CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2014)

A two-layer model for improving the energy efficiency of file sharing peer-to-peer networks

Paolo Trunfio*,†

DIMES, University of Calabria, Rende, Cosenza, Italy

SUMMARY

As peer-to-peer networks gather and share large sets of computing resources, their aggregate energy consumption is an important problem to be addressed, given the economic and environmental impact of energy production and use. This is particularly true for file sharing peer-to-peer networks, considered the vast number of nodes participating to these systems. To address such problem, this paper proposes a peer-to-peer file sharing model organized in two logical layers. The lower layer is composed of a set of peers providing the files to be shared. The upper layer includes a set of super peers organized to form a distributed hash table, whose purpose is indexing files and peers, including their availability status. Following a sleep-and-wake approach, energy saving is achieved by letting peers cyclically switch between normal and sleep modes, where the time passed in normal mode by a peer depends on the number of files it provides. An energy-saving algorithm and an associated file search and retrieval strategy are proposed within this model. Simulation results show that more than 50% of energy can be saved using the proposed model and that more than 80% of file downloads start with no additional delay compared with a network in which all peers are always powered on. Copyright © 2014 John Wiley & Sons, Ltd.

KEY WORDS: peer-to-peer; file sharing; energy efficiency; performance analysis

1. INTRODUCTION

After years of intensive development and research, peer-to-peer is widely employed today as an effective computing paradigm to implement decentralized, self-organizing, and self-coordinating large-scale distributed systems. As peer-to-peer networks gather and share large sets of computing resources, their aggregate energy consumption is an important problem to be addressed, given the economic and environmental impact of energy production and use. This is particularly true for file sharing peer-to-peer networks, considered the vast number of users and nodes participating to these systems. The latter is also witnessed by the impact of peer-to-peer file sharing on the overall Internet traffic, which some studies have quantified between 40% and 70% [1, 2].

The importance of the problem has stimulated several researches aimed at improving the energy efficiency of peer-to-peer networks. Common approaches toward this goal include the use of proxies, optimizing task allocation, message reduction, location-based mechanisms, overlay structure optimization, and the 'sleep-and-wake' strategy [3]. The latter is one of the most popular approaches, based on the principle that the overall energy consumption of a peer-to-peer system can be significantly reduced if peers periodically switch from normal mode (high-power) to sleep mode (low-power) and vice versa. Sleep-and-wake is based on the observation that a main cause of wasted energy in peer-to-peer networks is represented by hosts which are kept powered on even when they

*Correspondence to: Paolo Trunfio, DIMES, University of Calabria, Via P. Bucci 41C, 87036 Rende, Cosenza, Italy.
†E-mail: trunfio@dimes.unical.it

are not active. The popularity of the approach, other than on its performance, depends also on the fact that it can be effectively implemented in a real peer-to-peer system. In fact, there are libraries providing methods to easily switch off or schedule a PC wake-up directly from a program, including a peer-to-peer client application. This is a point of strength that highly motivated the adoption of the sleep-and-wake approach in the present work.

A problem of the sleep-and-wake approach is that, because hosts play also the role of file providers, temporarily turning off a subset of them lowers the probability for a user of finding a file of interest at a given time. A way to solve this problem is resubmitting the query on a periodical basis, but this introduces excessive overhead to the network if the period is too short or causes late responses if the period is too long. Alternatively, a centralized index of current peers availability could be maintained, but this would negate the decentralized nature of peer-to-peer networks.

To take the most of the sleep-and-wait approach while limiting its inherent drawbacks, this paper proposes a peer-to-peer file sharing model organized in two logical layers. The lower layer is composed of a set of peer nodes, each one providing a set of files to be shared. The upper layer is composed of a set of super-peer nodes, organized to form a distributed hash table (DHT), whose purpose is indexing the peers and the files at the lower layer. Hence, in addition to file indexing, the DHT is used in the proposed model for keeping a fully decentralized index of peers, of their current availability status (where sleeping peers are considered unavailable), and of the next time when they are expected to return available. Each peer autonomously takes decisions about its sleep–wake cycles. In particular, it is followed the principle that the time passed in normal mode by a peer depends on the number of files it provides. This design choice is based on the observation that the higher the number of files shared by a peer P_i, the higher the probability that P_i owns a file matching a query submitted to the network. Therefore, keeping powered on for a longer time the peers sharing a higher number of files, it is increased the chance that a file can be found and retrieved with no additional delay compared with a network in which all peers are always powered on.

The search for an existing file returns a reference to the providing peer, which can be either available or not. In case the providing peer is unavailable, its expected availability time is also returned to the requestor, thus allowing to postpone the file download until the availability time is reached. The system has been evaluated through simulations, by using a custom discrete-event network simulator. The experimental results show that more than 50% of energy can be saved using the proposed approach, while preserving the overall quality of service perceived by the user. Regarding this last aspect, the simulation results show that more than 80% of file downloads start with no additional delay compared with a network in which all peers are always powered on, while in the remaining cases the download is postponed for a limited amount of time.

In summary, the main contributions of this paper are

- the definition of a two-layer model for improving the energy efficiency of file sharing peer-to-peer networks, in which a DHT keeps a fully decentralized index of files, peers, and their availability status;
- the definition of an energy-saving (ES) algorithm in which every peer autonomously takes decisions about its sleep–wake cycles, following the principle that the higher the number of files owned, the lower the sleep duration;
- the definition of a file search and retrieval (FSR) algorithm that allows peers to search and retrieve files over the peer-to-peer network taking into account the availability/unavailability status of the file providers; and
- an experimental analysis of the proposed framework for assessing its behavior and for evaluating its performance.

The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 presents the two-layer system model. Sections 4 and 5 describe the ES and FSR algorithms, respectively. Section 6 presents an experimental evaluation of the system. Finally, Section 7 concludes the paper.

2. RELATED WORK

A recent comprehensive survey by Malatras, Peng, and Hirsbrunner [3] has thoroughly studied existing research works on energy-efficient peer-to-peer systems, classifying them under six categories, according to the approach they follow in reducing energy consumption: proxying, task allocation optimization, message reduction, location-based, sleep-and-wake, and overlay structure optimization.

The proxying approach is based on the use, by peer-to-peer hosts, of proxies to delegate some of their activities, such as file downloading. Using proxies, peer-to-peer hosts do not need to stay constantly online, this way reducing the overall energy consumption. Examples of proxy-based approaches are the system proposed by Anastasi *et al*. [4] for reducing the energy consumption of hosts running the BitTorrent application [5] and the system by Purushothaman *et al*. [6] for Gnutella networks [7].

Task allocation optimization is based on the observation that significant energy savings can be achieved in a peer-to-peer network by carefully scheduling the allocation of tasks to peers, that is, deciding on which peer will satisfy the request of another peer. One example is the work by Enokido *et al*. [8, 9], who proposed a model for peer-to-peer data transfers in which computation time and power consumption are minimized by optimizing the allocation of file requests.

Message reduction aims at minimizing the number of messages exchanged through the peer-to-peer network with the goal of lowering processing and transmission times, thus reducing energy consumption. One example of energy-saving peer-to-peer system based on this approach is the work by Kelenyi and Nurminen [10], who adopted a selective message dropping mechanism for reducing the number of messages exchanged in a Kademlia network.

The location-based approach exploits positioning information about nodes to make peer-to-peer overlays more closely matching the underlying physical connections with the goal of reducing multihop transmissions and consequently the overall energy consumption. This approach is particularly effective in mobile peer-to-peer networks, as proven by the research works proposed by Joseph *et al* [11], Park and Valduriez [12], and Tung and Lin [13].

The sleep-and-wake approach aims at reducing the overall energy consumption of a peer-to-peer network by letting peers cyclically switch between normal and sleep states. The critical point of this approach is deciding when peers should be in a normal or sleep state, in order to avoid excessive degradation of system performance. Several systems fall in this category, including the ones by Lefebvre and Feeley [14], Blackburn and Christensen [15], Lee *et al*. [16], Gurun *et al*. [17], Sucevic *et al*. [18], Jourjon *et al*. [19], Andrew *et al*. [20], and Hlavacs *et al*. [21].

