Journal of High Speed Networks 20 (2014) 113–129
DOI 10.3233/JHS-140491
IOS Press

Energy saving mechanisms on high communication layers

Mohamed Oulmahdi [a,*], Christophe Chassot [a,b] and Ernesto Exposito [a,b]
[a] *CNRS, LAAS, 7 avenue Colonel Roche F-31400 Toulouse, France*
E-mails: {mohamed.oulmahdi, christophe.chassot, ernesto.exposito}@laas.fr
[b] *Université de Toulouse, INSA, LAAS; F-31400 Toulouse, France*

Abstract. With the large deployment of interconnected objects and systems, energy consumed by computing equipment is becoming more and more important, and saving energy consumption, particularly at the network layer, becomes a very pertinent issue. Energy saving solutions are generally considered at low communication layers; however, higher layers, typically transport and application ones, have not only an important impact on energy consumption, but also can negatively influence the energy saving mechanisms applied at lower MAC and physical levels. In this context, we propose two energy saving mechanisms for the transport and application layers of the Internet stack. The first one consists in an improvement of a classical but not well-known mechanism aimed at maintaining opened transport level connections. The second one is related to the adaptation of the qos offered to multimedia applications in order to minimize energy requirements of underlying layers. NS-3 simulation results show the benefits of each mechanism.

Keywords: Energy saving, multimedia application, transport layer, NS-3 simulation

1. Introduction

Recent years have seen significant evolution of computer networks dealing with their capacity and their deployment density. This has led to an important increase of their part of energy consumption, and statistical studies show that this part will increase more and more in the future [8]. The energy is therefore becoming a very important parameter on network equipment design. Although research work on energy saving is generally limited to low-level communication protocols, energy consumption can also be negatively influenced by high-level protocols behavior, typically at application and transport layers.

Among network equipment, end hosts (i.e., users terminals) represent a non-negligible energy consumption source, mainly because of their very large deployment.

In this context, this paper presents two energy saving mechanisms aimed at being deployed at end hosts:

- The first one is related to a classical but not well-known mechanism (the *keepalive messages* mechanism) whose implementation is shared between the Application and Transport (TCP) protocols. We first study and demonstrate the negative impact of this mechanism on the energy solutions applied at the lower layer, then we propose and evaluate the benefits of an energy saving alternative solution.
- The second mechanism is aimed at improving energy saving by taking advantage of the loss tolerance of multimedia applications. This tolerance is exploited at the transport level by reducing the provided qos to the minimum acceptance level required by the application. This leads to decrease the overall duration of the data transmission and consequently to increase the duration of the network interface sleep.

*Corresponding author. E-mail: mohamed.oulmahdi@laas.fr.

The following of the paper is structured as follows: Section 2 exposes a relevant state of the art related to the energy saving in the Internet and particularly at the high communication layers. Section 3 is dedicated to the energy saving models of network interfaces studied in this work. Sections 4 and 5 present in detail the two energy saving mechanisms proposed in this paper. Finally, Section 6 concludes the work.

2. Related work

This section presents a general survey of energy saving works on the Internet, and more specifically at the transport layer.

2.1. Energy saving on the Internet

The concept of energy saving is far from being a novelty in computer systems. Launched in 1989, Intel 486-DX processors already had mechanisms for efficient energy management [8]. Over the time, the needs in energy saving solutions have become increasingly important due to the continuing increase of computer equipment and their energy requirements. Several mechanisms have been proposed and have resulted in more efficient and more economical equipment. But this has been still not enough, because the consumption of energy is growing faster than its economy. Currently, and with energy consumption statistics estimated for the future Internet, energy saving has become one of the most active research fields.

This need can be expressed in two different contexts: a *constraint context* within which the energy source is limited, like in laptops and mobile phones, and an *optimization context* where the energy consumption is high enough, like in core networks and data:

- In a constraint context, the need is rather apparent. In this context, energy sources are limited due to the mobile nature of the terminals. Moreover, their running time depends as well on the use of these sources as on their kind [21]. It is consequently very important to carefully manage this energy;
- In an optimization context, the problem is completely different. Indeed, the availability of energy is not the main issue. The need is rather economic, strategic, and more recently, ecologic, giving rise to the green computing and green networking concepts [3,7,8]. According to a study provided by the IETF [20], the amount of energy consumed in the Internet is very important and figures are alarming. In United States, for example, Internet equipment use up to 7% of the total of energy consumption, which in turn corresponds to an annual amount of more than $250 billion and more than 180 million tons of CO_2. Several statistics also indicate that in the absence of energy efficient solutions, these figures will increase drastically by 2020 [8]. In such contexts, environments are very sensitive to any type of consumption, as small as it may be, as any optimization can lead to significant economic gains.

The rate of energy consumption in networked equipment is largely related to their performances (capacity and real use) [20]. Energy saving solutions must find a compromise between performances provided and energy consumed [3]. However, the problem is rather complex due to the fact that the relationship between performance and energy consumption is not always the same: for instance, decreasing the sending rate of a given network equipment does not necessarily lead to a reduction of the energy consumption; similarly, increasing a sending rate does not necessarily lead to increase its energy consumption. This relationship depends on the network nature [8], on the protocol layers that are involved in the transmission [5] and on the protocol mechanisms themselves [19,23,29].

Energy saving solutions are aimed at either *recovering energy losses* (i.e., by targeting loss sources in order to decrease them) or at *optimizing energy consumption* (i.e., by targeting high consumption sources in order to optimize them). It is not a rule but generally, solutions based on loss recovery are proposed for wireless networks, while optimization-based solution are proposed for wired networks.

- The first type of solution (aimed at recovering energy losses) tries to better manage the sources of losses, which constitute an important part of energy consumption. Solutions have been proposed both at the software (high) level by optimizing protocols [3,8], and at the physical level by optimizing the hardware architecture of the network [20];
- The second type of solutions (aimed at optimizing energy consumption) tries to reduce the bitrate at high energy consumption points. This may involve some periods or flow of network, some protocols or some behaviors of these protocols:
  - Periods concerned are those where the network is overloaded or has a very high bit error rate. Reducing retransmissions frequency in such cases allows reducing packets loss probability due to congestion or incorrigible bit errors;
  - Similarly, it is possible to avoid some greedy flows with as cost a non-significant decrease of performances;
  - Finally, some protocols are completely or partially unsuitable for some contexts and present therefore chaotic behaviors that leads to high loss rate or bandwidth wastage. In such situations, the decrease in bitrate can lead to significant energy savings. The reduction of the flow rate can be achieved by absorption, load sharing, or simply by reducing performances.

For several years, research on energy saving has been mainly focused on physical and data link layers, and in the early 2000s much of the work had already been done [3]. Although the economy rate of equipment has continued to increase, the applied solutions were always the same, and the rate of consumption evolves faster [8]. In parallel, the progresses of energy sources life of mobile devices are very low [3]. The introduction of new equipment with low energy consumption cannot be considered isolated from these trends and cannot constitute sufficient solutions [8].

