

Advantages of Virtual Addressing for Efficient and Failure Tolerant Routing in Sensor Networks

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

Abstract—We study the capabilities of virtual addressing schemes for efficient and failure tolerant routing in sensor networks. In particular, we present the Virtual Cord Protocol (VCP) that uses techniques known from peer-to-peer networks, i.e. Distributed Hash Tables (DHTs) are used to associate data items in sensor networks with particular node addresses. The addresses of nodes are dynamically maintained by the protocol to form a virtual cord. VCP uses two mechanisms for finding paths to nodes and associated data items: First, it relies on the virtual cord that always points towards the destination. Furthermore, locally available neighborhood information is exploited for greedy routing. Our simulation results show that VCP is able to find paths close to the possible shortest path with very low overhead. The routing performance of VCP, which clearly outperforms other ad hoc routing protocols such as Dynamic MANET On Demand (DYMO), is similar to other virtual addressing schemes, e.g. Virtual Ring Routing (VRR). However, we improved VCP to handle frequent node failures in an optimized way. The results presented in this paper outline the capabilities of VCP to handle such cases.

I. INTRODUCTION

In recent years, Wireless Sensor Networks (WSNs) have changed from purely academic research testbeds into real-world applications. Nevertheless, many of the original research issues still apply [1]. Among others, routing has attracted many research projects. Over the last decades, a wide variety of routing protocols has been developed in the domain of sensor networks [2]. However, still most practical approaches rely on standard Mobile Ad Hoc Network (MANET) protocols such as Ad Hoc on Demand Distance Vector (AODV), Dynamic MANET On Demand (DYMO), or Dynamic Source Routing (DSR). On the other hand, WSNs show specific capabilities that demand completely different routing approaches. One of the major requirements in the domain of sensor networks is the need for network-centric operation [3], [4]. This property relies on the main working principles of WSNs, i.e. data is collected in a distributed way and needs to be analyzed as close as possible to the data source. This working behavior saves communication and energy resources in sensor networks to a large extend. Combined with data base technology such as data stream query processing, further optimization can be achieved [5].

These observations motivate the need to support the combination of data storage and communication capabilities. Interestingly, similar requirements can be observed in a different application domain, i.e. in peer-to-peer networks working

over the Internet. Here, Distributed Hash Tables (DHTs) are successfully used to distribute data over a large number of peers and to find optimal paths towards this data [6]. This operation is supported by hashes, i.e. virtual addresses.

In this paper, we analyze the advantages of using virtual addressing schemes in WSNs. Actually, a number of proposals have been made that are based on virtual addresses for routing or data storage [7]–[10]. Motivated by studies on dynamic address allocation techniques that do not scale well in sensor networks [11], we address two objectives in this paper: efficient routing towards clearly identified data items and fault tolerance w.r.t. frequent node failures. Our study is based on an extended version of our Virtual Cord Protocol (VCP) [9], [12], which follows similar concepts as used in DHTs to provide $O(1)$ complexity for storing and retrieving data items in the network.

A. Related Work

DHT based approaches for data management in WSNs can be classified in three main categories: real location based, virtual location based, and location independent. Geographic Hash Tables (GHTs) [7] hash keys into geographic locations, so the data items are stored on the sensor node geographically nearest the hash of its key. For routing, protocols such as Greedy Perimeter Stateless Routing (GPSR) [13] are used that use the physical location of nodes. Thus, it is assumed that all nodes in the network exactly know their location. Since greedy forwarding routing mode fails in case of voids, GPSR uses another mode for routing, which uses a planar subgraph without crossing edges.

One of the first virtual coordinate based protocols was Geographic Routing Without Location Information (GRWLI) [14]. Instead of using real node locations, it constitutes an n -dimensional virtual coordinate system, which is based on finding the perimeter nodes and their locations. Then, a relaxation algorithm is used to find the virtual location of all nodes. However, the drawback of having many dimensions resulting from a large n is that forming virtual coordinates requires a long time to converge [15]. Subsequently, it consumes more communication power. Again, the problem of possible dead ends exists here, so the greedy forwarding algorithm can not guarantee reaching the correct destination.

Virtual Ring Routing (VRR) [8] is a routing protocol inspired by overlay DHTs. VRR uses a unique key to iden-

tify nodes. This key is a location independent integer. VRR organizes the nodes into a virtual ring in order of increasing identifiers. For routing purposes, each node maintains a set of virtual neighbors of cardinality r that are nearest to node identifier in the virtual ring. Each node also maintains a physical neighbor set with the identifiers of nodes that it can communicate with directly. A proactively maintained routing table identifies the next hop towards each virtual neighbor. The forwarding algorithm used by VRR is quite simple. VRR picks the node with the identifier closest to the destination from the routing table and forwards the message towards that node. The problem of such protocols is that the adjacent nodes in ring can be far away in the real network. As a result, forwarding to the nearest node can result in a very long path. Moreover the scalability is a problem because, as the network gets larger, the protocol needs to maintain routing tables of increasing size.

