DOI: 10.1109/INFOCOM.2015.7218661

Adaptive Online Power-Management for Bluetooth Low Energy

Philipp Kindt, Daniel Yunge, Mathias Gopp, Samarjit Chakraborty
Institute for Real-Time Computer Systems (RCS), Technische Universität München (TUM)
Email: kindt/yunge/gopp/chakraborty (at) rcs.ei.tum.de

***Abstract*—Bluetooth Low Energy is a time-slotted wireless protocol aimed towards low power communication for battery-driven devices. As a power-management capability, whenever there is less data to send, the slave is allowed to remain in a low power mode during a given number of time-slots in a row. However, since the master does not know the exact sleep behavior of the slave, it has to wake-up at every time-slot and repeat its packets until the slave is awake. As a result, applications with variable throughput lead to many energy-consuming idle-slots at the master. In such applications, usually the connection parameters are chosen considering the worst case at design time and remain constant during operation. In this paper, we propose a novel power-management framework for BLE. Rather than skipping slots at the slave side, the proposed system updates the interval between two consecutive time-slots during runtime by applying online algorithms. To avoid data-loss or high delays, the framework guarantees that constraints on latency are met and buffers never overflow. Energy measurements of three different test-cases show that up to 42 percent of the energy consumption of a BLE master can be saved with our power management system.**

I. Introduction

Energy consumption is one of the central problems in the design of wireless devices ranging from small sensors and wearable user interfaces to nodes participating in the emerging Internet-of-Things. Currently, most of these are configured for static data rates. For example, heart rate sensors and fitness trackers send their information usually in fixed time intervals. However, there are many devices that communicate using variable data rates. An example for such devices are BLE modules that implement serial port profiles such as the terminal I/O profile [1] or the SPP-over-BLE profile [2]. These profiles are designed to transmit any possible data of any application to a remote node. It is not known at the design time which applications will use it, and since no assumptions on the characteristics of the traffic can be made, the devices have to consider applications that generate variable data rates. Other devices intrinsically produce variable data rates, such as wireless mice that send the relative coordinates of a mouse movement. The data rate depends on the amount of movement the user causes to the device. Moreover, with recent efforts to utilize BLE as a gateway technology for the internet-of-things [3], the problem of variable throughput rates is likely to appear frequently in many applications. By exploiting the fact that these devices communicate using variable data rates, a new opportunity for energy savings may be used. However, as described below, there are a number of obstacles towards this.

Bluetooth Low Energy is a newly introduced wireless protocol, which is not compatible with the original Bluetooth, referred to as Bluetooth BR/EDR. BLE is aiming towards ultra-low power applications with throughputs between a few bits per second up to $236.7kBits/s$ [4]. It is expected to become extremely widespread. A study expects that 2.5 billion BLE chips will be sold within 2014 [5]. BLE allows applications using it to adjust different parameters that affect throughput, range, latency and energy-consumption to its needs. One of the most important parameters is the *connection interval* T_c that determines the amount of time between two adjacent time-slots data is exchanged at. Its value greatly affects the energy consumption, communication latency and throughput of the wireless link. Usually, this parameter is chosen at design time of a wireless device. Therefore, this connection parameter is determined offline by taking into account the worst case without changing it adaptively. For example, in the SPP-over-BLE profile [2], the connection interval is set to $20ms$ by default and is not updated during runtime automatically. To account for variable throughput, the BLE-protocol [6] specifies a capability for power management on slave nodes. The slave may skip some time-slots without waking up if there is nothing to send. However, only the slave saves considerable amounts of energy, since the master does not know the exact sleep behavior of the slave and hence has to be awake during all time-slots. Especially in networks where the master has a limited energy supply (e.g, a battery-driven wall display that visualizes the temperature measured by a wireless sensor), energy-savings on both sides would be desirable.

To achieve energy-savings also for the master, an online-algorithm could analyze the data rate generated by the application. Based on this, it could decrease T_c whenever a higher BLE throughput is required by the application and increase it in idle-phases. However, constraints imposed by the BLE protocol make the design of such algorithms challenging. In particular, the following challenges need to be addressed:

1) An update of T_c consumes a considerable amount of energy and should therefore not be performed too often. The overhead expressed in the amount of time after which a connection interval decrease pays off is given by Equation 8. For example, a reduction of the connection interval from $25ms$ to its next larger value $26.25ms$ would redeem if the new connection remains constant for at least 13 intervals on a BLE112 [7] module.

2) If an update is scheduled, it takes at least 6 connection intervals until the new parameter value takes effect. Hence, all reactions on the decisions of the power-management algorithm are delayed by an amount of time which depends on the current value of T_c.

3) As a consequence, constraints on latency and buffer-overruns must be considered by the power management system when it schedules the interval updates. In particular, the buffer that contains unsent data might overflow if the application suddenly increases its data rate and the appropriate connection interval update is delayed (e.g, because a previous update has not been completed yet).

For some special applications, dedicated solutions on how to adjust T_c online have been presented. For example, the BLE find me profile [8] specifies a *fast connection interval* which is used whenever encryption parameters need to be exchanged. At all other points in time, the master uses a connection interval which is defined by the slave. To the best of our knowledge, a generic approach that separates the task of online adaptation of the connection interval for power savings from the application design has not been presented, yet. In other domains, like in dynamic voltage and frequency scaling (DVFS) on CPUs, a power-management system chooses relevant parameters online and applications do not need to care about managing the underlying hardware. Developers of computer programs or smartphone applications can save the effort to develop power-aware systems and benefit from platform-independent solutions. However, existing solutions from other domains are not well suited, since for an online-adjustment of the connection interval, the constraints described above exist.

In this paper, we propose a novel power-management framework for BLE that can be implemented within BLE-modules or -stacks. The proposed system analyzes the throughput demands of the application and adjusts the connection interval adaptively. In addition to the application throughput, the buffer fill level is monitored to make online decisions. The system guarantees that latency constraints are met and buffers never overrun. Different power management strategies can be used as a part of the proposed system. These strategies must be light-weight in terms of memory and computational demand in order to be implemented on a BLE module. By applying them, significant amounts of energy can be saved.

