Andreas Reinhardt, Olivia Morar, Silvia Santini, Sebastian Zöller, Ralf Steinmetz: *CBFR: Bloom Filter Routing with Gradual Forgetting for Tree-structured Wireless Sensor Networks with Mobile Nodes*. In: Proceedings of the 13th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1-9, June 2012. IEEE Press, ISBN 978-1-4673-1239-4.

CBFR: Bloom Filter Routing with Gradual Forgetting for Tree-structured Wireless Sensor Networks with Mobile Nodes

Andreas Reinhardt*, Olivia Morar*, Silvia Santini†, Sebastian Zöller*, Ralf Steinmetz*

*Multimedia Communications Lab, Technische Universität Darmstadt, Germany
{andreas.reinhardt, olivia.morar, sebastian.zoeller, ralf.steinmetz}@kom.tu-darmstadt.de

†Wireless Sensor Networks Lab, Technische Universität Darmstadt, Germany
santinis@wsn.tu-darmstadt.de

***Abstract*—In tree-structured data collection sensor networks, packets are routed towards a sink node by iteratively choosing a node's immediate parent node as the next hop. It is however beyond the scope of these routing protocols to transfer messages along the reverse path, i.e., from the sink to individual nodes in the network. In this paper, we present CBFR, a novel routing scheme that builds upon collection protocols to enable efficient point-to-point communication. We propose the use of space-efficient data structures known as Bloom filters to efficiently store routing tables on the networked devices. In particular, each node in the collection tree stores the addresses of its direct and indirect child nodes in its local Bloom filter. A packet is forwarded down-tree only if the node's local filter indicates the presence of the packet's destination address among the node's descendants. In order to cater for the presence of mobile nodes, we apply the concept of counting Bloom filters to allow for the removal of elements from the filter by means of gradual forgetting. The effectiveness of our approach in achieving both high delivery rates and low overhead is demonstrated by means of simulations and experiments.**

I. Introduction

The flow of data in Wireless Sensor Networks (*WSN*s) is commonly directed from a large number of sensing devices towards a data-collecting sink node. In order to deliver the collected readings to the sink, convergecast routing protocols like MintRoute [1] or the collection tree protocol (*CTP*) [2] are commonly being used. While these collection protocols cater for an efficient collection of sensor data, directed communication from the sink to individual sensor nodes is beyond their scope. A back channel is however essential in many scenarios, e.g., in order to alter the reporting intervals of specific sensors or change their configuration [3].

Existing approaches to realize back channels mostly either rely on dissemination protocols or use point-to-point routing schemes. However, neither approach fulfills the requirements of the application scenario at hand. Dissemination protocols like Trickle [4] enable the sink node to propagate information to all devices in the WSN, but do not support addressing individual nodes. In contrast, point-to-point routing schemes like DSDV [5] enable message passing between any two nodes in the network, but come at the cost of a large overhead to build and maintain the routing tables.

We leverage the tree structures established by collection protocols for enabling lightweight point-to-point communication. On each sensor node, a Bloom filter (*BF*) [6] is used to maintain the identities of all its descendants at a constant memory overhead. This routing information is forwarded towards the sink (i.e., *up-tree*) along the existing collection tree. Once information about the BFs of all nodes has reached the sink, each node in the network possesses knowledge about the addresses of all of its direct and indirect child nodes. This knowledge is then used by the nodes to verify their list of reachable nodes before forwarding a message addressed to a specific node with greater depth in the tree (i.e., *down-tree*).

Since Bloom filters are probabilistic space-efficient data structures, they can be used conveniently even on memory-constrained sensor nodes. Thanks to the combination of collection trees and Bloom filters, our Counting Bloom Filter Routing (*CBFR*) scheme caters for efficient point-to-point communication at: (1) lower messaging overhead than approaches based on flooding; (2) lower memory demand than traditional point-to-point routing schemes; and (3) lower delay than typical dissemination protocols.

The main contributions of this paper are the following:

- We discuss the operation of our routing scheme for the case of static network topologies.
- We introduce the gradual forgetting component, based on counting Bloom filters, in order to account for node mobility.
- We evaluate CBFR in both simulations and experiments and show its performance in terms of packet delivery ratio and overhead.

After revisiting the targeted scenario and our basic assumptions in Sec. II, we present our CBFR approach in detail in Sec. III. In Sec. IV, we describe the design of the gradual forgetting component, while the evaluation of CBFR through simulations and real-world experiments is reported in Sec. V. After discussing related work in Sec. VI, we summarize the key contributions of this work and highlight possible directions for further research in Sec. VII.

II. Scenario and Assumptions

Before elaborating on the details of CBFR, we briefly introduce the assumptions made for the underlying network.

A. Data Collection Protocol

The majority of WSNs is based on the collection of data from sensor nodes deployed within a region of interest. Common application domains include environmental or habitat monitoring, industrial process surveillance, or structural health monitoring (e.g., [7], [8]). In such data collection networks, sensor readings captured by the deployed nodes are generally forwarded to one or multiple sink nodes in a multipoint-to-point manner. To this end, collection protocols are predominantly employed, which establish a spanning tree rooted at the sink. In order to route packets towards the sink, each node retains local knowledge about the next routing hop, i.e., knowledge about the address of its *parent* in the tree. Due to the acyclic property of spanning trees, nodes in a stable tree can be assumed to have at most one parent.

