence, in our simulations l roughly represents the above ratio.

In each experiment in the NSF network 100 connections are created. In each connection the same number of packets ρT are injected in each terminal node towards the other in each T-frame. As said above, these packets are routed via shortest paths when using SWOC and via decomposition lines when using SWC.

The results of the simulations are presented in Figures 7 and 8. For small l (e.g., $l = 1$ in Figure 7) SWC

Line label	Decomposed lines for NSF network
1	$1-2-0-8-9-7-5-12-13$
2	$0-1-3-4-6-8$
3	$3-10-13-9-11-12$
4	$2-5-4$

Table 1: Path decomposition of the NSF network.

and SWOC have comparable performances. However, as l becomes larger (e.g., $l = 10$ in Figure 8), SWC provides significantly better delays than SWOC, both on average and in the worse case. Table 2 summarizes several parameters observed in the experiments for the values $l = 1, 5, 10$. It is easy to verify in this table that l in fact behaves as the ratio κ. Hence, from this table and the previous figures one can conclude that as the ratio κ grows (which is $2.0/0.63 < 10.5/1.17 < 20.7/1.1$, see Table 2), SWC gives much better end-to-end delay performance than SWOC.

	# of	R_e		Connect. length		Line switches	
l	nodes	Average	Max	Average	Max	Average	Max
1	14	0.20	0.46	2.0	4.0	0.63	2.0
5	94	0.21	0.40	10.5	21	1.17	4
10	194	0.21	0.42	20.7	44	1.1	4

Table 2: The length of connection paths and the number of line switches.

6 Conclusions and Future Work

In this paper we study the problem of reducing energy consumption while satisfying the bandwidth requirements of a set of connections via powering down network links. Two variants of the problem are explored, depending on whether the routes of the connections are given or have to be chosen with a routing algorithm. Under both variants, a trade-off between power consumption and delay guarantees has been unveiled. It is shown that energy optimality is possible at the cost of a potentially large delay, but that by allowing a larger power consumption the delay bounds can be highly improved.

We want to note that our delay-friendly schedules require some coordination with global information. However, some of these schedules exhibit distributed nature, for example, using FTG for contention resolution locally at each link. Coming up with completely distributed schedules remains a challenge. Another way to extend this work is to consider a more general packet arrival model. For instance, the leaky-bucket model assumed could be generalized to a model with some degree of randomness in the arrivals. This study is beyond the scope of this work, but it is of significant interest. At a more practical level, it is worth noting that the energy saved by powering down links is relatively small, since it occurs only at the physical level[2]. The solutions proposed in this paper do not consider powering down queues or even whole nodes. This option may lead to higher energy savings and it is worth further exploring.

[2]Reviriego et al. [47] have proposed techniques to further increase these savings.

References

[1] Smart2020: Enabling the low carbon economy in the information age, webpage. www.smart2020.org.

[2] I. Abraham, Y. Bartal, and O. Neiman, Nearly tight low stretch spanning trees, FOCS, IEEE Computer Society, 2008, pp. 781–790.

[3] M. Ajmone Marsan, A. Fernández Anta, V. Mancuso, B. Rengarajan, P. Reviriego Vasallo, and G. Rizzo, A simple analytical model for energy efficient ethernet, IEEE Comm. Letters. 15 (July 2011), 773–775.

[4] M. Andrews, S. Antonakopoulos, and L. Zhang, Minimum-cost network design with (dis)economies of scale, Proc 51st Ann Symp Foundations Comput Sci, FOCS, Las Vegas, NV, October 2010, pp. 585–592.

[5] M. Andrews, B. Awerbuch, A. Fernández, F.T. Leighton, Z. Liu, and J.M. Kleinberg, Universal-stability results and performance bounds for greedy contention-resolution protocols, J. ACM 48 (2001), 39–69.

[6] M. Andrews, A. Fernández, M. Harchol-Balter, F.T. Leighton, and L. Zhang, General dynamic routing with per-packet delay guarantees of O(distance + 1/session rate), SIAM J. Comput. 30 (2000), 1594–1623.

[7] M. Andrews, A. Fernández Anta, L. Zhang, and W. Zhao, Routing for power minimization in the speed scaling model, IEEE/ACM Trans. Netw. 20 (2012), 285–294.

[8] B. Awerbuch and Y. Azar, Buy-at-bulk network design, FOCS, 1997, pp. 542–547.

[9] N. Bansal, T. Kimbrel, and K. Pruhs, Speed scaling to manage energy and temperature, J. ACM 54 (Mar. 2007), 3:1–3:39.

[10] Y. Bartal, Probabilistic approximations of metric spaces and its algorithmic applications, FOCS, 1996, pp. 184–193.

[11] Y. Bartal, On approximating arbitrary metrices by tree metrics, STOC, ACM, 1998, pp. 161–168.

[12] H.L. Chan, W.T. Chan, T.W. Lam, L.K. Lee, K.S. Mak, and P.W.H. Wong, Energy efficient online deadline scheduling, Proc Eighteenth Ann ACM-SIAM Symp Discr Algorithms, SODA, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, 2007, pp. 795–804.

[13] K.J. Christensen, C. Gunaratne, B. Nordman, and A.D. George, The next frontier for communications networks: Power management, Comput Commun 27 (2004), 1758–1770.

[14] K.J. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J.A. Maestro, IEEE 802.3az: The road to energy efficient ethernet, IEEE Commun Magazine 48 (2010), 50–56.

[15] C.S. Computing, webpage. http://www.climatesaverscomputing.org.

[16] T.E.C.C. (ECCJ), Top runner program, webpage. Asia Energy Efficiency and Conservation Center (AEEC), http://www.asiaeec-col.eccj.or.jp/top_runner.