GSpring [16] tries to improve the performance of greedy forwarding. In a first phase, each node assigns itself an initial coordinate. Subsequently, nodes adjust their coordinates by simulating a system of springs and repulsion forces. Based on this, greedy routing is performed with high only about 15% overhead compared to using real addresses.

In the hop id routing scheme [17], each node maintains a hop id, which is a multidimensional coordinate based on the distance to some landmark nodes. Fundamentally, landmarks can be randomly selected in the network. However, to obtain better performance and reduce the effect of dead ends, the authors present several methods for landmark selection. To construct and maintain the hop id system, a voluntary node first floods the entire network to build a shortest path tree. Then, landmarks are selected. Finally, each node adjusts its hop id periodically and broadcasts its new hop id using a hello message.

The special case of unidirectional links has been investigated in [10]. The developed virtual coordinate assignment protocol (ABVCap_Uni) supports routing in sensor networks with unidirectional links. Based on available unique network IDs of all nodes, the protocol tries to assign nodes with unidirectional links into rings and to treat a ring as an extended node. Routing is performed on virtual addresses assigned to real nodes and extended nodes.

B. Contributions

In the first part of the paper, we analyze the routing performance of virtual address based protocols. In particular, we compare our protocol VCP to one of the most promising approaches of virtual routing presented at SIGCOMM 2006, namely VRR [8]. As a basis measurement, we compare both to a typical MANET protocol, i.e. DYMO [18]. From the results, we will see that both virtual address based protocols clearly outperform the most recent MANET approach.

Therefore, in the second part of the paper, we study the protocol behavior in the case of frequent node failures. Such a scenario is not unusual in WSNs because nodes may fail, e.g. due to energy outages, or just become “invisible” due to changing conditions of the radio communication. For the

evaluation, we explicitly modeled a scenario that is similar to the one described in the GHT paper [7] to be able to compare the results to this approach as well. For this scenario, we only consider VCP and VRR (and, relying on the published simulation results, GHT).

The main contributions can be summarized as follows:

- We present an extended version of VCP that is able to provide efficient routing and data storage and that is capable to tolerate frequent node failures (Section II).
- In a comprehensive simulation study, we compare the routing performance of the two virtual address based protocols VCP and VRR to classical MANET protocols using DYMO as an example (Section IV).
- The capability of the protocol to handle frequent node failures is investigated in comparison to VRR and to GHT – for the latter relying on measures available in the literature (Section V).

II. VIRTUAL CORD PROTOCOL

The idea behind the Virtual Cord Protocol (VCP) is to combine data lookup with routing techniques in an efficient way. VCP accomplishes this by placing all nodes on a virtual cord, which is also used to associate data items with. A hash function is used to create values in a pre-defined range $[S, E]$ and all available nodes capture this range. Thus, each node maintains a part of the entire range. The routing mechanism relies on two concepts: First, the virtual cord can be used to find a path to each destination in the network. Additionally, locally available neighborhood information is exploited for greedy routing towards the destination. In the following, the operation of VCP is introduced including special extensions for failure management.

A. Joining operation

One node must be pre-programmed as initial node, i.e. it gets the position S . Furthermore, a number of initial variables are initialized in the startup phase as listed in Table I. We employ `hello` messages to discover the network structure, i.e. all neighboring nodes and their position in the cord. In the current implementation of VCP, the `hello` messages are transmitted by means of broadcasting, i.e. each node broadcasts a `hello` every T_h s. Basically, the joining operation can also be executed using an on-demand mechanism, which has advantages in static networks or those with a high density. Another requirement is that the first node is pre-programmed with the smallest value of the entire range, i.e. S .

Based on the periodically transmitted `hello` messages, the joining node gets information about its physical neighbors and their adjacent nodes. Algorithm 1 depicts the handling of `hello` messages. If the node has not yet joined the network, it calls the `SetMyPosition()` function listed in Algorithm 2 to get a relative position in the cord. The artificial join delay T_{ps} is used to prevent conflicts between multiple nodes that simultaneously ask for relative positions.