The existence of a tree-based routing structure is an essential prerequisite for the operation of our CBFR scheme. For the sake of simplicity, we will assume this underlying protocol to be CTP [2], a well-known best-effort collection protocol for WSNs. However, CBFR can operate on top of any arbitrary tree-based collection protocol.

B. Node Mobility

Collection protocols in general (and CTP in particular) are tailored to purely static sensor networks and thus only have indirect ways to cope with node mobility. In order to reduce the messaging overhead for the tree construction and maintenance, CTP progressively increases the time interval at which beacons are transmitted for neighbor discovery. Only when inconsistencies in the routing tree (e.g., loops) are detected, CTP resets its beaconing interval to a default minimal value of 128ms [2]. In other cases, however, the maximal length of the beaconing interval increases up to 512,000ms, i.e., more than 8 minutes, by default. As a result, topology changes incurred by the motion of mobile nodes may remain undiscovered or even lead to routing loops. To accommodate a more reactive behavior in the presence of mobile nodes, we thus reduce the upper bound of CTP's exponentially increasing beaconing interval to 10,240ms. As a result, the faster integration of new devices and mobile nodes, such as sensors attached to autonomous robots, into the collection tree is enabled. If mobile nodes are moving at a slower speed or their presence needs to be detected less accurately, the upper limit of the adaptive beaconing should however be adapted to the scenario requirements at hand.

For the remainder of this paper, we assume a WSN in which the sink node is static, whereas intermediate nodes are possibly mobile. All mobility-related parameters used in our simulations are discussed in detail in Sec. V along with the experimental setup.

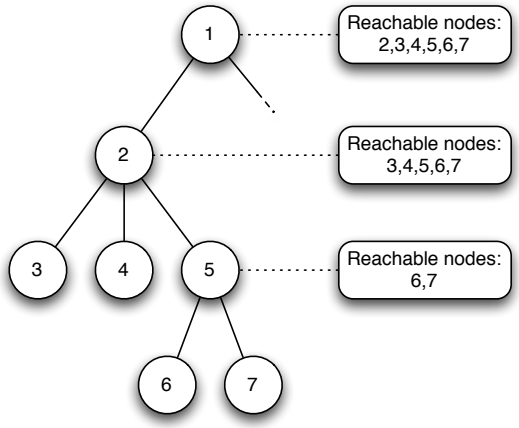

Figure 1. Local knowledge of addresses of direct and indirect child nodes

III. Bloom Filter-based Routing for WSNs

Our CBFR scheme provides point-to-point routing functionalities between nodes in a WSN. Similar to other proactive distributed routing schemes, each node maintains a routing table that stores the addresses of its directly and indirectly reachable nodes[1]. In contrast to fully distributed schemes, however, CBFR exploits the presence of an existing collection tree to cater for a directed routing of messages towards their destination. Fig. 1 exemplary shows an excerpt of a tree-based routing topology with corresponding annotations about the reachable descendants of three nodes.

A major difference to traditional routing schemes is the fact that CBFR does not store the set of direct and indirect descendants as a dynamically expanding list. Especially in operating systems without support for dynamic memory allocation, such as TinyOS [9], worst-case assumptions need to be made when allocating memory to the routing tables. In an exemplary tree topology comprised of 511 nodes, each of the one-hop neighbors of the sink node would thus have to allocate memory for at least 509 addresses in their routing table. Under the assumption that a 16-bit-wide address field is used, the memory demand for the routing table exceeds 1,000 bytes. However, in the more likely case of a balanced binary tree, only 254 entries, i.e., less than half of the worst-case requirement, would be needed. Instead of relying on a priori estimations of the expected number of entries in the routing table, we utilize a BF in order to store only the actual set of reachable children. In contrast to allocating memory according to the worst-case assumptions, the space-efficient probabilistic data structure only incurs a constant memory demand of the routing table, and thus strongly contributes to its applicability on resource-constrained embedded systems.

[1] An extension of our concept would be to store the routable destination addresses for each direct child node. This would however require an individual list for each direct child and thus infer an additional demand for memory.

A. Bloom Filters Revisited

The operation of a BF comprises two methods: (1) *element insertion* to store new elements into the filter; and (2) *membership query* to verify whether an element is present in the filter. It needs to be remarked at this point that BFs do not provide support for the deletion of entries by design.

1) Element Insertion: A BF is typically implemented as an array of m bits, which are initially all set to '0'. In order to insert a data element into the bit array, h hash functions are calculated over the element to insert. In our case, such an element is a unique address of a sensor node. Each of the hash functions returns the index of an element of the bit array, which is accordingly set to '1'. The insertion of an element thus corresponds to setting the h bits signaled by the hash functions to '1'. In case a bit has already been set to '1' prior to the insertion operation, its state is not altered. Fig. 2 displays the insertion of two elements $input_1$ and $input_2$ into a BF with m=16 bits and h=3. The figure also shows a collision at the sixth position of the BF, where the hash functions map to the same entry for both input sequences. Such collisions cause the occurrence of false positives, as detailed below.