For all those reasons, ideas began to turn towards higher layers with the aim of exploiting new opportunities and to propose new solutions. Studies have begun to focus on the network layer (like in [26,27]) and transport layer [5,6]. Regarding the transport layer, the studies are still now very modest. In addition, they are limited to wireless environments (generally WLAN). However, during the last three years, some studies began to focus also on the application layer as well as on the wired networks [16,24]. Synthetic surveys of all these solutions and studies can be found in [3,7].

2.2. Energy saving on high communication layers

Most of the works targeting the energy issues on high communication layers concern TCP. Several studies have been conducted in order to measure the energy performances of TCP; they conclude that good performances of the protocol do not always led to a good use of energy. Similarly, higher performance is not always synonymous of higher energy consumption.

TCP is recognized as having a bad behavior when the channel quality degrades. This is mainly due to the *congestion control* mechanisms implemented by TCP, which were designed under the assumption of a low error bit within the underlying networks. As this hypothesis is no more valid in wireless networks, *congestion control* mechanisms have a direct impact in energy consumption. More generally, TCP causes significant energy loss when the hypothesis of reliability of the medium is not verified. Those losses are largely due to the high frequency of retransmissions which most of the cases are unnecessary [29]. Also, it was verified that the versions of TCP congestion control mechanisms that respond poorly to the degradation of the channel are more energy efficient [29]. The reason is quite logical: when TCP invokes the congestion control mechanism, confusing losses due to transmission errors with those due to network congestion, it avoids many unnecessary retransmissions at link level.

Solutions proposed at high communication layers are very few and are based on the following two main principles:

- The first one consists to turn off the network interface when it is inactive [6,17]. The lack of activity may be related to the simple fact that there is no data to transmit or to the voluntary interruption of transmission by, for instance, the flow control mechanism. This principle is based on some assumptions that are however not

true in real situations. TCP must first have access to the network interface and its flow must be the only one at transport level. It is a type of solutions that is applicable in embedded systems. There is similar principle that uses the same idea, with the difference that instead of expecting that an inactivity period occurs, TCP uses this time to predict it based on previous activity scheme [2].

- The second principle is based on the *proxy* concept [18,24]. It consists in delegating a maximum work to the equipment having enough energy. This can be motivated either to decrease the load of equipment with limited energy sources, or to prevent the establishment of connections for transferring small data amounts. In the first case, the proxy station performs most of the work of routing, computing or other tasks that can be assigned to it. In the second case, the station initiates a connection only in the presence of a significant amount of data. Obviously, this technique should consider the quality of service requirements, which opposes the delayed data transfer mechanism.

In order to complete the lack of works at high protocol layers, and to contribute to energy saving in end-hosts Internet equipment, we propose in this work two energy saving mechanisms at high communication layers. The first one is situated at transport/application layers in a constraint context and follow the losses recovery principle. While the second is situated at the transport layer in an optimization context and follows the consumption optimization concept.

3. Energy saving models of studied interfaces

Providing energy saving solutions at the higher layer of the communication stack requires first analyzing what has been done at the lower level. This section is devoted to the presentation of the energy saving models that have been proposed and implemented for the three most used end-hosts interfaces, namely: IEEE 802.3 for wired networks, IEEE 802.11 for wireless ones and UMTS for 3G mobile interfaces.

Let us already note that there is a common characteristic for these three models that has an important impact in our work: they all consider two levels on energy consumption, *active* and *passive* mode levels. This means that if the interface is activated, the energy consumed is the same whatever the transmission bitrate might be.

3.1. IEEE 802.3

For the IEEE 802.3 interfaces, we based our study on the IEEE 802.3az standard (see Fig. 1) which is dedicated to the energy efficiency. As the other models, this energy saving model defines two energy states corresponding to only two energy consumption levels. The transition rule between the two levels depends on the presence or no of packets to send or receive in the buffer. Once the buffer is empty, the interface switches immediately to the passive mode, and the reverse process occurs when new packets are now present in the buffer to be sent or received. During the inactive (or *sleep*) mode, the interface periodically checks the presence of new packets. If any,

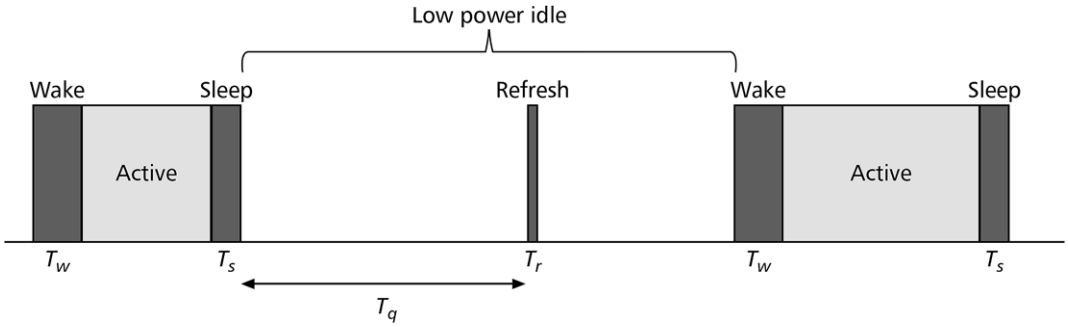

Fig. 1. IEEE 802.3az energy saving model [9].

the interface switches to the active mode, otherwise, it stays sleeping and will check again for new packet after the same period. Let us note that the time taken to switch from an interface mode to another has no significant impact on performances. The real values of all timers depend on the physical characteristics of the considered interface. As shown in Fig. 1, the active mode only appears during sending or receiving time. Thus, the time spent during the checking operation or to switch the interface from one mode to another is not considered in the period of active mode (the energy consumption of each one is slightly lower than in the active mode).

3.2. *IEEE* 802.11

For IEEE 802.11 interfaces, the energy saving principle is almost the same as for IEEE 802.3. The only difference is that an IEEE 802.11 interface does not check its buffers, but checks the access point. In practice, at the access point, there are two modes to check the receiving hosts activity before sending data to them: the *synchronous* and *asynchronous* modes.

- In the first mode (also called *proactive mode*), the receiving host and the access point agreed on a "wake up" interval.
- In the asynchronous mode (also called *reactive mode*), the end host is in charge of notifying the access point when it wakes up as well as of checking the presence of new packets.

Because of its large deployment, we only consider in this paper the asynchronous mode.

