Data-Centric Cooperative Storage in Wireless Sensor Network

Abdalkarim Awad, Reinhard Germany and Falko Dressler
Computer Networks and Communication Systems
University of Erlangen, Germany
{abdalkarim.awad,german,dressler}@informatik.uni-erlangen.de

***Abstract*—Although commonly used sensor nodes are resource-limited devices, enabling cooperation among these devices can help building very powerful systems. Besides other characteristics, storage capabilities of individual sensor nodes have to be considered as this might be very small compared to the collected sensor data. The temporal availability of sink nodes or the importance of collected data are not equal for all nodes and, therefore, demand storage of the generated data locally for later retrieval. Furthermore, data items (detected events) may be collected at different rates, which can lead to hotspot-like storage requirements. In this paper, we introduce a data-centric cooperative storage mechanism for wireless senor networks. Our approach is based on Virtual Cord Protocol (VCP), a virtual relative position based efficient routing protocol that also provides means for data management, e.g. insert, get, delete, and replicate, as known from typical Distributed Hash Table (DHT) services. Data items are distributed deterministically over several nodes in the same vicinity. Thus, storing and retrieving data items typically require communication with local nodes. To maintain information about stored information, we use a bloom filter to track the nodes that are storing particular items.**

I. Introduction

Earlier generations of sensor networks, which have been used in a regular topographic mesh, like the radar network used in air-traffic control and nationwide weather stations, use special computers and communication protocols making them very expensive. For some application scenarios, a network of sensors and actuators can be built using the existing wired technologies. For many other application types, however, wiring is not practical, expensive, and can make it difficult to install the sensors close to the phenomenon under observation [1]. Therefore, Wireless Sensor Networks (WSNs) present a cost-effective, practical, and capable solution to support many application scenarios. In spite of the fact that the capabilities of sensor nodes are very limited, WSN application domains are diverse and they can operate a variety of data types, including simple payload such as temperature, light intensity, and humidity; or more complex data types such as sound, images, or even more complicated data like video.

Distributed Hash Tables (DHTs) [2] ensure $O(1)$ complexity to insert and lookup data items. Moreover, they work in a distributed and self-organized manner. These characteristics make them attractive for use in WSNs. The main idea is simple: Data items are associated with numbers and each node in the network is responsible for a range of these numbers. Therefore, it is easy to identify the node at which a data item is stored. Usually, DHTs are built on the application layer and rely on an underlying routing protocol that provides connectivity between the nodes. Systems like Chord [3], Pastry [4], and CAN [5] have been implemented to work on the Internet for scalable file sharing applications. The nodes communicate taking advantage of the already existing routing protocols in the Internet. Several DHTs offer cooperative-storage (and data replication) mechanisms, for example in Chord, the successor list can be used to store data on behalf of other nodes. The main drawback of implementing DHTs as an overlay in WSNs is the separation between the logical (overlay) and physical (underlay) networks. This separation poses extra, possibly avoidable overhead and complexity to the system as each layer has its own routing schemes.

Virtual Cord Protocol (VCP) [6]–[8] is a DHT-like protocol in which all data items are associated with numbers in a pre-determined range $[S, E]$, i.e. a one-dimensional cord. All the available nodes capture this range. Thus, each node in the network is responsible for a portion of the entire space defined by its relative position to physical neighbors. This way, it is possible to store data on the nodes by mapping data items deterministically in space using a hash function. The corresponding key-value pair is then stored at the node whose position is closest to the key. Routing of packets is performed based only on the position of the physical neighbors. To retrieve data items, nodes have to apply the same hash function to find the key value. They then can route the request to the node whose position is closest to the key.

To this end the authors have investigated the performance of VCP assuming each node can store all data items it is responsible for. In this paper, we extend VCP to support cooperative storage among nodes. The rest of the paper is organized as follows. In Section II, we discuss relevant related work. In Section III, we provide an overview of VCP and, in Section IV, we present our cooperative storage scheme. Afterwards, we evaluate the performance of the proposed scheme in Section V. Finally, Section VI concludes the paper.