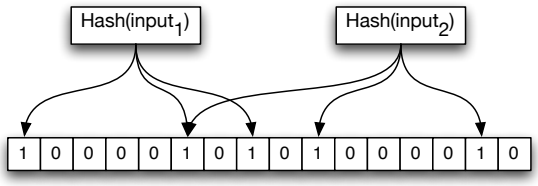

Figure 2. Insertion of two elements into a Bloom filter (m=16, h=3)

2) Membership Query: In order to check whether an element is contained in a BF, the hash functions of the input element are calculated first. If any of the bits at the h positions in the BF is set to '0', the input element is certainly not contained in the BF. When all bits at the h positions are set to '1', the input symbol is a member of the BF with a certain probability. Depending on the number of hash functions and the length of the bit array, a membership query might however return a false positive statement.

The likeliness of false positives depends on the values of m and h as well as on the number t of elements already inserted into the BF. Under the assumption that the outputs of the hash functions are independent and uniformly distributed, the probability of a membership query to return a false positive, indicated as P_{FP}, is calculated as [6]:

$$P_{FP} = (1 - (1 - \frac{1}{m})^{ht})^h \approx (1 - e^{-\frac{ht}{m}})^h$$

User-specified estimations for the desired false positive probability and for the expected number elements (t) allow to specify the parameters of a BF as $h = \lceil -log_2 P_{FP} \rceil$ and $m = \lceil \frac{ht}{ln(2)} \rceil$.

B. Bloom Filter Dimensioning and Population

Let us refer back to the example introduced in the beginning of this section. Using the above equations, a BF capable of storing an average number of 254 elements at a false positive probability below 0.07 would require the use of h=4 hash functions and a BF size of m=1,466 bits (i.e., about 184 bytes). This represents an 82% gain in terms of space efficiency with respect to the aforementioned example of a routing table with 509 entries. However, in order to avoid the propagation of incomplete routing information due to packet losses, we have imposed a constraint on the BF size, namely that it should be sufficiently small to completely fit into an IEEE 802.15.4 frame [10]. The only way to reduce the size of the filter for a given number of expected entries is by increasing the allowance for false positives.

Each node stores a BF, which can be queried for the presence of node addresses in the down-tree neighborhood. The size of the BF as well as the number and definition of the hash functions are identical across all nodes in the network. The BF is initially only populated with the node's own address, resulting from the application of the h hash functions to the node's address and setting the corresponding entries in the local BF. Upon each reception of BF update from a child node (described below), the local BF is updated in order to reflect the presence of all direct and indirect child nodes.

C. Bloom Filter Forwarding

After its deployment, the WSN is organized as a tree by CTP. Instead of adapting the internals of the CTP implementation to carry specific packet payloads for the required routing messages, our design relies on the interfaces publicly exposed by CTP as visualized in Fig. 3. Concretely, the following two interfaces to the CTP engine are used by our BF-based routing:

1) The up-tree collection interface to forward messages to the sink node.
2) The interface to intercept messages received by a child node on their way towards the sink.

Besides the interface provided by the collection protocol, our scheme employs a direct connection to the radio stack in order to send and receive broadcast messages. The regular operation of our routing scheme commences by the aforementioned step of initializing the local BF, followed by the publication of the routing table to the next hop in the direction of the tree root. In other words, the up-tree

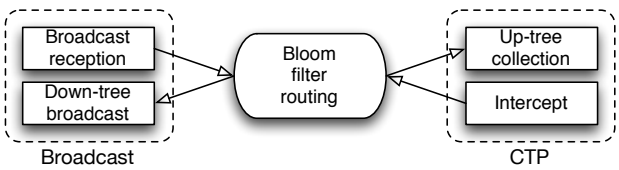

Figure 3. Interfaces of our routing scheme to the wireless channel

collection interface is used to transfer the BF towards the sink and thus update the BFs of all intermediate nodes. In its practical realization, the updates of the BF can also be piggybacked on CTP beaconing messages in order to reduce the additionally introduced communication overhead.

In contrast to the regular operation of CTP, which transparently forwards packets towards the sink, we have attached CBFR to CTP's interception interface. If an intercepted packet contains regular sensor data, it is passed on to the regular up-tree collection mechanism. However, if a representation of the BF is present in the intercepted message, the receiving node removes the message from the CTP retransmission queue and forwards it to the BF routing mechanism, which calculates the bitwise logical disjunction of both bit vectors and stores the result into its own BF. As a result of the bitwise logical OR operation, the local BF also contains a combined list of directly and indirectly reachable nodes.

$$BF_{local} := BF_{local} \vee BF_{recv}$$

After updating the local Bloom filter's bit array according to the intercepted BF update message, the node again uses the up-tree collection interface to forward its own updated BF towards the root. In the regular operation mode, the BF is transmitted at a defined update period ϕ only; this interval is slightly randomized [4] to avoid continuous collisions of BF update messages. In order to quickly integrate new nodes into the BFs of all up-tree nodes, we have supplemented the periodic transmission of updates by a *fast update* mechanism. Whenever the BF of a down-tree node is received, its content is matched against the local BF, and the local BF content is immediately forwarded up-tree in case a difference is detected. The fast update approach has been chosen to improve the responsiveness of the approach, while ensuring that immediate updates are only transmitted when changes to the topology occur. The up-tree communication of BFs is summarized in Algorithm 1.