Finally, overlay structure optimization aims at improving the energy efficiency of a peer-to-peer network by either controlling its topology during construction or maintenance or introducing new layers to the overlay. An example of the first type is the work by Leung and Kwok [22], where topology control is used for improving the energy efficiency of wireless file sharing peer-to-peer networks. An example of the second type is the double-layered system by Han *et al*. [23], which is more closely related to the model proposed in this paper and thus will be shortly reviewed later in this section.

According with the taxonomy proposed in [3], the model proposed in this paper falls in the last two categories, namely sleep-and-wake and overlay structure optimization. In fact, from one hand, the proposed model allows peers to switch between normal and sleep states to save energy; from the other hand, it introduces an additional layer to the peer-to-peer overlay for maintaining a distributed index of peers' availability. In what follows, some related systems belonging either to the sleep-and-wake or the overlay structure optimization categories are shortly reviewed.

One of the first theoretical works on the sleep-and-wake approach was proposed by Lefebvre and Feeley [14]. The authors argued that idle nodes should be turned off or placed in low-power hibernation to reduce energy consumption. They further argued that a peer-to-peer storage system should distinguish between powered off idle nodes and permanently departed ones, in order to avoid the significant data-copy cost of re-replicating data stored on idle nodes. The key to this distinction is to place an upper bound on the amount of time that nodes are allowed to sleep. Given the system's

replication factor, a theoretical model allows calculating an upper bound on the time an idle node can sleep without affecting the durability of the data stored in the system.

Following the same approach, Blackburn and Christensen [15] proposed a 'green' version of the BitTorrent protocol by introducing the concept of sleeping peers, which are considered temporarily unavailable to other BitTorrent nodes. A sleeping peer can wake up when contacted by other members of the BitTorrent network, in which case an active Transmission Control Protocol (TCP) connection to the latter is reestablished. Simulation results showed that green BitTorrent could consume less than 25% of the energy as standard BitTorrent (where all clients are always fully powered on) with a limited penalty in increased download time. Similar in aim and focus, the work by Lee *et al*. [16] introduces modifications in the BitTorrent protocol to enable idle peers to go into sleeping mode in order to reduce their energy consumption.

The sleep-and-wake approach is also the basis of the work by Gurun *et al*. [17], which proposes a scheduling approach to promote energy conservation in mobile peer-to-peer networks. Using actual energy measurements, the work derives the energy consumption of different types of operations in a peer-to-peer overlay. Based on these results, the authors introduced the concept of idle time. According to their approach, an energy-aware peer-to-peer protocol running on a mobile peer lets the peer go in sleep mode and thus preserve energy for the duration of the idle time period. When the peer wakes up, it can handle all buffered output requests and respond to incoming messages from other peers. The paper reports simulation results that indicate significant energy savings, obtained at the cost of some delay in message forwarding.

Among the systems based on overlay structure optimization, the work most related to the model proposed in this paper is the one by Han *et al*. [23]. It is a double-layered wireless peer-to-peer system in which files are searched mainly through the peers in the upper layer, called super peers, which are selected among those with the highest residual energy. Three energy-efficient routing schemes are used within the system. The first routing scheme tries to utilize the energy of the peers on the routes more evenly. The second scheme chooses a route with the 'strongest' peer among the peers each of which is the 'weakest' peer on a route. The third scheme selects a route with the second scheme among the routes with the smallest number of hops.

Table I summarizes the main features of related work in comparison with the system proposed in this paper (last row in the table). For each system, the table indicates (i) the energy-saving technique used; (ii) whether or not the system focuses on mobile networks; and (iii) whether or not it is based on a structured overlay (e.g., a DHT). As shown in the table, the proposed system is the only wired peer-to-peer network combining the scalability of a structured overlay with the effectiveness of the

Table I. Comparison with related systems.

System	Strategy	Mobile	Structured
Anastasi *et al*. [4]	Proxying	No	No
Andrew *et al*. [20]	Sleep-and-wake	No	No
Blackburn and Christensen [15]	Sleep-and-wake	No	No
Enokido *et al*. [8, 9]	Task allocation optimization	No	No
Gurun *et al*. [17]	Sleep-and-wake	Yes	Yes
Han *et al*. [23]	Overlay structure optimization	Yes	No
Hlavacs *et al*. [21]	Sleep-and-wake	No	No
Joseph et. al [11]	Location-based	Yes	Yes
Jourjon *et al*. [19]	Sleep-and-wake	No	No
Kelenyi and Nurminen [10]	Message reduction	No	Yes
Lefebvre and Feeley [14]	Sleep-and-wake	No	No
Lee *et al*. [16]	Sleep-and-wake	No	No
Leung and Kwok [22]	Topology control	Yes	No
Park and Valduriez [12]	Location-based	Yes	No
Purushothaman *et al*. [6]	Proxying	No	No
Sucevic *et al*. [18]	Sleep-and-wake	No	No
Tung and Lin [13]	Location-based	Yes	No
Proposed system	*Sleep-and-wake + Overlay structure optimization*	*No*	*Yes*

sleep-and-wake approach. In addition, thanks to its two-layer approach, it is also one of the few systems employing overlay structure optimization to improve the energy efficiency of a peer-to-peer network. The double-layer approach is shared with the system proposed by Han *et al*. [23], as described earlier. There are, however, three main differences with Han's system: (i) the reference scenario assumed in the proposed system is a wired peer-to-peer network, thus the residual energy of peers is not a concern; (ii) the proposed system relies on a sleep-and-wake strategy to achieve energy efficiency, rather than on energy-efficient routing schemes; and (iii) the upper layer of the proposed system forms a DHT overlay, which ensures logarithmic time complexity of file search operations, as demonstrated in Section 5.1.

3. DESIGN PRINCIPLES AND SYSTEM MODEL

The guiding principles that have been followed in the design of the system are (i) let peers autonomously decide on their availability status, to avoid the need for centralized coordination; (ii) keep a fully decentralized index of files, peers, and their availability status, to avoid single points of failures; and (iii) ensure scalable performance in file search operations.

The decision-making autonomy of peers is ensured by the ES algorithm, which let peers decide about their sleep–wake cycles based exclusively on local information (the number of files owned), as described in Section 4.

Decentralization is achieved by employing a DHT as indexing technology. Differently from other peer-to-peer systems based on a DHT, the proposed system is organized in two logical layers. This allows the nodes that are not part of the higher level to be switched off without affecting the overall indexing of files, which is the key for saving energy while preserving the system functionality. In addition, the proposed system extends standard DHTs by introducing an additional data structure, the *availability table* (AT), to keep decentralized information about the availability status of all the nodes in the network.

The use of a DHT as indexing technology also ensures scalable performance in file search operations. In fact, the FSR algorithm allows peers to find the files of interest with logarithmic performance bounds, as demonstrated in Section 5.1.

3.1. The two-layer model

The proposed system organizes the peer-to-peer network in two logical layers. The lower layer is composed of a set of *peer* nodes, each one providing a set of *files* to be shared. The upper layer is composed of a set of *super-peer* nodes, organized in a DHT-based overlay, whose purpose is indexing the peers and files at the lower layer.

Every peer, file, and super-peer is identified by an integer i in the range [0..2^m-1], where m is the number of bits used to represent the identifiers.[‡] For every i, the following assumptions hold: there is at most one peer with id i (denoted P_i); there is at most one super peer with id i (denoted S_i); there can be 0, 1, or multiple files with id i (all of them denoted F_i). Files with the same id are identical to each other; that is, they are multiple copies of the same data.

For the sake of explanation, Chord [24] is used for the super-peer layer, even though any other similar DHT-based system (e.g., Pastry [25], Tapestry [26], and Kademlia [27]) could be used instead. Following the Chord model, super peers are organized into a virtual circle with 2^m positions, where each super peer occupies the position corresponding to its identifier (i.e., S_i lies at position i of the circle).

Every super peer maintains a *finger table* (FT) and a *key table* (KT). The FT points to super peers at exponentially increasing distance, and allows locating the super peer responsible for any file identifier (or *key*) in $O(\log n)$ hops, where n is the number of super peers in the network. A super-peer S_i is responsible for a file F_j if $S_i = successor(F_j)$; that is, S_i is the first node that can

[‡]As usual in many peer-to-peer systems, it is assumed that identifiers are generated by applying a consistent hash function to a native identifier of each resource, so that identifiers are uniformly distributed across the range.

be found on the Chord circle clockwise from position j. The KT keeps association between each key the super peer is responsible for and the identifiers of all the peers providing files identified by that key.