II. Related work

Virtual Ring Routing (VRR) [9] is a routing protocol inspired by overlay DHTs. Besides routing, it provides traditional DHTs functionality. VRR uses a unique key to identify nodes. This key is a location-independent integer. Similar to Chord [3], VRR organizes its nodes in a virtual ring in the

order of increasing identifiers. VRR reduces the routing tables from two in overlay implementation to one by combining the routing and DHTs. Nevertheless, the successor and predecessor of a node may be located far away. Thus, due to the distance of candidate nodes to store data items on behalf of a particular node, cooperative storage may require multi-hop data transmission, which is considered unsuitable for WSNs.

Geographic Hash Tables (GHTs) [10]–[12] hash keys into geographic locations, so data items are stored on the sensor node that are geographically nearest to the hash of its key. They replicate the stored data locally to ensure persistence in the case that nodes fail. Like normal DHTs, GHTs are built as overlays and rely on underlay routing. In fact, it uses the Greedy Perimeter Stateless Routing (GPSR) [13] algorithm for low-level routing. GPSR uses the physical location of nodes for routing purposes. Thus, it is assumed that all nodes in the network know their location by using localization methodologies like GPS [14]. GHTs also employ a scheme called Structured Replication in DCS (SR-DCS) to achieve load-balancing in the network. SR-DCS uses a hierarchical decomposition of the key space and associates each event-type e with a hierarchy depth d. It hashes each event-type to a root location. For a hierarchy depth d, it then computes $4^d - 1$ images of root. When an event occurs, it is stored at the closest image node. Queries are routed to all image nodes, starting at the root and continuing through the hierarchy. The performance of SR-DCS depends on the location of nodes that detect an event. SR-DCS performs well if the locations of nodes detecting an event are uniformly distributed over the area. However, all values of an event will be stored on the same image if the event is detected by one node (e.g., a node has specific sensor) or the nodes who detect an event are located in the same region. The image nodes are located not in the same vicinity, which can cause extra communication overhead to retrieve all values (or finding a statistical value, e.g. average or maximum values) of an event.

EnviroStore [15] is a cooperative storage scheme for sensor networks. EnviroStore focuses on redistribution of data when the storage capability of a sensor node exceeds a given threshold. In this approach, each node stores information about storage capabilities of the nodes in its communication range. This way, when a node becomes highly loaded, it migrates part of its load to another, less loaded node. Furthermore, mobile data mules are used in this scheme to carry data form loaded partitions to unloaded ones. The mules also responsible for transferring the data to the base station. It is not clear in this scheme how to use nodes that are not in the communication range of the loaded node if the nodes in the communication range are loaded.

III. Virtual Cord Protocol (VCP)

In this section, we briefly describe the design of the VCP before studying the cooperative storage in the next section. As stated previously, VCP is a DHT-like protocol. All data items are associated with numbers in a pre-determined range $[S, E]$

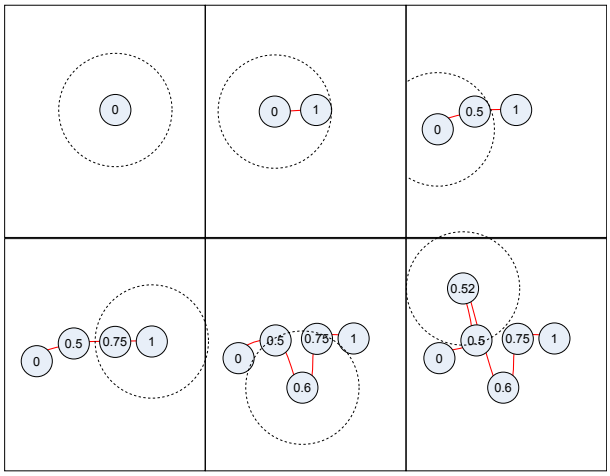

Fig. 1. Join operation in VCP

and the available nodes capture this range. Thus, each node captures a part of the entire range.

When a node joins the VCP network, it must set three important variables: its position, its predecessor, and its successor in the virtual cord. Each node determines these values based on the positions of its single-hop neighbors. First, one node must be pre-programmed as the initial node, i.e. it gets the position S. The joining node has to discover the network structure, i.e. all neighboring nodes and their position in the cord. In the proactive implementation of VCP, the nodes that already joined the network send `hello` messages by means of broadcasting, i.e. each node broadcasts a `hello` periodically.

