Virtual Cord Protocol (VCP): A Flexible DHT-like Routing Service for Sensor Networks

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

Abstract

Efficient data management techniques are needed in Wireless Sensor Networks (WSNs) to counteract issues related to limited resources, e.g. energy, memory, bandwidth, as well as limited connectivity. Self-organizing and cooperative algorithms are thought to be the optimal solution to overcome these limitations. On an abstract level, structured peer-to-peer protocols provide O(1) complexity for storing and retrieving data in the network. However, they rely on underlayer routing techniques. In this paper, we present the Virtual Cord Protocol (VCP), a virtual relative position based efficient routing protocol that also provides means for data management, e.g. insert, get, and delete, as known from typical Distributed Hash Table (DHT) services. The key contributions of this protocol are independence of real location information by relying on relative positions of neighboring nodes, short virtual paths because successors and predecessors are in their vicinity, and high scalability because only information about direct neighbors is needed for routing. Furthermore, VCP inherently prevents dead-ends and it is easy to be implemented.

1 Introduction

Wireless Sensor Networks (WSNs) provide an interesting research domain because they represent a class of massively distributed systems in which nodes are required to work in a cooperative and self-organized fashion to overcome scalability problems [2]. Additionally, WSNs are facing strong resource limitations as a large number of sensor nodes with strong CPU, energy, and bandwidth restrictions need to be operated to build stable and operational networks. This includes the need to solve problems with high dynamics introduced by joining and leaving nodes. As of today, WSNs are used in a wide range of applications such as wildlife monitoring, disaster prediction and management, health care, precision agriculture, and intelligent homes [6, 7].

In many cases the produced sensor data is enormous, thus an efficient data management is essential. There are several methods to retrieve data from a network. One method is to use a central server that maintains the current location of data items. Thus, all requests are directed to the server that returns the current location of the data item. Another method is to use flooding search. Thus, to retrieve a data item, the network is flooded with the request and a certain number of nodes is queried whether or not they have stored the data item. The drawbacks of the central server approach are that the server represents a single point of failure and that it provides a possible bottleneck reducing the scalability. On the other hand, the flooding approach consumes high communication power and search results are not guaranteed if the requests are restricted to a limited number of hops.

Distributed Hash Tables (DHTs) [20] 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 find 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 [21], Pastry [18], and CAN [15] have been implemented to work on the Internet for scalable file sharing applications. The nodes communicate taking advantage of the already existing routing protocols on the Internet.

Implementing DHTs in WSNs as an overlay and relying on typical Mobile Ad Hoc Network (MANET) routing protocols [1] (best known examples are DSR [8], DSDV [12], or AODV [13]) has the drawback that these routing protocols already need to maintain globally valid topology information of the entire network. This cannot scale well because additional overhead is needed to maintain the overlay.

978-1-4244-2575-4/08/$20.00 ©2008 IEEE

In addition, the DHTs have not been designed to take advantage of physically neighboring nodes. On the other hand, routing protocols that use geographic location like Greedy Perimeter Stateless Routing (GPSR) [9] and Geographic Routing Without Location Information (GRWLI) [14] can scale well. Unfortunately, obtaining the location is not only costly and susceptible to localization errors, but also is not always available and greedy forwarding cannot guarantee reachability of all destinations because of possible dead ends [11].

This paper presents the Virtual Cord Protocol, a DHT-like protocol that offers in addition to standard DHT functions (e.g., insert, get, and delete) an efficient routing mechanism [3]. The key important characteristics of this protocol besides the efficient routing are:

- The successors and predecessors of a node are in its direct vicinity, which reduces the communication load when nodes join and leave the network.
- The exact physical location is not required, which can be expensive in terms of communication or system requirements. Instead, we use an easy-to-be-obtained relative position.
- VCP is scalable because it only needs information about direct neighbors for routing.
- Greedy routing on the cord always leads to a path to the destination (it cannot suffer from packets getting stuck in dead-ends).
- The protocol is easy to be implemented on top of the MAC layer.

The rest of the paper is organized as follows. In Section 2, we outline relevant related work. In Section 3, an overview to the working principles of our Virtual Cord Protocol (VCP) protocol is presented. Afterwards, the implementation and selected evaluation results are presented in Section 4. This also includes a comparison to a typical MANET routing protocol to evaluate the routing performance of our VCP protocol. Finally, Section 5 concludes the paper.