3.3. 3*G mobiles interfaces*

Let us now present the energy saving model for 3G radio interfaces (Fig. 2). The principle is similar to the previous models, but there is a small detail that leads to an important difference in the energetic behavior of the interface. Unlike to the previous models, the 3G interface activates an idle timer after sending the last packet, and before its expiration, the interface stays in active power mode. If other packets arrive during this time, the interface transmits them directly and the idle timer is reset. Otherwise, when the idle timer has expired, the interface enters in energy saving mode during which it uses almost no energy. However, the interface is periodically waked up to check if other packets arrived during its sleep. If new packets are present, the interface returns in active power mode in order to transmit them, otherwise it stays in power saving mode. Let us note that the checking period, called "DRX cycle", and the idle timer value are two important factors in the energy saving efficiency of the radio interface. A detailed study of those two parameters can be found in [28]. The idle timer is a specificity of 3G mobile

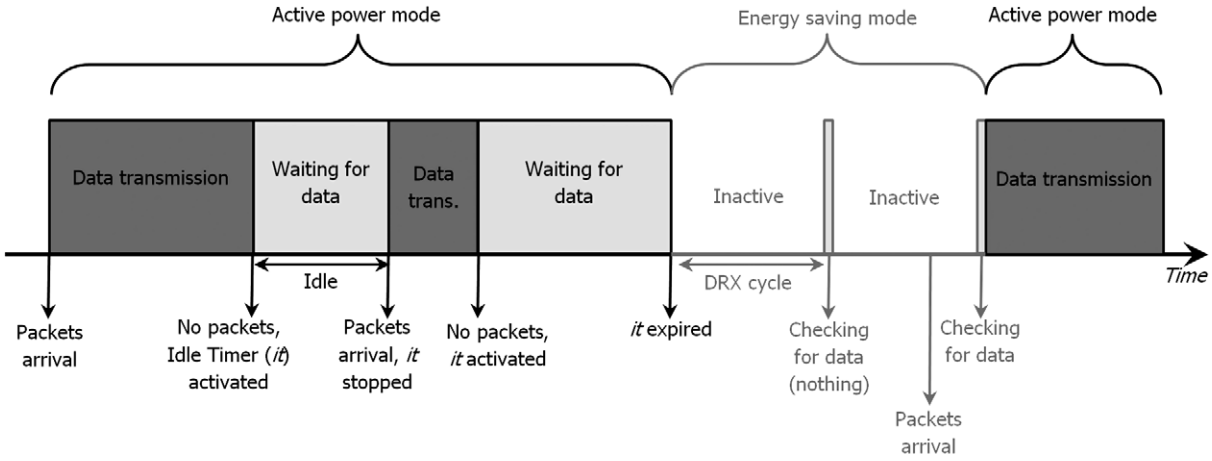

Fig. 2. Energy saving model for 3G UMTS systems.

interfaces. The main reason that explains this difference is the important energy that is consumed for switching the interface from a mode to the other; this is not the case for the others.

As previously written, the energy consumption in the active mode of all previous interfaces is the same whatever the transmission bitrate. By the way, the real energy consumption in active and inactive modes, and the necessary energy to switch between those two modes and for checking packets presence, depend on the physical characteristics of each equipment. Thus, to make our study as general as possible, we only consider the activity time of the interface; it is a sufficient metric to express the general energy consumption.

Based on this analysis of the energy saving model of the three main end-hosts interfaces, the following Sections 4 and 5 present two different contributions aimed at illustrating the importance of considering the energy saving issue not only at the lower layers but also at the upper application and transport layers.

4. 1st mechanism: Reducing energy cost of keepalive messages

This section presents the first energy saving mechanism proposed for the application and transport layer. This mechanism is aimed at solving the negative impact of "*keepalive messages*" exchanged at the transport level with the aim of maintaining active the opened TCP connections. An alternative energy saving-aware solution is proposed and evaluated by simulation through NS-3.

4.1. Keepalive mechanism description

The *keepalive* mechanism is a classical but not well-know mechanism performed between TCP-based applications and TCP itself. It allows the server of a client/server application to check the client connectivity when it uses permanent TCP connections.

The notion of *keepalive* messages appeared after the changes in HTTP protocol for the duration of TCP connections. Initially, HTTP has been designed to create a TCP connection for each file to be transmitted. In 1999, and for some performances-oriented motivations, HTTP designers replaced this method with *permanent connections*. By this way, a TCP connection could be used to transfer multiple files [11,13]. The problem is that, in contrast with the initial architecture of HTTP, this method cannot allow differentiating client inactivity and network failure. For this reason, a new mechanism to ensure this differentiation, called *keepalive mechanism* has been created.

The principle of this mechanism (see Fig. 3) is to periodically send from the server to the client, a kind of test to verify the client connectivity state. This test is a TCP acknowledgement (ACK) message with a sequence number of one unit less than the expected sequence number. When it receives this ACK, the TCP client replies to the client with another ACK having a sequence number of the last segment correctly received. When the server receives this ACK, it can confirm to the server that the client is active.

The *keepalive* mechanism is managed by the server. The client does not even realize that it has answered to a *keepalive* message. Once the connection is established, the server waits for a data request. After a given period, which is implemented by an "idle timeout", if the server receives nothing from the client, it goes to another state to check the client activity by sending it periodically *keepalive* messages. If the client replies, this means that it is always connected; otherwise the server can deduce that the client is offline.

Both TCP and the application level manage this mechanism. The application chooses if the mechanism has to be used or not, and determines the values of the different mechanism's parameters. Once the mechanism is activated, TCP retakes its management [14].

The *keepalive* mechanism is not part of any standard, but its use is so large that it has been integrated in major implementations like Linux and Windows sockets. Its integration is also approved by the IETF, subject to certain conditions [15].

Theoretically, the energetic impact of the *keepalive* messages may be explained by two reasons. The first one is the energy required to the transfer of these messages. More important, the second one is related to the energy saving state of the network adapter (the host interface). Indeed, *keepalive* messages prevent the network adapter to

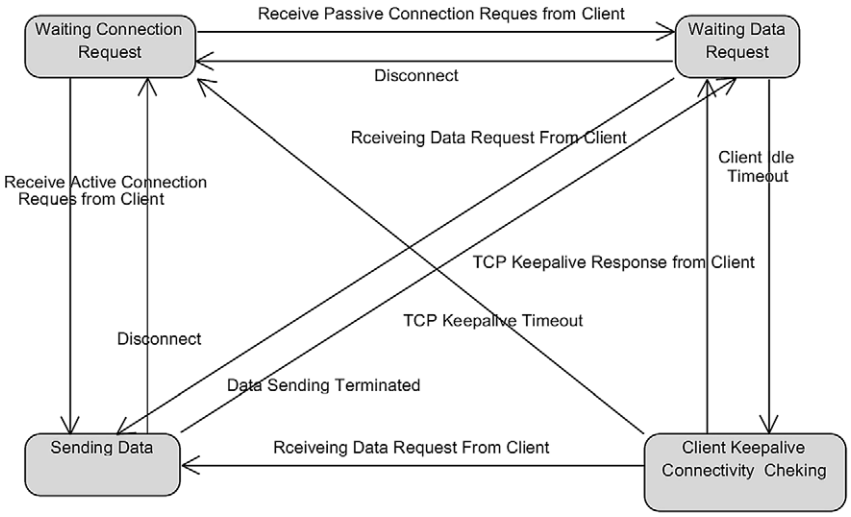

Fig. 3. Keepalive mechanism machine-state.

be put in standby (i.e., inactive) mode. The frequency of the transitions between standby and active modes is also to be considered.

