BurstMAC — An Efficient MAC Protocol for Correlated Traffic Bursts

Matthias Ringwald, Kay Römer, Institute for Pervasive Computing, ETH Zurich, Switzerland

***Abstract*— Many sensor network applications feature correlated traffic bursts: after a period of idle time with almost no network traffic, many nodes have to transmit large amounts of data simultaneously. One example is volcano monitoring [17], where rare eruptions trigger many nodes simultaneously to transmit seismic data traces.**

For such applications, a MAC protocol should exhibit a low overhead both in idle mode *and* during correlated traffic bursts. Existing MAC protocols only meet one of these requirements, but not both at the same time. Contention-based protocols such as SCP-MAP have very low overhead in idle situations, but exhibit high overhead during correlated traffic bursts due to contention. Scheduled protocols such as LMAC avoid contention overhead during correlated bursts, but exhibit a significant overhead in idle mode due to control traffic [10].

We propose BurstMAC, a new MAC protocol specifically designed for applications with correlated traffic bursts. BurstMAC's idle overhead is almost as low as that of SCP-MAC and at the same time provides better throughput and lower energy overhead than LMAC during correlated traffic bursts.

I. Introduction

Many early applications of sensor networks were *time-triggered*, where sensor nodes sample their sensors at regular intervals and report these readings to a sink. In *event-triggered* applications, in contrast, sensor nodes do not transmit any data unless a relevant real-world event occurs. In a volcano monitoring application [17], for example, sensor nodes detect volcanic eruptions by sampling their sensors. Only when a node detects an eruption, it sends a lengthy time series of sensor values to the sink, generating a *traffic burst* in the network. Because an eruption typically triggers many nodes simultaneously, the occurrence of traffic bursts produced by different nodes are highly *correlated* in time. Here, it is important that all data is collected in a reliable and timely manner with low energy overhead. However, it is equally important that the energy overhead during idle periods between eruptions is also very low.

Existing MAC protocols are not well-suited for such event-triggered applications with correlated traffic bursts as they are efficient either in idle mode or during correlated traffic bursts, but not both. Contention-based protocols such as SCP-MAC are very efficient in idle mode because they minimize the control overhead required for node coordination. However, during correlated traffic bursts when many nodes compete for the radio channel, their efficiency is rather low due to the control traffic required for contention. In contrast, scheduled protocols such as LMAC eliminate contention overhead during correlated traffic bursts, but exhibit a high overhead in idle mode due to the control traffic required for node coordination.

The contribution of this paper is BurstMAC, a MAC protocol specifically designed for event-triggered applications with correlated traffic bursts. BurstMAC's idle overhead is almost as low as that of contention-based protocols such as SCP-MAC and at the same time provides better throughput and lower energy overhead than extsing scheduled protocols such as LMAC during correlated traffic bursts. The key to achieve this goal lies in a novel combination and tight integration of different medium access techniques. In particular, BurstMAC uses TDMA techniques and multiple radio channels to efficiently handle correlated traffic bursts. However, to eliminate most of the control overhead typically induced by these techniques, we apply physical layer approaches such as efficient transmission of single bits without preambles or concurrent, non-destructive transmissions by multiple senders, resulting in a low overhead in idle mode.

While BurstMAC also supports broadcasts, it is optimized for unicast transmission. This is grounded in the belief that nodes may rarely want to disseminate large amounts of data to many other nodes. That is, BurstMAC is optimally suited for applications where large amounts of data have to be extracted from the sensor network (e.g., using convergecast with a single or multiple sinks) or where large amounts of data have to be transferred between pairs of nodes.

The remainder of this paper is structured as follows. We first put BurstMAC into the context of related work in Sect. II, before we give an overview of BurstMAC and its key techniques in Sect. III. The protocol is described in detail in Sect. IV. An implementation of BurstMAC on BTnodes is discussed in Sect. V, while performance results are reported in Sect. VI before drawing conclusions.

II. Related Work

Among the large number of MAC protocols for sensor networks [10], two categories are of particular relevance for our work. Firstly, MAC protocols that achieve a very low energy overhead in the idle case, and, secondly, MAC protocols that have a low overhead during correlated traffic bursts.

State-of-the-art MAC protocols with respect to low idle overhead are WiseMAC [4], SCP-MAC [20], and Dozer [3]. All of these achieve idle radio duty cycles well below 1%. While WiseMAC and SCP-MAC are general purpose MAC protocols designed for low data rates, Dozer is an integrated data collection stack for ultra-low data rates, consisting of MAC layer, topology control, and routing. However, being

The work presented in this paper was partially supported by the Swiss National Science Foundation under grant number 5005-67322 (NCCR-MICS).

designed for low data rates, these protocols suffer from significant overhead due to contention during correlated bursts. While Dozer uses local one-hop schedules to avoid collisions, transmissions of nodes in different scheduling domains still collide. WiseMAC improves on the low-power listening technique [10] where unsynchronized nodes periodically wake-up from sleep to detect a radio transmission. To assert that the receivers wakes up in time, a long preamble has to be prefixed to the packet. WiseMAC reduces this extra preamble by keeping track of neighbors' sleeping schedules. Multiple parallel transmissions (e.g., during correlated bursts) are not scheduled and lead to collisions. SCP-MAC is also based on low-power listening but synchronizes the sleep schedules of all nodes to send a packet just after all neighbors wake up. It also provides a streaming mode that allows packets to be forwarded through the network in a staggered fashion. The synchronized medium access, however, leads to a higher chance of collisions, which have to be resolved by back-off mechanisms.

On the other end of the design space, collision-free protocols have been designed for higher data rates by employing scheduling techniques. The most well-known and widely used protocol in this category is LMAC [15]. In LMAC, each node is assigned to a slot in a TMDA schedule and can transmit during this slot. All nodes must listen during the slots of all its neighbors to be able to receive data, which implies a significant energy overhead in all but very sparse networks, resulting in a radio duty cycle $\gg 1\%$ even in idle mode.