Based on the received `hello` messages, the joining node gets information about its physical neighbors and their adjacent nodes. If the node that has sent the `hello` message is not in the physical neighbors table, it is added. Otherwise, the entry is updated accordingly. Of course, the `hello` message updates also the successor and predecessor information. If the node has not yet joined the network, it calls the `SetMyPosition()` function – an artificial join delay T_{ps} must have elapsed before re-asking for a relative position.

If a node can communicate with an end node, i.e. a node that has either position S or E, the new node takes over this end value as its virtual cord position. The old node gets a new position between the end value and its successor or predecessor, depending on the its old position. The new node becomes predecessor of the old node if it received position S. Otherwise it becomes its successor.

If a node can communicate with two adjacent nodes in the cord, the new node gets a position between the values of the two adjacent nodes. Additionally, the new node becomes successor of the old node with the lower position value and predecessor to the node that has the higher position value.

Finally, if the new node can communicate with only one node in the network, which is neither at S nor E, then the new node asks that node to create a virtual position . This virtual node gets a position between the position of the real node and its successor or predecessor. The new joining node

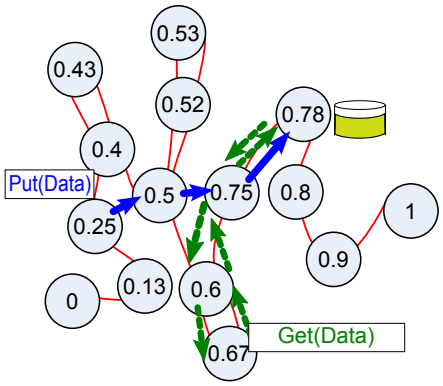

Fig. 2. An example for a Inserting/retrieving data using the virtual cord and greedy routing exploiting local neighborhood information

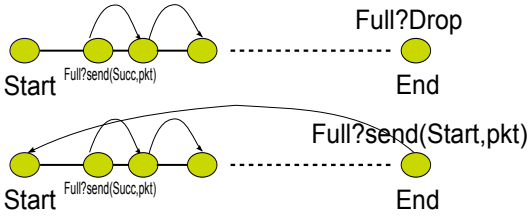

Fig. 3. VCP-based cooperative storage:Ring disabled(up) and Ring enabled (bottom)

can now get a position in between the real and the virtual position of the node in the cord. Notice that the node has to wait some time before asking for a virtual node. This timeout is used to encourage the node to find multiple neighbors, i.e. to get a proper position in the cord without the need to setup a virtual position. In previous experiments, we discovered that fewer virtual nodes lead to better routing paths [7].

Figure 1 shows the joining process for a six node network. The outer circle indicates the communication range of the newly joining node. In the first five steps, nodes are placed in the cord according to the simple rule to create new addresses either at an end or in the middle of the cord. In the sixth step, a virtual node is created to join node 0.52.

For routing, each node has to know its successor and predecessor as well as the physical neighbors. Then a greedy algorithm is employed to send packets to the node of the physical neighbors that has the closest position to the destination until there is no more progress and the value lies between the positions of the predecessor and successor. VCP inherently relies on a previously established cord. Therefore, greedy routing will always discover a path to the destination – it is not possible to run into a dead end. Additionally, VCP allows to take shortcuts whenever a physical neighbor with a virtual number is available that is closer to the destination.

For example, if node 0.25 in Figure 2 produces a data item, then it has to hash this item to the correspondent hash value. Continuing with our example, we assume a hash value of 0.781. Thus, node 0.25 will forward the message towards the destination node, i.e. in our case to node 0.5, which has the closest position to the value among the physical neighbors. Afterwards, node 0.5 will send it to node 0.75, then node 0.75 will send it to node 0.78. Node 0.78 will finally store the data and will not send it any more because there is no more progress possible and the value lies between the positions of the predecessor and the successor.

IV. Cooperative Storage

VCP inherently offers a cost-effective mechanism to find alternative nodes which can offer part of their storage capacity to store data in place of other nodes. In VCP, these nodes are the successors and predecessors on the cord. This way, if the storage capacity of a node is full, this node can send new data messages to either its successor or its predecessor. In our system, we use the successor nodes. If the storage capacity of successor also reached its limits, it can in turn send the new data items to its own successor until reaching a node that can store the data item. Recall that successors (and predecessors) of a node are located in the same vicinity, therefore, storing data on the successor always requires only single-hop data transmission. In order to update a data item that is not available at the node itself, it has to ask its succeeding nodes, whether the item has been stored at this place.