Let us note that the application/transport layers are not the only ones using *keepalive* messages. For instance, this principle is also implemented in routing protocols with the aim of seeing the other routers' activity. In our work, we only treat the *keepalive* mechanism applied at the application/transport levels.

4.2. Proposed solution: *A sender based keepalive mechanism*

Since the *keepalive* mechanism is needed despite its important energy cost, the idea is to replace it by another mechanism, named *sender-based keepalive mechanism* that meets the same requirements but following an energy saving approach. This mechanism has mainly to be implemented in the HTTP application since all the considered applications use this protocol. Initially, both client and server must negotiate the use of this mechanism.

With the sender-based *keepalive* mechanism, the server does check the client connectivity; indeed, it is the client that has to notify the server that it does not need more data for an estimated period of time, while still staying connected.

States machines depicted in Figs 4 and 5 provide a more detailed view of the mechanism behavior. Figure 2 provides the server side and Fig. 3 the client side.

When the server establishes an http session, it either waits a GET request or a *standby notification* from the client, otherwise it considers that it is offline and cuts off the connection. The client can send a standby notification after receiving all the needed data or directly after the connection establishment, depending on if it needs immediate data or not. In both cases, the server waits nothing from the client during the estimated standby duration that is sent by the client within the standby notification. The estimated duration depends on the nature of the application and it is to each client application to measure it.

Once the client is in the *standby* state, it is not obliged to wait for all estimated standby duration to request data from the server: in this case, it has only to send an ordinary GET request to the server. Also, even if this duration expires, the client is not obliged to request data from the server: in this case, it has simply to renew its standby notification with a new estimated standby duration. Finally, if the client has not renewed a standby notification, the server considers that the client is offline and then cuts off connection.

Let us note the following important point: with this strategy, the only energy cost that is needed corresponds to the sending of the standby notifications. However, since the interface stays always activated after receiving the last packet, the sending of a standby notification is always performed in an active mode and thus does not generate any supplementary energy cost.

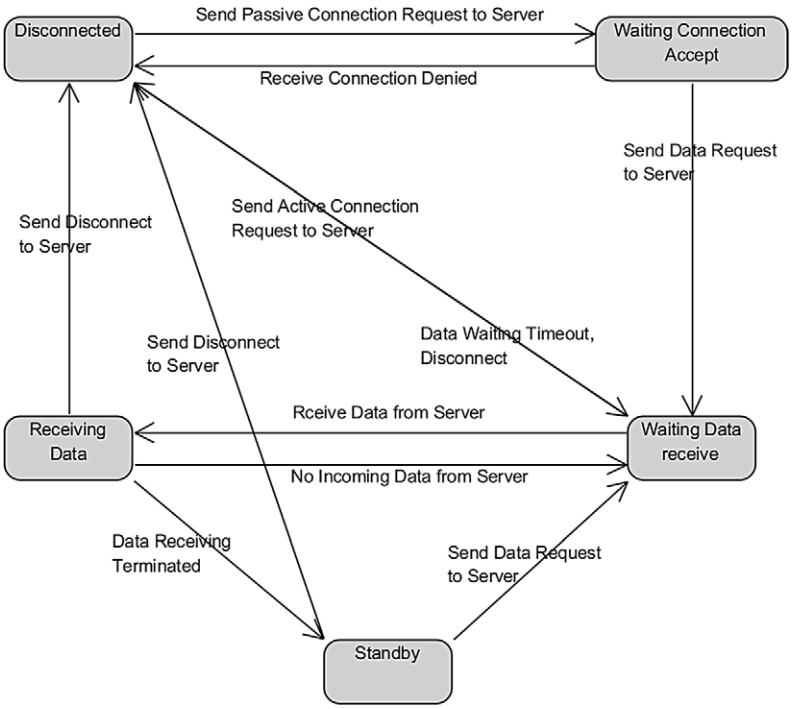

Fig. 4. State machine for application using standby notification: server side.

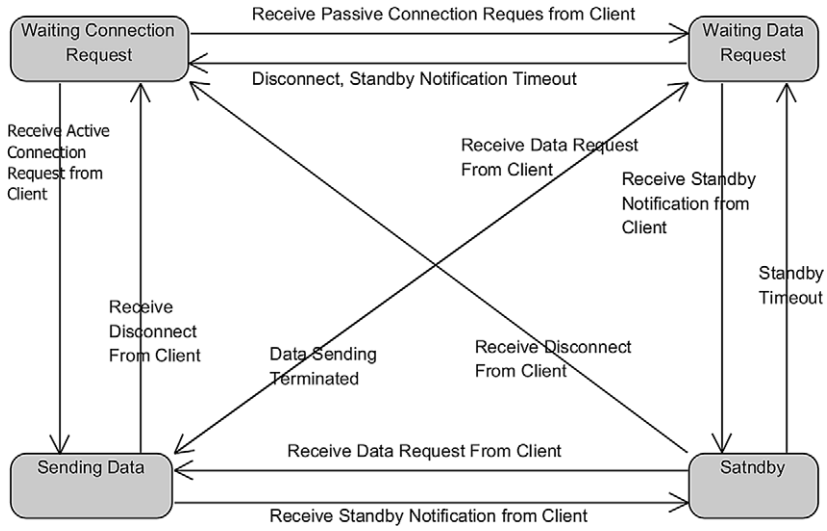

Fig. 5. State machine for application using standby notification: client side.

4.3. Experimentations

In this section, we study the impact of both classical and sender based *keepalive* mechanism on the interface activity, and we measure their energetic cost, expressed in terms of activity duration of the radio interface.

Let us note that only the applications whose traffic scheme presents inactivity periods are really concerned. We then only consider such type of applications chosen from previous works published in [4] that present traffic scheme of the most used applications in the Internet. Those applications are *Gmail* (web mail), *Deezer* (Music) and *Reader* (News). The traffic scheme of these applications is presented in Fig. 6.