Algorithm 1 Local forwarding decision for up-tree traffic

BF_{local}: Locally stored Bloom filter
$BF_{received}$: Bloom filter in intercepted up-tree packet
ϕ: Periodic up-tree update interval

if (packet intercepted from up-tree forwarding) **then**
  if $((BF_{recv} \vee BF_{local}) \neq BF_{local})$ **then**
    Immediately forward the packet up-tree
    $BF_{local} := BF_{recv} \vee BF_{local}$
  else
    Queue forwarding until expiry of current interval ϕ
  end if
end if

D. Hash Function Selection

In order to ensure the efficient operation of the BF, a fast hash function with low energy demand is required. The task of the hash function is to map an input value, i.e., the address of a node in the network, to a set of values that point to offsets in the BF, which are then set to '1' in the bit vector. Collisions, i.e., identical hash values for different input sequences, are almost impossible to avoid because the universe of keys is usually larger than the number of possible output values. Ideally, however, the used hash function should have a low number of collisions whilst being applicable on nodes with limited resources and tight energy budgets.

Table I
EXECUTION TIME AND RESOURCE DEMAND OF THE HASH FUNCTIONS

Hash function	Execution (μs)	ROM (bytes)	RAM (bytes)
Thomas Wang	27	178	2
hash7shift	28	188	2
Bob Jenkins	28	196	2
hash32shift	59	198	2
hash32shiftmult	104	162	2
MurmurHash	204	522	32
CRC16	438	410	1060

In order to select a suitable hash function, we have implemented several hash functions in TinyOS and assessed their performance. Each hash function calculates the hash values for 16 bit wide input values which have been generated using a random number generator. The size of 16 bits has been particularly selected, because it represents the size of node addresses when the 16 bit addressing mode of the IEEE 802.15.4 standard is selected. The generated random node addresses have been hardcoded into our TinyOS implementation to minimize possible delays for data access during our experimental evaluation. Table I shows the average execution time for processing each input values and the memory consumption of each function for the same input data set. While the required program memory does not vary much between the implementations, the differences in execution times are remarkable. The hash functions proposed by Thomas Wang [11] and Bob Jenkins [12] are clearly faster than a standard 16 bit CRC. In our implementation, we have thus selected Thomas Wang's hash function.

E. Packet Routing

The CBFR scheme caters for the transmission of messages in a point-to-point manner. Each routing process begins with a transfer of the message to the root node of the tree via the up-tree collection interface. Forwarding to the root is essential due to the inherent possibility of false positives, i.e., the local knowledge that the destination node might be among the direct and indirect children is insufficient to exclude the possibility that the destination is located in another part of the network. The transfer towards the sink node is performed by encapsulating the source and destination address as well as the payload within a CTP message.

Once the packet has been received at the sink, its further processing is shown in Algorithm 2. The sink node starts

by applying the h hash functions to the destination address. Subsequently, it checks if the h offsets are all set to '1' in its local BF. If this step fails, the message is discarded, because the destination node cannot be part of the network due to the fact that false negatives can never occur in the BF. In case the check succeeds and all h bits are set in the BF, the message is broadcast to the one-hop neighborhood of the sink, where the process is restarted. To minimize the traffic, nodes only accept broadcast messages from their parent and ignore the messages broadcast by other nodes. It should be noted at this point that the down-tree forwarding of messages is realized by broadcasting and does not rely on the address-free CTP. As a result, the recipient of a broadcast message can easily identify whether the originating node is identical to its CTP parent or not.

Algorithm 2 Local forwarding decision for down-tree traffic

$H(x)$: Hash values for input x
A_{dest}: Destination address of the packet
A_{local}: Local node address
A_{parent}: Local node's parent address

if (packet source != A_{parent}) **then**
  Discard the packet
else if (A_{dest} == A_{local}) **then**
  Forward packet to application
else
  Compute $H(A_{dest})$ for each of the h hash functions
  if (all fields $H(A_{dest})$ are set in BF_{local}) **then**
    Re-broadcast the packet down-tree
  else
    Discard the packet
  end if
end if

IV. Gradual Forgetting for Mobility Support

Up to this point, we have shown the algorithm's behavior for a static deployment of nodes. However, as outlined in Sec. II-B, nodes in the envisioned application area may expose mobile behavior or lose connectivity to the network due to other reasons. Despite the fact that traditional BFs do not allow for the deletion of elements, nodes disconnected from the network would be unable to transmit unregistration messages in any case. A corresponding functionality to permit updating the tree to changed topologies without specific unregistration announcements has thus been regarded when adapting our routing algorithm for the presence of node mobility. In order to achieve this, we have used counting Bloom filters (*CBF*s) [13] instead of traditional BFs. In contrast to the traditional BFs, CBFs allocate an integer value to each position in the bit vector instead of using binary fields. To maintain the applicability of CBFs on resource-constrained WSN hardware, we have limited the counters to a size of 4 bits each.