Other than for indexing files, the DHT is used in this model also for indexing peers, as it is assumed that peers are not part of the Chord overlay. To this purpose, each super peer includes an additional data structure, called Availability Table (AT), which stores the *availability time* (described below) of every peer the super peer is responsible for. Following the principle used for indexing files, a super-peer S_i is responsible for a peer P_j if $S_i = successor(P_j)$. For every P_i, notation $S(P_i)$ is adopted to indicate the super peer responsible for P_i. It is assumed that P_i knows $S(P_i)$; that is, there is a bidirectional link between any super peer and the peers it is responsible for.

The availability time t_j of a peer P_j is an integer number. A negative value of the availability time (e.g., $t_j = -1$) means that P_j is currently *available*; that is, it can immediately respond to file download requests. Conversely, a positive value of t_j indicates that P_j is currently *unavailable* and that it will become available at time t_j. In this second case, t_j is measured using an appropriate time unit (e.g., number of seconds from a reference date).

It is assumed that all the hosts that run a super-peer instance run also a peer instance. Based on this assumption, hosts are classified either as *core hosts* or *edge hosts*. A core host runs one super peer and one peer, while an edge host runs only one peer. Therefore, both core hosts and edge hosts provide files to be shared, but only core hosts are necessary for indexing and discovery purposes. To preserve the indexing and discovery infrastructure, core hosts are assumed to be always available, while edge hosts can switch between normal and sleep power modes over the time to reduce energy consumption, based on the ES algorithm that will be described in Section 4.

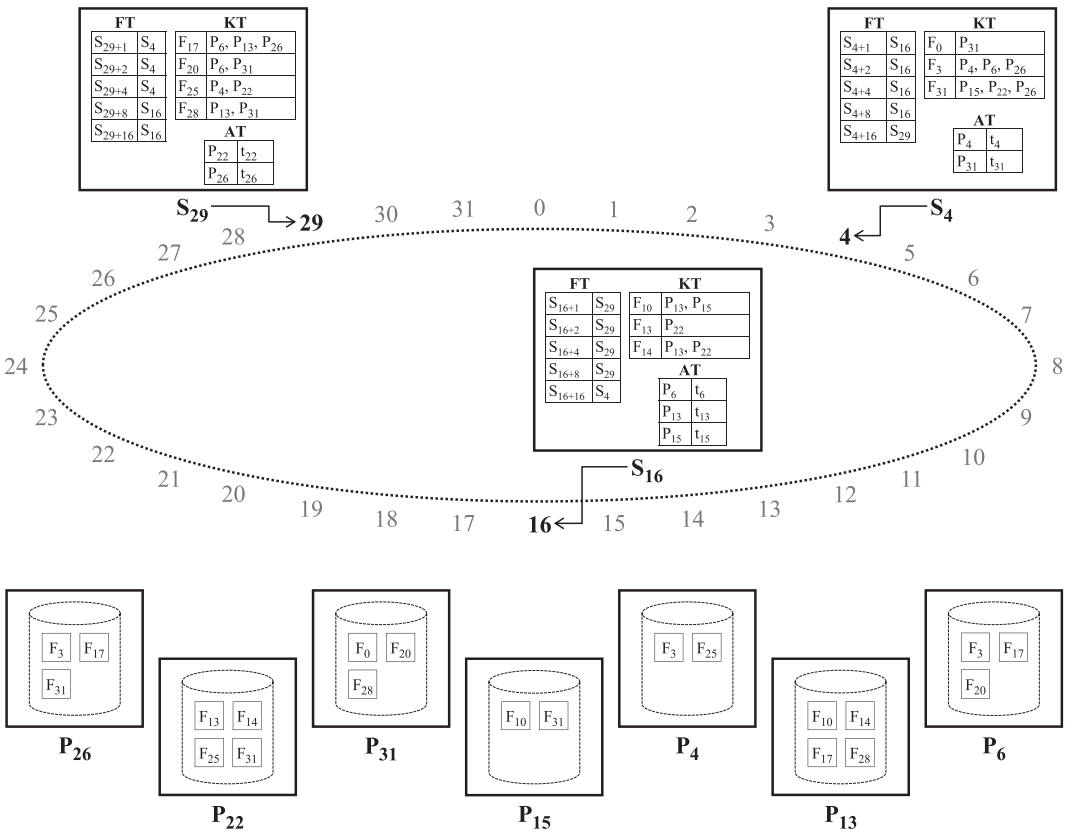

Figure 1. Example showing how files and peers are indexed by super peers.

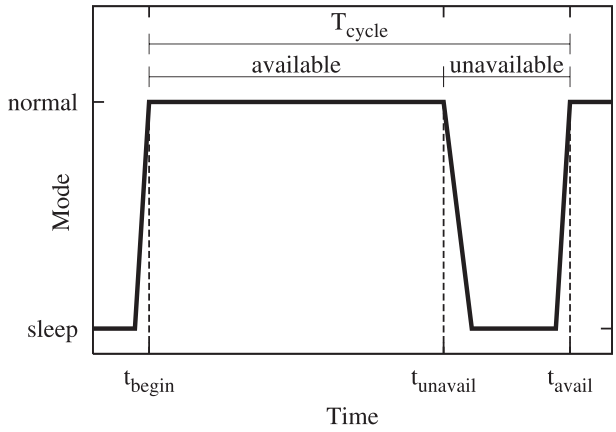

Figure 2. Relationship between availability status and power mode of a peer.

3.2. A small-scale network example

Figure 1 provides a small-scale network example showing how files and peers are indexed by super peers in the proposed model. In this example, a 5-bit identifier space is used; therefore the identifiers of peers, files, and super peers lie in the range [0..31]. The network includes three super peers and seven peers; each peer locally stores a set of files.

The KT of a super-peer S_i contains an entry for each file whose identifier lies in the range assigned to S_i. For example, S_{16} is responsible for the identifier range [5..16], because its predecessor on the circle is S_4. Therefore, all the existing files identified by a number in that range (F_{10}, F_{13}, F_{14}) have an entry in the KT of S_{16}. Each KT entry contains the identifiers of all the peers that locally store the corresponding file. For instance, from the first entry of S_{16}'s KT, one knows that file F_{10} can be provided by peers P_{13} and P_{15}.

Similarly, the AT of a super-peer S_i contains the availability time of each peer identified by a number in the range assigned to S_i. For example, because S_{29} is responsible for the identifier range [17..29], all the peers in that range (P_{22} and P_{26}) are listed in the AT of S_{29}.

It is worth noticing that, according with the system model introduced earlier, the logical network shown in Figure 1 is actually implemented as a system composed of three core hosts and four edge hosts: each core host running one of the three super peers and one of the seven peers; each edge host running one of the four remaining peers.

4. ENERGY-SAVING ALGORITHM

Following the sleep-and-wake approach, it is assumed that every peer running on an edge host periodically passes from *normal* to *sleep* power mode and vice versa. When a peer is in normal mode, it is available for download requests and works at normal power level (p_{high}). Conversely, a peer in sleep mode is unavailable, but it works at reduced power level (p_{low}), thus consuming a limited amount of energy.

Figure 2 illustrates the relationship between the availability status of a peer and its power mode. The standard duration of a sleep–wake cycle, indicated as T_{cycle}, is fixed and equal for all the peers. On the contrary, the *availability ratio* (AR), that is, the amount of time a peer is available in a cycle divided by T_{cycle}, changes from peer to peer. In particular, the principle followed in the proposed model is that the AR of a peer P_i depends on the number of files owned by P_i, as specified by the following equation:

$$AR = \begin{cases} AR_{min} & \textbf{if } NF < NF_{min} \\ AR_{max} & \textbf{if } NF > NF_{max} \\ AR_{min} + \frac{(NF - NF_{min})(AR_{max} - AR_{min})}{NF_{max} - NF_{min}} & \textbf{otherwise} \end{cases} \quad (1)$$

where NF is the number of files owned by P_i, $0 \leqslant AR_{min} \leqslant AR_{max} \leqslant 1$, and $0 \leqslant NF_{min} < NF_{max}$.