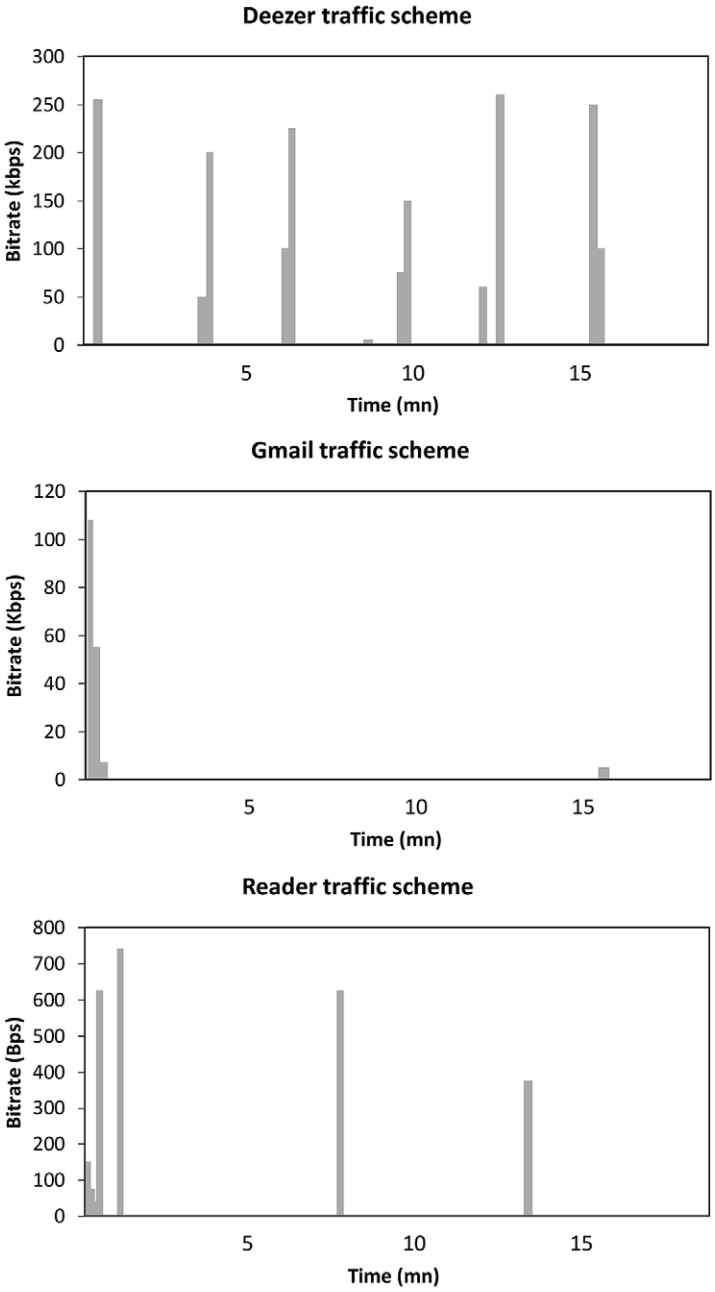

Fig. 6. Traffic schemes of selected applications [4].

4.3.1. Methodology and simulation tools

The methodology applied for the different measurements is the following one:

- In a first step, we measure the activity of the interface when it transmits traffic of the selected applications, using the sender-based *keepalive* mechanism;
- In a second time, the same traffics are transmitted using this time the classical *keepalive* mechanism, and we measure the activity of the interface;
- We finally compare the two set of results and we estimate the (expected) benefits induced by the sender-based *keepalive* mechanism.

Experimentations have been performed using the NS-3 simulator. The real values of the different parts of the energy model described in Fig. 1 are not standardized and depend entirely on the constructor. The values of all the parameters that have been used in simulation are based on values used in Nokia laboratories [12]. Let us insist that the goal was not measuring the real energy consumption, but only the activity duration of the radio interface.

For this mechanism, we only present results for 3G mobiles interfaces because *keepalive* message have no impact on the other interfaces. In fact, the two others interfaces switch to energy saving mode directly after sending the last packet in the buffer, and it is this difference which causes energy losses on 3G interfaces.

4.3.2. Interface activity with sender-based keepalive mechanism

Simulation results associated with the measurement of the interface activity with activation of the sender-based *keepalive* mechanism are presented through the gray vertical lines of Fig. 7.

The first (*and already well-known*) *point that can be observed is that the energy consumption of the radio interface does not depend on the transmission bit rate. Indeed, we can see that at each activity of any application is associated a peak of power consumption, whatever the corresponding bit rate may be.*

A more interesting point can also be noted related to the difference between the transmission lengths and the corresponding interface activity lengths. Indeed, the interface activity duration is more important than the transmission length in all cases. This is due to the important time during which the interface waits before turning into energy saving mode.

4.3.3. Interface activity with classical keepalive mechanism

Let us now analyze the measures of the interface activity for the same traffics, but with the application of the classical *keepalive* mechanism.

The frequency of *keepalive* messages depends only on the application. For the simulations, we used the default value that is 75 seconds, defined by the constant TCP_KEEPALIVE_INTVL in file *linux/include/net/tcp.h*.

Simulation results are presented on the gray and dark vertical lines of Fig. 5. The *keepalive* energy cost is represented by the dark lines. We can easily observe the impact of the *keepalive* messages generated by the classical *keepalive* mechanism on the interface activity.

Let us note that the impact due to the effective sending of the messages is not the problem. The real cause is the time spent by the interface to wait before moving to the energy saving mode. Indeed, the interface must be waked up for each *keepalive* message arrival, and although it sends back an acknowledgement quickly, it stays activated pending possible new messages.

4.3.4. Comparison between the two simulation sets

The different values of the interface activity corresponding to the previous simulations are summarized in Table 1.

The impact of the classical *keepalive* mechanism is more important for application having longer activity duration. This is logical because more time the application is inactive, the number of needed *keepalive* messages become more important. Let us note that the ratio in the table is just an indicator value which is related to the measurement length (20 minutes here); its value increases when time increases.

