BeeAdHoc: An Energy Efficient Routing Algorithm for Mobile Ad Hoc Networks Inspired by Bee Behavior

Horst. F. Wedde, Muddassar Farooq, Thorsten Pannenbaecker, Bjoern Vogel, Christian Mueller, Johannes Meth and Rene Jeruschkat
Informatik III
University of Dortmund
44221, Germany

ABSTRACT

In this paper we present *BeeAdHoc*, a new routing algorithm for energy efficient routing in mobile ad hoc networks. The algorithm is inspired by the foraging principles of honey bees. The algorithm mainly utilizes two types of agents, scouts and foragers, for doing routing in mobile ad hoc networks. *BeeAdHoc* is a reactive source routing algorithm and it consumes less energy as compared to existing state-of-the-art routing algorithms because it utilizes less control packets to do routing. The results of our extensive simulation experiments show that *BeeAdHoc* consumes significantly less energy as compared to *DSR*, *AODV*, and *DSDV*, which are state-of-the-art routing algorithms, without making any compromise on traditional performance metrics (packet delivery ratio, delay and throughput).

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: [Distributed networks, Wireless communication]; C.2.2 [**Network Protocols**]: [Protocol architecture, Routing protocols]

General Terms

Algorithms, Design, Theory

Keywords

Swarm Intelligence, Mobile Ad Hoc Networks, Self-Organization, Energy Efficient Routing

1. INTRODUCTION

Mobile Ad Hoc Networks (MANETs) is becoming an active area of research [13]. All nodes in such networks take two roles: producer/consumer of data packet streams, and routers for data packets destined for the other nodes. The most important challenges in MANETs are: mobility and limited battery capacity of the nodes. Mobility of nodes results in continuously evolving new topologies and the routing algorithms have to adapt the routes according to these changes. The limited battery capacity poses yet another challenge for the routing algorithms: to distribute the packets on multiple paths in such a manner that the battery of different nodes deplete at an equal rate, as a result, the life time of the network could be increased [16] [8]. The metrics for energy efficient routing are also introduced in [8] and it is evident that an energy aware routing algorithm is expected to degrade the traditional performance metrics of a routing algorithm i.e. throughput and packet delay [11]. The real dilemma in MANETs is: *how to to design a routing algorithm which is not only energy efficient but also provides the same performance as that of the existing state-of-the-art algorithms.*

The routing algorithms for MANETs can be broadly classified as proactive algorithms or reactive algorithms. Proactive algorithms periodically launch control packets which collect the new network state and update the routing tables accordingly. On the other hand, reactive algorithms find routes on-demand only. Reactive algorithms look more promising from the perspective of energy consumption in MANETs. Each category of the above-mentioned algorithms is further classified based on the routing scheme as source routing or next hop routing algorithms. In source routing algorithms, the complete route to a destination, which consists of a sequence of nodes leading to the destination, is added as a header to each data packet. In next hop routing a packet is forwarded to a neighbor node, based on the information in the routing table, lying on the route leading toward the destination.

DSR (Dynamic Source Routing) is a reactive source routing algorithm [7] while *AODV* (Ad-Hoc On-demand Distance Vector Routing) is a reactive next hop routing algorithm [9]. *DSDV* (Dynamic Destination-Sequenced Distance-Vector) is a proactive next hop routing algorithm [10]. *AODV* and *DSR* are considered to be state-of-the-art routing algorithms developed by the networking community for MANETs. However, all of these algorithms are not designed for energy efficient routing. Feeney reported in [4] the energy consumption behavior of *DSR* and *AODV* and concluded that the algorithms are not optimized for energy consumption. She believes that metrics for energy efficient routing are completely different than traditional performance metrics (packet delivery ratio and packet delay). The energy aware algorithms

reported in [16] [8] also use *DSR* or *AODV* as an underlying route discovery and maintenance mechanism, and then use energy as a cost metric for routing. To our knowledge, little attention has been paid in developing an energy efficient routing algorithm from scratch with one primary objective: optimizing energy consumption without any degradation of the performance.
In this paper we present a new MANET routing algorithm, *BeeAdHoc*, which is primarily designed for energy efficient routing. The algorithm proposes a solution to the energy-performance dilemma. *BeeAdHoc* achieves similar/better performance as that of *DSR*, *AODV*, *DSDV* but consumes significantly less energy as compared to these state-of-the-art algorithms. The algorithm achieves the objectives by sending less control packets and distributing data packets on multiple paths. Such a behavior is made possible by taking inspirations from the foraging behavior of honey bees which is discussed in [17][15].