Each node joining the network has to receive at least one `hello` message from a node that already joined the cord in

TABLE I
INITIAL PARAMETERS FOR THE JOIN PROCESS

Parameter, Value	Description
Start $S = 0.0$	lowest position on the cord
End $E = 1.0$	highest position on the cord
Position $P = -1.0$	current position in the cord, -1 means the position is still unset
HelloPeriod $T_h = 1$ s	time interval between hello messages
SetPosDelay $T_{ps} = 1$ s	time interval before re-requesting a new position
SetVPosDelay $T_{vps} = 1$ s	time interval before requesting a virtual position
BlockDelay $T_b = 1$ s	blocking period to prevent assigning the same position to more than one node
Interval $I = 0.1$	interval between the two end positions $[S, E]$ and successor or predecessor position
VirtInterval $I_v = 0.9$	interval between node position and virtual node position

Algorithm 1 Handle hello messages

Require: Locally stored state of all neighbors in set N
Ensure: Maintain neighbor set N and set virtual address

- 1: Receive neighbor information from node N_i
- 2: **if** $N_i \notin N$ **then**
- 3: $N \leftarrow N_i$
- 4: **else**
- 5: Update $N_i \in N$
- 6: **end if**
- 7: **if** $P == -1$ AND $(\text{Time}() - \text{OldTime}) > T_{ps}$ **then**
- 8: $\text{OldTime} \leftarrow \text{Time}()$
- 9: SetMyPosition()
- 10: **end if**

order to get a relative position in the cord. If a node can communicate with an end node (lines 2–17 in Algorithm 2), 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 (lines 20–27). 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. The call of SendBlockReq() requests a position update at both neighbors. If successful, this is replied with an acknowledgment, which, in turn, activates the temporally stored position information.

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 (lines 29–33). This virtual node gets a position between the position of the real node and its successor or predecessor. The new joining node 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, i.e. T_{vps} , before asking for a virtual node.

Algorithm 2 SetMyPosition()

Require: Neighbor information stored in set N

- 1: **for** $\forall N_i \in N$ **do**
- 2: **if** $\text{Position}(N_i) == S$ **then**
- 3: **if** $\text{Successor}(N_i) < S$ **then**
- 4: $P_{temp} \leftarrow E$
- 5: **else if** $\text{Successor}(N_i) == E$ **then**
- 6: $P_{temp} \leftarrow (S + E)/2$
- 7: **else**
- 8: $P_{temp} \leftarrow \text{Successor}(N_i) - I \times (\text{Successor}(N_i) - \text{Position}(N_i))$
- 9: **end if**
- 10: SendNewPositionToNeighbor(N_i, P_{temp})
- 11: **else if** $\text{Position}(N_i) == E$ **then**
- 12: **if** $\text{Successor}(N_i) == S$ **then**
- 13: $P_{temp} \leftarrow (S + E)/2$
- 14: **else**
- 15: $P_{temp} \leftarrow \text{Predecessor}(N_i) - I \times (\text{Predecessor}(N_i) - \text{Position}(N_i))$
- 16: **end if**
- 17: SendNewPositionToNeighbor(N_i, P_{temp})
- 18: **else**
- 19: $\text{found} \leftarrow 0$
- 20: **for** $\forall N_j \in N : i \neq j$ **do**
- 21: **if** $\text{Predecessor}(N_i) == \text{Position}(N_j)$ **then**
- 22: $\text{found} \leftarrow 1$
- 23: $P_{temp} \leftarrow (\text{Position}(N_i) + \text{Position}(N_j))/2$
- 24: temporally store positions of N_i and N_j
- 25: SendBlockReq(N_j, P_{temp})
- 26: **end if**
- 27: **end for**
- 28: **if** $\text{found} == 0$ **then**
- 29: **if** $(\text{Time}() - \text{OldVTime}) > T_{vps}$ **then**
- 30: $\text{OldVTime} \leftarrow \text{Time}()$
- 31: temporally store position of N_i
- 32: SendCreatVirtualNode(N_i)
- 33: **end if**
- 34: **end if**
- 35: **end if**
- 36: **end for**

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. We noticed that fewer virtual nodes lead to better routing paths. More details on the creation of virtual nodes can be found in [9].

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.

Figure 2 depicts the network after adding 15 nodes. Routing in VCP is done using the virtual cord. Additionally, local neighborhood information is exploited for greedy routing. The

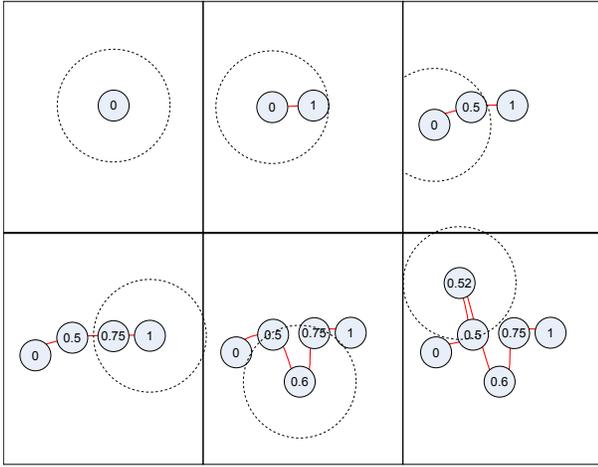


Fig. 1. Basic join operation in VCP, six nodes are joining the network according to the rules described in Algorithm 2