2 Related work

DHT based approaches that are targeted to manage data in WSNs can be classified in three main categories: real location based, virtual location based, and location independent. Geographic Hash Tables (GHTs) [16, 17] hash keys into geographic locations, so the data items are stored on the sensor node geographically nearest the hash of its key. They replicate the stored data locally to ensure persistence when nodes fail. Like ordinary DHTs, GHTs are built as overlays and rely on underlay routing. In fact, it uses GPSR [9] for routing. GPSR uses the physical location of nodes for routing purposes. Thus, it is assumed that all nodes in the network know their location. Since greedy forwarding routing mode fails in case of voids (even in static networks), GPSR uses another mode for routing, which uses a planar subgraph without crossing edges. In comparison, VCP focuses on efficient routing on a virtual cord. It features a pre-defined hashing range that allows applications to clearly associate data items to places in the cord.

A virtual coordinate routing protocols is GRWLI [14]. Unlike GPSR, this routing protocol does not use real coordinates, which is costly, not available in many situations, and susceptible to localization errors. Instead, it constitutes an n-dimensional virtual coordinate system. The construction of the coordinates is based on finding the perimeter nodes and their locations, if they are not pre-defined. Then, a relaxation algorithm is used to find the virtual location of the nodes in the network. However, the drawback of having many dimensions resulting from a large n is that forming virtual coordinates requires a long time to converge [10]. 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) [4] is a routing protocol inspired by overlay DHTs. Besides the routing, it provides traditional DHT functionality. VRR uses a unique key to identify 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 routing table entry identifies the next hop towards virtual neighbor. This information is maintained proactively, i.e. it is maintained even when there is no traffic along the path. 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.

In the hop id routing scheme [23], 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, three basic steps should be followed. First, a voluntary node floods the entire network to build a shortest path tree rooted at this node. Then, landmarks are selected. Finally, each node adjusts its hop id periodically and broadcasts its new hop id using a hello message. To deal with dead ends when greedy forwarding fails, a landmark guided detour is designed and is applied with an expanding ring search algorithm to route out of dead ends.

3 Virtual Cord Protocol

As stated previously, Virtual Cord Protocol (VCP) is a DHT-like protocol. All data items are associated with numbers in a pre-determined range $[S, E]$ and the available nodes capture this range. Thus, each node captures a part of the entire range.

3.1 Joining operation

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 seconds. 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.

Based on the `hello` messages, the joining node gets information about its physical neighbors and their adjacent nodes. The first node is pre-programmed with the smallest value of the entire range (S, for "start"). The second node joining the network gets the largest number of this range (E, for "end"). The basic join algorithm is shown in Algorithm 1.

Each further node joining the network has to received at least with one `hello` message from a node that already exist in the network to get a relative position, i.e. its value, in the cord. If a node can communicate with an end node, i.e. a node which has either S or E position, the new node gets this end value (S or E), i.e. its position. The old node gets a new position between the end value and its successor or predecessor depending on the its old position (we use the function position() in Algorithm 1 for this purpose). The new node becomes predecessor of the old node if it received position S, otherwise it becomes 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 (again using position()). 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, then the new node asks that node to create a virtual node. 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 between the position of the real node and the position of the virtual node.

Algorithm 1 VCP join algorithm

```
Require: One node must be pre-programmed as initial
    node, i.e. it gets position $S$; all nodes in the cord pe-
    riodically send hello messages
 1: if $NeighbourPosition = S$ then
 2:   $MyPosition \leftarrow S$
 3:   $Successor \leftarrow Neighbour$
 4:   $Predecessor \leftarrow NULL$
 5:   if $NeighbourSuccessor$ = NULL then
 6:     $NewNeighbourPosition \leftarrow E$
 7:   else
 8:     $NewNeighbourPosition \leftarrow$ position($S$,
        $NeighbourSuccessorPosition$)
 9:   end if
10:   SendUpdatePredecessor($Neighbour$,
      $NewNeighbourPosition$)
11: else if $NeighbourPosition = E$ then
12:   $MyPosition \leftarrow E$
13:   $Successor \leftarrow NULL$
14:   $Predecessor \leftarrow Neighbour$
15:   $NewNeighbourPosition \leftarrow$
      position($NeighbourPredecessorPosition$, $E$)
16:   SendUpdateSuccessor($Neighbour$,
      $NewNeighbourPosition$)
17: else if $Neighbour1$ is predecessor to $Neighbour2$
    then
18:   $MyPosition \leftarrow$ position($Neighbour1Position$,
      $Neighbour2Position$)
19:   $Predecessor \leftarrow Neighbour1$
20:   $Successor \leftarrow Neighbour2$
21:   SendUpdateSuccessor($Neighbour1$)
22:   SendUpdatePredecessor($Neighbour2$)
23: else
24:   CreateVirtualNode($Neighbour$)
25: end if
```