1.1 Related Work

The first algorithm which presented a detailed scheme for MANET routing based on ant colony principles is *ARA* [6]. The algorithm has its roots in *ABC* [14] and *AntNet* [2] routing algorithms for fixed networks, which are inspired by the pheromone laying behavior of ant colonies. The algorithm floods ants to the destinations while establishing reverse links to the source nodes of the ants. Nodes launch ant agents in a reactive manner in order to limit the overhead caused by them. *AntHocNet* has been recently proposed in [3] which is a hybrid algorithm having both reactive and proactive components. The algorithm tries to keep most of the features of the original *AntNet* and shows promising results in the simulation tests over *AODV*. *Termite* is another MANET routing algorithm inspired from termite behavior [12]. In this algorithm, no special agents are needed for updating the routing tables rather data packets are delegated this task. Each data packet follows the pheromone for its destination and leaves the pheromone for its source. Pheromone is a quality metric representing the goodness of a link. The data packets are biased toward the paths that have higher pheromone values. An exponential pheromone decay is introduced as a mean of a negative feedback to prevent old routes from remaining in the routing tables.
Recently, Wedde, Farooq and Zhang have proposed a novel routing algorithm for fixed networks which is inspired by foraging principles of honey bees [18]. The algorithm is simple but delivers the same/better performance as that of *AntNet* [2]. The success of *BeeHive* motivated us to take the foraging principles of bees as an inspiration for designing our new routing algorithm, *BeeAdHoc*, for MANETs. A honey bee colony has many features that are desirable in MANETs: efficient allocation of foraging force to multiple food sources, different type of foragers for each commodity, foragers evaluate the quality of visited food sources and then recruit optimum number of foragers for their food source by dancing on a dance floor inside the hive, no central control, foragers try to optimize the energetic efficiency of nectar collection and foragers take decisions without any global knowledge of the environment. The principles discussed in [17] [15] form the basis for our *BeeAdHoc* algorithm. We skip the details for the sake of brevity.
The rest of the paper is organized as follows. In Section 2 we will introduce our bee agent model and on its basis we will describe our routing algorithm, *BeeAdHoc*, in Section 3. We will first explain the complete experimental framework in Section 4 and then discuss the results obtained from the extensive simulations. Finally, we conclude the paper with an outlook to our future research.

2. BEE AGENT MODEL

Our *Bee Agent Model* is inspired from the foraging principles of a honey bee colony. Our agent model consists of four types of agents: packers, scouts, foragers, and swarms. In the rest of the paper we use the term scout for scout agent, forager for forager agent etc. until otherwise specified.

2.1 Packers

Packers mimic the task of a food-storer bee. They always reside inside the node, receive and store the data packets from the transport layer. Their major job is to find a forager for their data packet and they die once they hand over it to the foragers.

2.2 Scouts

Scouts discover new routes from their launching node to their destination node. A scout is transmitted using the broadcasting principle to all the neighbors of a node with an expanding time to live timer (TTL), which controls the number of times a scout could be re-broadcasted. Each scout is uniquely identified with a key based on its id and source node. Once a scout reaches at the destination then it starts the backward journey on the same route that it followed to the destination. A destination node sends back all of the received scouts to ensure discovery of multiple paths. Once a scout returns to its source node then it recruits the foragers for its route by using the metaphor of dance (as scout bees do in Nature). A dance is abstracted as the number of clones that could be made of a scout (equivalent of recruiting forager bees in Nature).

2.3 Foragers

Foragers are the main workers in our *BeeAdHoc* algorithm. They receive the data packets from packers and then transport them to their destination. Each forager has a special type: delay or lifetime. The delay foragers collect the delay information from the network while the lifetime foragers collect the remaining battery capacity of the nodes that they visit. The first ones try to route packets along a path that has a minimum delay while the second ones try to route packets in such a manner that the life time of the network is increased.
A forager gets the complete route, in the form of a sequence of nodes leading to a destination, from a scout or another forager. A forager follows point-to-point mode of transmission till the destination and collects the information about the network state depending upon its type. Once a forager reaches at the destination then it remains there until it could be piggybacked on the network traffic from the destination node to its source node. This optimization reduces the overhead of control packets and hence saves energy as well. A reliable transport protocol, like TCP, acknowledges the received packets and *BeeAdHoc* piggybacks in the acknowledgments the waiting foragers. The foragers also use the metaphor of dance once they return to their source node in a similar way as scouts do.

2.4 Swarms

An unreliable transport protocol, like UDP, sends no explicit acknowledgments for the received data packets. Such a protocol may not be able to provide an implicit return path to a waiting forager and therefore it could never return to its source node. Consequently, its source node might run out of the foragers and unable to continue the communication. We solved this problem with the help of swarms. Once the difference between the incoming foragers from a certain node i and the outgoing foragers to the same node i reaches above a threshold value at a node j then the node j launches a swarm of foragers to the node i. We put one forager in the header of the swarm while the others are put in the payload part of the swarm. Once the swarm arrives at the node i then the foragers are extracted from the payload part and they are stored like they would have arrived at the node in a normal fashion.

3. ARCHITECTURE OF BEEADHOC

Each node in MANET has a hive, which consists of three parts: packing floor, entrance and dance floor. The structure of the hive is shown in Figure 1. The entrance is an interface to MAC (Medium Access Control) layer while packing floor is an interface to transport layer. All packets depart/enter the hive through the entrance. The dance floor contains the foragers (routing information) for routing of data packets originated at the node.