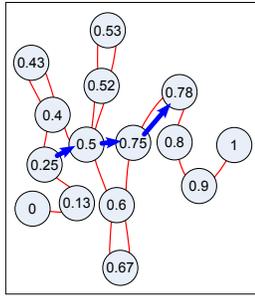


Fig. 2. An example for a routing path using the virtual cord and greedy routing exploiting local neighborhood information

greedy forwarding works as follows: a node with relative position P forwards a packet to its neighbor N_i that has the closest virtual position to the destination D_p . The forwarding is terminated if no more progress is possible, i.e. the local coordinate P is closest D_p . Based on the established cord, VCP will never get stuck in dead ends. Preliminary studies have shown that the path stretch is quite optimal. Also, it is clear that the joining of a new node only affects a small number of nodes in the vicinity of the node and it is independent of the total number of nodes in the network. In fact, the insertion of a new node only affects $O(m)$, where m is the number of successors.

B. Failure management

The presented cord management is working very well as long as all nodes stay available after forming the cord. Greedy forwarding can guarantee the reachability of the destination only if there is no failure. However, in case of node failures, greedy forwarding might fail and the cord becomes unstable. To overcome the problem of finding the a path towards the destination in case of node failures, we propose a new schema to find an alternative path.

We use `hello` messages to identify failed nodes. In particular, we store the timestamp of the last `hello` message in the routing table, i.e. the physical neighbor table, in addition to

successor and predecessor positions. If a node did not receive a `hello` message from a neighbor for $n \times T_h$ s, where T_h is the hello message period, this neighbor is marked as a dead node. From the available information in the routing table, each node can locally check whether the correct destination of a packet is this dead neighbor itself or one of this neighbor's physical neighbors.

During packet routing there are two cases in which greedy forward cannot reach the correct destination because of a dead end in the cord. The first case is to reach a physical neighbor of the failed node. In this case, the packet can be either dropped or stored within the neighbor of the failed node. If the operation was to retrieve data items, the connection is counted as not successful. In future work, we plan to add data replication techniques to counteract this case.

Secondly, the failed node is the next hop towards the destination but not the final destination itself. In this case, we have to find an alternative path. The procedure is as follows. The neighbor of the failing node locally creates a so called *no path interval* NP-I. This interval corresponds to the range of IDs that the dead node was responsible for. Then, the node sends a *no path* (NP) packet, which includes NP-I to another active node in its neighborhood. This node is selected according to its position in the cord that should be as close as possible to NP-I. In order to prevent routing loops, this information needs to be stored on all nodes involved in this process. However, the stored NP-I data is expected to be expired after T_{np} s. From now on, each node either transmits the data using greedy forwarding towards the destination if there is a neighboring node closer to the destination available, or it continues to send NP packets. Using the stored NP-I data, this information will never be sent twice. If a NP packet reaches a node, which already has NP-I in its table, it has to send a *no path back* (NPB) packet as an indicator of a detected loop. The procedure of treating routing packets is shown in more detail in Algorithm 3. The interval $[P_{min}, P_{max}]$ is maintained by evaluating the distance between the current node and the neighbors on the cord. In particular, this interval is used to identify the final destination for each packet. The *no path interval* NP-I is maintained by

Algorithm 3 Handle routing packets

Require: Received data packet D for destination position D_p ,
locally maintained data set $[P_{min}, P_{max}]$

- 1: **if** $P_{min} \leq D_p \leq P_{max}$ **then**
- 2: StoreData()
- 3: **else if** $D \in \text{NP-I}$ **then**
- 4: Send(NPB, D)
- 5: **else if** $\exists N_i \in N : |\text{Position}(N_i) - D_p| < |P - D_p|$ **then**
- 6: Send(N_i , D)
- 7: **else**
- 8: ComputeNoPathInterval()
- 9: Send(NP, D)
- 10: **end if**

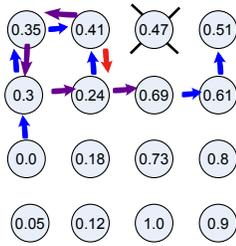


Fig. 3. Routing in case of a node failure: node 0.41 generates a NP packet that is forwarded until node 0.69 continues to regularly forward the packet

the function `ComputeNoPathInterval()`. In short, it just checks the locally stored NP information, updates the timeouts and expires old entries.

An example for packet forwarding in case of a node failure is illustrated in Figure 3. In this example, node 0.0 produces a data packet destined to address 0.52. According to the greedy routing principles of VCP, the data packet will be forwarded by nodes 0.30 and 0.35 until it reaches node 0.41. At this node, a dead end is detected because the previously existing node 0.47 has failed. Thus, node 0.41 will create a *no path interval* NP-I and, accordingly, send a NP packet back the path to node 0.35. Similarly, the NP packet is forwarded until it reaches node 0.24. This one, according to the programmed rules, tries node 0.47 again, however, 0.47 detects a loop and sends a NPB packet to node 0.24. In turn, node 0.24 tries to find another path by sending a NP packet to node 0.69. Finally, this node can resume greedy forwarding toward destination node 0.51.