Given its AR, a peer can calculate the point of a cycle where it must change its status from available to unavailable. If t_{begin} is the time at which the cycle begins, the time at which the peer must become unavailable, $t_{unavail}$, is given by the following equation:

$$t_{unavail} = t_{begin} + AR \cdot T_{cycle} \tag{2}$$

Figure 3 shows the ES algorithm, which implements the strategy outlined earlier (see also Table II for the notation used in the algorithm). The algorithm is composed of two procedures: *power_management*, which specifies the cyclic sleep–wake operations performed by every peer P_i running on an edge host, and *publish_availability_time*, to publish the availability time of a peer on a super-peer S_i.

The *power_management* procedure works as follows. At the beginning of each sleep–wake cycle, t_{begin} is initialized (line 2), and the availability of P_i is published on the super peer responsible for P_i (line 3). Then, the next instant of time at which P_i will become unavailable, $t_{unavail}$, is calculated through Equation 2 (line 4). As soon as the current time is equal to $t_{unavail}$ (line 5), the peer calculates the next availability time t_{avail} (line 6) and publishes it on its super peer (line 7). At this point, the unavailability of P_i begins. However, before trying to go in sleep mode, P_i waits for an amount of time $T_{unavail\_to\_sleep}$ (line 8). This wait is carried out to give enough time

```
// executed by every peer running on a edge host
P_i.power_management()
 1: while true do
 2:    t_begin := t_current;
 3:    S(P_i).publish_availability_time(P_i, -1); // peer available (avail. time = -1)
 4:    t_unavail := t_begin + AR · T_cycle;
 5:    wait until t_current = t_unavail;
 6:    t_avail := t_begin + T_cycle;
 7:    S(P_i).publish_availability_time(P_i, t_avail); // peer unavailable (avail. time > 0)
 8:    wait until t_current = t_unavail + T_unavail_to_sleep;
 9:    wait until N_downloads = 0 and N_uploads = 0 and N_pending_queries = 0;
10:    if t_current ≤ t_avail − T_sleep_to_normal − T_normal_to_sleep − T_min_sleep then
11:       schedule wake up at t_avail − T_sleep_to_normal;
12:       go in sleep mode; // after wake up the peer restarts from here
13:    end if
14: end while

// ask S_i to publish the availability time of peer P_j
S_i.publish_availability_time(P_j, t_j)
 1: AT[P_j] := t_j;
```

Figure 3. Energy-saving (ES) algorithm.

Table II. Meaning of the symbols used in the ES algorithm (shown in Figure 3).

Symbol	Meaning
AR	Availability ratio of the peer
$t_{current}$	Current time
t_{begin}	Time at which the current cycle of the peer begins
$t_{unavail}$	Time at which the unavailability of the peer gets published
t_{avail}	Time at which the availability of the peer gets published
T_{cycle}	Duration of a sleep–wake cycle
$T_{sleep\_to\_normal}$	Amount of time for switching from sleep to normal mode
$T_{normal\_to\_sleep}$	Amount of time for switching from normal to sleep mode
T_{min_sleep}	Minimum duration of a sleep phase
$T_{unavail\_to\_sleep}$	Amount of time between unavailability and attempt to go in sleep mode
$N_{downloads}$	Number of downloads currently being performed by the peer
$N_{uploads}$	Number of uploads currently being performed by the peer
$N_{pending\_queries}$	Number of queries submitted by the peer still waiting for response

for requesting a file transfer to P_i, to those peers that received P_i's id as file provider in a query response just before that P_i published its unavailability. Before trying to sleep, P_i also waits until the number of downloads, uploads, and pending queries is equal to 0 (line 9). After these waits, P_i checks whether there is enough time to go in sleep mode, that is, if $t_{current}$ is not greater than t_{avail} minus the amount of times to switch from sleep to normal and vice versa, and the minimum duration of a sleep (line 10). If the condition holds, first P_i schedules a wake up at $t_{avail} - T_{sleep\_to\_normal}$ (line 11) and finally enters the sleep mode (line 12).

The *publish_availability_time* procedure is executed by a super-peer S_i when it is asked to publish the availability time t_j of a peer P_j it is responsible for. The availability time publication is performed by assigning t_j to the AT entry associated with P_j.

4.1. On the use of file popularity for availability ratio calculation

As discussed earlier in the section, the AR of a peer P_i depends linearly on the number of files owned by P_i, unless such number is smaller or bigger than given thresholds. This choice is based on the observation that, keeping powered on for a longer time the peers sharing a higher number of files, it is increased the chance that a file can be found and retrieved with no additional delay compared with a network in which all peers are always powered on. On the other hand, Equation 1 could be extended to take into account also other relevant parameters, such as the popularity of the files owned by a peer. Considering also the popularity of files is likely to further improve the system performance but would require two main issues to be addressed: (i) defining a scalable algorithm to estimate the popularity of all the files owned by each peer in the network and (ii) deciding which popularity-related parameter(s) should be used, and how AR should be related with such parameter(s).

For the first issue, it should be considered that naïvely estimating the popularity of all the files of a peer P_i has complexity $O(f \log n)$, where f is the number of files owned by P_i, and n is the number of super peers. In fact, for each file F_j owned by P_i, the DHT must be searched to find the super-peer S_k responsible for F_j. Then, S_k can estimate the popularity of F_j by counting the number of peers associated with F_j's entry in its KT. According to [24], searching the DHT for S_k requires $O(\log n)$ hops and messages, which when multiplied for the f files owned by P_i results in a complexity of $O(f \log n)$, as stated earlier. Because this operation must be performed for each peer, the overall complexity is $O(p \cdot \bar{f} \log n)$, where p is the number of peers, and $\bar{f}$ is the average number of files per peer. In a large-scale peer-to-peer network, this cost can be excessive, particularly if the popularity must be reevaluated frequently because of fast changing content. Therefore, a more scalable algorithm should be defined to efficiently estimate the popularity of all the files in the network.

For the second issue, assuming to have the popularity values of all the files of a peer P_i, it should be decided how to aggregate such values into one or more parameters to effectively summarize the overall popularity of P_i's content. Several popularity-based parameters may be used for a peer P_i, for example, the sum of the popularity values of all the files in P_i, their mean and standard deviation, or the top-k values. The choice of the most effective parameter(s) is likely to be greatly influenced by the distribution function of the popularity values, which can change over time. In addition, as mentioned earlier, it should be defined how to incorporate such parameter(s) into Equation 1, to make AR a (multivariable) function of their values.

As a concluding remark, it would be worth investigating in a future work how the AR of a peer could be adapted to reflect the popularity of the files owned by the peer, other than the number of files owned. Fundamental, to achieve this goal, is the definition of practical solutions to the issues discussed earlier.

5. FILE SEARCH AND RETRIEVAL ALGORITHM

After having introduced the ES algorithm, this section describes how files are searched and retrieved in the proposed model taking into account the availability/unavailability status of the file providers. Figure 4 shows the FSR algorithm, which is composed of the four procedures described in the succeeding text.

```
// search and retrieve file F_j
P_i.search_and_retrieve(F_j)
 1: S_k := S(P_i).successor(F_j);
 2: ⟨P_p, t_p⟩ := S_k.find_provider(F_j);
 3: if P_p ≠ null then
 4:    if t_p > 0 then
 5:       wait until t_current = t_p;
 6:    end if
 7:    download F_j from P_p;
 8: end if

// ask S_i to find a peer providing file F_j
S_i.find_provider(F_j)
 1: P_p := KT[F_j];
 2: if P_p ≠ ∅ then
 3:    ⟨P_p, t_p⟩ := S_i.best_provider(P_p);
 4:    return ⟨P_p, t_p⟩;
 5: else
 6:    return ⟨null, 0⟩;
 7: end if

// select the peer with the lowest availability time in P_p
S_i.best_provider(P_p)
 1: P_min := ∅;
 2: t_min := 0;
 3: for each P_j ∈ P_p do
 4:    S(P_j) := S_i.successor(P_j);
 5:    t_j := S(P_j).find_availability_time(P_j);
 6:    if P_min = ∅ or t_j < t_min then
 7:       P_min := {P_j};
 8:       t_min := t_j;
 9:    else if t_j = t_min then
10:       P_min := P_min ∪ {P_j};
11:    end if
12: end for
13: P_p := random peer from P_min;
14: return ⟨P_p, t_min⟩;

// ask S_i to find the availability time of peer P_j
S_i.find_availability_time(P_j)
 1: t_j := AT[P_j];
 2: return t_j.
```

Figure 4. File search and retrieval (FSR) algorithm.