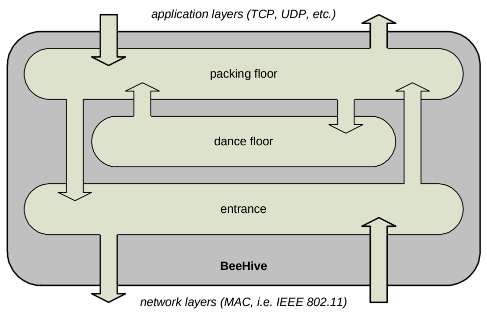

application layers (TCP, UDP, etc.)
packing floor
dance floor
entrance
BeeHive
network layers (MAC, i.e. IEEE 802.11)

Figure 1: Overview of the BeeAdHoc architecture

3.1 Packing Floor

The packing floor is an interface to higher level transport layer like TCP or UDP. Once a data packet arrives from the transport layer, a packer is created in the packing floor which stores the data packet. After that the packer tries to locate a suitable forager for the data packet from dance floor. If it finds one then it hand overs the data packet to the forager and dies. Otherwise, it waits for a time (may be a returning forager is on its way toward the current hive) and if no forager arrives within this time, then it launches a scout which is responsible for discovering new routes to the destination of the data packet. Figure 2 explains the series of actions performed at a packing floor.

3.2 Entrance

The functions performed in entrance are shown in Figure 3. The entrance is an interface to lower level MAC layer. The entrance handles all incoming/outgoing packets. A scout received at the entrance is broadcasted further if its time to live (TTL) timer has not expired or if it has not

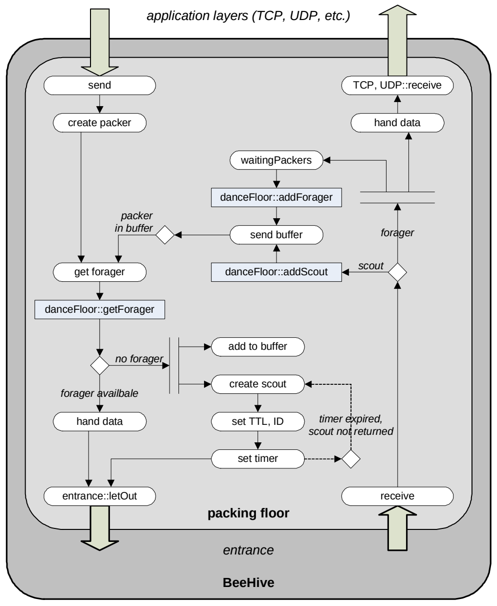


Figure 2: The packing floor

arrived at the destination. The information about the id of the scout and its source node is stored in a table. If another replica of an already received scout arrives at an entrance of a hive then the new replica is killed here. If a forager with a same destination as that of the scout already exists in the dance floor then the route to the destination is given to the scout by appending the route in the forager to its current route.
If the current node is the destination of a forager then it is forwarded to the packing floor else it is directly forwarded to the MAC interface of the next hop node.

3.3 The Dance Floor

The dance floor is the heart of the hive because it takes important routing decisions. Once a forager returns after its journey it recruits new foragers by dancing according to the quality of path that it traversed. However, the quality metric for each forager is different. As mentioned before, a lifetime forager evaluates the quality of its route based on the average remaining battery capacity of the nodes on its route. A lifetime forager might allow itself to be cloned many times (forager bees in Nature dance enthusiastically and consequently recruit more foragers) in two scenarios: one, the nodes on the route have enough remaining battery capacity (good route), two, if large number of packers are waiting for it even though its route might be having nodes with little battery capacity. In second case, it is sensible to send the packets through less good routes as well. On the other hand, if none of the packers are waiting then a forager with a very good route might not dance because its colleagues are doing a nice job in transporting the data packets. This concept is directly borrowed from the behavior of scout/forager bees in Nature, and it helps in regulating the number of foragers for each route.
The dance floor also sends a matching forager to the packing floor in response to a request from a packer. The foragers whose life time has expired are not considered in the matching function. If multiple foragers match the criteria then a