Figure 1 (left) shows the joining process of six nodes. The outer circle indicates the communication range of the newly joining node. Figure 1 (right) depicts the network after adding 15 nodes. In this example, a range of $[0, 1]$ for the virtual addresses is assumed. Note that when the fifth node joins the network, it finds two adjacent nodes (node 0.5 and 0.75). So it becomes the successor of the first node (0.5) and predecessor of the other node (0.75). The new address is a number between 0.5 and 0.75. However, when

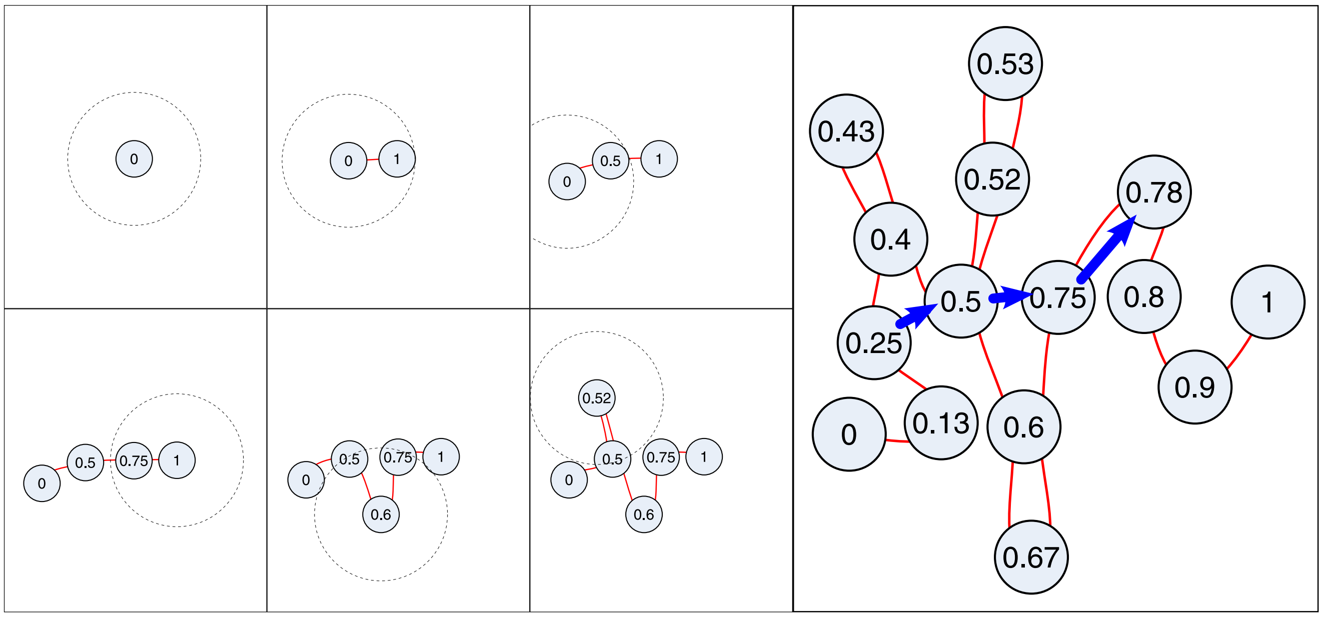


Figure 1. Basic join operation in VCP (left) and resulting routing paths (right)

the sixth node joins the network, nodes 0.5 and 0.75 are no longer adjacent. Thus, it asks node 0.5 to create a virtual node as shown in Figure 2. The arrow indicates the nodes are adjacent in the virtual cord.

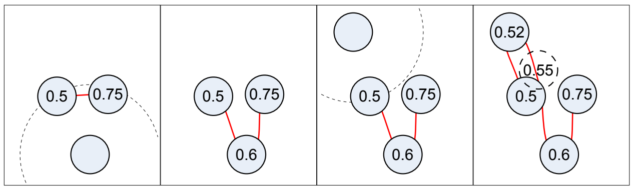


Figure 2. Operation of creating virtual node

It is clear that the joining of a new node only affects a small number of nodes in the vicinity 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.

3.2 Routing

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 lead to 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 1 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 (0.781) 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 as shown in Figure 1 (right). 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.

3.3 Node failures

We assume that node failures can be detected if a node (successor or predecessor) does no longer communicate. In our case, the periodic `hello` can be exploited for this operation. Moreover, the responsible nodes for the data items of the failed node are the predecessor and successor. Different treatments can be performed to overcome the problem of node failure depending on the situation.

1. If an end node fails (i.e., either S or E), then the successor or predecessor gets the end position.

2. If the successor and predecessor of the failed node can communicate directly (including its virtual nodes if ex-

isting), then it is easy to recuperate the cord by marking the predecessor and successor of the failed node as adjacent.