[17] E.N.P. et al., Vision and roadmap: Routing telecom and data centers toward efficient energy use, webpage. http://sites.energetics.com/ICT_roadmap09/pdfs/ICT_Vision_and_Roadmap_051309_draft.pdf.

[18] EuP, Networked standby DG ENER Lot 26, webpage. The European Commission (DG ENER), http://www.ecostandby.org.

[19] J. Fakcharoenphol, S. Rao, and K. Talwar, A tight bound on approximating arbitrary metrics by tree metrics, STOC, ACM, 2003, pp. 448–455.

[20] P.C. Fishburn, Interval orders and interval graphs, Wiley and Sons, New York, 1985.

[21] A. Francini and D. Stiliadis, Rate adaptation for energy efficiency in packet networks, Bell Labs Tech J 15 (Sept. 2010), 131–146.

[22] M. Garrett, Powering down, Commun. ACM 51 (2008), 42–46.

[23] G. Ginis, Low-power modes for ADSL2 and ADSL2+, webpage. White Paper, Broadband Communications Group, Texas Instruments.

[24] GreenTouch, webpage. http://www.greentouch.org.

[25] C. Gunaratne, K.J. Christensen, and B. Nordman, Managing energy consumption costs in desktop pcs and lan switches with proxying, split tcp connections, and scaling of link speed, Int. J Network Manage 15 (2005), 297–310.

[26] C. Gunaratne, K.J. Christensen, B. Nordman, and S. Suen, Reducing the energy consumption of ethernet with adaptive link rate (alr), IEEE Trans. Comput 57 (2008), 448–461.

[27] M. Gupta and S. Singh, Greening of the internet, Proc 2003 Conference Appl, Technologies, Architectures, Protocols Comput Commun, SIGCOMM '03, New York, NY, USA, ACM, 2003, pp. 19–26.

[28] M. Gupta and S. Singh, Dynamic ethernet link shutdown for energy conservation on ethernet links, IEEE Int Conference Commun, ICC '07 (June 2007), 6156–6161.

[29] M. Gupta and S. Singh, Using low-power modes for energy conservation in ethernet lans, INFOCOM, IEEE, 2007, pp. 2451–2455.

[30] IEEE, IEEE Std 802.3az: Energy Efficient Ethernet-2010, webpage. IEEE Standard.

[31] Intel, Enhanced Intel speedstep technology for the Intel Pentium M processor, webpage. Intel White Paper 301170-001.

[32] S. Irani, R.K. Gupta, and S.K. Shukla, Competitive analysis of dynamic power management strategies for systems with multiple power savings states, DATE, IEEE Computer Society, 2002, pp. 117–123.

[33] S. Irani and K. Pruhs, Algorithmic problems in power management, SIGACT News 36 (2005), 63–76.

[34] S. Irani, S.K. Shukla, and R.K. Gupta, Online strategies for dynamic power management in systems with multiple power-saving states, ACM Trans. Embedded Comput. Syst. 2 (2003), 325–346.

[35] S. Irani, S.K. Shukla, and R.K. Gupta, Algorithms for power savings, ACM Trans Algorithms 3 (2007).

[36] R. Kumar and L. Mieritz, Conceptualizing 'green' it and data center power and cooling issues, webpage. Gartner Research Paper No G00150322, 2007.

[37] P. Kurp, Green computing, Commun. ACM 51 (2008), 11–13.

[38] F.T. Leighton, B.M. Maggs, and S. Rao, Packet routing and job-shop scheduling in O(congestion + dilation) steps, Combinatorica 14 (1994), 167–186.

[39] M. Li, B.J. Liu, and F.F. Yao, Min-energy voltage allocation for tree-structured tasks, COCOON, Vol. 3595 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 283–296.

[40] J. Matoušek, On embedding trees into uniformly convex banach spaces, Israel J Math 114 (1999), 221–237.

[41] M. Mitzenmacher and E. Upfal, Probability and computing: Randomized algorithms and probabilistic analysis, Cambridge University Press, 2005.

[42] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, Reducing network energy consumption via sleeping and rate-adaptation, NSDI, USENIX Association, 2008, pp. 323–336.

[43] B. Nordman and K.J. Christensen, Reducing the energy consumption of network devices, webpage. tutorial, IEEE 802 LAN/MAN Standards Committee Plenary Session.

[44] P. Patel-Predd, Energy-efficient ethernet, IEEE Spectrum (May 2008), 13.

[45] P. Ranganathan, A science of power management, a systems perspective, webpage. http://scipm.cs.vt.edu/Slides/ParthaRanganathan.pdf.

[46] P. Reviriego, K.J. Christensen, J. Rabanillo, and J.A. Maestro, An initial evaluation of energy efficient ethernet, IEEE Commun Lett 15 (2011), 578–580.

[47] P. Reviriego, K.J. Christensen, A. Sánchez-Macián, and J.A. Maestro, Using coordinated transmission with energy efficient ethernet, Networking (1), Vol. 6640 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 160–171.

[48] P. Reviriego, J.A. Hernández, D. Larrabeiti, and J.A. Maestro, Performance evaluation of energy efficient ethernet, Comm. Letters. 13 (September 2009), 697–699.

[49] P. Reviriego, J.A. Hernández, D. Larrabeiti, and J.A. Maestro, Burst transmission for energy-efficient ethernet, IEEE Internet Comput 14 (2010), 50–57.

[50] E. STAR, webpage. http://www.energystar.gov.

[51] F.F. Yao, A.J. Demers, and S. Shenker, A scheduling model for reduced cpu energy, FOCS, 1995, pp. 374–382.