The rest of this paper is organized as follows: In Section II, we briefly describe the BLE protocol including energy considerations when updating the connection interval. Related work is presented in Section IV. In Section V, our proposed power management framework is presented along with the theoretical machinery that is needed to comply with constraints on latency and throughput. Based on that framework, three power management strategies are described in Section VI. Next, using two representative traffic patterns and the signal of an electrocardiograph (ECG) with lossless compression as an example, we evaluate the power savings achieved with our proposed strategies in Section VII. Finally, our results are discussed along with an outline for future research in Section VIII.

II. The BLE Protocol

In this section, we briefly describe the parts of the BLE protocol that are essential to understand our proposed power management framework. We consider two devices exchanging some data via the BLE protocol. Our descriptions are based on the official Bluetooth Specification 4.0 [6].

Fig. 1. Packet exchange between a master and a slave in the connected mode.

A. Data Transmission

In this paper, we consider two devices that exchange data in the *connected mode* of BLE, which is the mode that is usually used for the transmission of payload data. One device acts as the communication master and defines the timing of the other device, which acts as the slave. As depicted in Figure 1, data transmission is organized in so-called *connection events*.

These connection events are repeated periodically with the *connection interval* T_c. In such a connection event, the following steps take place:

1) Both the master and the slave wake up at the beginning of the event. The master begins its wireless transmission on a RF channel. The slave starts listening at the same point in time, thus receiving a packet from the master.

2) After the master has finished its transmission, the slave sends a packet to the master.

3) If there is more data to be sent by the master or by the slave, more pairs of packets are exchanged within the same connection event. The maximum packet size in BLE is limited to 47 bytes in order to keep packet transmission times short. Short packets allow for cheaper BLE radios, as temperature-driven frequency drifts remain low within short time intervals [9]. In Figure 1, one pair of packets is shown as an example.

4) If there is no more data to be sent or the maximum number of packets per event has been reached, the connection event ends. The maximum number of packets per event is a device-specific parameter. The next connection event will take place after T_c time units.

Up to a number of N_{sl} events in a row may be skipped by the slave without waking up. N_{sl} is a connection-specific parameter that can have a value between 0 and 500 events. N_{sl} is referred to as *slave latency*. Using the scheme described above, BLE defines a send-and-confirm procedure, which is considered in this paper. This procedure works as follows. For exchanging payload, the master sends a data packet to the slave in one connection event. Within the same event, the slave sends an empty response packet since the packet from the master has not been processed, yet. After one connection interval, the master sends an empty polling packet to the slave, who answers with a confirmation that the payload was received. In the next event, the master may send its next payload packet. If larger volumes of data need to be sent in a given amount of time, shorter connection intervals are required. In addition, larger values of T_c lead to higher latencies. Whenever there is no connection event taking place, both devices can go into a low power mode. Figure 2 shows the mean power consumption of a BLE master and a slave for different connection intervals

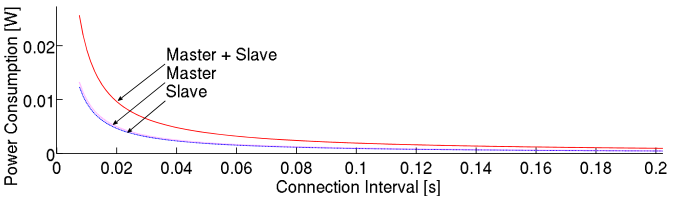

Fig. 2. Power consumption for a BLE master and slave for different connection intervals. Both devices consume similar amounts of power.

varying from $7.5ms$ to $0.2s$. As can be seen, the master and the slave consume nearly the same power. The joint curve for both devices is nearly the double of each single curve. The figure is valid for the following assumptions. The master sends the maximum number of 20 payload bytes per packet using the confirmed transmission procedure described above, while the slave has no payload to send. The power consumption has been computed using a BLE energy model proposed in the literature(e.g., see [10] or [11]), assuming the device sends on its maximum transmission power level. For the slave, we assume that it wakes up during every connection interval. As can be seen from Figure 2, by reducing the connection interval from its minimum value $T_{c,min} = 7.5ms$ to, e.g., $50ms$, the joint power consumption of both devices can be reduced by approximately 85%. Hence, when the data volume is varying, substantial energy savings can be achieved by adjusting the connection interval in an online fashion. However, the connection interval must be chosen such that there is no data loss. In addition, the adjustment incurs an overhead and cannot be enabled instantaneously. The connection interval is negotiated in the connection establishment phase and can be updated by a parameter update procedure, which is described below.

B. Connection Parameter Updates

The connection interval T_c is set by the master during the connection establishment phase. During an existing connection, it can be changed using an update procedure. In this procedure, not only a packet that contains the new connection interval must be sent from the master to the slave, but also the clocks of both devices have to be re-synchronized to a new anchor point for future connection events. The energy E_{up} that needs to be spent to update the connection interval is higher for the slave than for the master. For both devices, one has to take into account a connection-update packet, which induces an energy cost denoted as E_p. In the send-and-confirm procedure under consideration, an empty polling packet of the master which has a length of 10 Bytes is replaced by a 22-Byte update packet. Hence, E_p is the difference of the energy consumed by the two packets.

For the device acting as a master, the energy needed for a connection update from the interval $T_{c,o}$ to $T_{c,n}$ can be estimated well by $E_{up,M} = E_{p,M}$. In this estimation, the sleep mode energy which is small compared to the energy consumed in the active mode is neglected. For the slave, the clock resynchronization induces an additional idle-listening period. A precise model for this cost as well as for E_p has been adopted from the literature (e.g., [10] and [11]). Based on this model, a coarse worst-case approximation is given by

$$E_{up,S} = E_{p,S} + (2 \cdot \Phi \cdot 10^{-6} \cdot (T_{c,o} + T_{c,n}) + 10ms) \cdot I_{rx} V_{cc} \quad (1)$$

In the equation above, Φ is the slave clock accuracy of the device in parts per million (PPM), I_{rx} is the current needed for reception (including idle-listening) and V_{cc} is the supply voltage of the device. The equation takes into account the energy consumed due to idle listening and potential clock jitter as well as the energy for the packets that are exchanged.