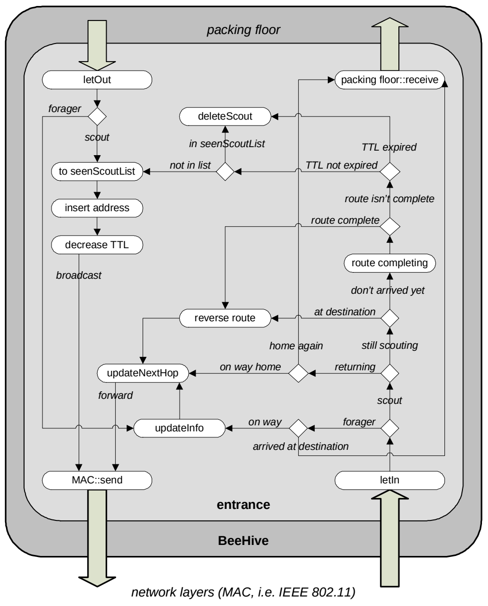


Figure 3: The entrance

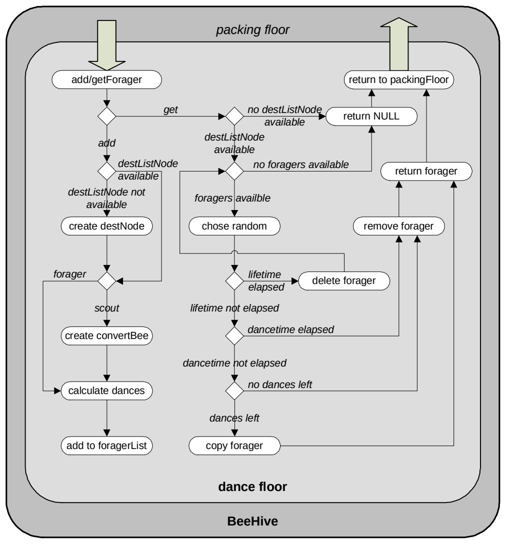


Figure 4: The dance floor

forager is stochastically chosen among them. This helps in distributing the packets over multiple paths that serves two purposes: avoid congestion under high loads and battery of different nodes are depleted at an equal rate. A clone of the selected forager is sent to the packing floor and the original forager is stored in the dance floor after reducing its dance number. If the dance number is zero then the original forager is sent to the packing floor and its entry is deleted from the dance floor. Using the above-mentioned principle, young foragers, which represent latest routes and which are likely to remain valid in future, are favored over the older ones.
If the last forager for a destination leaves a hive then the hive does not have a route to the destination. We believe that if a route to the destination exists then soon a forager would be returning toward the hive and if no forager comes within a certain time then the node has probably lost route to the destination node. This mechanism eliminates the need for explicitly monitoring the validity of the routes by using special hello packets and then informing other nodes through Route Error Messages (RERR). This results in transmitting less control packets, as a result, the algorithm has less energy expenditure. Figure 4 explains in detail the actions taken at a dance floor.

4. SIMULATION FRAMEWORK

We evaluated the performance of our algorithm *BeeAdHoc* using mobility enhancements made to ns-2 simulator by the authors of [1]. The authors also evaluated the performance of different state-of-the-art algorithms like *AODV*, *DSR*, *DSDV* and *TORA* in their work. Our test scenarios are derived from the base scenario used in [1]. We use the same implementations of *DSR*, *AODV* and *DSDV*, which are distributed with the ns-2 simulator to factor out any implementation related error in the algorithms.
The scenario consists of 50 nodes which are moving in a rectangular area of $2400 \times 800m^2$. The rectangular area ensures that longer paths exist between the nodes as compared to a square one provided the node density (nodes per unit area) remain the same. The nodes move according to the "random waypoint" model [7]: each node randomly selects a destination point and then moves to that point with a certain randomly selected speed. Once the node arrives at the destination point then it stops there for a certain pause time and then again randomly selects a new destination point and moves toward it with a new speed. The speed is selected from a uniform distribution between a minimum speed of 1 m/s (walking speed) and the maximum speed of 20 m/s (car speed within cities). All the nodes generate a constant bit rate (CBR) peer-to-peer data traffic with x packets/s. The size of a data packet is kept constant at 512 bits. We use the same models of physical and MAC layers as the authors of [1] did. The reported results are an average over five independent different runs to factor out any stochastic elements in the environment or in the algorithms. The simulation time for the algorithms is set to 1000 seconds.

4.1 Metrics

We now define the metrics which we used in the comparison of the algorithms.

- **Energy per user data.** *The total energy consumed, including the energy consumed by the control packets, to transport one kilobyte of data to its destination.* This metric is minimum when the same number of bytes could be delivered at the destinations in less hops and with small number of control packets. We used the model presented in [5] to estimate the send/receive energy of broadcasting or point-to-point mode of transmitting packets. This metric is also referred to as energy expenditure in rest of the paper.

- **Success rate.** *The ratio between the number of packets successfully received by the application layer of a destination node and the number of packets originated at the application layer of each node for that destination.* This parameter is also referred to as packet delivery ratio in rest of the paper.

- **Delay.** *The difference between the time once the packet is received by the application layer of a destination node and the time when the packet was originated at the application layer of a source node.* This definition takes care of the time that a packet has to wait at the source node while the route to its destination is to be found (reactive wait time). We always report the 100th percentile of the delays distribution because it provides an insight on the spread of the delays which is an important criterion for quality of service (QoS) applications, in which all packets should arrive at the destination within an acceptable variance from the mean.

- **Throughput.** *If y number of bits are delivered within t time at a node then the throughput at the node could be defined as* $\frac{y}{t}$. This definition assigns a higher throughput value to an algorithm that delivers the same number of y bits in a smaller time. This definition of throughput implicitly strikes a good balance between the number of packets delivered at a node and their delays.