III. SIMULATION MODEL AND PARAMETERS

For analyzing the performance of the virtual address based routing protocols VCP and VRR, we implemented these models in OMNeT++. Furthermore, we used our recently implemented model of the DYMO routing protocol, which is now publicly available [19]. OMNeT++ is a discrete event based simulator free for academic use. Additionally, we used the *INET framework* that provides detailed simulation models of the MAC and the physical layer. In our simulation, we built our protocol on the top of the IEEE 802.11 WLAN protocol.

For all communications, the complete network stack is simulated and wireless modules are configured to closely resemble IEEE 802.11b network cards transmitting at 2 Mbit/s with RTS/CTS disabled. For the simulation of radio wave propagation, a plain free-space model is employed, with the transmission ranges of all nodes adjusted to a fixed value of 50 m. All simulation parameters used to parameterize the modules of the *INET Framework* are summarized in Table II.

In our experiments, 100 nodes are deployed either in form of a grid or randomly on a rectangular area. A single node is dedicated as the sink node and placed on the upper left corner of the playground. We allowed for an initial transient period of 400 s in which VCP and VRR initialize their address information and routing tables. Afterwards, each node starts transmitting at a time in the range [400, 418] s for a data rate of 1 pps and in the range [400, 580] s for a data rate of 0.1 pps.

TABLE II
INET FRAMEWORK MODULE PARAMETERS

Parameter	Value
mac.address	auto
mac.bitrate	2 Mbit/s
mac.broadcastBackoff	31 slots
mac.maxQueueSize	14 Pckts
mac.rtsCts	false
snrEval.bitrate	2 Mbit/s
snrEval.headerLength	192 bit
snrEval.snrThresholdLevel	4 dB
snrEval.thermalNoise	-110 dB
snrEval.sensitivity	-85 dB
snrEval.pathLossAlpha	2.5
snrEval.carrierFrequency	2.4 GHz
snrEval.transmitterPower	1 mW

TABLE III
SIMULATION PARAMETERS

Input Parameter	Value
Number of Nodes	100
Playground size	1 80 × 180 m ² or 4 00 × 400 m ²
Node placement	Grid and random
Data rate	CBR, 1 pps or 0.1 pps
Initialization time	400 s
Start of data transmission	uniformly distributed in [400, 418] s or [400, 580] s
End of data transmission	490 s or 1300 s
Destination node	Upper left node
Fraction of failing nodes	0 %, 20 %, 40 %, 60 %, 80 %, or 100 %
On time	uniformly distributed in [0, 120] s
Off time	uniformly distributed in [0, 60] s
Start of node failures	400 s
End of node failures	1300 s

The experiment is terminated at 490 s or 1300 s, respectively.

For the second set of experiments, we analyzed the failure tolerance of the routing protocol by periodically switching a fraction of the nodes on and off (except the sink node). Both the on and off intervals are uniformly distributed in a pre-defined range. After a node is re-activated, it needs to re-join the network and re-establish its routing information. For statistical evidence, for each experiment, we performed ten runs. All the simulation parameters are summarized in Table III. For evaluation, we selected four basis measures. First, we analyzed the success ratio, which describes the capability of the protocol to ensure correct data delivery. Then, the MAC collisions were analyzed to get an impression of the overall network load. Furthermore, the path length is evaluated and, finally, the end-to-end delay.

All results discussed in the following sections are shown as boxplots. For each data set, a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the edges of the box towards the minimum and maximum of the data set. Data points outside the range of box and whiskers are considered outliers and drawn separately. Additionally, the mean value is depicted in form of a small filled square.

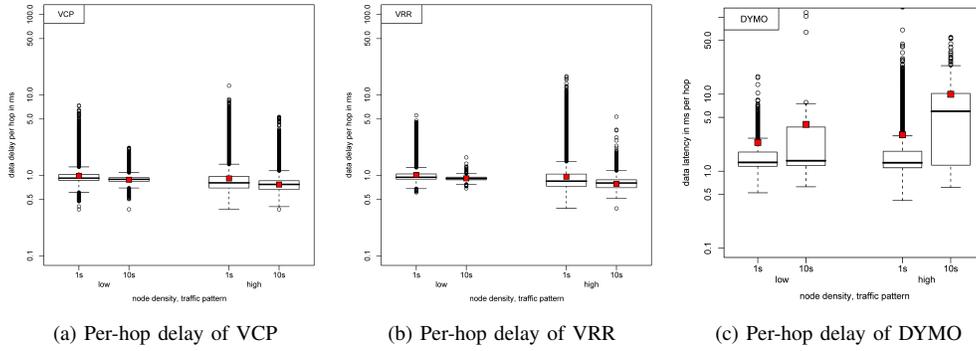


Fig. 4. Delay performance of VCP, VRR, and DYMO in the grid scenario: depicted is the latency as observed by the application normalized to the path length; all figures are plotted using a log scale y-axis