Once a connection parameter update has taken place, the new connection interval cannot be changed again for at least $N_{i,m} \cdot T_{c,n}$ time units. $N_{i,m}$ has a minimum value of 6 intervals defined by the BLE specification [6]. $N_{i,m}$ has a value of 8 intervals for the device we used (BLE112).

III. Problem Formulation

With the descriptions above, the problem considered in this paper can be expressed in abstract terms as follows. A BLE module transmits application data which is generated using a variable data rate R_{app}(t) with a connection interval T_c. At any point in time, it may update T_c to a new value or leave it constant. An update incurs a cost $E_{up}(T_c)$ and might pay off if R_{app} remains constant for a sufficiently large amount of time. The effect of an update is delayed by an amount of time $t_d(T_c)$, which depends on the connection interval before the update. In addition, sometimes the system is forced to change T_c to avoid data loss caused by overflowing buffers. A power-management system needs to control T_c such that the energy consumption is minimized while buffers do not overrun and latencies do not exceed a given threshold. In the next section, we describe how this problem is related to problems known from the literature.

IV. Related Work

Online algorithms have been widely used for energy optimizations in different applications. A problem which is related to the one described in this paper is the well-studied k-page-migration problem [12]. In a network, a set of servers hold copies of a page. A client requests this page from a server that contains the page. The system can choose whether to serve the request from the next server that holds the page, or it can migrate the page to a server that is closer and can serve the request with lower cost. A migration procedure has an update cost and incurs a time delay. While this problem shares many similarities with the connection interval update, existing algorithms cannot be applied easily to BLE due to additional constraints. For the BLE connection interval, updates might be enforced because otherwise constraints would be violated as described in Section V (i.e., buffers might overflow).

Another related problem has been widely studied in the context of Bluetooth BR/EDR, which we describe below. Since the Bluetooth specification [6] does not suggest how a master should poll its slaves, several algorithms to schedule the polling intervals for individual slaves have been proposed. In [13], this interval is adjusted online based on the ratio of successful polls (i.e., polls that result in a data transmission)

and the total number of polls. Another approach adjusts the sleep period based on the expected arrival time of a packet flow [14]. The latter solution use the so-called *hold-mode* of Bluetooth. A solution using Bluetooth's *sniff mode* is presented in [15]. Updates in the sniff mode take effect after a single sniff interval, whereas updates in BLE take effect after at least 6 connection intervals. Unlike for BLE, updating the sleep interval in the Bluetooth hold mode incurs no significant cost. Hence, the existing solutions are not expected to perform well for BLE. In further literature (i.e., [16]), it was claimed that scheduling strategies for Bluetooth BR/EDR which minimize the number of idle packets without using sleep modes achieve energy savings up to 20%. Related problems also occur in other domains, e.g., dynamic voltage and frequency scaling (DVFS) of CPUs. The on-demand governor [17] of the Linux-kernel adjusts the frequency and voltage of CPU-cores adaptively to the load. However, there are different requirements and constraints in BLE. For example, DVFS algorithms do not need to satisfy constraints on buffer-overflows and latency.

Our contribution:
Whereas algorithms for scheduling the polling of slaves for Bluetooth BR/EDR are still an active topic of research, to the best of our knowledge, no power-management scheme for Bluetooth Low Energy that dynamically adjusts the connection interval has been presented, yet. In this paper, we for the first time propose a generic power-management scheme for BLE, and make the following contributions:

1) We propose a novel framework for updating the connection interval online, that can ensure given bounds on the maximum communication latency while buffers are always guaranteed not to overrun. As a part of this framework, different online algorithms for different applications can be used, analogous to governors for DVFS.

2) We propose three online algorithms for adjusting the connection interval. These algorithms are based on the current application throughput and the buffer backlogs. Each of them has different advantages and drawbacks, which we evaluate using comparative measurements.

3) Using two representative application traffic patterns and a wireless ECG as an example, we evaluate our proposed scheme with comprehensive power measurements and show that significant amounts of energy can be saved using the proposed algorithms.

V. Power Management System

A. System Overview

Figure 3 depicts the setup on which our proposed system works. We consider two devices *A* and *B*. Without loss of generality, we assume that device *A* acts as the BLE master which sends data to device *B*. On each of them, an application generates traffic that is passed to the BLE radio for wireless transmission via a FIFO-buffer. Each time the device adds data to the buffer, it is assumed to do so in small bursts with variable length. We denote these bursts as *write-requests*. Since there could also be single-byte write requests (e.g., as parts of a stream), this assumption does not constrain the

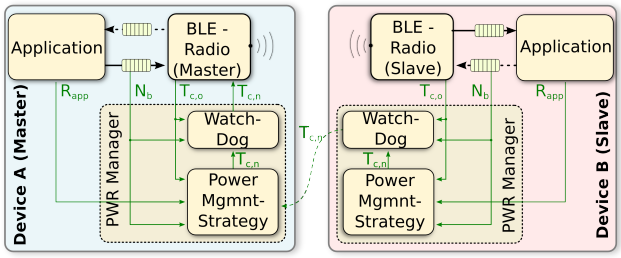

Fig. 3. Power management system for a link between two devices A and B.

possible applications. The maximum number of bytes the application writes in one request is denoted as $N_{b,w}$. The number of bytes per write-request together with the points in time these requests take place are monitored by a power-manager, which also has access to the fill level N_b of the buffer between the application and the BLE module. As a main part of the power-manager, an online-algorithm monitors the traffic rate R_{app} of the application along with N_b. We denote these algorithms as *power-management strategies* (PMS). Based on the informations available, it decides when to update the connection interval and its future value. The second part of the power-manager is a BLE-specific watchdog. It may override the connection interval decided by the PMS to ensure that the buffer never overflows. The watchdog is described in detail in Section V-D.

Our system provides a simple API to the application, which can be used for setting parameter values. Before the power-manager can start operating, the application must make some parameters known to the system. In particular, these are:

- The maximum traffic rate $R_{app,m}$ of the application.
- The maximum allowed latency $t_{lat,m}$, defined as the maximum time between an application writing a byte and the BLE module sending it over-the-air.
- The maximum number of bytes per write-request $N_{b,w}$.