- **Network life.** *The average remaining battery capacity of the nodes.* A higher value means less depletion of the batteries and hence is a desirable property of any routing algorithm.

4.2 Node Mobility Behavior

The purpose of the experiments was to study the behavior of the algorithms by varying the speed of the nodes. Higher speeds reduce the stability of a network topology, as a result, an algorithm has to adapt itself with the changes in topologies. In these experiments the packet rate was 10 packets/s (CBR source) and the pause time was 60 seconds. Figure 5 shows the effect of mobility on different metrics. The packet delivery ratio (see Figure 5(a)) reduces with the increasing speed but *BeeAdHoc* is able to deliver approximately the same number of packets as that of *DSR*, the best performing algorithm. However, *BeeAdHoc* has a significantly smaller delay (see Figure 5(b)). Consequently, *BeeAdHoc* is able to maintain higher throughput (see Figure 5(c)) as compared to all other algorithms. Please remember that our definition of throughput favors one algorithm over the others if it is able to deliver the same number of packets but with smaller delays.
We investigated the problem of higher packet delays of *DSR* by looking at 80th, 90th, 95th and 100th percentile of the delays distribution (see Table 1). It is evident from Table 1 that *BeeAdHoc* is able to deliver majority of the packets with in an acceptable deviation from mean while *DSR* delivers about 5% of packets with quite large delays, as a result, 100th percentile of the delays distribution is significantly larger than that of the other algorithms. We further looked into this problem and it appeared that the most important contributor for higher delay is the packet salvaging technique used in *DSR*. Once a node finds out that the next hop in the header is down then it looks at its routing table and if it finds a route to the destination in it then it replaces the remaining part of the header with this route. However, by the time, the packet arrives at the next node, the route again needs to be repaired. Consequently, a packet keeps on taking hops until it arrives at the destination. The basic behavior of MANETs could then be summarized as follows: *if a node finds that the next hop to a destination is no more available then it should not try to repair the route with the old information in its routing table because there is a high probability that this old route would be no more valid as well.* Therefore, *BeeAdHoc* simply deletes the packet if it finds that the next hop is down. It is clear from Figure 5(a) that this simple approach results, at maximum, in loss of about 0.3% in packet delivery ratio.

	BeeAdHoc	*DSR*	*AODV*	*DSDV*
80th percentile	105.64	167.73	156.85	117.89
90th percentile	153.84	278.58	220.52	176.01
95th percentile	191.36	396.29	269.31	223.76
100th percentile	280.97	969.39	387.96	372.55

Table 1: Different Percentile of the delays distribution for node speed of 1-5 m/s

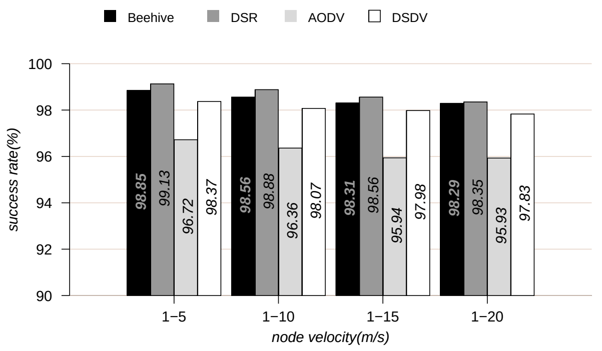
The simplicity of *BeeAdHoc*, which results because of its simpler architecture and using smaller number of control packets (see Table 2, please note that all of the 50 nodes are transmitting packets), pays off once we look at its energy expenditure (see Figure 5(d)) in transporting the packets from their source to their destination. *BeeAdHoc* employs a simple bee behavior to monitor the validity of the routes by controlling the number of foragers, their dance and age parameter, rather than explicitly using hello/RERR messages. This results in the smallest amount of energy expenditure for *BeeAdHoc* (see Figure 5(d)).
Finally, Table 3 shows the average remaining battery capacity (%) of the nodes in the network at the end of the above-mentioned simulations. *BeeAdHoc* has higher remaining battery capacity under all of the circumstances. The battery level of *BeeAdHoc* is better because it tries to spread the data packets over different routes rather than always sending them on the best routes. Different routes could be established to the destination nodes at higher node speeds, as a result, data packets are routed through different nodes and this explains the increasing network life behavior of *BeeAdHoc*, with an increase in the speed of the nodes. *AODV* and *DSR* utilize significantly larger number of control packets at higher nodes speed (see Table 2) therefore the batteries of the nodes are almost completely depleted.