Analog to the use of traditional BFs, only bit vectors containing routing information are being forwarded to a node's parent. On each reception of a routing information message, the local CBF is incremented by one at each position which is set in the received BF. As soon as the full capacity β of the position is reached, no further incrementation operation is performed. In order to manage node mobility and outages, the CBF is periodically decremented, i.e., each of its entries is reduced by one after a given timeout ρ. Each position in the CBF can thus be modeled as a leaky bucket of equal capacity β and equal leak rate ρ. In case a node fails, no update is received and it is accordingly removed from the filter after the deletion timeout period, which is at most equal to $\beta \cdot \rho$. For stability reasons, the timeout ρ needs to be larger than the typical update interval ϕ, described in Sec. III-C.

V. Evaluation

In the previous sections, we have presented the concept of CBFR. We now analyze CBFR's behavior with respect to packet delivery rates and delays considering different network settings both with and without the presence of mobile nodes. To this end, we have implemented CBFR in TinyOS and simulated it in the the COOJA simulator [14] with the Mobility extension. The reference platform for our simulations is the TelosB [15]. We have designed the BF in our CBFR scheme to use two hash functions (h=2) and optimized the BF for the insertion of t=25 elements. Accordingly, we have chosen m=64 bits, i.e., 8 bytes, in order to achieve a false positive probability of less than 0.25 while catering for a low memory demand. An overview of the notations used in the following simulations and experiments can be found in Table II.

A. Static Network Topology

The first simulation has been performed for the static network topology presented in Fig. 4. The network has 10 sensor nodes with the root having identifier 1. The root periodically creates data messages and sends them to node 4, which is 3 hops away from the root. Following the established tree topology, the messages sent from node 1 are relayed by nodes 7 and 10 before reaching node 4. In the simulations, the parameters have been set to $h = 2$, $m = 64$, $\rho = 40s$, $\beta = 15$, and $\phi = 25s$. These numbers represent a CBF with 64 entries of 4 bits size each. The filter is forwarded to the node's parent at least every 25

Table II
Summary of used notations

Symbol	Interpretation
h	Number of hash functions
m	Total number of entries in the BF
ρ	Time interval after which the CBF is decremented by one
β	Maximum value of an entry in the CBF
ϕ	Time interval for the up-tree transmission of CBF updates

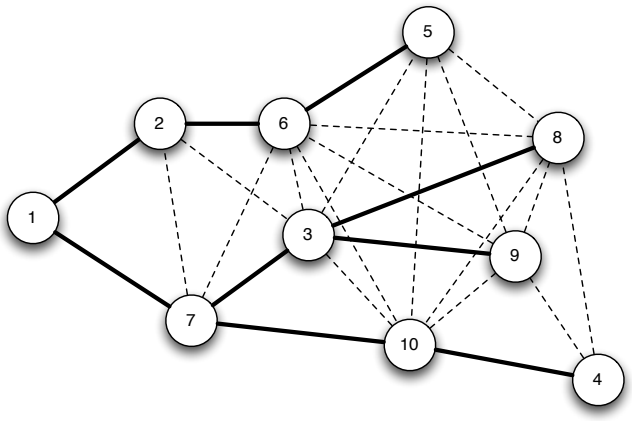

Figure 4. Static network topology (CTP tree indicated in the form of bold and connectivity as dashed lines)

seconds, whereas its counters are decremented every 40 seconds. The particular numbers have been adapted to an application scenario in which no mobile nodes are present. As a result, comparably long timeouts have been selected, e.g., the node deletion timeout is equal to ten minutes.

In our simulation, the root node transmits 500 data messages to each node in the network, using intervals of 100ms, 250ms, 500ms, and 1 second between messages. After the simulation had finished, we have analyzed the packet delivery ratio of all 4,500 packets, and determined a probability of successful packet delivery of 99.6%, regardless of the transmission interval. These results confirm that the algorithm performs very well in a simple scenario with a static network topology. In an additional experiment, we have analyzed the time required to detect the presence of a new node in the network. To this end, we first simulated the network without the presence of node 10, and only added the device after the collection tree had been established. Across ten repetitions of the experiment, the average delay between the physical addition of the node to the network and its presence in the root node's CBF has been shown to be 22 seconds. The fact that this period is smaller than ϕ results from the use of our fast update mode (cf. Sec. III-C).