Very recently, first MAC protocols for WSN have been published that make use of multiple radio channels. While these protocols aim to increase the effective network bandwidth, they perform worse than existing single-channel protocols with respect to energy efficiency. In particular, two such multi-channel protocols have been proposed and implemented for the ChipCon CC2420 radio transceiver. The protocol by H. Le et. al. [11] scatters nodes over different channels to increase the channel bandwidth. However, energy-efficiency has not been addressed and the additional periodic channel update packets that are used to maintain connectivity increase energy consumption significantly for the idle case. Y-MAC [7] uses channel hopping to cluster all nodes that send to the same receiver, such that neighboring clusters use different channels. However, contention is still used for each transmitted packet similar to SCP-MAC. In particular, all nodes that try to send to the same receiver (which is the common case in tree-based collection protocols) suffer from the same contention overhead as SCP-MAC.

Recently, several protocols for data collection and dissemination on top of other MAC protocols have been proposed. Most notably, Flush [6] is a protocol for reliable delivery of bulk traffic. However, Flush makes the fundamental simplification that there is no inter-flow interference, i.e., only one node "flushes" at a time in the whole network. This requires a global scheduling of the nodes, which is not practical in many situations, for example, if a dynamically changing subset of nodes has data to report. Typhoon [12] is a code dissemination protocol that extends Deluge by introducing multiple radio channels to increase spatial diversity. While Typhoon focuses on broadcast dissemination, BurstMAC is tailored for unicast transmissions.

BurstMAC is loosely based on our previous work on BitMAC [14]. While building on similar basic techniques and sharing some goals, BurstMAC is very different on a technical level. In particular, it supports dynamic topology changes (BitMAC only works for fixed topologies), supports different communication patterns (BitMAC only supports convergecast), and has significantly relaxed requirements on the accuracy of time synchronization.

Physical layer techniques, in particular cooperative transmissions have also been considered by other authors, e.g., [8], [9]. There, multiple senders transmit concurrently a jam signal, such that a receiver can detect that at least one sender transmitted a jam signal. To reduce the effect of destructive interference, the senders in [8] use slightly different frequencies such that the different signals can cancel out each other only during short periods of time. If the receiver listens long enough it will detect the jam signals with high probability. We use a variant of this technique described later in the paper.

III. Protocol Overview

In this section, we present the key ideas behind BurstMAC and outline the basic protocol structure. A detailed discussion of the various BurstMAC components can be found in Sect. IV. BurstMAC combines a number of techniques to provide high throughput and efficiency under load *and* low idle overhead. Most notably, scheduling and the use of multiple radio channels enable high throughput, while cooperative transmissions and techniques to eliminate preambles guarantee low idle overhead. Also, the different techniques are integrated in an innovative manner. For example, scheduling of time slots and assignment of channels is combined to reduce the overhead further.

One important underlying assumption on the radio is that a sufficient number of radio channels is supported. In particular, that number should be larger than the maximum number of two-hop neighbors of a node in the network. In our implementation on the ChipCon CC1000 radio, we use 32 data and two additional control channels as detailed below.

General Approach. To avoid collisions, BurstMAC operates in synchronous rounds. Each round consists of N frames with N=32 in our implementation. Every frame contains a small CONTROL section and a large DATA section as depicted in Fig. 1. To maximize throughput and to allow for collision-free communication during the DATA section, BurstMAC uses N interference-free data channels and one control channel. The CONTROL section is used for time synchronization, to broadcast other information to all network neighbors, and to assign color ids to nodes. As a result of the latter, each node is assigned a color id $c \in 1..N$ that is unique within two hops. The color id c is used for two purposes. Firstly, a collision-free TDMA schedule of the control channel is implemented, such that a node with color id c sends a control message in the CONTROL section of frame c. Secondly, node c coordinates multiple senders on radio channel c during the DATA section, which allows it to receive data without collisions. The set of

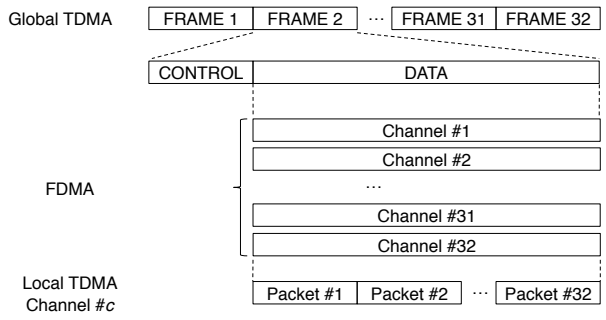

Fig. 1. BurstMAC protocol overview: Synchronous rounds consist of 32 frames (global TDMA). Each frame contains a CONTROL and a DATA section. Multiple radio channels are used for interference-free communication of collocated node clusters (FDMA). Communication within a node cluster is coordinated by local scheduling (local TDMA).

nodes which act as coordinators during a frame is determined by a *coordination-free transmission scheduling*, as described in the next paragraph. By using multiple channels, contrary to basic TDMA, nodes are able to send packets during multiple frames, which increases the total network bandwidth.

To achieve a low duty cycle in idle situations, the frame length is chosen rather large, 1s in our implementation. Therefore, a node is required to turn its radio on for the duration of the (short) control message in every frame. During the data section, coordinator nodes need to check for neighor transmissions. Due to the coordination-free transmission scheduling described below, a node is a coordinator in every other frame on average. This results in an idle duty cycle below 1%.