In order to meet constrains on the maximum latency, the power-manager uses these values to compute the size of the buffer $N_{b,m}$ it should allocate and the maximum connection interval $T_{c,m}$ that can be used safely. For their computation, a relation between T_c and the throughput of the link is described in the next section. Afterwards, the computation of $T_{c,m}$ and N_b is explained in detail.

B. Connection Interval and Throughput

As described in Section II, in each connection interval, N_{seq} pairs of packets can be sent from the master to the slave and vice-versa. To compute the over-the-air throughput of BLE, one must also take into account that not all attempts on transmitting a packet are successful. We account for that by introducing a factor η which reduces the throughput. This parameter is device- and link dependent. If there is interference with other wireless devices or a high attenuation, the data rate decreases. Without loss of generality, in this paper, we assume having an error-free link, which is normally true for small distances without other devices interfering. If interference is present, our approach can be used by decreasing η either adaptively based on the measured loss rate or statically taking

into account a safety margin. A flow control mechanism that can be used by any BLE device also influences the value of η. If the BLE stack on a remote device is not ready to receive a packet, it may request a retransmission. The device investigated in this paper (Bluegiga BLE112) requests the opposite device to resend its packet frequently. While other devices and other firmware stacks might have different values for $\overline{\eta_a}$, its value is independent from the application. By analyzing more than $710,000$ packets, we found that on an average, the throughput is degenerated by $1-\overline{\eta_a} = 0.3\%$ due to flow control. However, when shorter intervals are considered, retransmissions can happen more frequently. This has to be accounted for in worst-case computations. We define a worst case analysis window of $d_{wc} = 0.5s$. For all intervals longer than d_{wc}, our experiments revealed that the throughput is reduced by no more than $1 - \eta_{wc} = 26.3\%$. In every interval smaller than d_{wc}, we assume that the traffic rate might drop to $0\ bytes/s$. The actual length of the worst case window is unimportant as η_{wc} changes accordingly.

Under these assumptions, the throughput of the BLE link can be computed as in Equation 2 by taking into account that one data packet can carry up to 20 bytes of payload.

$$R_{BLE}(T_c) = \frac{20bytes \cdot N_{seq}}{T_c} \cdot \eta \tag{2}$$

As already mentioned, a send-and-confirm-procedure is applied in BLE. As a result, one data packet is sent within two connection intervals and therefore N_{sec} has a value of 0.5. In Equation 2, η might be set to $\overline{\eta_a}$ when considering the overall throughput, or to η_{wc} when considering the worst case throughput. In the worst case, when the application writes with its maximum rate and the link is stalled for d_{wc} amounts of time, the number of bytes that can be written maximally within one window of length d_{wc} is:

$$N_{b,wc} = R_{app,m} d_{wc} \tag{3}$$

C. Buffer dimensioning

Our proposed system is capable of providing guarantees that the maximum latency $t_{lat,m}$ is never exceeded. This can be accomplished by choosing small buffer sizes $N_{b,m}$ or by choosing small maximum connection intervals $T_{c,m}$ - or a combination of both. We assume that there is a given amount of memory available on the device, which is partially used for buffering. Within this amount of memory, the buffer size can be chosen freely. Since we also guarantee that the output buffer never overflows, both $N_{b,m}$ and $T_{c,m}$ are determined by a set of linear equations which are described and solved below.

As already mentioned, when a connection interval update from an old interval $T_{c,o}$ to $T_{c,n}$ is scheduled at any point in time, the actual connection interval update takes place after $N_{i,m} \cdot T_{c,o}$ time units. A buffer overflow might occur if the application writes with a higher data-rate than the current throughput of the BLE link, and an interval update cannot take place before the buffer is entirely filled. As depicted in Figure 4, there might be situations in which a connection interval update takes effect after $2 \cdot N_{i,m} \cdot T_{c,o}$ time units. Consider an interval update from $T_{c,o}$ to $T_{c,n}$ with $T_{c,o} \approx T_{c,n}$. From the time the update is scheduled, it takes $N_{i,m}$ intervals until the new interval becomes valid. Another update to $T_{c,em}$ in the meantime can only be scheduled after $N_{i,m} \cdot T_{c,o}$ time units in the worst case, and it takes another $N_{i,m} \cdot T_{c,n}$ time units until the connection interval of the link becomes $T_{c,em}$.

In the worst case, the application might increase its throughput to $R_{app,m}$ right after the first interval update. The update that is required to adjust the throughput R_{BLE} to the new value of $R_{app,m}$ is delayed by $2 \cdot N_{i,m}$ connection intervals and in the meantime, the buffer must accept the additional bytes without overflowing. Hence, the minimum required buffer size can be computed as follows.

$$N_{b,m} = \lfloor 2 \cdot N_{i,m} T_{c,m} (R_{app,m} - \overline{R_{BLE}}(T_{c,m})) + N_K \rfloor \tag{4}$$

N_K is a constant number of bytes. It contains the number of bytes $N_{b,w}$ which are written to the buffer within one write-request and the bytes needed for flow control in the worst case, $N_{b,wc}$. In addition, it includes a number of bytes $N_{b,c}$, which makes the buffer larger than its minimum size to allow for control without interventions of the watchdog. It is a parameter of the power management algorithm used. N_K is defined as $N_K = N_{b,w} + N_{b,wc} + N_{b,c}$. In Equation 4, $T_{c,m}$ is the maximum connection interval that the power management system is allowed to choose. If the available memory is lower than $N_{b,m}$, then the buffer size is reduced to this limit.