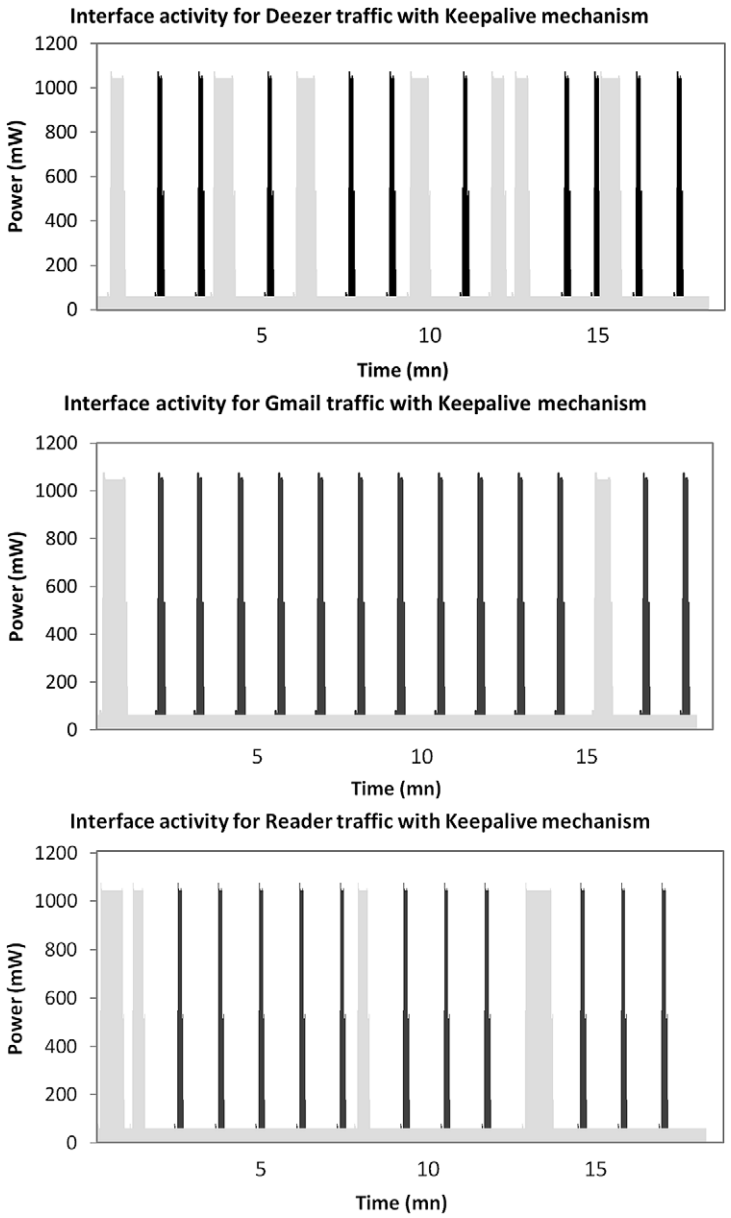

Fig. 7. Interface activity for applications traffics with/without keepalive mechanism.

Table 1
Value of the interface activity with sender-based/classical keepalive mechanism

Application	Interface activity with sender-based keepalive, s	Interface activity with classical keepalive, s	Ratio
Gmail	81	263	3.24
Deezer	253	376	1.48
Reader	139	308	2.21

5. 2nd mechanism: An energy aware TCP for multimedia streaming

In this section, we present the second mechanism that consists on adding a new functionality to TCP allowing making it capable to know the application tolerances in terms of reliability (e.g., acceptable loss rate) with the aim of reducing the offered qos to the minimum required. Multimedia applications (based on audio and/or video codec) are the targeted ones. The goal is in turn to reduce the global data transfer time, and to reduce consequently the needed energy to transmit it.

5.1. Proposed mechanism

The proposed solution is aimed at optimizing the overall energy consumption. The idea is to take advantage of the loss tolerance of multimedia applications with the aim of reducing the number of packets transmitted, while still providing the minimal qos required by the application in terms of reliability. The reduction of transmitted packets can occur by reducing transmissions or retransmissions:

- In the first case, i.e., when the qos that can be offered is greater than the minimal quality required by the application, we voluntary eliminate (at the sending side) a set of packets in order to reduce the qos offered to the application, while satisfying its minimal need;
- In the second case, a lost packet is retransmitted (regardless the reason of its loss) only if the application tolerance has been reached. The difference with the first case is that packets are not lost voluntarily. Nevertheless, the reduction of retransmissions is theoretically more energy-efficient for several reasons. The most important one is that the retransmission of lost packets induces energy loss because packets are transmitted several times but received only once. Additionally, the risk of loss of a retransmitted packet is higher if the first loss is due to congestion [3]; thus, if we do not retransmit it, we avoid losing energy on sending uselessly a packet that will likely loss. However, we will show later that in practice, this is not true.

The expected contribution of our proposal is to optimize the energy consumed by the end hosts during multimedia data transfer, knowing that their consumption part in the overall Internet traffic is not negligible [8], and that their traffic scheme is generally characterized by an absolute continuity [4], which makes energy saving mechanisms at low layers completely inefficient. This absolute continuity implies that the number of transitions between active and inactive modes have no effect, because the interface is activated during all the download period. Thus, our mechanism will not influence it; it will just send some packets instead of some others.

Our proposition can only be applied to particular types of multimedia applications such as video streaming like youtube, which are based on pre-recorded files stored in a server that it displays gradually during their download. For other applications, such as TV streaming or visio-conferencing systems, we cannot modify the downloading duration of the multimedia data because they are generated in real time. By the way, we suppose that the end users do not launch simultaneous multimedia streaming applications in parallel. So, when the downloading is finished, and until the end of its playing, there will not be another multimedia downloading susceptible to continuously activate the network interface, which in turn, can sleep. Thus, it is this time separating the end of data downloading and the end of its playing that we try to maximize in order to maximize the time of the network interface sleeping.

Our solution is to be applied within the underlying transport protocol, for instance TCP in our study. The protocol has to be modified to allow it, on one hand to be able to know the applications tolerable loss rate, and on the other hand to manage the transmissions and retransmissions accordingly. Our study is limited to multimedia applications running directly over TCP, and do not target applications using the HTTP protocol.

Once the information is known by TCP (during the connection set up), the client (receiver) and server (sender) share the task of minimizing packets transmissions and retransmissions accordingly.

The basic algorithm is the following one. We first estimate the number of packets lost. If the corresponding loss rate is greater than or equal to the one tolerated by the application, no packets are discarded. By cons, there is no retransmission. If now the loss rate is less than allowed by the application, the server (packets sender) voluntarily discards some packets until reaching the loss rate tolerated by the application.

By this way, we can theoretically reduce the amount of data transmitted during the video transfer by a fraction equal to the loss rate tolerated by the application. For instance, if the application tolerates 10% of loss, the data amount will be reduced by 10% of the total amount required for its transfer without loss. Let us note that this decrease is not related to the amount of data received by the client, but to the one that is sent by the server (taking into account the retransmissions). For instance, if a video requires 15 minutes to be transferred without loss in the case of a fully reliable network, and 20 minutes in an unreliable network, then for the same application tolerating (for instance) 10% of loss, the transfer time will be 18 minutes (and not 15). By this way, as the video is longer and the loss rate is higher, the reduction of the transfer time will be more important. This reduction will of course result in reducing the network interface activity and will lead therefore to reduce the energy consumption.

5.2. Implementation details

Work to be done deals with the TCP transport protocol and first consists in the definition of a mean allowing this protocol to know the application tolerance in terms of data loss; secondly, it consists in managing TCP transmissions and retransmissions accordingly in an energy saving approach guarantying the minimal qos required by the application.

Above all, this mechanism must be implemented so that the application using it will have the possibility to use it or no, according to the IETF exigencies for all non-standard functionality (the policy of the mechanism's use is not studied here). Thus, the application must explicitly define, during the connection initialization process, if it wants or not to use the mechanism. On its turn, the TCP client must negotiate the mechanism to be applied with the server entity when it requests the connection. If the TCP server does not accept or does not support this mechanism, its usage will be refused.

During the connection set up, the data loss tolerance information will be transmitted by the application to TCP in order to be known by both server and client TCP entities. This information must reach the two extremities because they share the transmission and retransmission management. The client must determine if lost packets have to be retransmitted, and the server has to determine if the network loss rate is lower than the application tolerance. Client and server's work are complementary.

The mechanism has been implemented based on the cumulated acknowledgement policy applied by TCP. The difference is that the TCP client acknowledges both received and lost packets, and in order to inform the TCP pair entity with the number of lost packets, the TCP client also transmits the number of not received packets within the cumulated acknowledgement. By this way, the two TCP entities have a real time vision of the loss state of the application flow.