Coordination-free Transmission Scheduling. As a node cannot send and receive at the same time due to the half-duplex nature of typical low-power radios, some form of coordination is required among the nodes to achieve agreement on when to send and when to receive. To realize this coordination without introducing additional control messages, BurstMAC uses the following approach. During each frame, a node is either in transmit or receive mode, that is, it can either only transmit or only receive data during the whole frame. The choice of mode is controlled by a pseudo-random number sequence that is seeded with the unique 16-bit node id. Knowing the node ids of its neighbors, a node can not only compute its own current mode, but also the current modes of its neighbors. If node A wants to send to neighbor B, then A has to wait for a frame when it is in send mode and B is in receive mode. A uses B's channel for the actual transmission. This approach is loosely related to pseudo-hopping sequences used, e.g., in Bluetooth, or the uniform distribution of wake-up times in JAVeLEN [5], and avoids extra traffic for coordination among nodes.

Cooperative and Single-Bit Transmissions. While scheduled protocols are efficient during correlated bursts, they introduce coordination overhead that adds to a rather high idle overhead. We use two physical layer techniques to largely eliminate this overhead. The first technique allows for efficient checking if any neighbor needs to transmit a message. Neighbors that want to send a message transmit a short jamming signal [9] simultaneously. A receiver can use the Received Signal Strength Indicator (RSSI) to detect that at least one node is sending. This *cooperative transmission* technique requires only minimal radio-on time to detect an idle situation, see Sect. IV-B for details. A further source of overhead is the need to precede each packet with a long preamble to synchronize sender and receiver at the bit level. In many cases, these preambles are longer than the actual payload data. This is especially problematic for control traffic in a MAC protocol, which often consists of only few bits of information. To reduce this overhead, we employ *single-bit transmission* in BurstMAC, where a coordinating node broadcasts a short synchronization packet to provide a bit-accurate time reference for a set of receiver nodes. These nodes can then send single bits of information without any preambles to the coordinator.

Packet Bursts. To increase throughput and reduce communication overhead, a sender can request the transmission of multiple packets in a row, eliminating lengthy preambles for all but the first packet. In contrast to the standard approach of sending packets back-to-back, BurstMAC reliably transmits the packet lengths of each individual packet and provides each packet with a checksum to detect bit errors. By this, the checksum of each packet can be checked separately and a single bit error does not cause the whole packet train to be dropped. The receiver acknowledges the packet burst reception with a bit vector that contains a set bit for each packet that has been received correctly.

Cross-layer Optimizations. Typical routing protocols such as MintRoute [19] need to perform neighbor discovery and link quality estimation, which requires each node to broadcast beacon packets at regular intervals. However, due to the existence of the control packets in BurstMAC, we can integrate neighbor discovery and link estimation into BurstMAC without significant overhead.

IV. Protocol Details

In this section, we provide a detailed description of the BurstMAC protocol. Key functional components are channel assignment, cross-layer support for routing (i.e., neighbor discovery and link estimation), actual data transmissions, as well as time synchronization (i.e., establishing a common time among the nodes in the network) and network startup (i.e., how nodes join a BurstMAC network).

The key protocol element to enable the above functions is the CONTROL section, where a node broadcasts a control message to all its neighbors on a common control channel in frame c of each round, where c is the color id that has been assigned to the node. As shown in Fig. 2, the control message consists of a fixed part which is always present with basic information for time synchronization, and coloring, as well as a dynamic section which contains one or more optional headers. A `flags` field indicates which of the optional headers are present in a message. We will reference the various field of the control message throughout this section.

```
struct CONTROL {
  u_short node_id;         // MAC address of sender          (Coloring)
  u_long  occupied;        // vector of occupied control slots (Coloring)
  u_char  current_frame;   // frame number                   (Timesync)
  u_char  round;           // round counter, incremented by sink (Timesync)
  u_char  hops;            // hop distance to gateway        (Timesync)
  u_short timestamp;       // SOP time relative to frame start (Timesync)
  u_char  flags;           // type of dynamic data           (Dynamic Data)
  u_char  dynamic_length; // total length of dynamic data    (Dynamic Data)
  u_char  dynamic_data[0];                                  (Dynamic Data)
  u_short crc;
};
```

Fig. 2. Contents of the control message.

A. 2-Hop Coloring

Each node has to be assigned a color id c that is unique within two hops. As discussed in Sect. III, c is used for two purposes: as a channel id for payload data transmissions and to schedule broadcasting of control messages on the control channel.

For coloring, all nodes keep track of the frames used by their neighbors for sending control messages similar to LMAC [15]. As a node with color id c transmits a control message in frame c, each node is aware of the colors assigned to its neighbors and periodically broadcasts a bit vector of these occupied color ids in its control message (field `occupied`). A newly joining and yet uncolored node with id i receives the list of used color ids in the control messages of all of its neighbors. The union of these sets equals the set of color ids used in its 2-hop neighborhood. The new node then randomly picks a color id c from the remaining free colors and transmits its control packet in frame c in the next round. A special flag in the control message requests other nodes to echo the node id contained in the control packet of frame c. If another node with node id j simultaneously picks the same color c, both node ids i and j will be reported for frame c by different neighbors. In this case, both newly colored nodes pick another free color at random.

Topology changes in the network may lead to a situation where two nodes with the same color id are two hops or less apart. This situation must be detected and at least one of the nodes must pick a new color id. For detecting such a situation, each node monitors the node id (field `node_id`) contained in the control messages. If a node observes that the node id of the control message in frame c is different from the node id of the control message in frame c during the previous round, the observing node reports in its next control message that nodes with color c should pick a new color according to the above procedure. Here we exploit the capture effect [18] – where a node will receive the stronger of two concurrent transmissions on the same channel rather than seeing a collision – paired with the spatial diversity of the nodes.

B. Transmission Scheduling

As discussed in Sect. III, each node is either in receive or in send mode during each frame, using a pseudo-random generator to control the choice of mode. Using these pseudo-random sequences, a node can compute whether it can send to a certain node during a certain frame. However, more than one sender may want to transmit to a single receiver during the same frame on the same channel. Hence, these senders have to be coordinated in some way to avoid collisions. A key goal of BurstMAC is to minimize this coordination overhead in the idle case, where no sender wants to transmit.