It can happen that storage capacity of the nodes at the end of the cord gets exhausted. In this case, we can either treat the cord as a ring (*ring enabled*) or simply drop the packets (*ring disabled*). If the ring is enabled, the successor of the end node is the node at the start of the cord. This approach implies on average a path length of $O(\sqrt{N})$ between the nodes at E and S. Taking into consideration that this case does not happen often, the effectiveness of our cooperative storage approach should not be affected. Figure 3 shows cooperative storage based on VCP.

Upon receiving a query for a key, the node has to search locally and send the query to its successor(s). However, if there are no constraints on the number of succeeding nodes to search on, the search process will take long paths. In order to avoid the extra search overhead that can be introduced as a result of long searches on the succeeding nodes, Bloom filters [16] can be used. Bloom filters offer a way of representing a set of elements as a compact summary.

Consider a set of n elements to be represented using a m-bit vector, which is initially set to 0. A set of k independent hash functions $h_1, \ldots, h_k$ is chosen where each function maps an input item to a random number uniformly distributed in the range $[1, \ldots, m]$. For each element x to be represented, the bits at positions $h_i(x)$ are set to 1 for $1 \leq i \leq k$. Obviously, a bit may be set to 1 multiple times and the bit vector only serves as a summary.

To figure out if an element y is in the data list, we check the positions $h_1(y), \ldots, h_k(y)$ in the bit vector. If they are all set to 1, then we can infer that y is in the data-list with a high probability – though there is a probability of a false positive. The salient feature of a Bloom filter is that the probability of false positives decreases exponentially with m if the number

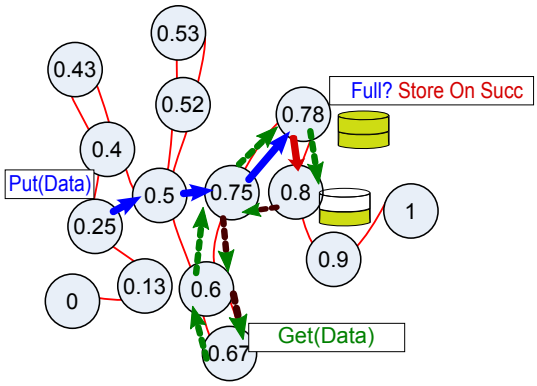

Fig. 4. An example for a inserting/retrieving data and using the successor on virtual cord to store data on behalf of a storage-exhausted node

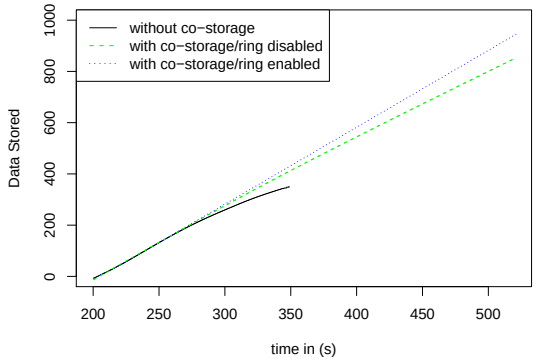

Fig. 5. Number of stored data items in the network

of hash functions k is chosen optimally. It has been shown in [17], [18] that the probability of a false positive is given by $(1-(1-\frac{1}{m})^{nk})^k$.

We now explain how this concept can be applied to the problem of data storage in a sensor network using VCP. Each node maintains a Bloom filter for its successor. Thus, the Bloom filter can be used to compactly represent which nodes store data on behalf of other nodes. By means of Bloom filter based matching, we can infer if a node stores a certain data item for other nodes. Hence, a node can then forward the query to its successor only if this node has sent the data item corresponding to the query to its successor.

Figure 4 depicts an insertion/retrieving example on a storage-exhausted node. We use the same example discussed for the routing of VCP. If the data item with hash value 0.781 is detected frequently, then, because sensor nodes have limited resources, it can happen that the storage capacity of node 0.78 gets exhausted. Now node 0.78 has to store this data item on its successor. Hence, it will send the data items to node 0.8 and insert this value in the successor Bloom filter.