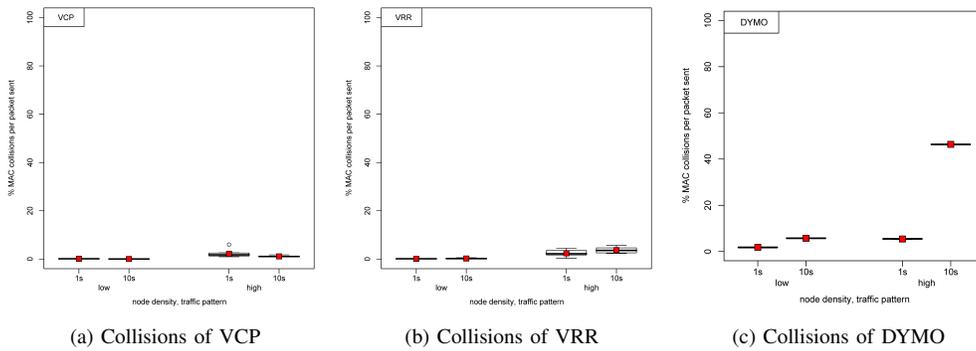


Fig. 5. MAC layer collisions per date packet sent for VCP, VRR, and DYMO in the grid scenario

IV. ROUTING PERFORMANCE OF VIRTUAL-ADDRESS BASED PROTOCOLS

In a first set of simulation experiments, we evaluated the routing performance of VCP. In particular, we compare VCP to VRR, a competitive approach relying on virtual coordinates for routing, and to DYMO, which is the most recent standard of ad hoc routing protocols as developed by the IETF MANET working group. We rely on the simulation settings as described in Section III. Basically, these settings represent a basis for analyzing the routing performance in sensor networks [8], [9].

In a first measure, we evaluated the end-to-end delay as observed by the application. In a number of different experiments, we analyzed the protocol behavior for different network densities and traffic rates. Also, we evaluated the influence of the network topology, i.e. grid or random. The results for the grid scenario are depicted in Figure 4 (for random deployment, the results are similar). For better comparability, we normalized the latency to the path length, i.e. to a per hop delay. As can be seen, the per hop delay for VCP and VRR is quite similar. Both, the mean and the median are at about 1 ms. Some outliers can be observed up to about 10 ms. Both protocols are very robust w.r.t. the network density and the traffic load. Differently, the MANET routing protocol DYMO performed slightly worse for higher traffic load (depicted as 1 s traffic pattern). For lower traffic rates, the observed delay increases largely. This effect can be explained by the route timeouts used by DYMO in our experiment. In the ten seconds example, DYMO has to set up a route for almost each packet because the available routes have timed out. Thus, each time an additional route setup delay adds to the transmission delay.

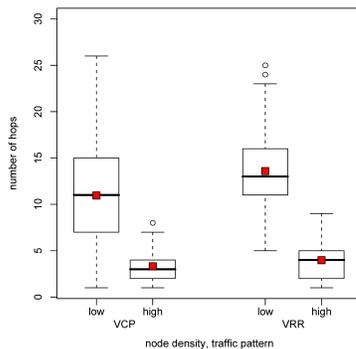


Fig. 6. Path lengths of VCP and VRR for low and high density scenarios

For the analysis of routing protocols designed for wireless networks, a main measure to observe is the number of MAC layer collisions. This metric helps to evaluate the overall load in the network. Figure 5 shows the results for the grid scenario (for random deployment, the number of MAC collisions shows

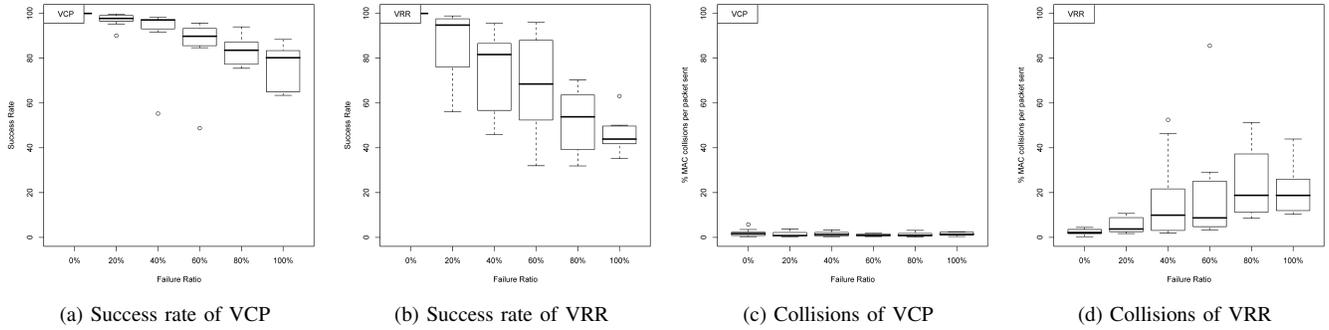


Fig. 7. Failure performance: depicted are the success rate and number of MAC collisions for the grid scenario