Our solution is illustrated in Fig. 3, which shows the DATA section in more detail, time increases from left to right. One node is in receive mode, and two nodes in send mode (with colors $c = 1$ and $c = 3$) want to transmit a packet to the receiver. In segment A, both senders employ cooperative transmission and concurrently transmit a short jamming signal (i.e., unmodulated carrier signal) during 500 us to indicate a send request. If the receiver does not detect a jamming signal (i.e., if RSSI always smaller than a certain threshold during these 500 us), it can go back to sleep, optimizing the overhead in the idle case.

However, the simultaneous transmission of jam signals of multiple senders may result in destructive interference at the receiver, such that the receiver won't detect the presence of multiple jam signals. This problem can be addressed if senders transmit on slightly different frequencies, such that a beat with a certain period time will result at the receiver. If the receiver listens for at least that period time, it will encounter non-destructive interference at some point in time and detects the jam signal. Fortunately, there is a natural frequency diversity due to variations of crystal frequencies among nodes. For a 868 Mhz carrier signal, for example, a crystal frequency difference of 10 ppm between two transmitters results in a beat with a period length of 115 us, and a high probability for the receiver to detect the transmission if it listens for at least this amount of time. In our implementation, we used 500 us, which further increases the chance that a receiver will encounter non-destructive interference. In tests, we could not observe a significant drop in the number of detected jam signals when using multiple transmitters instead of a single one. If this natural diversity should not be enough to prevent errors from destructive interference on other hardware, additional frequency diversity could be created explicitly as detailed in [8].

If a receiver detects a jam signal (i.e., if there is at least one sender), then the receiver exerts the single-bit transmission technique by transmitting a minimal sync packet consisting of a preamble and a start-of-packet symbol in segment B in Fig. 3. The sync packet allows a sender to accurately synchronize with other senders and the receiver. Each sender then transmits a single jamming "bit" in the bit slot which corresponds to its color id c in segment C in Fig. 3. Based on the list of senders, the receiver then computes and broadcasts the transmission schedule in segment D. This schedule is essentially a copy of the bit vector received in section C, where bit i is set iff a sender with channel i has submitted a send request. However, if the receiver has insufficient buffer space to store all packets, it will randomly erase a bit until the number of remaining bits equals available buffer space. This way, we achieve a simple form of flow control to avoid packet loss due to buffer overruns.

A sender with channel i checks if bit i in the received schedule bit vector is set. If this is the case, the sender transmits a data packet in slot E_j, where j is the number

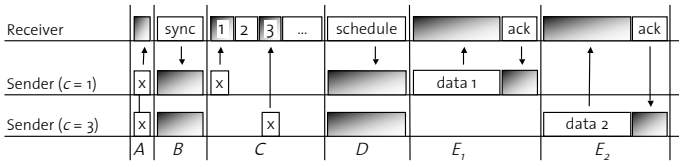

Fig. 3. Data SECTION with two senders transmitting a packet to the receiver. Cooperative transmission is used in segment A and single bit transmission in segments B and C to identify the senders.

of "1" bits in the schedule bit vector with an index smaller than i. The slots are of fixed length such that each sender can compute start and end of its slot from the schedule bit vector. To deal with packet loss caused by bit errors, each data packet is immediately acknowledged by the receiver. The sender retransmits the packet if it does not receive the acknowledgment in time. To eliminate duplicates caused by lost acknowledgments, each data packet contains a small header with a one byte sequence number, which also allows FIFO delivery of packets at the receiver.

The header of the data message also contains several flags. One of them, the `more` flag, can be set by the sender to indicate that it needs another slot to send a further packet. If available, the receiver will reply an unused slot k in the acknowledgment, such that the sender can send the next packet in slot E_k.

C. Packet Bursts

To reduce the probability of packet loss due to bit errors, packet sizes are rather small in BurstMAC as well as in all other MAC protocols for sensor networks. However, this results in substantial overhead due long preambles and due to the fact that both sender and receiver need to switch the radio back and forth between transmit and receive mode.

To reduce this overhead, BurstMAC offers a so-called burst mode, where a sender can use almost the whole DATA section of a frame to send a sequence of packets back-to-back preceded by only a single preamble and acknowledged with a single message. Each individual packet in the sequence has its own CRC, such that a bit error destroys only a single packet and not the whole burst[1]. A sender requests a burst transmission by sending a message with the `burst` and `more` flags set. In case of burst requests by multiple senders, the receiver grants burst access to a randomly selected sender to ensure faireness. Next, the sender transmits a message containing the lengths of all packets in the burst to allow the receiver to detect packet boundaries and check CRCs. Next, the sender sends the data packets back-to-back without individual premables.

D. Time Synchronization

As BurstMAC makes use of synchronous rounds, we need some form of global time synchronization to make sure that rounds and frames begin at the nodes at approximately the same time. Note that the required synchronization accuracy is rather low as the time-critical events in BurstMAC are the transmission of control messages and the single-bit transmission at the beginning of the DATA section. We set the duration of the single-bit to 500 us as described in Sect. IV-B. A maximal synchronization error of half that duration between two neighboring nodes ensures an overlap of at least 250 us among neighbors for detection of the jam signal.

For time synchronization, we assume that a dedicated node provides a time reference for the whole network (e.g., the data sink). In contrast to typical tree-based synchronization protocols where each node synchronizes to a single parent node, BurstMAC uses a more robust and accurate approach where each node synchronizes to the average time of all its parents (i.e., all nodes that are closer to the time reference than itself).