Node Mobility	*BeeAdHoc*	*DSR*	*AODV*	*DSDV*
1-5 m/s	83095	122313	592454	165454
1-10 m/s	93895	224335	716235	211220
1-15 m/s	99240	310119	836058	240234
1-20 m/s	103119	396885	731279	253000

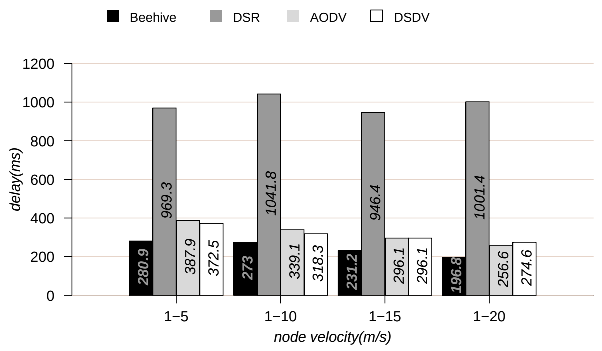
Table 2: Total number of control packets sent

4.3 Congestion Control Behavior

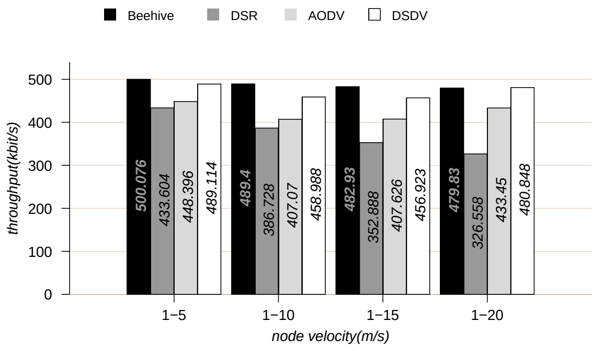
The purpose of these set of experiments was to investigate the congestion control behavior of the algorithms. The node's speed was chosen in the range 1-20 m/s and the packet send rate was gradually increased from 10 packets/s to 100 packets/s and the other parameters remain the same as in the previous experiments. All algorithms are able to cope up with an increased load (see Figure 6(a)), however, the performance of *AODV* is the worst. The throughput of



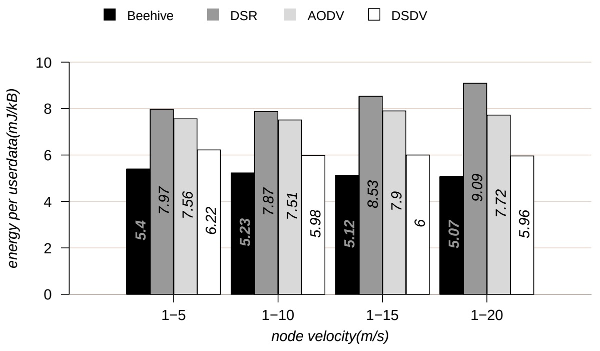
(a) Packet delivery ratio



(b) Packet delay



(c) Throughput



(d) Energy Expenditure

Figure 5: Effect of varying the speed of the nodes

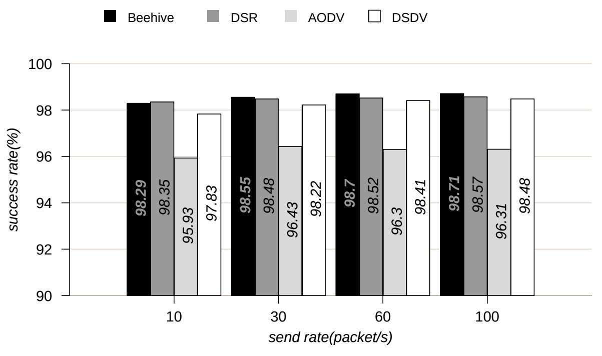
Node Mobility	*BeeAdHoc*	*DSR*	*AODV*	*DSDV*
1-5 m/s	6.2	1.4	1.8	2.4
1-10 m/s	5.5	1.3	2.4	2.6
1-15 m/s	6.1	1.0	1.6	2.6
1-20 m/s	11.4	1.1	1.4	3.3

Table 3: Effect of varying speed on Network life

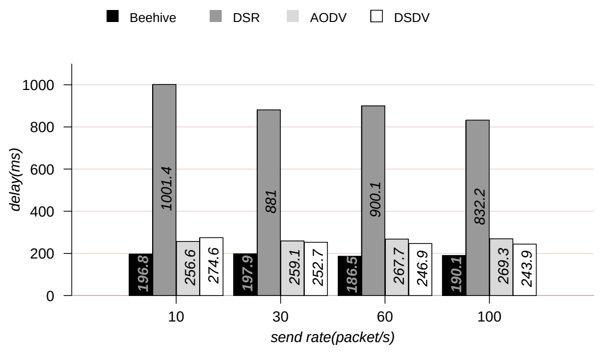
the algorithms increased with an increase in the send rate of packets (see Figure 6(c)). No significant queue delays were experienced even with a sending rate of 100 packets/s, as a result, y bits took approximately the same time t as in the previous experiments. According to our definition of throughput (see Section 4.1), it should approximately remain the same even though ten time more packets are delivered at the destinations. *BeeAdHoc* has the highest throughput value, because it provides a good compromise between packet delivery ratio and packet delay.
Figure 6(b) shows that the packet delay decreases with an increase in the load, which at first, appears to be counter intuitive. But this could be easily explained by looking at the definition of packet delay (see Section 4.1). The route discovery time, which is an overhead of discovering new routes, is now shared by more data packets. The energy expenditure of the algorithms also decrease with an increase in the send rate of packets because more data packets are delivered at their destination with the same number of control packets. However, *BeeAdHoc* has the lowest energy expenditure.