3. If the network gets partitioned, two virtual cords are created until a new node re-connects the network.

The effects of node failure can be diminished by replication on the successors and predecessors, which is also used in most DHT approaches. The advantage here is that the successors and predecessors are in the local vicinity. Thus, it will not demand too much communication overhead. So, if a node receives a packet that contains a request from its failed adjacent node then, by means of replication, the data is also available on this node locally. Similarly, a node can store the data item locally, if it received a data item that should be forwarded to the failed successor or predecessor and lies between its position and the position of the failed node.

In case of a packet that has to pass through the failed node to reach its destination, the routing algorithm has to be modified as follows. The next promising node is selected even if there is no progress. In order to prevent loops, nodes should be visited no more than one time. This can be done simply by adding the destination address of the packet in a special "no-path" list on each visited node. This procedure prevents loops and averts using this path again for the same destination that cannot be reached through this node. This list must be kept alive for some period of time. Another possible solution is to construct alternative paths to the m adjacent and physical neighbors on each node.

4 Performance Evaluation

For a first analysis of the VCP protocol, we implemented a simulation model of the protocol in OMNeT++ [22]. OMNeT++ is a discrete event based simulator free for academic use. In addition, there are some extensions such as the INET and the MF framework available released under GNU General Public License (GPL). In this section, we show preliminary simulation results. The results have been collected from several simulation experiments to investigate the impact of different parameters like network size and traffic load on the performance of our protocol.

4.1 Simulation Environment

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 Wireless LAN protocol. For the first set of experiments, the nodes are deployed either in a grid or randomly on a rectangular area. Each node can communicate only to direct neighbors that are aligned either horizontally or vertically but not on the diagonal. An example of a 25 node network is depicted in Figure 3. The figure includes the virtual relative positions and the virtual cord connecting all the nodes. We varied the network size from 25 to 225 hosts and adapted the plane size to keep the density constant. Moreover, we studied two different traffic patterns. After joining the network, each node uniformly selects a start time in the time interval $[0, 100)$ s. Then, in the first traffic scenario, we evaluated bursty traffic, i.e. all messages were sent with a uniformly distributed inter departure time in $[0, 1)$ s. Secondly, a constant packet stream was analyzed to compare the protocol behavior under artificial traffic conditions. For statistical correctness, each experiment was executed five times for different data item keys.

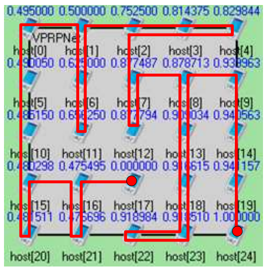


Figure 3. Simulation setup. Nodes are deployed either in a grid or randomly a rectangular area

For all communications, the complete network stack is simulated and wireless modules are configured to closely resemble IEEE 802.11b network cards transmitting at $2\,\mathrm{Mbit/s}$ 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 $50\,\mathrm{m}$.All simulation parameters used to parameterize the modules of the *INET Framework* are summarized in Table 1.

4.2 Quality of routing paths

In order to inspect the quality of the routing paths, we examined the stretch ratio, i.e. the ratio between length of the path traversed by VCP and the shortest path. For different network sizes, we measured the path length from all nodes to the upper left node.

Before showing the simulation results for different network sizes, we analytically evaluate the average path length. The average length of routing paths l_{avg} depends on the number of nodes n in the network. For simplified analysis, we still consider only nodes deployed in a grid network in a rectangular area. The maximum length of routing paths l_{max} in the grid occurs when routing between end

Table 1. 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
decider.bitrate	2 Mbit/s
decider.snirThreshold	4 dB
snrEval.bitrate	2 Mbit/s
snrEval.headerLength	192 bit
snrEval.snrThresholdLevel	3 dB
snrEval.thermalNoise	−110 dB
snrEval.sensitivity	−85 dB
snrEval.pathLossAlpha	2.5
snrEval.carrierFrequency	2.4 GHz
snrEval.transmitterPower	1 mW
channelcontrol.carrierFrequency	2.4 GHz
channelcontrol.pMax	2 mW
channelcontrol.sat	−85 dBm
channelcontrol.alpha	2.5

corners. To find a mathematical solution we inspect simple networks. For a network with only one node, l_{max} is 0, for four nodes it is 2, for nine nodes it is 4, and so on. Thus, l_{max} can be calculated recursively as shown in Equation 1. The close form solution is given in Equation 2.

$$l_{max}(1) = 0$$
$$l_{max}(n) = l_{max}(\sqrt{n} - 1) + 2 \quad (1)$$
$$l_{max} = 2\sqrt{n} - 2 \quad (2)$$