An important issue that needs to be dealt with is the dynamic nature of wireless links, that is, the set of parents of a node changes over time. In fact, a parent of a node may turn into a child (i.e., a node that is farther away from the reference as the node) due to a broken link. Here, we must avoid synchronizing to the former parent, which has become a child. To deal with this issue, the control message contains both a `hops` counter that holds the distance of the sending node from the reference, and a `round` number. The latter is incremented only by the reference node before broadcasting a control message, all other nodes broadcast a copy of this value without incrementing. If a node receives a control message from a parent (i.e., a node which has a smaller hop counter than the node), whose round counter is the same as in the previous message from this parent, then the parent may have turned into a child and is not used during time synchronization of the node.

Time synchronization is realized by accurately timestamping the transmission and reception of the first bit of the control message in the radio interrupt handler. With this, we obtain a per-hop synchronization accuracy of few microseconds.

A node performs these measurements for two consecutive control messages from each parent and computes the duration of a round for each of the parents, which may differ from parent to parent due to different clock rates and drift. The node then computes the average round duration of its parents and adjusts its local round duration, increasing/decreasing the number of clock ticks of a frame if the average round duration of the parents is larger/smaller than its own round length. As the duration of a clock tick of a typical 32 kHz real-time clock is too large for this correction (resulting in a minimal increment per round of 30.5 $us \times 32$ frames), we use a variant of Bresenham's algorithm [1] to adjust the *average* frame length in smaller increments using only integer arithmetic. This approach proved to be very robust to outliers.

E. Network Startup

Network startup is concerned with the problem of how nodes join a BurstMAC network. In fact, the main problem is getting in sync with the time reference. For example, during deployment nodes are switched on in random order, so each

[1]Note, that modern radio transceivers can receive arbitrarily sized packets without loosing the bit synchronization, even in the presence of bit errors. Therefore, the main caveat with sending packets back-to-back is the risk of a bit error in the length field of a header, as the start of all later packets would be missed and, hence, would have to be discarded.

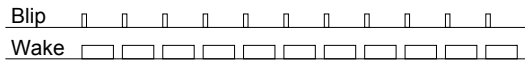

Fig. 4. (Top) Periodic start-of-frame signal: Blip, (Bottom) Wake-up signal with integrated start-of-frame information. The duration of Blip and wake-up in our prototypical implementation are 0.1 ms and 999.8 ms respectively.

node needs to wait until it gets connected to a time reference. As this waiting process may last quite long, we need to make sure that nodes do not spend much energy in this waiting state.

Our approach uses a dedicated wake-up channel where nodes transmit wake-up signals. Communication on this channel uses cooperative transmissions, i.e., concurrent transmissions by multiple nodes overlay in a non-destructive manner. On this channel, every node transmits a very short jamming signal (100 us), called *Blip*, at the begin of *every* CONTROL section as depicted in Fig. 4. By this, a new node joining the network has to scan the wake-up channel at 100% duty cycle only for a single frame instead of a whole round. On detection of the Blip, the node obtains rough time information (i.e., when CONTROL section starts), which is sufficient to receive at least one control message in the next 32 frames. Based on this first control message, a node sets its local clock to the time of the sender. After that, only the frame lengths are adjusted to stay synchronized as described in the previous section.

If a node starts up and the network is not active yet, it will not receive a Blip during one frame and start low-power listening with a period T slightly smaller than the duration of a frame. That is, the node wakes up every T time units, scans the wake-up channel for ongoing transmissions, and goes back to sleep.

When the time reference starts up, it sends a pulsed jamming signal, called *Wake*, during the first frame to wake up nodes from low-power listening. Each jamming pulse starts exactly at the begin of the CONTROL section, providing woken-up nodes with a rough time information similar to the Blips. The idle-listing period T equals the duration of a wake-up pulse, which is slightly shorter than the frame length. Note that Blips and Wakes have been designed such that the parallel transmission of Blips and Wakes results in Wakes.

Nodes that have been woken up this way will transmit Wakes themselves to wake up nodes further away from the time reference, such that eventually the whole network will be activated and synchronized. By this, the long Wake pulse has to be emitted at most once, which allows for energy-efficient deployment of the network.

V. Implementation

We implemented BurstMAC on BTnode Rev. 3 nodes [2] using its ChipCon CC1000 radio module. The CC1000 on the BTnode is configured for the 868 MHz ISM-band, where the actual baseband frequency within this band is configurable by software. We used 34 channels of which one is reserved for control broadcasts and one is used for the wake-up mechanism. The other 32 are used for data communication. The analog RSSI output of the CC1000 is used for clear-channel assessment, cooperative and single-bit transmission. We further make use of the CC1000's ability for precise MAC layer timestamping in the order of 10 us [13].

For the implementation, we used a bit rate of 19200 bps and a frame length of 1 s, which is split into 50 ms for the CONTROL section and 950 ms for the DATA section. By this, the DATA section provides 24 data slots for single bit transmission or up to 51 packets in burst mode. Each packet contains a type field and up to 32 bytes of payload. In this configuration, a node can send or receive a single packet and up to 51 burst packets of each 33 bytes in a single frame. Such a transfer of 1716 bytes results in a maximal usable bandwidth of 71.5% of the total bandwidth of 2400 bytes/s.

In fact, BurstMAC was also used to perform firmware updates during the experiments and to collect measurement results from the network, thus demonstrating BurstMAC's reliability.

VI. Evaluation

We study the performance of the BurstMAC implementation on BTnodes. In particular, we investigate the accuracy of time synchronization, the idle overhead, as well as overhead and time to completion of correlated bursts. We compare BurstMAC against two established protocols that represent the two ends of the design space that are relevant for our work. Firstly, we choose SCP-MAC as a state-of-the-art contention-based protocol with very low idle overhead. Also, SCP-MAC does contain an adaptive channel polling mode which allows it to adapt to bursty traffic. Secondly, we chose LMAC as a scheduling-based and, hence, collision-free protocol that has been shown to outperform many other MAC protocols under high data rates [10].