A peer P_i starts the FSR algorithm through the local execution of the *search_and_retrieve* procedure, which receives the id of the file F_j to be searched and retrieved. As the first operation, P_i asks its reference super peer, $S(P_i)$, to find the successor of file F_j; the result, S_k, is the super peer responsible for F_j (line 1). Then, P_i asks S_k to find a peer providing file F_j; the result is a pair $\langle P_p, t_p \rangle$, where P_p is the providing peer, and t_p is the time when P_p will return available (line 2). P_p is *null* if S_k could not find any peer providing F_j. If P_p is not null (line 3), two cases are possible: *i*) $t_p > 0$, which means that P_p is not currently available; in this case, P_i waits until the current time is equal to t_p, before downloading F_j from P_p (lines 4–7); *ii*) t_p is negative, which means that P_p is currently available; in this case, P_i can download F_j from P_p without any delay (line 7).

The *find_provider* procedure is executed by a super-peer S_i when it is asked to find a peer providing a file F_j it is responsible for. First, S_i accesses the local key table, KT, to obtain all the peers, $\mathcal{P}_p$, that provide F_j (line 1). If $\mathcal{P}_p$ is not an empty set, then the best peer P_p is taken from $\mathcal{P}_p$ and returned to the caller in pair with the time when P_p will be available (lines 2–4). Otherwise, a pair of null values is returned (line 6).

The *best_provider* procedure of a super-peer S_i selects the peer with the lowest availability time from a set of file providers $\mathcal{P}_p$. Basically, the procedure retrieves the availability time of each P_j in $\mathcal{P}_p$ by asking $S(P_j)$ and stores in a set $\mathcal{P}_{min}$ all the peers having the lowest availability time, called t_{min}. The value of t_{min} will be equal to -1 if the nodes with the lowest availability time are currently available. At the end, one random peer from $\mathcal{P}_{min}$ is returned in pair with t_{min}.

Finally, the *find_availability_time* procedure is executed by a super-peer S_i when it is asked to find the availability time t_j of a peer P_j it is responsible for. This is performed by returning the value of the AT entry associated with P_j.

5.1. Complexity analysis

The section concludes with an analysis of message and time complexity of the FSR algorithm. To this end, let P_i be the querying node, F_j the file to be retrieved, m the number of peers that provide F_j, P_p the providing peer returned to P_i, and n the number of super peers.

In order to analyze the message complexity, assume that the FSR algorithm is divided into two main phases:

(1) The reference super peer of P_i, $S(P_i)$, performs one DHT lookup to find the super-peer S_k responsible for F_j (see *search_and_retrieve*, line 1). The lookup is performed by invoking the *successor*(F_j) operation provided by the DHT, which requires $O(\log n)$ hops and messages to other nodes, as proven in [24].
(2) Super-peer S_k finds P_p and returns it to P_i(*search_and_retrieve*, line 2). The search for P_p is carried out by invoking *find_provider*(F_j), which requires $O(m \log n)$ messages to complete. In fact, *find_provider* first gets from its KT the identifiers of all the peers, $\mathcal{P}_p$, providing F_j (*find_provider*, line 1). Then, it finds the best provider in $\mathcal{P}_p$, by invoking *best_provider*$(\mathcal{P}_p)$ (*find_provider*, line 3). In turn, *best_provider* performs one DHT lookup for each $P_j \in \mathcal{P}_p$ (*best_provider*, lines 3–4). Because the size of $\mathcal{P}_p$ is m and each lookup produces $O(\log n)$ messages, the number of messages exchanged during the second phase is $O(m \log n)$.

Given the number of messages exchanged during the two phases—$O(\log n)$ and $O(m \log n)$, respectively—the total number of messages generated by the FSR algorithm is $O(m \log n)$.

The time complexity can be analyzed taking into account that the total time to perform a lookup is proportional to the number of hops, or messages, needed to complete the lookup [24]. That being so, the time complexity can be conveniently evaluated by assuming the FSR algorithm divided into the two phases described earlier:

(1) For the first phase, the asymptotic time complexity equals the asymptotic message complexity, $O(\log n)$, of the single lookup performed by the *search_and_retrieve* procedure (line 1).
(2) For the second phase, it must be taken into account that the m DHT lookups performed by *best_provider* (one lookup for each $P_j \in \mathcal{P}_p$, see lines 3–4) are independent from each other and can be executed concurrently, taking care only to atomically update $\mathcal{P}_{min}$ and t_{min} (lines 6–11). Therefore, also the asymptotic time complexity of the second phase equals that of a single lookup, that is, $O(\log n)$.

Considering the whole of the two phases, it can be concluded that the asymptotic time complexity of the FSR algorithm is $O(\log n)$.

6. PERFORMANCE EVALUATION

As discussed in the previous section, the search for an existing file returns a reference to a providing peer that can be either available or unavailable. In case the providing peer is unavailable, the file download is postponed until the next availability time of that peer is reached. Therefore, energy saving is achieved at the cost of an average increase of the amount of time needed to begin the download of the file of interest.

The goal of this section is to evaluate the amount of energy saved by the proposed model and the delay it causes to file retrieval. To this end, the *total energy consumed by the network* over a period of

observation, E_{tot}, and the *average delay to retrieve a file*, D_{avg}, will be evaluated. As an additional performance parameter, the *average percentage of sleeping peers*, L_{avg}, will also be considered.

For comparison purpose, the values of E_{tot}, D_{avg}, and L_{avg} will be measured in two different cases: (i) the edge hosts periodically go in sleep mode by executing the ES algorithm (case referred to as 'ES') and (ii) the edge hosts do not execute the ES algorithm, thus remaining always in normal mode ('NOES' case).

6.1. Experimental methodology

A custom discrete-event simulator has been implemented to carry out the performance evaluation. The simulator works in three phases:

(1) A two-layer peer-to-peer network composed of N_{hosts} hosts is created, N_{core_hosts} of which are randomly selected to play the role of core hosts.
(2) N_{files} are assigned to the peers on the basis of the popularity of each file and the number of files that can be owned by each peer.
(3) The simulation begins with each peer performing both client-side activities (query submission and file downloading) and server-side activities (query processing and file uploading).

Each simulation terminates when the clock reaches a value T_{sim}, which represents the simulation length. At the end, the performance parameters E_{tot}, D_{avg}, and L_{avg} are calculated by the following equations:

$$E_{tot} = \sum_{t=1}^{T_{sim}} \sum_{i=1}^{N_{hosts}} p_i(t) \cdot \Delta t \tag{3}$$

where Δt is the time resolution of the simulator (1 s), and $p_i(t)$ is the power consumed by the i-th host at time t, which is equal to p_{low} if the host is in sleep mode at time t and p_{high} otherwise.

$$D_{avg} = \frac{1}{N_{queries}} \sum_{i=1}^{N_{queries}} t_{down}(i) - t_{sub}(i) \tag{4}$$

where $N_{queries}$ is the number of queries processed during the simulation, $t_{sub}(i)$ is the submission time of the i-th query looking for a file F_j, and $t_{down}(i)$ is the instant of time when F_j starts to be transferred from the file provider to the requestor.

$$L_{avg} = \frac{1}{T_{sim}} \sum_{t=1}^{T_{sim}} \frac{N_{sleeping\_hosts}(t)}{N_{hosts}} \cdot 100 \tag{5}$$

where $N_{sleeping\_hosts}(t)$ is the number of hosts in sleep mode at time t.

For the purpose of the present analysis, it is assumed that no hosts join and leave the network over time. In addition, network formation and maintenance are not addressed, as these aspects are beyond the scope of the present work.