Of course the minimum number of hops l_{min} equals to 0, which means the node sends to itself. Because the grid is symmetrical, which means the number of nodes that have l_{max} is the same as the number of nodes that have l_{min} and the number of nodes that have $l_{max} - 1$ is the same as the number of nodes that have $l_{min}+1$, and so forth, the average length of routing paths l_{avg} can be calculated as:

$$\begin{aligned} l_{avg} &= \frac{l_{max} + l_{min}}{2} \\ &= \sqrt{n} - 1 \end{aligned} \quad (3)$$

Thus, in case of 25 nodes, the average path length $l_{avg} = 4$ and the maximum path length $l_{max} = 8$ (between end corners).

Figure 4 shows the measured stretch ratio in the simulations as we varied the network size from 25 to 225 nodes as stated before. The stretch ratio increases with the network size, however, this increase is reasonable and it stays below 25%. This low stretch level outlines the optimal path selection of VCP. For comparison, the stretch ratio of VRR [4] increased above 40% for network sizes larger than 200 nodes. This is because successors and predecessors, which are included in the routing table, can be far away from each other and, therefore, messages may visit many unnecessary nodes. This indicates that in VCP messages are traveling near optimal paths.

4.3 Influence of the network size

We performed a series of experiments to explore the effect of network size on the performance of VCP. Each node in the network sends packets to the same destination, which is equivalent to store the same data item collected from all the sensor nodes in the network. We varied the network size from 25 to 225 nodes.

Figure 5(a) shows the averaged results of the five runs for each size. It is clear the path length increases with the network size in a logarithmic manner. However, there are few nodes that used a path length larger than the shortest path to reach the destination. Taking a look on the average and mean path lengths, they are almost near around half of the worst case shortest path l_{max}, which is an indication of good path selection. Also more than 75% of the nodes have a path length smaller than l_{max}. On the other hand, the end-to-end delay is proportional to the path length as a result of only propagation delay because it is not necessary to queue packets. Moreover, the success ratio was 100 percent for all large network sizes.

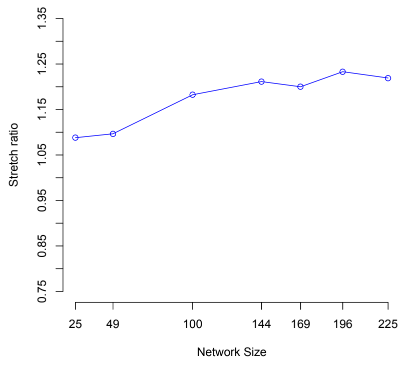
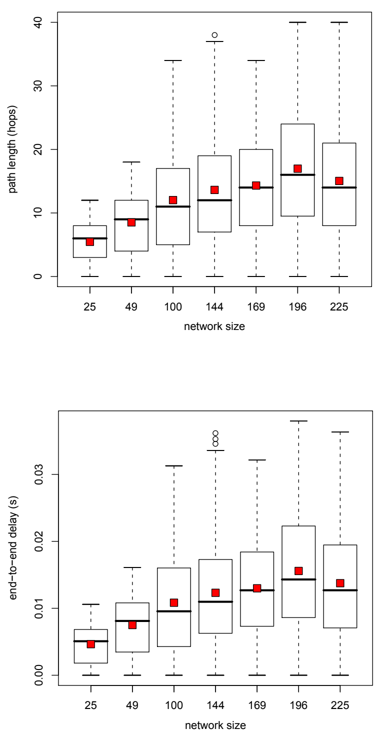
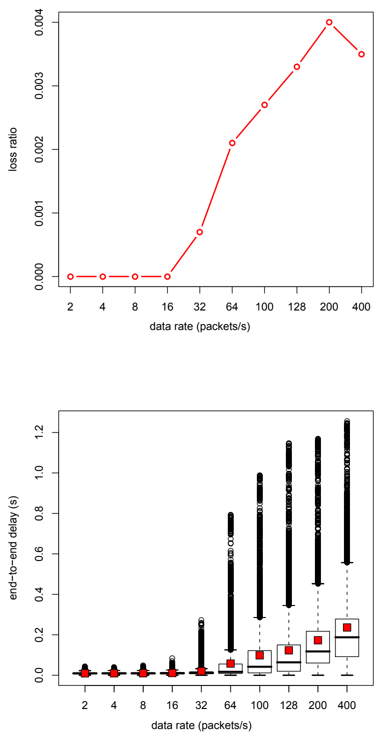