A. SCP-MAC and LMAC

In order to be able to compare the three protocols on a common hardware platform, we have implemented SCP-MAC and LMAC on the BTnode. In fact, we can switch between all protocols at runtime, performing firmware updates and collecting measurement results with BurstMAC, while the actual experiments make use of SCP-MAC or LMAC (or BurstMAC). The implementations are based on published papers and source code, with some additions and parameter choices to enable a fair comparison.

For LMAC, we used a frame length of 150 ms. By this, it is possible to send up to 300 bytes or up to 7 packets of maximal size in one frame similar to [15]. As LMAC provides only a single ACK per frame, increasing the frame length would result in a higher chance that all packets sent in a frame have to be dropped if a single bit error occurs. In LMAC, one or multiple payload packets are sent directly after the control packet. As described in a technical report [16], multiple packets are sent back-to-back with a single CRC at the end. The receiving node(s) acknowledge the reception of data packets in the control message of its own slot. If no acknowledgment is received, the packets will be retransmitted. If an acknowledgment gets lost, the destination will receive the packets a second time. Our only modification to LMAC

is the use of sequence numbers as in BurstMAC to suppress duplicate packets.

We implemented SCP-MAC as described in [20], using the published source code for clarification. We used 8 contention slots before the wake-up tone and 16 for the second contention window as described in the paper. The duration of a single contention slot is 427 us, which is the minimal time for a node to assert a free channel, switch to transmit, and allow other contenders to detect this transmission at the start of the next contention slot. We also implemented overhearing avoidance by checking the destination address in a packet before it is completely received. Our implementation differs from the original paper in two aspects: time synchronization and acknowledgements. For the time synchronization, we embedded SCP-MAC into the DATA section of BurstMAC, using the CONTROL section of BurstMAC for time synchronization. By this, the channel polling period in SCP-MAC is identical to the BurstMAC frame length of one second. As BurstMAC's CONTROL section serves more purposes than just time synchronization and because SCP-MAC could tolerate less frequent time sync updates, we exclude the radio time used for the CONTROL section from SCP-MAC's energy measurements. By this, SCP-MAC gains a slight advantage over both LMAC and BurstMAC, but this advantage is relevant only in an idle scenario. To allow for efficient and reliable link-layer packet transmission, we modified SCP-MAC to let nodes acknowledge the reception of a correct packet. We believe that this is favorable compared to other mechanisms, as for unicast packets, only the sender and receiver are active at this time allowing for a contention-free transmission of the acknowledgment.

B. Time Synchronization

We study the accuracy of time synchronization as a function of the network diameter to investigate the maximum network diameter that BurstMAC can support. For this, we arrange 10 nodes in a chain topology with the time reference at the end, forcing each node to use only its direct parent as a reference for synchronization. However, all nodes are within communication range of the time reference, such that each node can directly measure its synchronization error with respect to the time reference. We ran this setup for one hour, where each node measures its synchronization error once per round. After collecting the measurements from the network, we computed averages and standard deviations for all nodes in the chain as depicted in Fig. 5 (left). The results show that the average synchronization error increases by about 25 us per hop. Note, however, that the accuracy of our clock is only 30.5 us. As the required synchronization accuracy is in the order of few milliseconds (we only need to synchronize to rounds, not at the byte or bit level), we can easily support networks with a diameter in the order of several tens of hops.

C. Idle Case

We investigate the idle overhead of the protocols in terms of the radio duty cycle. For both BurstMAC and LMAC, the radio duty cycle is a function of the number of neighbors of a node

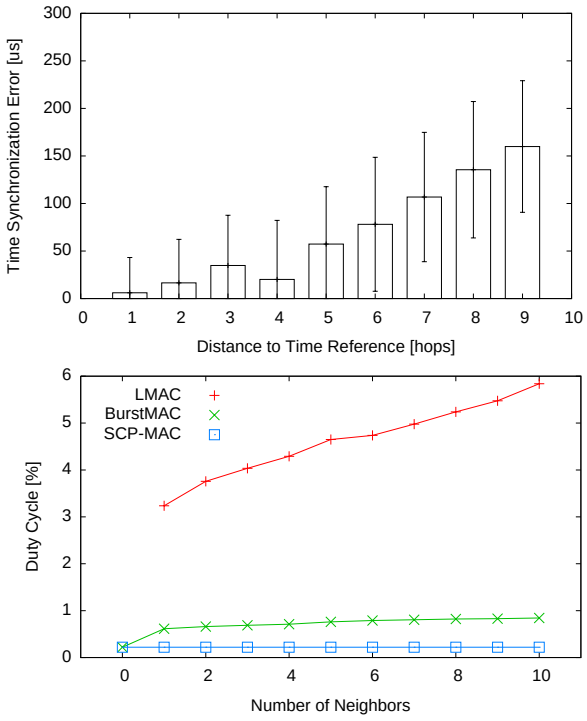

Fig. 5. Accuracy of BurstMAC time synchronization on a chain topology (top). Radio duty cycle in idle mode as a function of neighborhood size (bottom).

as a node has to receive the control message from each of its neighbors. Therefore, we study the radio duty cycle of a node with a varying number of neighbors. For each neighborhood size, we ran the network for 5 minutes, measuring the time the radio was on and compute the average duty cycle as depicted in Fig. 5 (right). A neighborhood size of zero represents a special case. In this situation, BurstMAC behaves as described in Sect. IV-E and switches to low-power listening. As the behavior or LMAC is not specified in the technical report [16], we have to omit this data point. For more than one neighbor, we find that the duty cycle increases by about 0.02% per added neighbor for BurstMAC and by about 0.5% for LMAC. The high duty cycle for LMAC is a consequence of its short frame rate which is required to deliver bursty traffic. The duty cycle for SCP-MAC, without time synchronization or neighborhood discovery packets, stays constantly at 0.22%, which is the cost of one clear channel assessment per second. When comparing idle overhead to other MAC protocols, we need to take into account that the above numbers for BurstMAC already contain the overhead for maintaining the routing topology and allow for the collision-free sending of short broadcast messages without extra overhead.