6.2. Simulation parameters

Table III reports the input parameters used for the simulations. The network has a number of hosts ranging from 1000 to 10,000, the 15% of which configured to play the role of core hosts, according with available statistics about the composition of two-layer peer-to-peer networks (e.g., Gnutella 0.6 [28]). The number of distinct files is proportional to the number of hosts. According with many works in the literature (e.g., [29–31]), it is assumed that the file popularity follows a Zipf distribution, where FP_{max} is the number of instances of the most popular file, FP_{min} is the number of instances of the least popular file, and FP_{skew} is the skew factor of the distribution. The same distribution also applies to the number of files owned by the peers, where no peers own less than FO_{min} files, and FO_{skew} is the skew factor.

The simulator models query submissions as Poisson processes: the interarrival times of the submission events generated by any peer are independent and obey an exponential distribution with a given *query submission rate* (QSR). It is assumed that the QSR of a peer reaches its maximum, QSR_{max}, at a given time of the day, denoted T_{peak}, and distributes around the maximum following a Gaussian with standard deviation QSR_{stdev} (Figure 5). The value used for QSR_{max} corresponds to the average value reported in [32]. The value of T_{peak} assigned to a peer is a random real number uniformly distributed in the range [0..24[h. This aims at reproducing the behavior of a wide-area peer-to-peer network in which peers, being located in countries with different time zones, reach their maximum client-side activity at different hours.

The duration of a sleep–wake cycle, T_{cycle}, is equal to 600 s. The amounts of time to switch from sleep to normal mode ($T_{sleep\_to\_normal}$) and vice versa ($T_{normal\_to\_sleep}$) are taken from [33]. The

Table III. Simulation parameters.

Parameter	Meaning	Values
N_{hosts}	Total number of hosts (equal to the number of peers)	1000–10,000
N_{core_hosts}	Number of core hosts (each one running one peer and one super peer)	$0.15 \cdot N_{hosts}$
N_{files}	Number of distinct files in the network	$30 \cdot N_{hosts}$
FP_{max}	Maximum file popularity (the most popular file is owned by FP_{max} peers)	$0.1 \cdot N_{hosts}$
FP_{min}	Minimum file popularity (the least popular file is owned by FP_{min} peers)	1
FP_{skew}	Skew factor of the file popularity (Zipf-distributed)	1.2
FO_{min}	Minimum number of files owned (no peers own less than FO_{min} files)	1
FO_{skew}	Skew factor of the number of files owned (Zipf-distributed)	1.2
QSR_{max}	Maximum value of the query submission rate (queries/min.)	1.2
QSR_{stdev}	Standard deviation of the query submission rate (Gaussian-distributed)	0.5
T_{cycle}	Duration of a sleep–wake cycle	600 s
$T_{sleep\_to\_normal}$	Amount of time for switching from sleep to normal mode	4 s
$T_{normal\_to\_sleep}$	Amount of time for switching from normal to sleep mode	9 s
T_{min_sleep}	Minimum duration of a sleep phase	120 s
$T_{unavail\_to\_sleep}$	Amount of time between unavailability and attempt to go in sleep	8 s
$T_{sub\_to\_resp}$	Average amount of time between query submission and response reception	6 s
$T_{init\_transfer}$	Average amount of time to initialize a file transfer	4 s
$T_{transfer}$	Average duration of a file transfer	100 s
T_{sim}	Duration of a simulation run	86,400 s
p_{high}	Power consumed by a host in normal mode	120 W
p_{low}	Power consumed by a host in sleep mode	5 W
AR_{min}	Minimum availability ratio	0.25
AR_{max}	Maximum availability ratio	0.75
NF_{min}	Number of files owned under which the AR of a peer is equal to AR_{min}	{2, 5}
NF_{max}	Number of files owned over which the AR of a peer is equal to AR_{max}	{20, 50}

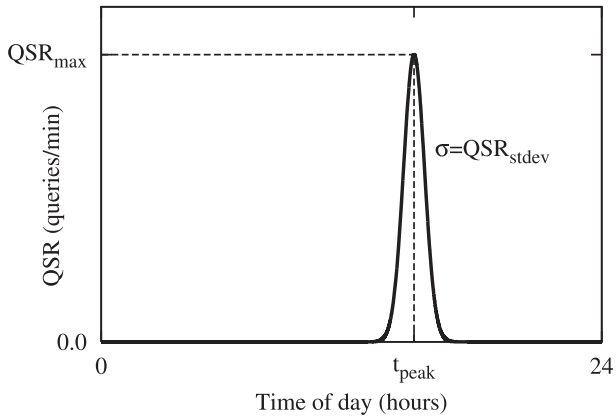

Figure 5. Query submission rate (QSR) of a peer as a function of the time of day.

minimum duration of a sleep, T_{min_sleep}, is set to 120 s (according to [14], usual values for this parameter range between 100 an 200 s). The table also reports the average values of the amount of time between query submission and response reception ($T_{sub\_to\_resp}$), of the amount of time to initialize a file transfer ($T_{init\_transfer}$), and of the duration of a file transfer ($T_{transfer}$).

According with [34], the power consumed by a host in normal mode, p_{high}, and that consumed in sleep mode, p_{low}, are assumed to be 120 and 5 W, respectively. The table finally reports the values of parameters AR_{min}, AR_{max}, NF_{min}, and NF_{max}, which are used in Equation 1 to calculate the AR of a peer.

6.3. Simulation results

In the first set of experiments, it has been evaluated how the shape of the AR function impacts on the performance of the system. To this end, fixed AR_{min} and AR_{max}, respectively, to 0.25 and 0.75, four combinations of values for NF_{min} and NF_{max} have been considered: (i) $NF_{min} = 2$ and $NF_{max} = 20$; (ii) $NF_{min} = 5$ and $NF_{max} = 20$; (iii) $NF_{min} = 2$ and $NF_{max} = 50$; and (iv) $NF_{min} = 5$ and $NF_{max} = 50$. Figure 6 shows how the AR function, as defined by Equation 1, appears using the four pairs of values $[NF_{min}; NF_{max}]$ considered.

One can notice that the higher NF_{min}, the higher the number of hosts with $AR = AR_{min}$. In contrast, the higher NF_{max}, the lower the number of hosts with $AR = AR_{max}$. As a result, one can expect that by choosing higher values for NF_{min} and NF_{max}, a lower AR is obtained, and therefore, longer periods are passed by edge hosts in sleep mode. To test this hypothesis, a network with $N_{host} = 5000$ has been simulated using the four $[NF_{min}; NF_{max}]$ combinations listed earlier. The results are presented in Figure 7.

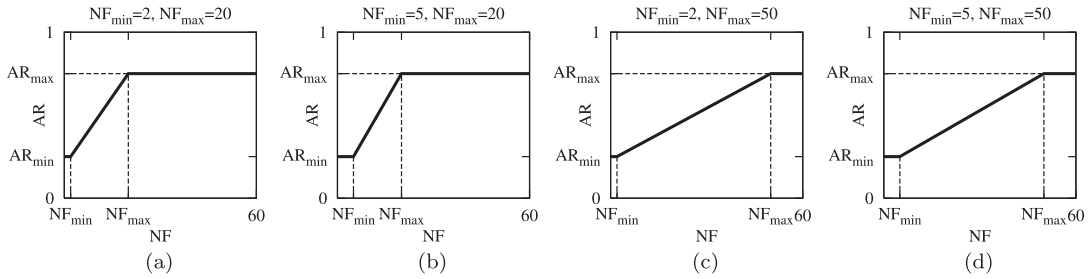

Figure 6. Availability ratio (AR) of a peer as a function of the number of files (NF) owned, with $AR_{min} = 0.25$, $AR_{max} = 0.75$, and the following $[NF_{min};NF_{max}]$ pairs of values: (a) [2; 20]; (b) [5; 20]; (c) [2; 50]; and (d) [5; 50].