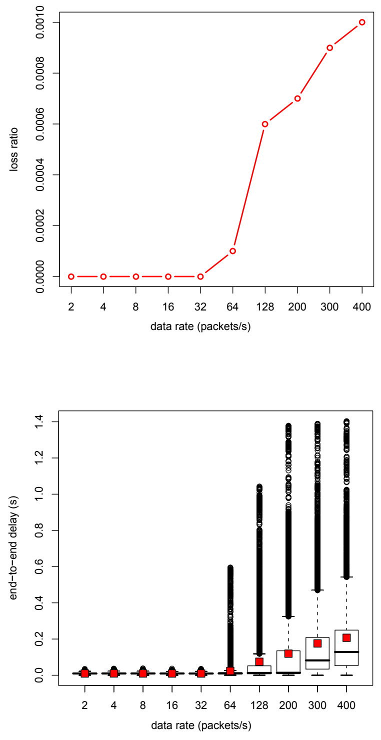
Figure 4. Stretch ratio for different network sizes



(a) Number of hops (top) and end-to-end delay (bottom) for varying network size



(b) Loss ratio (top) and end-to-end delay for bursty traffic behavior



(c) Loss ratio (top) and end-to-end delay for constant traffic rate

Figure 5. First performance evaluation results. Most figures are drawn as boxplots indicating the mean value and the quartiles. The small box shows the average value

4.4 Influence of the traffic load

To study the behavior of our protocol under varying traffic load, we kept the number of nodes in the network constant at 100 nodes and each node in the network sends 100 packets to the same destination. For the first experiments, the time between successive packets was randomly selected in the interval $[0, 0.5]$ seconds, which is equivalent to sending at least 2 packets per second. Afterwards, we decreased this time interval down to $[0, 0.005]$ seconds, which is equivalent to sending at least 400 packets per second. As shown in Figure 5(b) packets can be delayed at the MAC layer due to congestion. As a result the end-to-end delay is increased with increasing traffic load. Nevertheless, the delay is still in an acceptable range. For sending packets with rate below 16 packets per second, the effect of congestion is negligible. However, the delay reached a peak of $1.2\,\mathrm{s}$ when sending with rate of 400 packets per second compared to only $0.03\,\mathrm{s}$ in the other case. The effect of increasing traffic was not so big on the packet delivery rate. As can be seen from Figure 5(b), the loss ratio was below $0.4\,\%$, therefore the success rate is still above $99.6\,\%$.

We repeated the same experiment using a constant packet rate. Thus, we started sending packets every $0.5\,\mathrm{s}$ and decreased this duration down to $0.005\,\mathrm{s}$. As shown in Figure 5(c), the results are a little bit better. There was no impact of increasing traffic load until 32 packets per second and the mean delay was lower than before. However, the peak delay was slightly higher than sending packets within a random interval.

To explore the performance of VCP in random deployment, we performed several simulations using a network consisting of 200 nodes deployed randomly in a 600×120 plane, which is similar to scenarios described in [4] for VRR. The packet rate was set to one or two packets per second, which is equivalent to 200 or 400 CBR flows, respectively. In less than $20\,\mathrm{s}$, all the nodes joined the network. The average number of control messages sent by each node (excluding hello messages) was 10.54. Each node start sending a 100 byte packet to a random destination at a random time in the interval 50 to 230 seconds. All nodes stop transmission at time $950\,\mathrm{s}$. As shown in Table 2, the results are very promising. For example, the success rate is almost 100 % – even for high traffic load. In comparison, the success rate for VRR dropped to 60 % if either the network size increased to more than 200 nodes or if the traffic load doubled.

Table 2. Influence of the traffic load for random deployment

Measure	1 packet/s	2 packets/s
success rate	100%	99.95%
average delay	0.0068 s	0.0174 s
max delay	0.065 s	0.234 s
average hop count	5.65	5.79
max hop count	16	16

4.5 Comparison to standard ad hoc routing techniques

In order to provide a better understanding of the performance of VCP, we finally compared VCP to standard ad hoc routing techniques. This comparison outlines the usability of VCP especially in typical WSN scenarios where nodes are less mobile compared to MANETs.

Most recent proposals for routing protocols in both the WSN and the MANET domain have been evaluated in comparison with Ad Hoc on Demand Distance Vector (AODV). We decided to explicitly use DYMO for this comparison as it is the designated successor for AODV.

4.5.1 Scenario description

DYMO is the most recent reactive (on-demand) routing protocol, which is currently developed in the scope of the MANET working group of the Internet Engineering Task Force (IETF) [5]. DYMO builds upon experience with previous approaches to reactive routing, especially with the routing protocol AODV [13]. It aims at a somewhat simpler design, helping to lower the nodes' system requirements and simplify the protocol's implementation. DYMO retains proven mechanisms of previously explored routing protocols like the use of sequence numbers to enforce loop freedom. At the same time, DYMO provides enhanced features, such as covering possible MANET-Internet gatewaying scenarios and implementing path accumulation.