D. Constant Traffic

In most sensor network applications, data collected in the network is extracted by a small number of sink nodes using convergecast. Therefore, each sink and its direct neighbors form a star topology in which all neighbors of the sink compete for the right to send to the sink. Similarly, the inner nodes in a data gathering tree form such a star topology. Hence, the

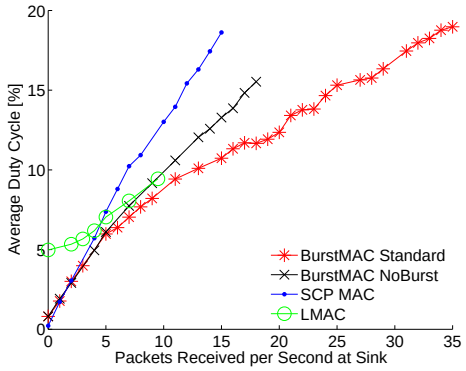

Fig. 6. Star topology with one sink and seven neighbor nodes: average duty cycle. Note that each protocol can only support constant traffic up to a certain packet rate.

performance of a star network, where N senders try to send to one receiver simultaneously, represents the performance bottleneck of tree collection protocols during correlated traffic bursts. Therefore, we first evaluate the performance of the protocols on such a star topology with $N = 7$ senders (a typical number to ensure connected networks) under different data rates.

In addition to the full version of BurstMAC, we also study BurstMAC's performance when the packet burst feature is disabled, termed BurstMAC NoBurst, to provide a better comparison to the other protocols. Each node creates packets with a constant data rate. Every 32 (resp. 38.4 for LMAC) seconds, the number of packets received without bit-errors and the total radio-on time are logged, and the packet rate is increased. We plot the average duty cycle per node in Fig. 6. Note that each protocol can handle a certain maximal packet rate depending on its design. To determine the maximal usable packet rate and the corresponding efficiency, we let the nodes send packets as fast as possible. Fig. 7 lists the results.

Although LMAC's idle duty cycle is higher than for other protocols, its collision and contention-free design allows to deliver packets more efficiently than SCP-MAC for packet rates higher than 6 packets/second. At its maximal packet rate of about 10 packets per second, its energy efficiency is even slightly better than BurstMAC without the packet burst mode. This results from the fact that LMAC only sends a single acknowledgement bit for up to 7 packets, whereas SCP-MAC and BurstMAC NoBurst both send complete acknowledgement packets. Overall, BurstMAC with its efficient burst mode has the lowest energy consumption and also delivers packets faster than the other protocols.

Below we study the performance of the protocols during correlated traffic bursts in a more realistic multi-hop network. As LMAC outperforms SCP-MAC already in the simple star topology for high data rates, we limit further comparisons to LMAC. Also, increased contention and potential hidden terminal problems in multi-hop networks makes things even worse for SCP-MAC.

E. Correlated Burst Case

To investigate protocol performance during correlated bursts in a more realistic scenario, we setup a multi-hop network of

	BurstMAC	BurstMAC (NoBurst)	SCP-MAC	LMAC
Packet Rate per Second	35.3	18.4	17.6	9.5
Radio Time per Packet [ms]	43.07	71.1	94.9	79.8

Fig. 7. Packet rate and radio time per packet for star topology with seven sender nodes.

30 nodes in our lab. The diameter of the resulting network is four hops and each node has at most ten neighbors. We simulate a traffic burst in the volcano monitoring application [17], where an eruption triggers all nodes simultaneously to transmit a burst of 10 KB (i.e., 320 BurstMAC packets). We measure the following metrics: firstly, the time it takes until all data has been successfully delivered to the sink, and secondly, the total radio energy spent in the network for delivering the correlated burst.

During preliminary tests, we realized that LMAC does not provide a flow control mechanism. In a burst traffic scenario with a high packet load, this would result in significant packet loss due to buffer overruns. Hence, we added a minimal flow control mechanism to LMAC by having a node announce whether or not it is ready to receive data in the control packet. A node is ready to receive when less than the maximum number of packets per LMAC frame is in the outgoing message queue. This simple stop-and-go flow control mechanism effectively avoids dropped packets due to buffer overruns. To evaluate the effect of our flow-control implementation for LMAC, we performed the experiment with BurstMAC, plain LMAC, and LMAC with flow control.

To compare the energy overhead of correlated bursts, we measure the radio-on time of each node from the start of the experiment until the node has finished sending all data and goes back to idle mode. By this, LMAC's comparatively high idle duty cycle does not affect the burst energy measurement. The results of the time measurement are shown in Fig. 8 (left).

For LMAC without flow control, we measured a packet retransmission rate of about 120% which means that each packet had to be sent more than twice. With our flow-control implementation, the packet retransmission rate drops to about 6%. Although BurstMAC also provides efficient flow control, we measured a packet retransmissions rate of about 12%. We attribute these packet losses to the fact that a lost burst acknowledgment (due to bit errors) currently requires to resend the whole burst although a significant number of packets might have been received correctly.

BurstMAC delivers the 9280 packets in 448 seconds which is about 5 times faster than both LMAC variants. The effectiveness of BurstMAC's multiple radio channels is limited as all traffic has to flow to the sink through a single channel. Hence, convergecast with a single sink can be considered as the worst case for BurstMAC with respect to LMAC. In scenarios with multiple (also collocated) sinks or point-to-point multi-hop traffic flows, BurstMAC's gain over LMAC will be even higher. In the convergecast scenario, the main benefit of BurstMAC's multi-channel approach is the reduction of collisions and energy consumption, as we show next.