A query to this data item will also reach node 0.78. Now, this node will check if data 0.781 was sent to its successor by examining the bloom filter. Because this test will return a positive answer, node 0.78 will forward the query to node 0.8. It is important to say that sending the answer to the originating node does not require to follow the same path which was used by the query. In the example, node 0.8 will send the answer to node 0.75 because it is closer to the destination (node 0.67) than node 0.78.

V. Performance Evaluation

We implemented a simulation model of the VCP in OMNeT++ [19]. OMNeT++ is a discrete event based simulator free for academic use. We also used the INET framework that provides detailed simulation models of typical Internet protocols – including our own implementation of VCP. In our simulation, we built our protocol on the top of the IEEE 802.11 Wireless LAN protocol. We investigated our approach for different network sizes between 100 and 400 nodes. We also adjusted the plane dimensions to keep the density of nodes per square meter constant.

To explore the effect of storage hot-spots, we simulated different number of keys. Furthermore, we studied the storage capacities of nodes in which we varied the percentage of storage capacity that nodes offer to store data on behalf of other nodes. For all experiments we used $200\,\mathrm{s}$ for network initialization. In the default configuration after the initialization period, 3 randomly selected nodes started to generate one data item per second. Each node started the data generation at a random time in the interval $[200-220]\mathrm{s}$ and all nodes stopped generating data at $520\,\mathrm{s}$. One node was configured as a base station. It started sending queries at time $300\,\mathrm{s}$ and stopped at $900\,\mathrm{s}$. The query should return an accumulated value by visiting all nodes that hold a data item for a specific key. The default network size was 200 nodes, the number of keys was 20, the storage capacity of all nodes was 20 units of which the nodes were allowed to offer $100\,\%$ to store data on behalf of other nodes. We ran each simulation experiment at least 5 times. We used cubic smoothing spline to show the results in this paper, because smoothing spline can be used well for estimating trends in time series.

A. With/Without co-storage

To explore the effectiveness of the proposed approach, we first investigated the ability of nodes to store data for later retrieval with and without cooperative storage schemes. We used a network with limited storage capacity.

Figure 5 shows the number of data items stored correctly in the network. In the configuration without cooperative storage the network was not able to store data after about 350 items have been generated. This can be explained as follows: If the keys were uniformly distributed as well as the virtual position of nodes, then at most 20 nodes will participate to store the generated data items. Thus, a maximum limit of 400 data items can be stored. For any non-uniform distribution, this limit is much smaller. Enabling cooperative storage, such a limit is no longer valid. If we limit the cooperative storage only to the successor (ring disabled), we notice that some items cannot be stored. However, this is limited and happens only to data items mapped on the last portion of the cord and occurs only when