4.4 Varying Pause Time

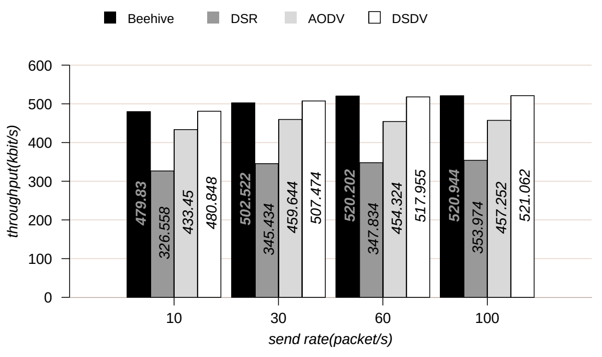
Another challenge in MANETS is to study the effect of pause time on the performance of the algorithms. Smaller pause time means that the nodes will stop for smaller times and as a result, the routes will never be stable. In these experiments we kept the maximum speed at 20 m/s and packet rate at 100 packets/s and all other parameters remained the same as in the previous experiments. We show the results for 1, 30 and 60 seconds of pause time, because according to the authors of [1] any increase in pause time after 60 seconds just improves the performance of the algorithms. Figure 7 shows the results for these experiments. Packet delivery ratio for *BeeAdHoc*, *DSR* and *DSDV* remains approximately the same with a decrease in pause time, however, the $AODV$ delivers about 2% less packets at a pause time of 1 sec as compared to a pause time of 60 seconds. The significant degradation in packet delay is experienced by all of the algorithms with a decrease in the pause time; *DSR* being the worst effected. All of the algorithms show an increase in the range of 300% to 600% in the delay. One could see from Figure 7(b) that $AODV$ has the smallest delay but then it delivers less packets as well (see Figure 7(a)). The routes are unstable at small pause times, as a result, packets, both on the average and in worst case scenarios, take more time to reach their destination. Consequently, throughput metric suffers as well (see Figure 7(c)) but *BeeAdHoc* manages to maintain higher throughput in all of the scenarios.
Energy expenditure of *BeeAdHoc*, as expected, is the lowest



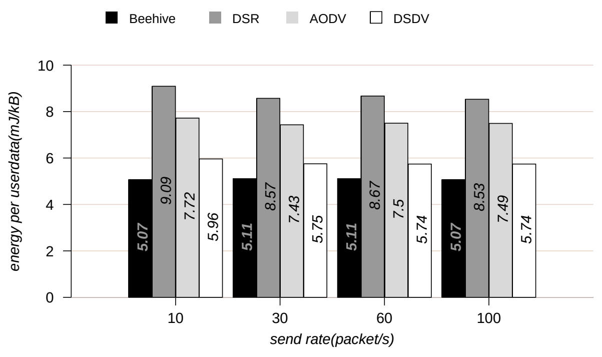
(a) Packet delivery ratio



(b) Packet delay



(c) Throughput



(d) Energy expenditure

Figure 6: Effect of varying packet send rates

among all algorithms in all of the scenarios. The reason is twofold: one, smaller number of control packets sent, and two, delivering more packets with smaller number of hops.

5. CONCLUSION

In this paper we presented a new routing algorithm for MANET which is inspired by the honey bee behavior. The algorithm is simple and mainly needs two types of messages for routing: scouts, which on-demand discover new routes to the destinations and forgers, which transport data packets and simultaneously evaluate the quality of the discovered routes. This simplicity results in substantially smaller number of control packets sent, as a result, the algorithm is energy efficient. We have verified through extensive simulations, which represent a wide spectrum of network conditions, that *BeeAdHoc* delivers the same/better performance as that of the state-of-the-art algorithms but at a significantly smaller energy expenditure.

We have already started our research in implementing *BeeAdHoc* in the network stack of the Linux kernel. We will then test and evaluate the algorithm on a mobile network of laptops. This effort is part of our *Natural Engineering* approach in which we want to develop engineering solutions for the real world problems under the resources of constraints (cost, labor, time etc.). We are also modifying *BeeAdHoc* so that it could scale to at least 1000 nodes MANETs. These enhancements will be the subject of our forthcoming publications.

Contact information. The email addresses of the authors are (wedde, farooq)@ls3.cs.uni-dortmund.de, thorsten.pannenbaecker@uni-dortmund.de, bjoernvogel@gmx.de, christian.mueller@uni-dortmund.de, J.Meth@LANdata.de and jeruschkat@web.de respectively.