We have furthermore simulated two static tree topologies comprised of 50 nodes each. Again, one of the nodes was acting as the sink node, while the remaining 49 nodes were deployed as follows. In scenario *S1*, the nodes were densely deployed around the sink at distances between one and three hops with an average distance of 1.5 hops. In contrast, scenario *S2* was comprised of a deeper tree, with nodes at distances between one to eight hops away from the sink node. In S2, the average distance to the sink was 4.0 hops. The resulting packet success probabilities for each number of hops to the sink are tabulated in Table III. From the table, lower success probabilities for the nodes farther away from the sink can be determined. This observation

Table III
PACKET DELIVERY RATES, 50 NODES

	Number of hops from the sink							
	1	2	3	4	5	6	7	8
S1	95.2%	75.7%	54%	–	–	–	–	–
S2	99.4%	90.5%	74.3%	94.7%	82.0%	90.0%	82.7%	78.5%

can be attributed to the higher node densities around the destination nodes (S1) or along the path (S2), respectively, which led to a slightly increased number of packet collisions. Still, CBFR has quickly established routes to all nodes; in less than 30 seconds after booting the nodes, a complete bidirectional routing tree is available.

B. Mobile Destination Node

In order to assess the impact of node mobility on the performance of CBFR, we verify whether our routing scheme is able to deliver data messages to a moving node. The topology used in this scenario is shown in Fig. 5, with the connectivity between nodes indicated by dashed lines and the CTP-tree indicated by bold lines. As in the previous small-scale experiment, the network is comprised of 10 sensors with node 1 being the root. All the nodes in the network are static, except for nodes 2 and 3, which move according to a Random Waypoint Model with a speed between 0.5 m/s and 1.5 m/s, simulating the regular walking speed of a human. The positions of nodes 2 and 3 at different points in time are also visualized in the figure.

The root node periodically computes data messages and sends them to node 3, which in turn logs a debugging statement upon receiving a message. The parameters have been again set to $h = 2$, $m = 64$, $\rho = 40s$, $\beta = 15$, and $\phi = 25s$ as in the previous evaluation and the root node 1 has again been configured to transmit 500 messages to each

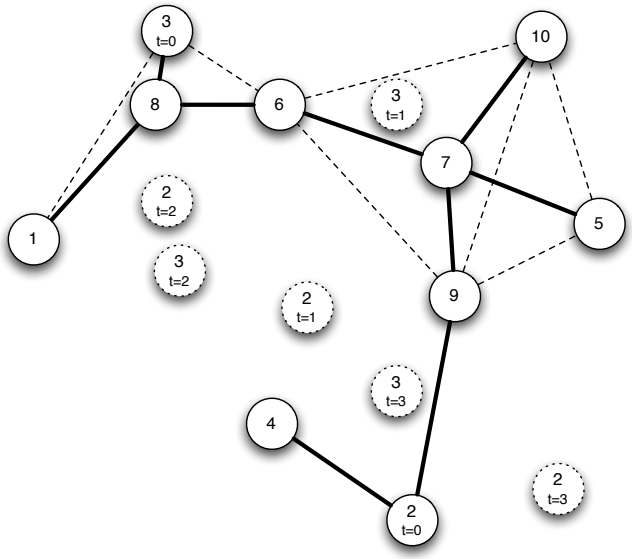

Figure 5. Network topology with mobile destination (initial CTP tree indicated as bold and initial connectivity as dashed lines, positions of mobile nodes as dashed circles annotated with respective time t)

Table IV
COMPARISON OF CBFR AND FLOODING IN TERMS OF MESSAGES

Parameter measured	Flooding	CBFR
Number of messages sent	9	9
Number of messages overheard	330	68
Number of messages forwarded	65	11
Number of Bloom filter updates	–	23

destination node. During the simulation, between 92.4% and 100% (on average 95.7%) of all messages transmitted to nodes 5 through 10 were successfully received by their destinations. In contrast, due to the frequent disconnection from their respective parents, only 56.2% of the packets could be successfully delivered to the two mobile nodes 2 and 3. At only 26.8%, the packet reception rate at node 4 was lowest in our simulations, which originates from the fact that it was generally disconnected from the network, but only integrated if one of the mobile nodes (i.e., nodes 2 or 3) could act as its parents and thus connect it to the tree.

C. Traffic Reductions by Directional Routing

We have compared our directional approach to a flooding-based dissemination of data. The static network topology visualized in Fig. 4 has been used and message counters have been added to the original routing scheme implementation. In the simulation, the root sends a data message to each node in the network at a rate of one packet per second. The number of intercepted and forwarded BF update messages has been counted for each node. Furthermore, the worst-case number of BF update messages that could have been transmitted during the simulation has been included in order to allow for a fair comparison. Additionally, a second implementation has been provided in TinyOS to implement a flooding-based dissemination of messages. In order to minimize the amount of traffic generated by flooding, a sequence number mechanism has been used. Based on this mechanism, a node forwards a data message only in case no other message with the same sequence number has been received before.

The results for the number of packets transferred are presented in Table IV. In both cases, nine packets have been emitted from the sink, one addressed to each node in the network. In the flooding-based solution, we regard repeatedly received messages that have already been forwarded by a node as overheard. Similarly, in CBFR, overheard messages are defined as messages that are received by a node, but not forwarded further, because the contained destination address is not present in the CBF. Finally, the number of forwarded messages indicates the total number of retransmissions of the nine messages throughout the entire network. As it may be seen, the CBFR scheme reduces the number of overheard messages by 79.3%, as well as eliminating the need for 83% of the messages forwarded when flooding is employed. Even when the BF update messages are regarded in the analysis, the total number of messages is reduced by more than 72% when CBFR is being used.