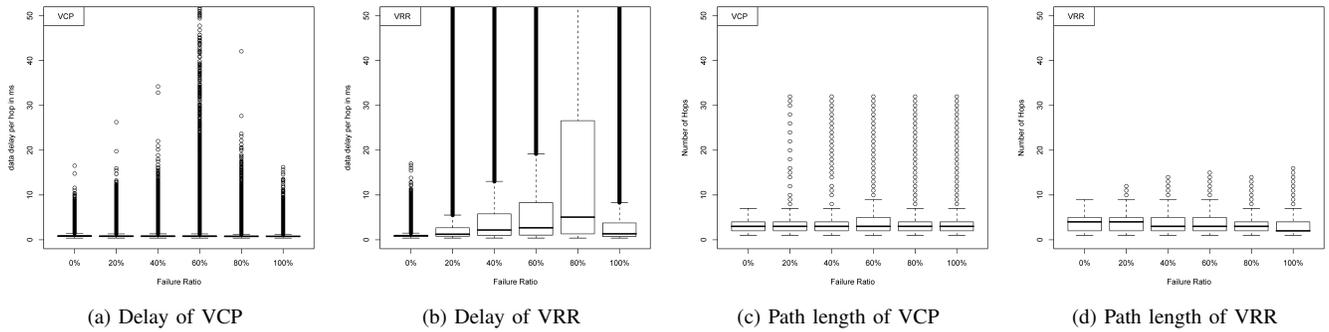


Fig. 8. Failure performance: depicted are the per hop delay and the path lengths for the grid scenario

a higher variance but a similar trend). In particular, the number of MAC collisions is almost zero for the virtual address based routing protocols (for the high density scenario, VRR shows about 5% collisions per data packet sent, which is negligible). However, the number of collisions is quite high for DYMO. This is not an effect of collisions among data messages but among the periodic hello messages. With decreasing data rates, the percentage of hellos to normal data messages increases and only effects (collisions) of hellos can be measured.

Finally, we evaluated the stretch ratio, i.e. the deviation of path length calculated by the virtual address based routing protocols to the shortest path. It turned out that in all the scenarios the measured stretch ratio was in the interval of $[1, 1.25]$. This measure was independent of the network size and density as well as of the deployment topology. Also, VCP and VRR provided almost similar results. A deviation of 25% from the shortest path can especially be considered acceptable if the observed end-to-end latency is not being influenced to a large degree. Figure 6 depicts the typical path lengths as observed during the simulation runs for the low and the high density scenarios. As can be seen, the path length used by VRR is slightly higher compared to the VCP paths.

V. PROTOCOL BEHAVIOR IN PRESENCE OF NODE FAILURES

In a second set of experiments, we focused on the protocol behavior in presence of frequent node failures. As a

node failure, we consider in general any event that prevents communication to a particular node at a given time, e.g. complete energy outages and node replacement, or interrupted communications due to changes in the radio propagation. For these experiments, we only consider VCP and VRR because the network load (and therefore the collision probability) increases too much for MANET protocols such as DYMO.

The general setup is the same as used for evaluating the routing performance. However, we further introduced node failures as described in Section III. We follow the simulation setup described in [7]. In particular, failing nodes are modeled as a uniformly distributed on/off process. We increase the number of nodes toggling their state from 0% to 100%. Four metrics were chosen for the performance comparison. In all the figures, we show results for the grid scenario. For random deployment, the variance was slightly higher but the trend of the results was exactly the same.

First, we investigated the success rate, i.e. the number of transmissions that were completed successfully. Figures 7a and 7b depict the measured success rate for VCP and VRR, respectively. As can be seen, the ratio of successful transmission degrades with the number of failing nodes. However, VCP still maintains a success rate of about 70%–80%. In contrast, the success rate degrades much faster for VRR (down to 50%). A look at the network load reveals some effects that explain the reduced success rate of VRR compared to

VCP. Figures 7c and 7d show the number of MAC layer collisions. As can be seen, there are almost no collisions for VCP, which outlines the capability of this protocol to work even in extreme failure situations. On the other hand, VRR needs many state maintenance operations that lead to increased network congestion.