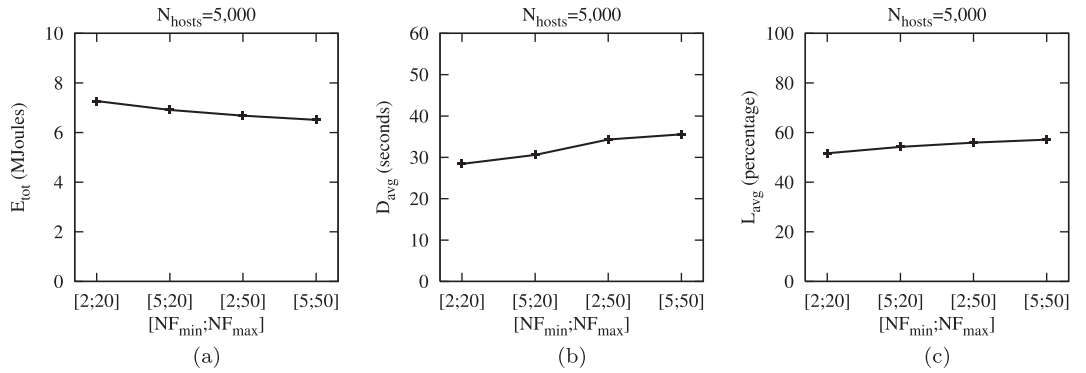

Figure 7. Simulation results on a network with $N_{host} = 5000$, using four $[NF_{min}; NF_{max}]$ combinations: (a) total energy consumed by the network (E_{tot}); (b) average delay to retrieve a file (D_{avg}); and (c) average percentage of sleeping peers (L_{avg}).

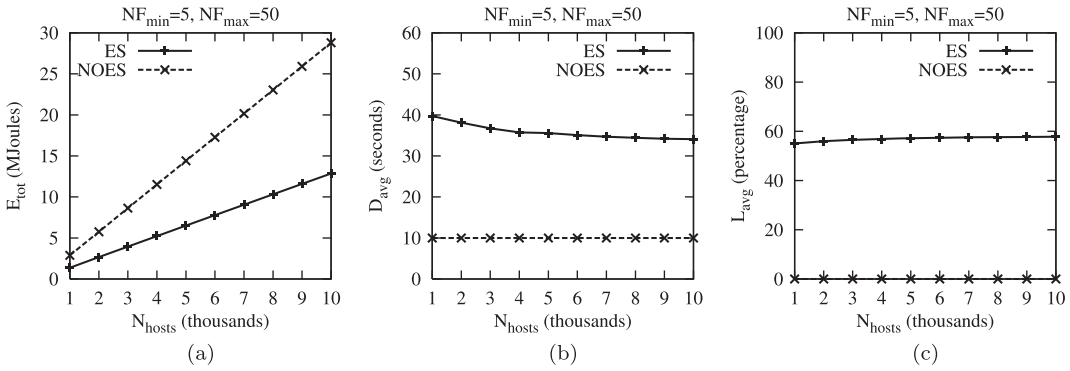

Figure 8. Simulation results on a network with N_{host} ranging from 1000 to 10,000: (a) total energy consumed by the network (E_{tot}); (b) average delay to retrieve a file (D_{avg}); and (c) average percentage of sleeping peers (L_{avg}).

The figure shows that the total energy consumed by the network decreases from 7.27 MJ with the combination [2; 20] to 6.51 MJ with the combination [5; 50]. At the same time, the average delay increases from 28.41 to 35.58 s, and the average percentage of sleeping peers increases from 51.67 to 57.17. As expected, the highest values of NF_{min} and NF_{max} lead to the highest value of L_{avg}, which in turn reduces E_{tot} and increases D_{avg}.

A second set of simulations has been conducted to compare the ES and NOES cases in a network with N_{hosts} ranging from 1000 to 10,000. For the ES case, $NF_{min} = 5$ and $NF_{min} = 50$ have been chosen as the default combination, because it produces a significant energy saving with a slight increase of D_{avg}. The results of the simulations are shown in Figure 8.

Figure 8(a) shows that, as expected, the total energy consumed by the network is proportional to the number of hosts. In the NOES case, in which peers do not execute the ES algorithm, E_{tot} passes from 2.88 MJ with 1000 hosts to 28.8 MJ with 10,000 hosts. In the ES case, E_{tot} passes from 1.36 MJ with 1000 hosts to 12.85 MJ with 10,000 hosts. Therefore, the amount of energy saved using the ES algorithm ranges between 53% and 55% with all the network sizes.

With regard to the average delay to retrieve a file, Figure 8(b) shows that the difference between the NOES and ES cases ranges between 24 and 30 s. In fact, in the NOES case, D_{avg} is equal to 10 s independently from the network size, while in the ES case, it passes from 39.7 s with 1000 hosts to 34.1 s with 10,000 hosts. The slightly higher value of D_{avg} registered with the smallest networks in the ES case is due to the higher impact that each single sleeping provider produces on a small-scale scenario.

Figure 8(c) shows that the average percentage of sleeping peers is almost independent from the network size. In fact, in the ES case, it ranges from a minimum of 55.1% with 1000 hosts to a maximum of 57.8% with 10,000 hosts. It is worth noticing that the percentage of sleeping peers is calculated over the total number of hosts, including the core hosts. Considering the edge hosts only, the number of sleeping peers in the ES case ranges from 64.8% to 68.0%. Of course, in the NOES case, the percentage of sleeping peer is always equal to 0.

The results presented earlier show that the additional delay to retrieve a file in the ES case on average is 24 to 30 s more than the NOES case. To better understand this result, the *delay* of each single query[§] in a network of 5000 hosts has been registered during 24 h of simulations. Figure 9 presents the results in the form of a cumulative distribution function. The x-axis represents time, while the y-axis represents the percentage of queries with delay lower than the corresponding time on the x-axis.

[§]According with the definition of D_{avg} formalized by Equation 4, the *delay* of a query Q looking for a file F_j is calculated as t_{down}-t_{sub}, where t_{sub} is the submission time of Q, and t_{down} is the instant of time when F_j starts to be transferred from the provider to the requestor.

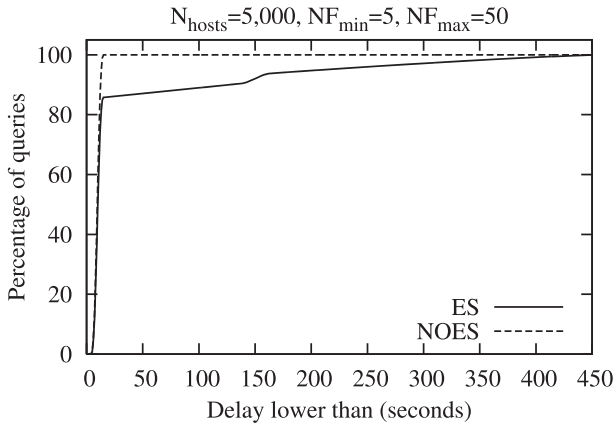

Figure 9. Distribution of the delays registered in a network of 5000 hosts during 24 h.

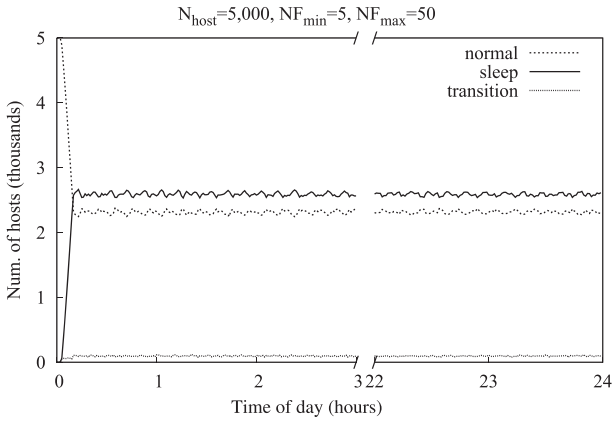

Figure 10. Simulation trace showing the number of hosts in normal and sleep modes over time.

The simulation results show that with the ES strategy, about 84% of the queries have no additional delay compared with the NOES case and that only 5% of the queries experience a delay greater than 180 s. These results demonstrate the good performance of the system also from a user's perspective.

Finally, it has been evaluated whether the energetic configuration of the network, expressed by the number of nodes in normal and sleep modes, is stable over time. The analysis of the simulation traces confirmed this point. As an example, Figure 10 presents a simulation trace for a network with $N_{hosts} = 5000$, showing the number of hosts in normal and sleep modes over 24 h.