To satisfy the latency requirement, the buffer must be flushed completely within $t_{lat,m}$ even for the lowest possible BLE throughput. The following equation applies:

$$\frac{N_{b,m}}{t_{lat,m}} \stackrel{!}{=} R_{BLE}(T_{c,m}) \tag{5}$$

By solving the linear system defined by the Equations 5, 4 and 2, one has:

$$T_{c,m} = \frac{40 N_{seq} N_{i,m} \eta_{wc} - N_K + \sqrt{\gamma}}{4 N_{i,m} R_{app,m}} \tag{6}$$

with

$$\begin{aligned}\gamma =& N_K^2 - 80 N_K N_{seq} N_{i,m} \eta_{wc} + 1600 N_{seq}^2 N_{i,m}^2 \eta_{wc}^2 \\ &+ 160 R_{app,m} t_{lat,m} N_{seq} N_{i,m} \eta_{wc}\end{aligned}$$

The Equations 4 and 6 define a minimum buffer size and a maximum connection interval based on the parameters of the application. The power management system allocates the required memory size for the buffer automatically, and the watchdog makes sure that $T_{c,m}$ is never exceeded. In addition, we define a quality-of-service parameter $T_{c,qos}$ which is always subtracted from the connection interval chosen by the power management system. This results in the BLE throughput being slightly higher than necessary, leading to reduced buffer backlogs and hence low average latencies. The maximum connection interval $T_{c,m}$ is reduced by it as well. With $T'_{c,m} = T_{c,m} - T_{c,qos}$, the buffer size can be dimensioned by using Equation 4.

D. Watchdog

By choosing $T_{c,m}$ and $N_{b,m}$ as described above, keeping the latency constraint is guaranteed intrinsically. In contrast, an online-algorithm controlling the connection interval using a best-effort approach could cause the buffer to overrun. This must be prevented by a watchdog that overrides the decisions of the algorithm whenever there is a potential buffer overrun. Let $T_{c,em}$ be the maximum connection interval that guarantees no buffer overrun at a given time instance t_0. Whenever the connection interval is updated from $T_{c,o}$ to $T_{c,n}$ by the online-algorithm, a value for $T_{c,em}$ needs to be computed by the watchdog. If necessary, $T_{c,n}$ is set to $T_{c,em}$.

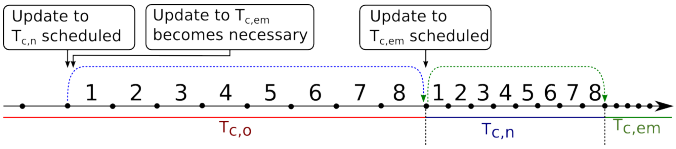

Fig. 4. Timing when updating the connection interval

Figure 4 illustrates the timing when updating from $T_{c,o}$ to $T_{c,n}$. We assume that $N_{b,f}$ bytes are left in the buffer when the update is scheduled. $T_{c,n}$ must be chosen such that the buffer cannot overrun before another connection interval update to $T_{c,em}$ scheduled by the watchdog takes effect. Therefore, $T_{c,n} < T_{c,em}$ must be set considering the worst case, which is defined as described below. The connection interval must be always low enough that a sudden increase of R_{app} does not lead to a buffer overrun. In case of a sudden increase of R_{app}, similar deliberations as for Equation 4 result in a maximum delay of $N_{i,m}T_{c,o} + N_{i,m}T_{c,n}$ until an update to compensate for this increase takes effect. As a result, at any point in time, a newly scheduled connection interval $T_{c,n}$ must not exceed an upper bound that is defined by the current buffer level N_b and the current connection interval $T_{c,o}$. This bound is defined by Equation 7.

$$T_c < \frac{N_{i,m} \cdot (40 N_{seq} \eta_{wc} - T_{c,o} R_{app,m}) + N_K}{R_{app,m} N_{i,m}} \qquad (7)$$

The watchdog depicted in Figure 3 evaluates the equation above for every write-request and at every connection update procedure. If needed, it overwrites the connection interval scheduled by the power management strategy. The maximum application data rate for which the watchdog guarantees that the buffer will never overrun is $\eta_{wc} R_{BLE}(T_{c,min})$. $T_{c,min}$ is the minimum required connection interval. It is defined as $T_{c,min} = max(7.5ms, \frac{20 \cdot N_{seq}}{R_{app,m}} \eta_{wc})$.

E. Bidirectional traffic

The system described above has been presented for traffic from the master to the slave only. However, it can be extended to include traffic in the reverse direction as well, as depicted in device B of Figure 3. For this purpose, the slave runs a similar power-management system, and sends appropriate update requests to the master via the wireless link. The master then chooses whether it should update to the connection interval requested by the slave, or override the slave's decision to satisfy its own traffic demands.

VI. Power Management Strategies

In this section, we describe three different online-algorithms that can be used as power management strategies in the framework depicted in Figure 3. Each of the proposed strategies has its advantages and disadvantages in terms of power-savings, average latency and implementation complexity, which we discuss within the next sections.

All strategies use the application data rate R_{app} as an input parameter. We define the following terminology: Let $R^*_{app}(t_0)$ be the application throughput rate defined as the number of bytes written during a write-request at time t_0 divided by the time between the current and the previous write-request. In contrast, $\overline{R_{app}(\Delta t, t_0)}$ is the application data rate defined by the number of bytes written within a given interval of time Δt divided by Δt. The time-instance t_0 is defined as the point in time at which the interval considered ends. If t_0 is omitted, the symbols refer to the values at the most recent point in time. $R_{BLE}(T_c)$ is the average throughput rate of the BLE link depending on the connection interval T_c. We define $T_{c,o}$ as the connection interval before a scheduled update and $T_{c,s}$ as the interval that is to be scheduled. $T_{c,n}$ denotes the connection interval after an update takes effect. Based on this terminology, we describe our proposed power-management algorithms below.

A. Periodic Adjust

We define the *periodic adjust strategy* as shown in Algorithm 1. The application throughput is monitored periodically with a period ΔT_{pa}. Once per period, the connection interval is adjusted such that $R_{BLE} = R_{app}$. The BLE throughput rate is updated periodically with a period of ΔT_{pa}. The procedure *wait*(ΔT) causes the calling procedure to sleep for ΔT amounts of time. Compared to the original interval

Algorithm 1 Periodic Adjust Algorithm

```
1: procedure PeriodicAdjust(ΔT_pa)
2:     repeat
3:         T_c,s ← 20Bytes·N_seq / R_app(ΔT'_pa) · η_a − T_c,qos
4:         wait(ΔT_pa)
5:     until forever
6: end procedure
```

length, $\Delta T'_{pa}$ in Line 3 of Algorithm 1 is a slightly modified period which is used for measuring the rate. If there are less than two write-requests within ΔT_{pa}, we set $\Delta T'_{pa}$ to ΔT_{pa}. Otherwise, we set $\Delta T'_{pa}$ to the time difference between the last write-request within the previous examination period and the last write-request within the current period. Without this mechanism, the measured rate $\overline{R_{app}}$ would be subjected to oscillations since the frequency of write-requests and the adjustment period are unsynchronized. The advantages of this strategy are its extremely simple implementation and its capability of reacting quickly on sudden changes in the application data rate.