When VRR enters the transmission phase after its initial join phase, it simply forwards packets to the nodes that has the closest ID to the packet ID. It needs to be noted that “closest” ID means the ID that is closest on the virtual ring. When the network size increases and many nodes fail periodically, the node’s forwarding table becomes incomplete and only represents a local view of the whole network. VRR has two different strategies to handle such failure situations: exact repair and local “vset-path” repair. The idea is to bypass the failed node. However, this technique only works for a few node failures.

This effect can also be observed when looking at the latency performance. Figure 8a shows that the median per hop delay of VCP is not affected by the failing nodes. Differently, the delay of VRR (Figure 8b) increases with the failure ratio.

Finally, we looked at the path length that outlines the capability of the routing protocol to find shortest paths even in case of many node failures. Figures 8c and 8d depict the simulation results. While the average path length is slightly shorter for VCP, some single outliers correspond to special cases in which the virtual cord needs to be used for routing instead of the optimal greedy routing between physical neighbors.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an extended version of VCP that supports improved failure tolerance. Basically, our goal was to emphasize the benefits of virtual coordinate based routing schemes in WSNs. Therefore, we compared the routing performance of virtual address based protocols, in particular VCP and VRR with a typical ad hoc routing protocol (DYMO). The results show that VCP shows similar routing performance as VRR in the optimal case, i.e. no node failures. However, in case of node failures, VCP demonstrates its strengths of efficient cord management. VCP shows a much better tolerance to node failures compared to VRR due to its low maintenance overhead. Together with the capabilities of virtual address based protocols to manage data using an application-specific hash function, we conclude that such approaches are better suited for WSNs, especially if these networks are dynamic w.r.t. node failures. Future work includes studies of the suitability of different hash functions for content replication in sensor networks. Additionally, we are working on an implementation on real sensor nodes for first lab experiments.

ACKNOWLEDGEMENTS

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

REFERENCES

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Elsevier Computer Networks*, vol. 38, pp. 393–422, 2002.
- [2] K. Akkaya and M. Younis, “A Survey of Routing Protocols in Wireless Sensor Networks,” *Elsevier Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, 2005.
- [3] B. Krishnamachari, D. Estrin, and S. Wicker, “The Impact of Data Aggregation in Wireless Sensor Networks,” in *International Workshop Distributed Event Based System (DEBS 2002)*, Vienna, Austria, July 2002.
- [4] R. Govindan, “Data-centric Routing and Storage in Sensor Networks,” in *Wireless Sensor Networks*, C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Springer, 2004, pp. 185–205.
- [5] J. Gehrke and S. Madden, “Query Processing in Sensor Networks,” *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 46–55, January–March 2004.
- [6] R. Steinmetz and K. Wehrle, Eds., *Peer-to-Peer Systems and Applications*. Springer, 2005, vol. LNCS 3485.
- [7] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, “Data-Centric Storage in Sensor Networks 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.
- [8] 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.
- [9] 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.
- [10] C.-H. Lin, B.-H. Liu, H.-Y. Yang, C.-Y. Kao, and M.-J. Tasi, “Virtual-Coordinate-Based Delivery-Guaranteed Routing Protocol in Wireless Sensor Networks with Unidirectional Links,” in *27th IEEE Conference on Computer Communications (IEEE INFOCOM 2008)*. Phoenix, AZ: IEEE, April 2008.
- [11] F. Dressler and F. Chen, “Dynamic Address Allocation for Self-organized Management and Control in Sensor Networks,” *International Journal of Mobile Network Design and Innovation (IJMNDI)*, vol. 2, no. 2, pp. 116–124, 2007.
- [12] 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.
- [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, MA, 2000, pp. 243–254.
- [14] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, “Geographic Routing without Location Information,” in *9th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2003)*, San Diego, CA, September 2003.
- [15] K. Liu and N. Abu-Ghazaleh, “Aligned Virtual Coordinates for Greedy Routing in WSNs,” in *3rd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS 2006)*. Vancouver, Canada: IEEE, October 2006, pp. 377–386.
- [16] B. Leong, B. Liskov, and R. Morris, “Greedy Virtual Coordinates for Geographic Routing,” in *15th IEEE International Conference on Network Protocols (ICNP 2007)*, Beijing, China, October 2007, pp. 71–80.
- [17] Y. Zhao, Y. Chen, B. Li, and Q. Zhang, “Hop ID: A Virtual Coordinate-Based Routing for Sparse Mobile Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, vol. 6, no. 9, pp. 1075–1089, September 2007.
- [18] I. Chakeres and C. Perkins, “Dynamic MANET On-Demand (DYMO) Routing,” Internet-Draft (work in progress) draft-ietf-manet-dymo-10.txt, July 2007.
- [19] C. Sommer, I. Dietrich, and F. Dressler, “A Simulation Model of DYMO for Ad Hoc Routing in OMNeT++,” in *1st ACM International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008): 1st ACM International Workshop on OMNeT++ (OMNeT++ 2008)*. Marseille, France: ACM, March 2008.