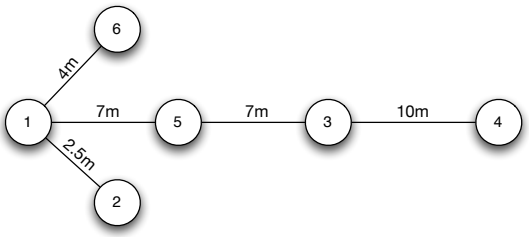

Figure 6. Visualization of the used real-world network topology

Table V
RESOURCE OVERHEAD OF OUR CBFR SCHEME

CBF size	64 entries	256 entries
Program memory	2,162 bytes	2,162 bytes
RAM	2,581 bytes	2,869 bytes

D. Real-world Evaluation

After analyzing the performance of CBFR by means of simulation, its applicability has been tested on real TelosB hardware. We have therefore compiled CBFR for the TinyOS operating system and integrated it with the existing implementation of CTP. The resource demands of CBFs of two different sizes are presented in Table V. The real-world tests have been performed in an indoor environment. Three nodes were positioned along a hallway and three other nodes in a room adjacent to it. The resulting topology is visualized in Fig. 6; all edges are annotated with the distance between nodes in meters.

The root node has been assigned the identifier 1 and been configured to send 50 data messages to each node in the network. On the nodes, we have implemented a function to respond to each incoming message with a specifically defined acknowledgment message, which is returned to the root by means of CTP's collection feature. The root node measures the delays between transmission of a packet to a destination in the network and reception of the corresponding acknowledgment. The purpose of these tests is to determine the time elapsed between the moment a data message was sent by the root and the time at which the acknowledgement was received by the root, i.e., the round trip time (*RTT*). The average RTTs as well as extremal

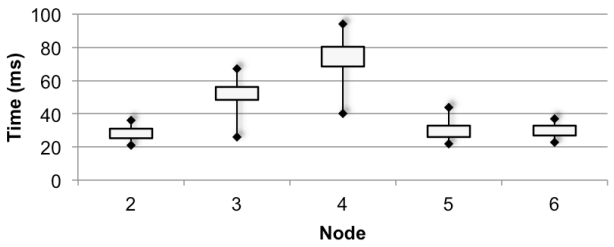

Figure 7. Visualization of the minimum, maximum, and average RTTs in real-world tests

values have been determined from the 50 measurements conducted per node and are presented in the box plot shown Fig. 7. As expected, nodes located further away from the sink (nodes 3 and 4) have a longer RTT, whereas node 5, which is directly connected to the sink, has the fastest response time. In comparison to Trickle [4], which operates on time scales of seconds to minutes, the observed RTTs of less than 100ms clearly show the faster delivery of messages when CBFR is being used.

VI. Related Work

Collection protocols for WSNs typically create tree-based routing structures to forward data packets from any node within a network towards a sink node. For instance, the Collection Tree Protocol [2] represents an efficient and reliable implementation of a collection service for WSNs. However, it does not provide any mechanisms to enable explicit communication from the sink to the nodes. Other collection protocols like Dozer [16] or DISSense [17] do provide such down-tree communication channels, but do not permit the addressing of individual nodes. Dozer constructs an explicit slot-based mechanisms to propagate packets from parent nodes to their children. In contrast, DISSense piggybacks information on the control packets used for tree construction and maintenance, thereby creating an implicit backward communication channel. In both cases, however, information is disseminated to all reachable nodes and communication to a specific node within the network is not supported. Instead, our approach exploits the tree-based structure built by collection protocols to offer a directed propagation of information between intermediate nodes.

Dissemination protocols, most notably Trickle [4] and DIP [18], provide reliable mechanisms to make sink nodes share a value with all other nodes in the network. Such protocols can thus be used to, e.g., propagate new parameter values. However, addressing one particular node is generally not supported by dissemination protocols either. Point-to-point routing approaches have been investigated extensively in research on mobile ad hoc networks (*MANET*s). The inherent differences between MANETs and WSNs however render most MANET routing protocols inapplicable in the application domain of WSNs [19].

Specifically tailored to WSNs, the ROLL initiative caters for routing over low-power and lossy networks by moving the burden of centrally collecting routing information to nodes with additional storage capacity, which act as intermediate routers [20]. Our approach overcomes the limitation of being confined to a small number of routers by leveraging the favorable properties of Bloom filters. Furthermore, our CBFR scheme guarantees prompt reactiveness to routing requests by updating the Bloom filter regularly. In reactive routing protocols like DSR [21], routes are computed on demand only when communication needs to take place. Although reactive solutions may be effective when point-to-point routing requests are infrequent, they also induce significant delays in packet delivery and do not take advantage of an existing collection tree. Publications on collection and dissemination in the presence of node mobility include [22], which introduces a protocol for per-packet based routing towards highly mobile sinks. Similarly, the HYPER protocol [23] can quickly repair collection tree topologies when nodes move within the network. By using Bloom filters, however, our approach introduces an element of novelty with respect to these and other related publications.