B. Lazy Decrease

A a slightly more sophisticated strategy is the *lazy decrease* strategy. It sets the connection interval to its minimum value whenever the application performs a write-request during which the application data rate is higher than the current rate of the BLE link. Whenever the application data rate remains below the BLE throughput for more than one time interval ΔT_{ld}, the BLE throughput rate is gradually decreased by a constant value $R_{reduced}$. This scheme shares some similarities with the strategy that is widely used for voltage-frequency scaling in the Linux ondemand governor [17], and with commonly known additive increase/multiplicative decrease schemes. However, it contains BLE-specific adaptations. A complete description of this strategy is presented in Algorithm 2. The procedure *OnWrite()* is called whenever a write-request takes place. The procedure *ResetTimer()* resets a timer which executes the procedure *TimerCallBack()* ΔT_{ld} time-units after the last execution of *ResetTimer()*. In each step, the BLE throughput is reduced by $R_{reduced}$, which is a given percentage γ_{dec} of the whole range of possible values. If the reduction in any step results in the same connection interval as in the period before, T_c is increased by its minimal subdivision, i.e., $T_{c,s} = T_{c,o} + 1.25ms$. In Line 9, a modified interval $\Delta T'_{ld}$ similar to the one described above for Algorithm 1 is used.

Algorithm 2 Lazy Decrease Algorithm

```
1:  procedure OnWrite(R*_app)
2:      if R*_app > R_BLE(T_c,o) then
3:          T_c,s ← T_c,min
4:          ResetTimer()
5:      end if
6:  end procedure
7:  procedure TimerCallBack
8:      R_reduced ← γ_dec(R_app,m − R_BLE(T_c,m))
9:      if R'_app(ΔT_ld) < R_BLE(T_c,o) − R_reduced then
10:         T_c,s ← η_avg · 20bytes·N_seq / (R_BLE(T_c,o) − R_reduced)
11:         ResetTimer()
12:     end if
13: end procedure
```

With the algorithm described above, oscillations in the measured application data rate can occur if the measurement interval T_{ld} is smaller than the time between two write-requests. Implementations must account for this. If ΔT_{ld} is chosen appropriately, the algorithm guarantees that not more energy than when leaving T_c constant is spent even in the worst case. In addition, the mean buffer backlog during runtime is small, leading to short average latencies. A drawback is that it takes much time and many energy-consuming interval updates until the algorithm's response on a sudden rate decrease stabilizes.

C. Adaptive Buffer Level Adjustment

An even more sophisticated strategy is updating the connection interval based on the current fill level of the output buffer depicted in Figure 3. The aim of our strategy is to try to keep the buffer fill level always between zero and a threshold $N_{b,inc}$ which is updated during runtime. The buffer level strategy should suppress changes in the application throughput rate that last very short and follow longer-lasting changes, only. The buffer along with multiple defined watermarks is depicted in Figure 5.

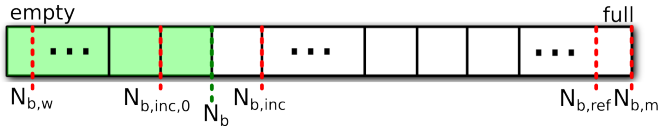

Fig. 5. Output buffer with relevant fill levels and the current fill-level N_b.

We define the *reference level* as $N_{b,ref} = N_{b,m} - N_{b,w}$. The reference level is used to calculate a variable threshold $N_{b,inc}$. If the buffer level exceeds this threshold, a connection interval update is initiated and $N_{b,inc}$ is set to the buffer level at the time of the update plus an additional offset $\Delta N_{b,inc}$. With β_b being a predefined percentage, $\Delta N_{b,inc}$ is defined as $\Delta N_{b,inc} = \beta_b N_{b,ref}$. We initialize $N_{b,inc}$ as $N_{b,inc,0} = \Delta N_{b,inc} + N_{b,w}$ and adjust it during runtime. The adaptive buffer level adjustment strategy is shown in Algorithm 3.

Algorithm 3 Buffer Level Monitoring Algorithm

```
1:  procedure OnWrite(R*_app, N_b)
2:      R'_app = αR'_app + (1 − α)R*_app
3:      if N_b > N_b,inc then
4:          T_c,s ← 20Bytes·N_seq / R'_app · η_a − T_c,qos
5:      end if
6:  end procedure
7:  procedure OnRead(R*_app, N_b, t_now, t_lastInc)
8:      if (N_b < N_b,w) ∩ (t_now − t_lastInc > d_block) then
9:          T_c,s ← 20Bytes·N_seq / R'_app · η_a − T_c,qos
10:     end if
11: end procedure
12: procedure OnDecrease(T_c,o, T_c,n, t_now)
13:     N_b,inc ← N_b + ΔN_b,inc
14:     OnRead()
15: end procedure
16: procedure OnIncrease(T_c,o, T_c,n, t_now)
17:     N_b,inc ← ΔN_b,inc + N_b,w
18:     t_lastInc ← t_now
19:     OnRead()
20: end procedure
```

The algorithm contains 4 procedures:

- *OnWrite()* / *OnRead()* is called whenever the application performs a write-request / reads from the buffer.
- *OnIncrease()* / *OnDecrease()* is called whenever an update with $T_{c,n} > T_{c,o}$ / with $T_{c,n} < T_{c,o}$ takes effect.