To estimate the protocol overhead introduced by the MAC protocols, we measured for all protocols the total radio-on time based on the collected metrics. In addition, we calculated the

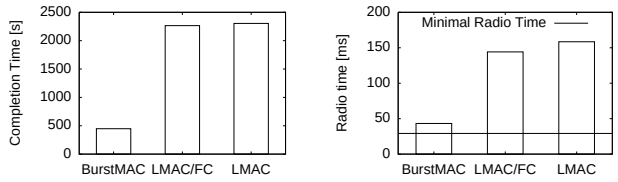

Fig. 8. Burst experiment: time to burst completion (left), average and minimal radio time per packet (right). LMAC/FC stands for LMAC with flow control.

minimal radio-on time based on a packet length of 35 bytes (1 byte type, 32 bytes payload and 2 byte CRC) to be 14.5 ms at 19200 kbps. As the transmission of a packet involves both the sender and the receiver, we define the minimal radio on-time as 29 ms. Fig. 8 (right) depicts the average radio time per packet per protocol (bars) together with the minimal radio-on time (horizontal line). BurstMAC required 43.24 ms which is at the same level as during the constant traffic experiment and shows that BurstMAC can achieve its energy efficiency also in multi-hop data gathering applications without extra configuration. Note that the complete overhead is only about 50% higher than the minimum and that this already includes the control messages and the overhead caused by sending preambles and acknowledgments. LMAC with flow control requires more than three times the minimal radio-on time. LMAC without flow control requires even more energy but not as much as we expected due to dropped packets caused by overruns, hence, the flow control did not significantly improve the delivery time, but it does reduce LMAC's energy consumption and should be added if LMAC is used in the future.

VII. Dense Networks

The number of available radio channels currently limits the maximal density of sensor networks. For random network topologies, the 32+2 channels used in BurstMAC allow for an average degree of about 8 nodes, which is enough to provide a well-connected network. To support more dense networks or to reduce the number of required radio channels, nodes can act as cluster heads for nodes that could not be assigned a free channel. Even without an assigned channel, such a *leaf node* can still receive broadcast messages, like all other nodes, e.g., to receive configuration data or a sensor query. As it does not control a channel, however, it cannot receive unicast messages from other nodes. But, it can still send unicast messages to other nodes on their respective channel. For this to work, the number of single request bits in the DATA section might be increased by $e + 1$ and each leaf node has to be assigned one of these e extra IDs by its new cluster head. While the first additional bit is used to signal the cluster head that a leaf node requests to be accepted, the other e bits are used to enumerate additional leaf nodes. If the first bit is set, the cluster head schedules a data slot in which all unassigned leaf nodes contend to send a packet with their node ID. If the cluster head correctly receives the node ID of a leaf node, it will announce in the next control message that node with ID X can use the additional request bit Y. In the case of collisions, the leaf nodes follow an exponential back-off strategy until all leaf nodes are associated with a cluster-head. With this scheme, the number of nodes in a given area can be increased by a factor of $e + 1$.

VIII. Conclusions

Applications with correlated traffic bursts require MAC protocols that have a low idle overhead *and* can efficiently handle simultaneous transmissions of bulk data by many or all nodes. Existing MAC protocols feature either low idle overhead *or* can efficiently handle correlated traffic bursts, but not both at the same time. Therefore, we proposed BurstMAC, which combines these two features in a single MAC protocol. This is achieved by combining scheduling and the use of multiple radio channels for efficient handling of traffic bursts with physical layer techniques to minimize the idle overhead. We could show that BurstMAC has a similar idle overhead as state-of-the-art low-power data collection MAC protocols under comparable conditions and that BurstMAC significantly outperforms existing scheduled protocols with respect to efficient handling of correlated traffic bursts.

References

[1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal 4, 1*, 4(1), 1965.
[2] BTnodes. A distributed environment for prototyping ad hoc networks. www.btnode.ethz.ch.
[3] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *IPSN*, 2007.
[4] A. El-Hoiydi, J. D. Decotignie, C. Enz, and E. L. Roux. WiseMAC, an Ultra Low Power MAC Protocol for the WiseNET Wireless Sensor Network. In *SenSys*, 2003.
[5] J. Redi et. al. Javelen – an ultra-low energy ad hoc wireless network. *Ad Hoc Networks*, 6(1), 1 2006.
[6] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *SenSys*, 2007.
[7] Y. Kim, H. Shin, and H. Cha. Y-MAC: An energy-efficient multi-channel MAC protocol for dense wireless sensor networks. In *IPSN*, 2008.
[8] A. Krohn, M. Beigl, C. Decker, and T. Riedel. Syncob: Collaborative time synchronization in wireless sensor networks. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS)*, pages 283–290, Braunschweig, Germany, June 2007.
[9] A. Krohn, M. Beigl, and S. Wendhack. SDJS: Efficient Statistics in Wireless Networks. In *ICNP*, 2004.
[10] K. Langendoen. Medium access control in wireless sensor networks. In H. Wu and Y.Pan, editors, *Medium access control in wireless networks, volume II: practive and standards*. Nova Science Publishers, 2007.
[11] H. K. Le, D. Henriksson, and T. Abdelzaher. A practical multi-channel media access control protocol for wireless sensor networks. In *IPSN*, 2008.
[12] C.-J. M. Liang, R. Musaloiu-Elefteri, and A. Terzis. Typhoon: A reliable data dissemination protocol for wireless sensor networks. In *EWSN*, 2008.
[13] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys*, 2004.
[14] M. Ringwald and K. Römer. BitMAC: A deterministic, collision-free, and robust MAC protocol for sensor networks. In *EWSN*, 2005.
[15] L. van Hoesel and P. J. M. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *INSS*, 2004.
[16] L. van Hoesel and P. J. M. Havinga. Design aspects of an energy-efficient, lightweight medium access control protocol for WSN. Technical Report TR-CTIT-06-47, University of Twente, Enschede, July 2006.
[17] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI*, 2006.
[18] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *EmNets*, 2005.
[19] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, 2003.
[20] W. Ye, F. Silva, and J. S. Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *SenSys*, 2006.