For this second set of experiments, we followed the experiments described for some recent DYMO studies [19]. We evaluated the performance for the following combinations of scenarios:

1. Nodes were arranged either to form a *grid* or in a completely *random* manner.

2. The playground size was adjusted so that the average distance between neighboring nodes corresponded to either *one hop* or *three hops* (according to the grid scenario).

3. The packet sink was either globally well identified, i.e. *node 0*, or *randomly* selected.

4. Nodes sent a new packet either following an exponential distribution with a mean value of *one second* or a mean value of *ten seconds*, or following a random, *bursty* pattern.

Primarily, we analyzed two parameters, the collision rate on the MAC layer as a measure for the congestion in the wireless network and the end-to-end delay as experienced from an application's perspective as a measure for the protocol efficiency. In all experiments, we placed 100 nodes in a rectangular area either on a grid or randomly. Also, we varied the node density and the traffic rate as described above.

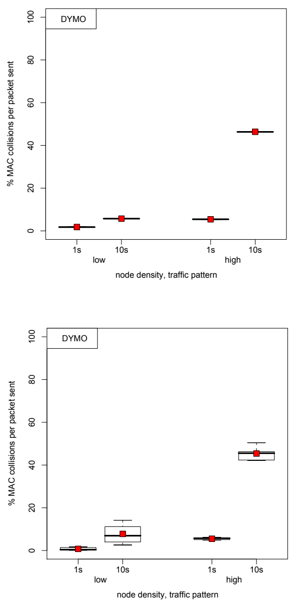
4.5.2 Collisions on MAC layer

In order to make sure that none of the effects discussed in later sections occurred due to message loss resulting from collisions in the overloaded shared medium, the first measures that we evaluated were the number of collisions and the number of successfully received transmissions, as observed by the MAC layer.

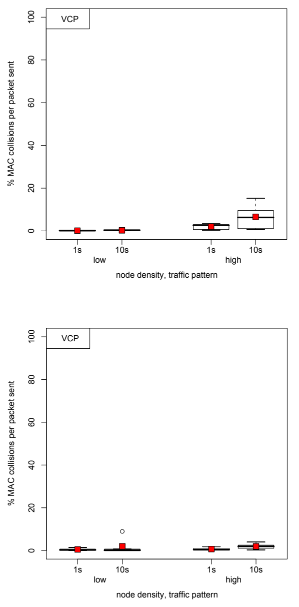
Figures 6(a) and 6(b) show the ratio of MAC collisions per link-layer packet sent for nodes deployed in a grid or randomly for DYMO and VCP, respectively. As can be seen, only in the case of high node density and a mean interval between two application-layer messages of 10 s, the collision ratio in the DYMO experiments exceeds 40 %. MAC collisions for other scenarios were much more seldom, reaching only insignificant ratios. VCP keeps the collision ratio in almost all experiments close to 0 %. In the case of grid deployment and one message every 10 s, the collision ratio for VCP exceeds 5 %. 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.

4.5.3 End-to-end delay

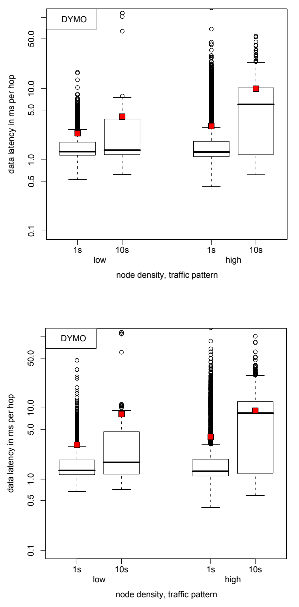
For the applications, one of the most important measures is the delay of messages from generation until storage to the destination node. Figures 6(c) and 6(d) show the results of



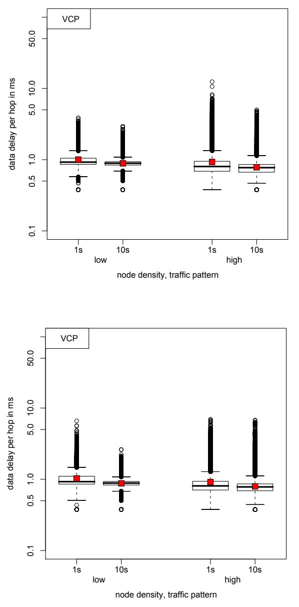
(a) Number of collisions for DYMO: grid deployment (top) vs. random deployment (bottom)