The variable t_{now} contains the point in time the corresponding function is called. $t_{lastInc}$ contains the point in time when the last connection interval increase took place. To get a robust measurement of the current application rate throughput R'_{app} that filters out short-time variations, exponential smoothing [18] with a coefficient α is applied in Line 2 of Algorithm 3. To avoid to frequent updates caused by small decreasing traffic

gradients, in Line 8 of the algorithm, there is a blocking time d_{block}. For d_{block} time-units after a connection interval increase, no further update is allowed. Its value can be derived by comparing the energy saved by an increased connection interval with the cost of the update to the new interval. d_{block} is the amount time after which the update pays off. An approximate solution is given by the equation below.

$$d_{block} = \frac{T_{c,o}T_{c,n}E_{up,m+s}}{T_{c,n}E_c(T_{c,o}) - T_{c,o}E_c(T_{c,n})} \quad (8)$$

In this equation, $E_{up,m+s}$ is the update energy for both master and slave and $E_c(T_c)$ is the energy spent per connection interval by both devices for a given value of T_c. After each update, the buffer should be checked for underruns because in the time between two updates, N_b might have changed and might be below $N_{b,w}$ due to a reduced application throughput. Therefore, *onRead()* is called after every interval update in the Lines 14 and 19. The buffer strategy cannot detect idle lines (i.e., periods without any traffic) and therefore needs a separate idle line detection.

VII. Evaluation

In this section, our proposed power management strategies are evaluated by comprehensive energy measurements. First, we describe the setup of our experiments. After this, we present and discuss the results of our measurements.

A. Experimental Setup

To measure the power savings of our proposed system, we implemented the following setup: On a node consisting of a host-MCU (ARM Cortex-M) and a BLE112-module, a traffic generator creates test patterns as described below. The node acts as the BLE master and contains an implementation of our proposed power-management. The traffic is received by a BLED112-USB dongle at a laptop. To evaluate the power-consumption, we measured the current draw of the BLE module which has a constant supply voltage with a sampling-rate of 50 KHz. Since the computational overhead of our proposed strategies is negligible, we can measure the energy-savings of the three strategies described in Section VI using this setup.

B. Traffic Patterns

The energy savings depend on the application data rate. We define the three representative traffic patterns that are depicted in Figure 6 as our test-cases. These patterns contain both sudden rate changes as well as gradual rate increases/decreases. As a real-world example, we use the traffic generated by a wireless ECG. In particular, the patterns are defined as follows.

- **T1): Series of ramps**: Starting from 0 $Bytes/s$, the traffic is increased every $100ms$ by 10 $Bytes/s$ until the maximum application rate $R_{app,m} \approx 980$ $Bytes/s$ is reached. Afterwards, the throughput rate is decreased in the same manner. The situation repeats periodically and $N_{b,w}$ is 98 $Bytes$.
- **T2): Burst traffic**: Every 20 seconds, there is a burst of 980 $Bytes/s$ lasting for 10 seconds. For the rest of the time, the traffic rate is 100 $Bytes/s$. $N_{b,w}$ is 98 $Bytes$.

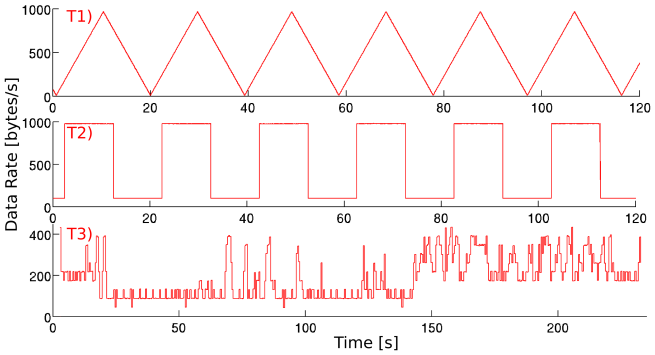

Fig. 6. Traffic patterns as test-cases.

- **T3): Wireless ECG**: As a real-world traffic pattern, we generated a signal which has a varying data rate by using a custom compression method together with an ECG signal. In our use-case, ECG electrodes are attached to a user's wrists while the person is typing on a computer keyboard and therefore generating movement artifacts in the ECG signal. These artifacts decreased the efficiency of the compression method, which is based on differential Huffman coding. As a result, the whole process generated a signal having a data rate that varied between 50 $Bytes/s$ and 435 $Bytes/s$, depending on the amount of movements. The data rate of this signal is depicted in the bottom of Figure 6. Using a record- and playback-infrastructure, a signal having the same data rate as the ECG signal is used as a third test traffic pattern in our experiments. $N_{b,w}$ is 200 $Bytes$. Since persons wearing ECGs for mobile health monitoring usually move during operation, motion artifacts are always present. Therefore, T3) is a realistic example.

Each measurement was started a bit more than 15 seconds after the establishment of the connection and lasted for 60 seconds for T1) and T2) or for the duration of the whole ECG pattern in the case of T3), respectively. We assume a maximum allowed latency of $15s$ in the system. As an example, the data of a wireless mobile ECG sending to a smart phone does not need to be transmitted in real-time in any case. A maximum latency of $15s$ can be tolerated if the evaluation of the results is not timing-critical. $N_{b,c}$ is set to 200 $Bytes$, $T_{c,qos}$ to $1.25ms$ except for the lazy-decrease strategy, for which $T_{c,qos} > 0$ makes no sense since it achieves low latencies anyway. For T1) and T2), the time between any two consecutive write-requests is $100ms$ and for T3), it is varying between $455ms$ and $475ms$. For the periodic adjust- and the buffer-level strategy, the exponential smoothing-factor α was set to 0.9.

C. Energy Measurements

The measured energy savings compared to keeping the connection interval at $T_{c,min} = 10ms$[1] for T1) and T2) or at $22.5ms$ for T3), respectively, are shown in Table I. The results are discussed below.

1) Periodic Adjust: The periodic adjust strategy performed well in all test-cases. For the burst traffic, the savings were

[1] 10 ms is the required connection interval to serve the maximum traffic of 980 $bytes/s$ in average.