6. REFERENCES

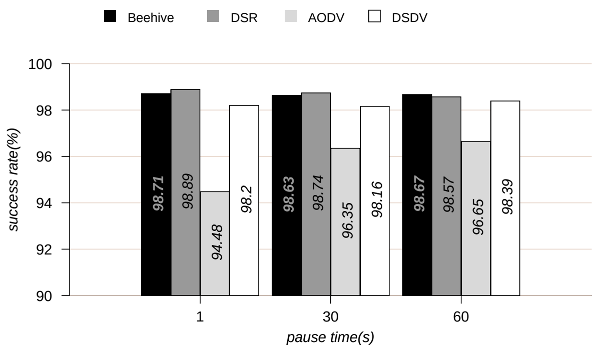
[1] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of Fourth ACM/IEEE Conference on Mobile Computing and Networking (MobiCom)*, pages 85–97, 1998.

[2] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communication networks. *Journal of Artificial Intelligence*, 9:317–365, December 1998.

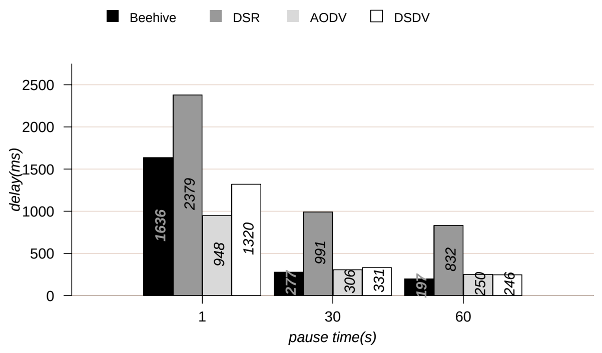
[3] Gianni Di Caro, Frederick Ducatelle, and Luca Maria Gambardella. AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In *Proceedings of Parallel Problem Solving from Nature (PPSN) VIII, LNCS 3242*. Springer-Verlag, 2004.

[4] Laura Marie Feeney. An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks. *Mobile Networks and Applications*, 6(3):239–249, 2001.

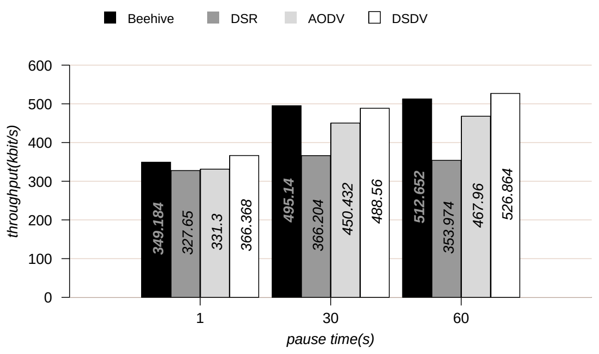
[5] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface



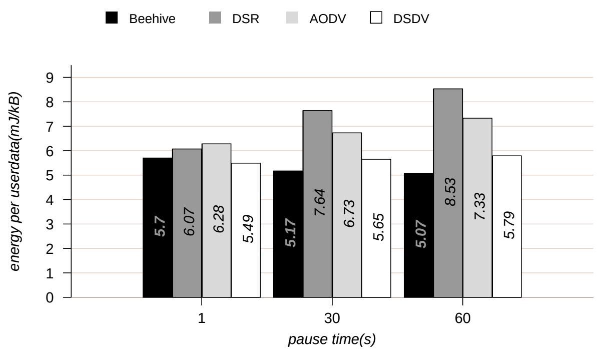
(a) Packet delivery ratio



(b) Packet delay



(c) Throughput



(d) Energy expenditure

Figure 7: Effect of varying pause time

in an ad hoc networking environment. In *Proceedings of IEEE INFOCOM*, 2001.

[6] M. Genes, U.Sorges, and I.Bouazizi. ARA – the ant-colony based routing algorithm for manets. In *Proceedings of ICPP Workshop on Ad Hoc Networks*, 2002.

[7] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[8] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh-Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wireless Networks*, 7(4):343–358, 2001.

[9] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[10] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[11] L.L. Peterson and B.S. Davie. *Computer Networks A Systems Approach*. Morgan Kaufmann Publishers, 2000.

[12] Martin Roth and Stephen Wicker. Termite: Emergent ad-hoc networking. In *Proceedings of the Second Mediterranean Workshop on Ad-Hoc Networks*, 2003.

[13] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 1999.

[14] R. Schoonderwoerd, O.E. Holland, J.L. Bruten, and L.J.M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[15] T.D. Seeley. *The Wisdom of the Hive*. Harvard University Press, London, 1995.

[16] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proceedings of Fourth ACM/IEEE Conference on Mobile Computing and Networking (MobiCom)*, pages 181–190, 1998.

[17] K. von Frisch. *The Dance Language and Orientation of Bees*. Harvard University Press, Cambridge, 1967.

[18] H.F. Wedde, M. Farooq, and Y. Zhang. Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In *Proceedings of ANTS Workshop, LNCS 3172*, pages 83–94. Springer Verlag, Sept 2004.