On these basis, four cases may occur:

- The network is fully reliable: the whole work is delegated to the server which has to eliminate as many packets as the application tolerance;
- The network is not fully reliable and its loss rate is equal to the application tolerance: the client then makes the whole work by acknowledging all lost packets;
- The network is not fully reliable but its loss rate is lower than the application tolerance: the client and the server share the work; the client acknowledges all the lost packets and the server eliminates the necessary packets number to reach the acceptable loss rate;
- The network is not fully reliable but now its loss rate is higher than the application tolerance: the whole work is delegated to the client, which acknowledges the allowed part of lost packets and reports the loss of the rest.

Thus, we can guarantee that the number of transmitted packets by TCP does not exceed the minimum required by the application, and that the number of lost packets does not exceed the application tolerance. This leads to a minimization of the data transmission duration and a satisfaction of the application qos needs.

5.3. Experimentations

As previously shown, the studied mechanism is based on the management of data losses. So, the only parameter that differentiates the considered interfaces from the point of view of the mechanism is the loss rate (regardless of the loss reason); all the other network parameters (like bitrate or topology) have no influence on the mechanism.

Therefore, we have based our simulation on this principle and we considered a basic point-to-point link with various loss rates; we then have measured the difference in the data downloading time. Finally, we have analyzed the obtained results regarding the interfaces characteristics.

Since the aim is to demonstrate that such a mechanism can reduce the data transfer duration, we have simplified the simulation parameters, but this does not influence the results in a real application of the mechanism. So, we have considered that the mechanism was not able to understand the data semantic (different importance of images types for example), and we have not considered the received file quality. We just start with application data loss tolerance principle. A complete implementation of the mechanism should differentiate images importance so as to eliminate only the non-critical ones in order to provide an acceptable video quality. This cannot be achieved simply by modifying the TCP protocol, and should require a more adapted transport protocol.

We used four simulation scenarios with different network loss rates corresponding to the four cases shown previously. We measured the transfer time of the same multimedia file both with and without the mechanism. The time gained with the mechanism can be interpreted as an increase in the interface sleep time. The multimedia file is an FLV video whose length is 7 minutes and 21 seconds (360p, H264, MP3). We considered that the application loss tolerance is of 10%. Simulations have been conducted under the NS-3 simulator.

Results obtained for each scenario are summarized in Table 2.

We note that the proportion of time recovered with this mechanism is around the loss rate tolerated by the application (10% here). However, as the network loss rate increases, the portion of recovered time decreases. This is due to two reasons:

- The first one is that as the loss rate of the network increases, the amount of data needed to transfer the video is important because of the retransmissions. For the first scenario (where the network is completely reliable), the amount of data transmitted by the server is equal to the minimum that is needed by the application. In the second scenario, there will be 7.5% of data to be transmitted in addition to the first scenario, and what will be eliminated is only the 2.5% remaining. So the amount of data sent when the network is not completely reliable is higher than the minimum of the application need, but of course, the application receives only the minimum because this amount in addition (7.5%) will be lost. So, when the network is completely reliable, the work of the mechanism is only an optimization of the transmission, but when the network has a certain level of losses, the mechanism will combine the recovery of losses principle (not retransmit lost data) and the optimization principle to fill what remains to reach the maximum tolerance of the application;
- The second reason for the reduction in the portion of recovered time with the increase of network losses is that when the loss rate exceeds the application tolerance, the application is limited to the part of lost data corresponding to the level of this tolerance. That is to say that the mechanism cannot use optimization consumption but can only use loss recovery; in addition, it cannot recover any losses because they exceed the application tolerance.

Table 2
Values of gained transfer time with the mechanism

Cases	Network loss rate	Without mechanism	With mechanism	Difference	Proportion
Case 1	00.00%	3 min 33 s	3 min 11 s	22 s	10.32%
Case 2	07.50%	3 min 48 s	3 min 27 s	23 s	10.08%
Case 3	10.00%	3 min 54 s	3 min 31 s	21 s	09.18%
Case 4	15.00%	4 min 10 s	3 min 43 s	17 s	08.92%

In conclusion, the more robust the network is (with regard to losses), the better is the optimization of the transmission time; however, in all cases, the portion of recovered time remains in the neighborhoods of the application tolerance. This is explained by the fact that this recovered time is not related to the amount of data received by the client, but to the amount sent by the server, because the mechanism combines recovery and optimization. It uses recovery as a first resort to take advantage of network losses, and if it does not reach the loss rate tolerance level, it uses optimization. However, and unlike to what we supposed previously, the optimization of transmission is more efficient than the recovery of losses; indeed, with the recovery, packets are sent but they are not always received, while with the optimization, packets are even not sent. In addition, the recovery of losses is limited by the application tolerance level: once this level achieved, it is no longer possible to ignore lost packets.

Table 2 shows that it is possible to use multimedia applications tolerance to reduce the data transfer time. This difference in transmission time results in a longer downtime of the network interface, which saves more energy.

To illustrate this, let us consider the example of the first scenario. The video should last 7 min 23 s. If the application finishes downloading it after 3 min 33 s, it will remain 3 min 50 s to watch. During this time, the interface will then enter in a standby mode and will stay there for 3 min 50 s. With our mechanism, the video is downloaded 21 seconds earlier and the interface remains idle for 21 supplementary seconds. As previously written, as the video is longer and the application tolerance rate is higher, as the contribution of this solution is important.

Note that in order to use this mechanism, it is necessary that the transmission rate of the video allow the download at a time that is at least equal to its length. This means that the network must display the video without cuts. Otherwise, even reducing the transfer time of the video, it will always be higher than its duration and the solution will have no interest.

Let us now analyze the simulation results according to the energy saving models for the different interfaces. There are two parameters that differentiate the energy behavior of each interface.

- The first one is the loss rate. As we said previously, our mechanism is more efficient when the network is more reliable. Otherwise, the loss reason can be the congestion or bit errors. Congestions depend on the access and core network, so we suppose that whatever the used interface, the loss by congestion probability is the same for all. For bit errors, it is clear that the interfaces do not have the same loss rate. Therefore, this mechanism will work more efficiently on IEEE 802.3 networks, a little less on IEEE 802.11, and much less on 3G. The recovery difference stays though it is not very important.
- The second parameter is the interface energy saving model behavior. For IEEE 802.3 and IEEE 802.11 interfaces, the energy saving principle is very simple: when there is no packet to be sent or received, the interface switches in the inactive mode. This way, the two interfaces are in the inactive mode during all optimized time; in other words, the transfer time gained fully results from the interface sleeping. However, for a 3G interface, it is not the case. After transmitting the last packet, the interface waits a moment before sleeping. The real value of this moment (Idle Time) depends on the interface, but anyway, it must be lower than the optimized time by the mechanism, otherwise, we do not obtain any gain. For 3G interfaces, the optimized time will be always diminished by the Idle Time's value. Another problem with 3G interfaces that does not exist with the others is the impact of other application's traffics. In fact, for IEEE 802.11 and IEEE 802.3, even if there are other traffics, the interface will transmit them and enter directly in the sleep mode. But with 3G interfaces, when the interface is waked up, even for transmitting only one packet, it will stay in active mode for all the Idle Time. This is to say that if the interface is activated frequently during the optimized time, all this will do not have any effect. However, the traffic scheme of mobile applications is particularly adapted to these devices, so all depend on the global context of the applicative traffic.