	Periodic Adjust			**Lazy Decrease**			**Buffer Level**		
	$\Delta T_{pa} = 0.1s$	$\Delta T_{pa} = 0.5s$	$\Delta T_{pa} = 1s$	$\Delta T_{ld} = 0.2s$	$\Delta T_{ld} = 0.5s$	$\Delta T_{ld} = 1s$	$\beta_b = 0.05$	$\beta_b = 0.1$	$\beta_b = 0.2$
T 1)	27.2%	27.9%	28.6%	18.8%	25.1%	16.5%	34.3%	34.1%	35.0%
T 2)	17.8%	18.4%	19.1%	21.4%	16.7%	11.5%	23.6%	24.3%	24.7%
T 3)	16.0%	29.9%	30.7%	-	20.5%	13.9%	42.0%	40.5%	35.2%

TABLE I
ENERGY SAVINGS OF THE PROPOSED STRATEGIES FOR ALL THREE TRAFFIC PATTERNS

reduced compared to the buffer level strategy for two reasons: 1.) At rate increases, the connection interval was sometimes not reduced quickly enough to prevent the watchdog from intervening. As a result, high connection intervals were set repeatedly. 2.) The exponential smoothing caused the strategy to follow rate increases with an update to an intermediate rate before the final rate was reached, whereas only one update was performed with the buffer level strategy. For T3), the strategy with $T_{pa} = 0.1s$ led to high buffer levels with frequent watchdog interventions which reduced the savings. Because of its simple implementation and the minimal computational cost, the algorithm is suitable to be implemented as a default strategy in BLE stacks. To obtain stable operations and high savings, T_{pa} should be chosen to be sufficiently long.

2) Lazy Decrease: In our measurements γ_{dec} was set to 10%. As already mentioned, the strategy does not work if T_{ld} is smaller than the time between two consecutive write-requests. Because of this, the measurement of T3) with $T_{ld} = 0.2s$ was not carried out. As expected, the lazy decrease strategy led to the lowest energy savings of all strategies in most of the measurements. Mainly, this was caused by unnecessarily high connection intervals whenever the strategy set T_c to $T_{c,min}$ and by frequent interval updates.

3) Buffer Fill Level: The buffer fill level strategy provided high savings in all measurements. Overall, the savings achieved with it exceeded the savings of all other strategies. With the disadvantage of a slightly more complex implementation, it appears to be the most robust strategy examined. The buffer fill level strategy has an advantageous property: In scenarios where frequent alternating throughput changes of small magnitude occur, the periodic adjust strategy would frequently toggle between two adjacent connection intervals. The buffer level strategy would avoid these costly updates.

VIII. CONCLUSION AND ACKNOWLEDGEMENTS

In this paper, we have presented a novel power-management framework for BLE. For applications with variable throughput requirements that allow for some latency, we have shown that substantial energy savings can be achieved by applying our proposed scheme. Applications with tighter latency constraints, such as wireless mice or keyboards, are expected to achieve lower savings, but can nevertheless benefit from our proposed system. We suggest that an algorithm such as the buffer-level based strategy or the periodic adjust strategy is included into future BLE stacks, as the effort that needs to be spent is low and the computational cost is negligible. We also believe that with some more fine-tuning, the strategies could be further improved in terms of latency and energy savings. If the number of intervals after which a connection parameter update takes effect in BLE could be decreased, the proposed algorithms are expected to achieve higher savings while the maximum latency can be reduced significantly. Additional constraints on the possible values of the connection interval are imposed if one master has to serve multiple slaves. Solutions on how to extend our concept to such scenarios need to be addressed in further research.

The authors would like to thank the anonymous reviewers for their helpful comments. This work was partially supported by *HE2mT - High-Level Development Methods for Energy-Saving, Mobile Telemonitoring Systems*, a project funded by the federal ministry of education and research of Germany (BMBF) and the *EIT ICT labs* in the medical CPS activity, a project funded by the European Union.

REFERENCES

[1] Stollmann Entwicklungs- und Vertriebs-GmbH, "Terminal i/o profile bluetooth low energy profile," 2013. [Online]. Available: www.stollmann.de

[2] "SPP-over-BLE application note," Bluegiga, 2013. [Online]. Available: www.bluegiga.com

[3] H. Wang, M. Xi, J. Liu, and C. C., "Transmitting ipv6 packets over bluetooth low energy based on bluez," in *15th International Conference on Advanced Communication Technology (ICACT)*, 2013.

[4] C. Gomez, I. Demirkol, and J. Paradells, "Modeling the maximum throughput of bluetooth low energy in an error-prone link," *IEEE Communications Letters*, vol. 15, no. 11, pp. 1187–1189, 2011.

[5] "2.5 billion bluetooth low energy chipsets to ship in 2014, says ABI research," ABI Research, 2009. [Online]. Available: www.reuters.com

[6] Bluetooth SIG, "Specification of the bluetooth system 4.0," June 2010, volume 0. [Online]. Available: bluetooth.org

[7] *BLE112 data sheet*, Bluegiga, 2011, version 1.21, www.bluegiga.com.

[8] "Find me profile specification v10r00," Bluetooth SIG, 2011. [Online]. Available: www.bluetooth.org

[9] R. Heydon, *Bluetooth Low Energy: The Developers Handbook*. Prentice Hall, 2012.

[10] P. Kindt, D. Yunge, R. Diemer, and S. Chakraborty, "Precise energy modeling for the bluetooth low energy protocol," *arxiv.org*, 2013. [Online]. Available: http://arxiv.org/abs/1403.2919

[11] M. Siekkinen, M. Hiienkari, J. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? comparative measurements with zigbee/802.15.4," in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2012.

[12] S. Albers, M. Charikar, and M. Mitzenmacher, "Delayed information and action in on-line algorithms," in *39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1998.

[13] M. Perillo and W. Heinzelman, "ASP: an adaptive energy-efficient polling algorithm for bluetooth piconets," in *36th Annual Hawaii International Conference on System Sciences (HICSS), 2003*, 2003.

[14] H. Z., G. C., G. Kesidis, and C. Das, "An adaptive power-conserving service discipline for bluetooth," in *IEEE International Conference on Communications (ICC)*, 2002.

[15] S. Garg, M. Kalia, and R. Shorey, "MAC scheduling policies for power optimization in bluetooth: a master driven TDD wireless system," in *51st IEEE Vehicular Technology Conference (VTC)*, 2000.

[16] D. Contreras and M. Castro, "Adaptive polling enhances quality and energy saving for multimedia over bluetooth," *IEEE Communications Letters*, vol. 15, 2011.

[17] V. Pallipadi and A. Starikovskiy, "The ondemand governor: past, present and future," in *Proceedings of Linux Symposium, vol. 2*, 2006.

[18] A. Watts, "On exponential smoothing of discrete time series (corresp.)," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 630–630, 1970.