Last but not least, the adoption of Bloom filters has been investigated for several purposes within and beyond the realm of address-centric routing in WSNs. For instance, scope decay Bloom filters have been proposed to steer the propagation direction of rumor routing and thus increase its success probability [24]. In DIP [18], the use of Bloom filters makes search for stale items in WSNs quicker and more efficient. Bloom filters have been used in the context of point-to-point routing in MANETs [25], as well as to store semantic data type information in WSNs [26]. Other uses include Web caching [13] or peer-to-peer networks [27]. To the best of our knowledge, however, we are the first to propose the use of Bloom filters to provide for efficient address-based routing in conjunction with tree-based collection protocols in WSNs.

VII. Conclusions and Future Work

This paper introduced a generic approach to extend tree-based collection protocols with point-to-point routing capabilities. Instead of maintaining complete and dynamically growing routing tables at each node, our CBFR scheme uses a Bloom filter to efficiently store information about the reachability of a node's direct and indirect child nodes. The constant memory overhead of such filters makes our approach readily usable on resource-constrained sensor nodes, which typically do not support dynamic memory allocation. We have implemented CBFR in TinyOS and tested its performance in different settings. Our experimental results show that, with respect to flooding-based routing, our approach significantly reduces the communication overhead needed to deliver a message to a specific node in the network.

The CBFR scheme leverages the CTP collection protocol both to forward the Bloom filter up-tree as well as to support an efficient down-tree broadcast to transmit packets to their destination. As a next step, we plan to embed our approach in CTP transparently, so that it can be used as a tunable CTP option. For instance, propagation of information could be performed leveraging CTP's own routing beacons. Future work also includes testing of our approach on WSN testbeds in order to assess its scalability in large networks.

Acknowledgment

The authors would like to thank Delphine Christin for her valuable comments. This research has been supported

by the German Federal Ministry of Education and Research (BMBF) and by the LOEWE research initiative of the State of Hesse, Germany, within the Priority Program Cocoon.

REFERENCES

[1] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, 2003.

[2] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 2009.

[3] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, "Constraint-Guided Dynamic Reconfiguration in Sensor Networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, 2004.

[4] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, 2004.

[5] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance Vector Routing Protocol (DSDV) for Mobile Computers," in *Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, 1994.

[6] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[7] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[8] J. Paek, K. Chintalapudi, J. Caffrey, R. Govindan, and S. Masri, "A Wireless Sensor Network for Structural Health Monitoring: Performance and Experience," in *Proceedings of the 2nd Workshop on Embedded Networked Sensors*, 2005.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Network Sensors," in *Proceedings of the 10th Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[10] IEEE Std, "802.15.4 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-rate Wireless Personal Area Networks," 2006.

[11] T. Wang, "Integer Hash Function," Online: http://www.concentric.net/~ttwang/tech/inthash.htm, 2007.

[12] B. Jenkins, "4-byte Integer Hashing," Online: http://burtleburtle.net/bob/hash/integer.html, 2006.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, 2000.

[14] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, "COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2009.

[15] *TelosB Datasheet*, MEMSIC Inc., 2011, http://memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152:telosb.

[16] N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: Ultra-Low Power Data Gathering in Sensor Networks," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, 2007.

[17] U. Colesanti, S. Santini, and A. Vitaletti, "DISSense: An Adaptive Ultralow-power Communication Protocol for Wireless Sensor Networks," in *Proceedings of the 7th International Conference on Distributed Computing in Sensor Systems*, 2011.

[18] K. Lin and P. Levis, "Data Discovery and Dissemination with DIP," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, 2008.

[19] T. Watteyne, A. Molinaro, M. G. Richichi, and M. Dohler, "From MANET to IETF ROLL Standardization: A Paradigm Shift in WSN Routing Protocols," *IEEE Communication Surveys and Tutorials*, vol. 13, no. 4, 2011.

[20] T. Winter (Ed.), P. Thubert (Ed.), and the RPL Author Team, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," Online: http://tools.ietf.org/html/draft-dt-roll-rpl, 2011.

[21] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," RFC 4728, 2007.

[22] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing Without Routes: The Backpressure Collection Protocol," in *Proceedings of the 9th International Conference on Information Processing in Sensor Networks*, 2010.

[23] T. Schoellhammer, B. Greenstein, and D. Estrin, "Hyper: A Routing Protocol to Support Mobile Users of Sensor Networks," UCLA, CENS Technical Report, 2006.

[24] X. Li, J. Wu, and J. Xu, "Hint-based Routing in WSNs Using Scope Decay Bloom Filters," in *Proceedings of the International Workshop on Networking, Architecture and Storages*, 2006.

[25] T. Osano, Y. Uchida, and N. Ishikawa, "Routing Protocol using Bloom Filters for Mobile Ad Hoc Networks," in *Proceedings of the 4th International Conference on Mobile Ad-hoc and Sensor Systems*, 2008.

[26] P. Hebden and A. R. Pearce, "Data-centric Routing using Bloom Filters in Wireless Sensor Networks," in *Proceedings of the 4th International Conference on Intelligent Sensing and Information Processing*, 2006.

[27] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, 2005.