6. Conclusion

The goal of this work was to illustrate the attention that has to be paid at high communication layers of the protocol stack in order to fully benefit from energy saving modes developed at lower layers of IEEE 802.3, IEEE 802.11 and 3G networks.

We have first analyzed the energy models performed at the lowest level of the communication stack for those three kinds of network interfaces. Then we proposed two different mechanisms aimed at illustrating our proposal, and we have analyzed their benefits in energy consumption.

The first solution consists in replacing the classical *keepalive* mechanism by an enhanced solution, which tackles the same requirements but following and energy saving approach. The second solution aims at optimizing the overall energy consumption by reducing the number of packets transmitted, still providing the minimal qos required by the application in terms of reliability. In both cases, simulations performed using NS-3 have allowed showing and discussing the benefits of the proposed contributions.

At the end of this work, we have confirmed our belief of considering high communication layers both to get more energy gains, and to optimize the underlying low layers energy saving mechanisms.

Acknowledgement

This work has been performed within a regional BQR project funded by the PRES University of Toulouse, France.

References

[1] J.-H. Yeh, J.-C. Chen and C.-C. Lee, Comparative analysis of energy-saving techniques in 3GPP and 3GPP2 systems, *IEEE Transactions on Vehicular Technology* **58**(1) (2009), 432–448.

[2] G. Anastasi, M. Conti, E. Gregori and A. Passarella, Balancing energy saving and QoS in the mobile Internet: an application-independent approach, in: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, January 2003, p. 10.

[3] C.E. Jones, K.M. Sivalingam, P. Agrawal and J.C. Chen, A survey of energy efficient network protocols for wireless networks, *Wirel. Networks* **7**(4) (2001), 343–358.

[4] B. Augustin and A. Mellouk, On traffic patterns of HTTP applications, in: *IEEE Global Telecommunications Conference* (*GLOBECOM 2011*), December 2011, pp. 1–6.

[5] J. Baliga, K. Hinton and R.S. Tucker, Energy consumption of the Internet, in: *32nd Australian Conference on Optical Fibre Technology. COIN-ACOFT 2007. Joint International Conference on Optical Internet*, June 2007, pp. 1–3.

[6] D. Bertozzi, A. Raghunathan, L. Benini and S. Ravi, Transport protocol optimization for energy efficient wireless embedded systems, in: *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 706–711.

[7] A.P. Bianzino, C. Chaudet, D. Rossi and J.-L. Rougier, A survey of green networking research, *IEEE Communications Surveys & Tutorials* **14**(1) (2012), 3–20.

[8] R. Bolla, R. Bruschi, F. Davoli and F. Cucchietti, Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures, *IEEE Communications Surveys & Tutorials* **13**(2) (2011), 223–244.

[9] K. Christensen, S. Florida, B. Nordman, M. Bennett and L. Berkeley, Energy efficiency in communications 802.3az: The road to energy efficient IEEE 802.3, *IEEE Communication Magazine* **November** (2010), 50–56.

[10] Y.W. Chung, J.H. Nat, Y.-J. Kim and H.-S. Cho, Performance analysis of IP paging and power saving mode in IP-based mobile networks, *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications* **3** (2004), 1533–1537.

[11] D. Comer, TCP/IP Architecture, protocoles et applications, in: *Pearson Education*, 2005.

[12] P. Eronen, TCP wake-up: Reducing keep-alive traffic in mobile ipv4 and ipsec NAT traversal, Nokia Technical reports, 2008.

[13] IETF, Hypertext transfer protocol – HTTP/1.1, IETF RFC 2616, 1999.

[14] T.F. Herbert, The Linux TCP/IP Stack, *Charles River Media*, 2004.

[15] IETF, Requirements for internet hosts–communication layers, IETF RFC 1122, 1989.

[16] L. Irish and K.J. Christensen, A 'Green TCP/IP' to reduce electricity consumed by computers, in: *Southeastcon'98. Proceedings*, April 1998, pp. 302–305.

[17] H. Izumikawa, I. Yamaguchi and J. Katto, An efficient TCP with explicit handover notification for mobile networks, in: *Wireless Communications and Networking Conference*, Vol. 2, 2004, pp. 647–652.

[18] M. Jimeno, Saving energy in network hosts with an application layer proxy: Design and evaluation of new methods that utilize improved bloom filters, PhD thesis, University of South Florida, 2010.

[19] L. Mamatas and V. Tsaoussidis, Transport protocol behavior and energy-saving potential, in: *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, 14–16 November, 2006, pp. 889, 896.

[20] B. Nordman and E. Davies, Energy efficiency in protocols and networks: Why might the IETF care?, in: *IETF70 Vancouver Technical Plenary*, 2003.

[21] K. Pentikousis, Pitfalls in energy consumption evaluation studies, in: *2009 6th International Symposium on Wireless Communication Systems*, 2009, pp. 368–372.
[22] G.P. Perrucci, F.H.P. Fitzek, G. Sasso, W. Kellerer and J. Widmer, On the impact of 2G and 3G network usage for mobile phones' battery life, in: *Wireless Conference, European*, May 2009, pp. 255–259.
[23] B. Wang and S. Singh, Analysis of TCP's computational energy cost for mobile computing, *SIGMETRICS Performances Evaluation Revue* **33**(1) (2003), 296–297.
[24] D. Wang, J. McNair and A. George, A smart-NIC-based power-proxy solution for reduced power consumption during instant messaging, in: *IEEE Green Technologies Conference*, April 2010, pp. 1, 10.
[25] X. Wang and W. Wang, An efficient negotiation protocol for real-time multimedia applications over wireless networks, in: *IEEE Wireless Communications and Networking Conference*, Vol. 4, 2004, pp. 2533–2538.
[26] M. Yamada, T. Yazaki, N. Matsuyama and T. Hayashi, Power efficient approach and performance control for routers, in: *ICC Workshops, IEEE International Conference on Communications*, June 2009, pp. 1–5.
[27] M. Yamada, T. Yazaki, S. Nishimura and N. Ikeda, Technologies to save power for carrier class routers and switches, in: *International Symposium on Applications and the Internet*, July 2008, pp. 385–388.
[28] S.-R. Yang, Dynamic power saving mechanism for 3G UMTS system, *Mobile Networks and Applications* **12**(1) (2007), 5–14.
[29] M. Zorzi and R.R. Raot, Is TCP energy efficient?, in: *IEEE International Workshop on Mobile Multimedia Communications*, 1999, pp. 198–201.