At the beginning of the simulation, all the hosts are in normal mode. As soon as the edge hosts start the execution of the ES algorithm, the number of hosts in normal mode decreases and the number of hosts in sleep mode increases consequently. After a transient of about 10 min, the network reaches a steady state, and the ratio between normal and sleeping nodes remains constant up to the end of the simulation. At any time, a small fraction of the nodes are in a transition state (normal-to-sleep or sleep-to-normal). In summary, the trace data show that the energetic configuration of the network is maintained as the simulation proceeds, thus demonstrating the effectiveness of the ES algorithm over time.

7. CONCLUSIONS

Achieving energy efficiency in distributed systems is a challenging task, as it involves design and optimization of energy-aware algorithms, architectural models, and applications. This is particularly

true with peer-to-peer file sharing systems, given the necessity of obtaining significant energy savings without affecting the performance perceived by a large set of users.

In this paper, the problem has been addressed by proposing an energy-efficient peer-to-peer file sharing system organized in two logical layers. The lower layer is composed of a set of peers, providing the files to be shared. The upper layer includes a set of super peers, organized to form a DHT-based overlay, whose purpose is indexing files and peers, including their availability status. Following a sleep-and-wake approach, energy saving is achieved by letting peers cyclically switch between normal and sleep modes, where the time passed in normal mode by a peer depends on the number of files it provides. Simulation results showed that more than 50% of energy can be saved using the proposed model and that more than 80% of file downloads start with no delay compared with a network with all peers always powered on.

There are some aspects that provide room for future work. First, as pointed out earlier in the paper, it would be worth investigating how the availability ratio of a peer could be adapted to reflect the popularity of the files owned by the peer, other than the number of files owned. Another interesting topic for future research is finding a network configuration that minimizes energy consumption, for example, by dynamically selecting the super peers among the peers having the highest availability ratios.

REFERENCES

1. Schulze H, Mochalski K. *The impact of peer-to-peer file sharing, voice over IP, Skype, Joost, instant messaging, one-click hosting and media streaming such as YouTube on the Internet*. IPOQUE Internet Study 2007.
2. Erman J, Mahanti A, Arlitt M, Williamson C. *Identifying and discriminating between web and peer-to-peer traffic in the network core*. WWW 2007.
3. Malatras A, Peng F, Hirsbrunner B. Energy-efficient peer-to-peer networking and overlays. In *Handbook of Green Information and Communication Systems*, Obaidat MS, Anpalagan A, Woungang I (eds). Elsevier: Waltham, MA, USA, 2013; 513–540.
4. Anastasi G, Giannetti I, Passarella A. A BitTorrent proxy for Green Internet file sharing: design and experimental evaluation. *Computer Communications* 2010; **33**(7):794–802.
5. Cohen B. Incentives build robustness in BitTorrent. *Workshop on Economics in Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.
6. Purushothaman P, Navada M, Subramaniyan R, Reardon C, George AD. Power-proxying on the NIC: a case study with the Gnutella file-sharing protocol. *LCN*, Tampa, FL, USA, 2006; 519–520.
7. Ripeanu M, Iamnitchi A, Foster IT. Mapping the Gnutella network. *IEEE Internet Computing* 2002; **6**(1):50–57.
8. Enokido T, Aikebaier A, Takizawa M. A model for reducing power consumption in peer-to-peer systems. *IEEE Systems Journal* 2010; **4**(2):221–229.
9. Enokido T, Suzuki K, Aikebaier A, Takizawa M. Laxity based algorithm for reducing power consumption in distributed systems. *CISIS*, Krakow, Poland, 2010; 321–328.
10. Kelenyi I, Nurminen JK. Optimizing energy consumption of mobile nodes in heterogeneous Kademlia-based distributed hash tables. *NGMAST*, Cardiff, UK, 2008; 70–75.
11. Joseph MS, Kumar M, Shen H, Das S. Energy efficient data retrieval and caching in mobile peer-to-peer networks. *PERCOMW*, Kauai Island, HI, USA, 2005; 50–54.
12. Park K, Valduriez P. Energy efficient data access in mobile P2P networks. *IEEE Transactions on Knowledge and Data Engineering* 2011; **23**(11):1619–1634.
13. Tung Y-C, Lin KC-J. Location-assisted energy-efficient content search for mobile peer-to-peer networks. *7th International Workshop on Mobile Peer-to-Peer Computing*, Seattle, WA, USA, 2011; 477–482.
14. Lefebvre G, Feeley MJ. Energy efficient peer-to-peer storage. *Technical Report (TR-2003-17)*, Dept. of Computer Science, University of British Columbia, 2003.
15. Blackburn J, Christensen K. A simulation study of a new green BitTorrent. *ICC*, Dresden, Germany, 2009; 1–6.
16. Lee Y-J, Jeong J-H, Kim H-Y, Lee CH. Energy-saving set-top box enhancement in BitTorrent networks. *NOMS*, Osaka, Japan, 2010; 809–812.
17. Gurun S, Nagpurkar P, Zhao BY. Energy consumption and conservation in mobile peer-to-peer systems. *MobiShare*, Los Angeles, CA, USA, 2006; 18–23.
18. Sucevic A, Andrew LLH, Nguyen TTT. Powering down for energy efficient peer-to-peer file distribution. *ACM GreenMetrics*, Seattle, WA, USA, 2009.
19. Jourjon G, Rakotoarivelo T, Ott M. Models for an energy-efficient P2P delivery service. *PDP*, Pisa, Italy, 2010; 348–355.
20. Andrew LLH, Sucevic A, Nguyen TTT. Balancing peer and server energy consumption in large peer-to-peer file distribution systems. *GreenCom*, New York, NY, USA, 2011; 76–81.
21. Hlavacs H, Hummel KA, Weidlich R, Houyou AM, de Meer H. Modeling energy efficiency in distributed home environments. *International Journal of Communication Networks and Distributed Systems* 2010; **4**(2):161–182.

22. Ka-Ho Leung A, Kwok Y-K. On localized application-driven topology control for energy-efficient wireless peer-to-peer file sharing. *IEEE Transactions on Mobile Computing* 2008; **7**(1):66–80.
23. Han J-S, Song J-W, Kim T-H, Yang S-B. Double-layered mobile P2P systems using energy-efficient routing schemes. *ATNAC*, Adelaide, Australia, 2008; 122–127.
24. Stoica I, Morris R, Karger DR, Kaashoek MF, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for Internet applications. *SIGCOMM*, San Diego, CA, USA, 2001; 149–160.
25. Rowstron A, Druschel P. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware*, Heidelberg, Germany, 2001; 329–350.
26. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz JD. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications* 2004; **22**(1):41–53.
27. Maymounkov P, Mazieres D. Kademlia: a peer-to-peer information system based on the XOR Metric. *IPTPS*, Cambridge, MA, USA, 2002; 53–65.
28. Stutzbach D, Rejaie R. Capturing accurate snapshots of the Gnutella network. *INFOCOM*, Vol. 4, Miami, FL, USA, 2005; 2825–2830.
29. Jin H, Chen H. SemreX: efficient search in a semantic overlay for literature retrieval. *Future Generation Computer Systems* 2008; **24**(6):475–488.
30. Stutzbach D, Zhao S, Rejaie R. Characterizing files in the modern Gnutella network. *Multimedia Systems* 2007; **13**(1):35–50.
31. Goh S-T, Kalnis P, Bakiras S, Tan KL. Real datasets for file-sharing peer-to-peer systems. *DASFAA*, Beijing, China, 2005; 201–213.
32. Lv Q, Ratnasamy S, Shenker S. Can heterogeneity make Gnutella scalable? *IPTPS*, Cambridge, MA, USA, 2002; 94–103.
33. Agarwal Y, Hodges S, Chandra R, Scott J., Bahl P, Gupta R. Somniloquy: augmenting network interfaces to reduce PC energy usage. *NSDI*, Boston, MA, USA, 2009; 365–380.
34. Knezek G, Christensen RR, Tyler-Wood T, Lim O, Neaville WE. Going green with IT: a study of energy consumption by Home and School Information Technology Systems in the College of Information at the University of North Texas. *iConference*, Urbana-Champaign, IL, USA, 2010.