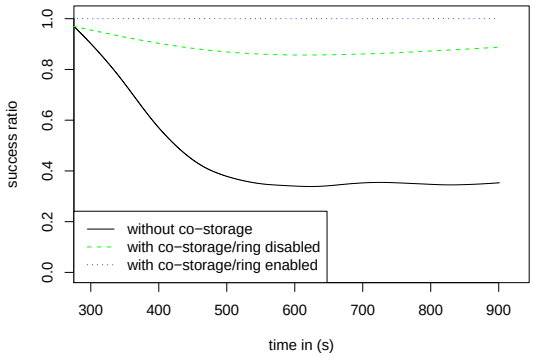

Fig. 6. Query success ratio

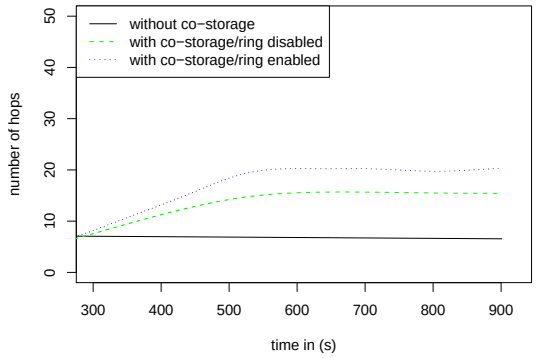

Fig. 8. Query path length

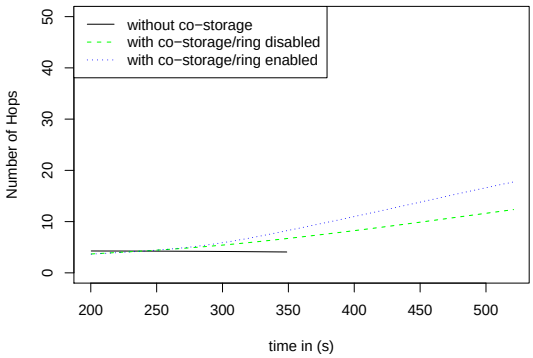

Fig. 7. Data insertion path length

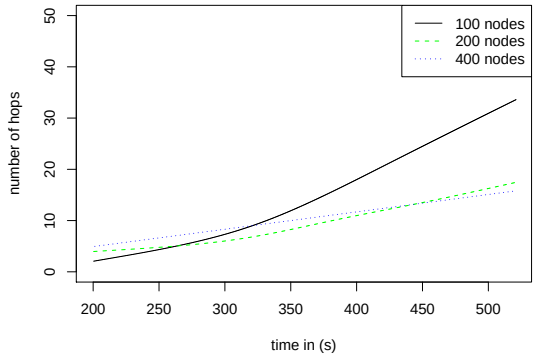

Fig. 9. Data insertion path length for different network sizes

this portion is already filled. Using the ring enabled mode to store data, the storage capacity in the network can be optimally utilized.

In order to measure availability, we use the success ratio as a key metric. This is measured after at least one event for each key has been inserted into the cord. We compute the fraction of events returned in each response, divided by the total number of events known to have been generated for that particular key. As can be seen in Figure 6, without cooperative storage the success ratio starts at an optimal value and then declines substantially as the number of detected events for a key exceeds the storage capacity of each node.

We see the price of cooperative storage in stretch of the path length as shown in Figure 7 for data insertion and in Figure 8 for data retrieval. The insertion path length increases each time a node's storage capacity get exhausted. Similarly, for queries that were sent after the insertion phase, each query has to visit all inserted data. Basically, the path length for inserts and queries depends on two parts: the path to the node with the ID closest to the current key and the path from this node to the node that is actually storing the data. For the queries, in theory the path length is twice as much as for insertion (the answer packet has to travel back the same path). However, as can be seen in Figure 8, there is only a little increase in the path length compared to the insertion path length. VCP optimizes the path for the answer by sending it on a near optimal path directly to the node originating the query.

B. Performance with different network sizes

In the remaining set of experiments presented in this paper, we evaluated the insertion path length in the ring enabled mode. In all these experiments, the network was able to store all generated data. Thus, the success rate was close to $100\,\%$.

The second set of experiments evaluates the performance of our cooperative storage scheme as the number of nodes increases while keeping the other parameters constant. We investigated three network sizes of 100, 200, and 400 nodes. Figure 9 shows the required path length to insert data items. At the beginning, we see that the larger the network size is the larger is the path length. Nevertheless, when individual nodes exhausted their storage capacity, and thus cooperative storage is required, this rule is no longer valid. Here, the number of nodes traversed to store data items in smallest network (100 nodes) is higher compared to the larger networks. This is a result of the fact that the probability to find a non-overloaded node in the smallest network is smaller than in the larger networks, which results in many successive overloaded nodes. Hence, the packet has to be transmitted through several nodes until it reaches a non-overloaded node.

C. Performance with different number of keys

To explore the effect of hot-spot storage nodes, we evaluated the performance of our cooperative storage scheme for different numbers of keys while keeping the other parameters constant. The result of using only a small number of keys in