(b) Number of collisions for VCP: grid deployment (top) vs. random deployment (bottom)



(c) Data delay per hop for DYMO: grid deployment (top) vs. random deployment (bottom)



(d) Data delay per hop for VCP: grid deployment (top) vs. random deployment (bottom)

Figure 6. Simulation results for the comparison between VCP and DYMOs. Again, the figures are drawn as boxplots indicating the mean value and the quartiles. The small box shows the average value

our simulation experiments. In order to produce comparable results, both figures depict the mean delay per hop, i.e. the end-to-end latency divided by the number of hops for this particular transmission.

Without looking at particular numbers, which mainly depend on the specific network scenario, we need to discuss a number of effects that became visible in these figures. Considering the DYMO measurements first and comparing the traffic scenarios with one and ten seconds inter-packet time, we see that the average per-hop delay increases. 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 packet transmission delay. Thanks to the route response messages created on behalf of the destination, the median is equal in both cases.

Looking at the VCP measurements, we see that the measured per hop delay varies much less compared to DYMO. The first reason is the consequent application of the virtual cord to locate destination nodes and to forward packets towards this destination. Furthermore, and this is confirmed by the per hop delay that is a bit smaller compared to the DYMO scenario, VCP is able to rely on better path information. The network stretch analyzed in Section 4.2 also supports this observation.

5 Conclusion

VCP is a new DHT-like protocol that is based on virtual relative positions for routing. It offers traditional DHT services as well as efficient packet routing. VCP has the following attractive characteristics:

- The successors and predecessors are in the vicinity, which reduces the communication load when nodes join and leave the network.
- The exact physical location is not required, instead we use an easy to be obtained relative position.
- VCP is scalable because it needs only information about direct neighbors for routing.
- There is no problem with dead-ends because the cord structure always ensures reachability of the destination. Additionally, greedy forwarding helps to find good paths.
- It is easy to be implemented using typical MAC protocols.

In this paper, we studied the performance of VCP for different scenarios. We showed that the path length is almost optimal and greedy forwarding guarantees packet delivery. Additionally, there is no extra delay needed before packet forwarding. Therefore, VCP represents a promising technique, however more investigation is needed to explore the performance of the protocol in different environments.

Future work on VCP includes further research on handling of node failures and rejoining partitioned networks. Additionally, we are working on an implementation VCP on sensor nodes in our lab environment.

Acknowledgemnts

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

References

[1] K. Akkaya and M. Younis. A Survey of Routing Protocols in Wireless Sensor Networks. *Elsevier Ad Hoc Networks*, 3(3):325–349, 2005.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Elsevier Computer Networks*, 38:393–422, 2002.

[3] 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*, pages 443–444, Hong Kong, China, May 2008. ACM.

[4] 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.

[5] I. Chakeres and C. Perkins. Dynamic MANET On-Demand (DYMO) Routing. Internet-Draft (work in progress) draft-ietf-manet-dymo-10.txt, July 2007.

[6] D. Culler, D. Estrin, and M. B. Srivastava. Overview of Sensor Networks. *Computer*, 37(8):41–49, August 2004.

[7] D. Estrin, D. Culler, K. Pister, and G. S. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, January 2002.

[8] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imielinski and H. F. Korth, editors, *Mobile Computing*, volume 353, pages 152–181. Kluwer Academic Publishers, 1996.

[9] 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)*, pages 243–254, Boston, Massachusetts, USA, 2000.

[10] 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)*, pages 377–386, Vancouver, Canada, October 2006. IEEE.

[11] M. Mauve and J. Widmer. A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. *IEEE Network*, 15(6):30–39, 2001.

[12] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Computer Communications Review*, pages 234–244, 1994.

[13] C. E. Perkins and E. M. Royer. Ad hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999.

[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, USA, September 2003.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM 2001*, pages 161–172, San Diego, CA, USA, 2001.

[16] 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*, 8(4):427–442, August 2003.

[17] 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.

[18] 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)*, pages 329–350, Heidelberg, Germany, November 2001.

[19] C. Sommer, I. Dietrich, and F. Dressler. A Simulation Model of DYMO for Ad Hoc Routing in OMNeT++. In *1st ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008): 1st ACM/ICST International Workshop on OMNeT++ (OMNeT++ 2008)*, Marseille, France, March 2008. ACM.

[20] R. Steinmetz and K. Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume LNCS 3485. Springer, 2005.

[21] 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)*, 11(1):17–32, February 2003.

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

[23] 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*, 6(9):1075–1089, September 2007.