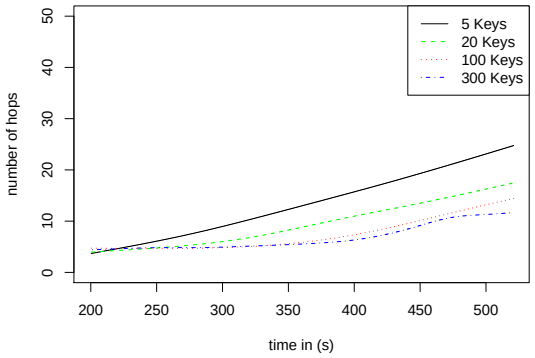

Fig. 10. Data insertion path length for different number of keys

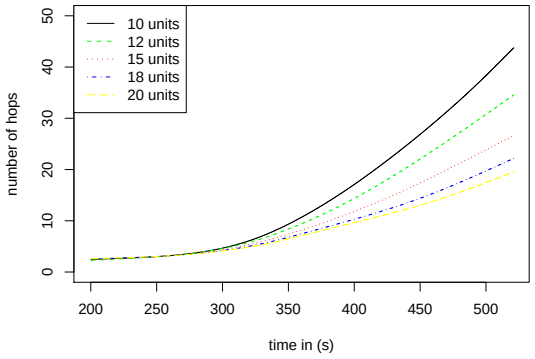

Fig. 11. Data insertion path length for different storage capacities

the network is that more data items will be mapped to the same position. This effect can be seen in Figure 10: As the number of keys decreases, the insertion path length increases.

D. Performance with partial storage capacity

Finally, to investigate the effect of offering only a part of the available storage capacity for cooperative storage, we changed the offered storage capacity from $100\,\%$ to $50\,\%$. We fixed the maximum storage capacity at 20 units in a network of 100 nodes. As can be seen in Figure 11, it is preferable to use the maximum available memory to store data in place of other nodes.

VI. Conclusion

In this paper, we presented a fully self-organizing data-centric cooperative storage system that maximizes the usage of network storage in presence of limited storage capacity nodes using the virtual position of nodes generated by VCP. In the proposed scheme, each node maintains a Bloom filter-based compact list of the inserted data. This list is used to efficiently restrict the query forwarding. Our evaluation study validates that our scheme can effectively utilize the network storage capacity of individual sensor nodes to accommodate the most sensory data.

Acknowledgment

This work is supported by DAAD grant “Peer-to-peer techniques for sensor networks” under grant number 331 4 04 001.

References

[1] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[2] R. Steinmetz and K. Wehrle, Eds., *Peer-to-Peer Systems and Applications*. Springer, 2005, vol. LNCS 3485.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications,” *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 17–32, February 2003.

[4] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network,” in *ACM SIGCOMM 2001*, San Diego, CA, USA, 2001, pp. 161–172.

[6] A. Awad, R. German, and F. Dressler, “P2P-based Routing and Data Management using the Virtual Cord Protocol (VCP),” in *9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2008), Poster Session*. Hong Kong, China: ACM, May 2008, pp. 443–444.

[7] A. Awad, C. Sommer, R. German, and F. Dressler, “Virtual Cord Protocol (VCP): A Flexible DHT-like Routing Service for Sensor Networks,” in *5th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2008)*. Atlanta, GA: IEEE, September 2008, pp. 133–142.

[8] A. Awad, L. R. Shi, R. German, and F. Dressler, “Advantages of Virtual Addressing for Efficient and Failure Tolerant Routing in Sensor Networks,” in *6th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (IEEE/IFIP WONS 2009)*. Snowbird, UT: IEEE, February 2009, pp. 111–118.

[9] M. Caesar, M. Castro, E. B. Nightingale, G. O’Shea, and A. Rowstron, “Virtual Ring Routing: Network routing inspired by DHTs,” in *SIGCOMM 2006*, Pisa, Italy, September 2006.

[10] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, “GHT: A Geographic Hash Table for Data-Centric Storage,” in *1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, Georgia, September 2002.

[11] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, “Data-Centric Storage in Sensornets,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 137–142, January 2003.

[12] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, “Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table,” *ACM/Springer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks*, vol. 8, no. 4, pp. 427–442, August 2003.

[13] B. Karp and H. T. Kung, “GPSR: Greedy Perimeter Stateless Routing for Wireless Networks,” in *6th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2000)*, Boston, Massachusetts, USA, 2000, pp. 243–254.

[14] T. A. Herring, “The Global Positioning System,” *Scientific American*, vol. 274, no. 2, pp. 44–50, February 1996.

[15] L. Luo, C. Huang, T. F. Abdelzaher, and J. Stankovic, “Envirostore: A cooperative storage system for disconnected operation in sensor networks,” in *INFOCOM*, 2007, pp. 1802–1810.

[16] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[17] 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, pp. 281–293, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.9672

[18] A. Broder and M. Mitzenmacher, “Network Applications of Bloom Filters: A Survey,” *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, January 2005.

[19] A. Varga, “The OMNeT++ Discrete Event Simulation System,” in *European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, June 2001.