

BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior

Horst F. Wedde, Muddassar Farooq, and Yue Zhang

Informatik III, University of Dortmund
44221, Dortmund, Germany

{wedde,farooq,zhang}@ls3.cs.uni-dortmund.de

Abstract. Bees organize their foraging activities as a social and communicative effort, indicating both the direction, distance and quality of food sources to their fellow foragers through a "dance" inside the bee hive (on the "dance floor"). In this paper we present a novel routing algorithm, *BeeHive*, which has been inspired by the communicative and evaluative methods and procedures of honey bees. In this algorithm, *bee agents* travel through network regions called *foraging zones*. On their way their information on the network state is delivered for updating the local routing tables. *BeeHive* is fault tolerant, scalable, and relies completely on local, or regional, information, respectively. We demonstrate through extensive simulations that *BeeHive* achieves a similar or better performance compared to state-of-the-art algorithms.

1 Introduction

Swarm Intelligence has been evolving as an active area of research over the past years. The major emphasis is *to design adaptive, decentralized, flexible and robust artificial systems, capable of solving problems through solutions inspired by the behavior of social insects* [2]. Research in the field has largely been focused on working principles of ant colonies and how to use them in the design of novel algorithms for efficiently solving combinatorial optimization problems. A major emphasis here is on ant colony optimization (ACO) techniques [2].

To our knowledge, little attention has been paid in utilizing the organizational principles in other swarms such as honey bees to solve real world problems although the study of honey bees has revealed a remarkable sophistication of the communication capabilities as compared to ants. Nobel laureate Karl von Frisch deciphered and structures these into a language, in his book *The Dance Language and Orientation of Bees* [13]. Upon their return from a foraging trip, bees communicate the distance, direction, and quality of a flower site to their fellow foragers by making waggle dances on a dance floor inside the hive. By dancing zealously for a good foraging site they recruit foragers for the site. In this way a good flower site is exploited, and the number of foragers at this site are reinforced. In our approach we model *bee agents* in packet switching networks, for the purpose of finding suitable paths between sites, by extensively borrowing from the principles behind the bee communication.

Previous and Related Work. The focus of our research is on dynamic routing algorithms therefore we now provide an overview of two such algorithms, *AntNet* and *Distributed Genetic Algorithm (DGA)*, that have been inspired through ants swarm behavior. This will help in understanding not only the motivation for our work, but also assist the reader in understanding the different direction of our algorithm. However, we do use a classic static routing algorithm, OSPF (see [5]), in our comparative simulation for comprehensiveness.

AntNet was proposed by Di Caro and Dorigo in [3]. In *AntNet* the network state is monitored through two ant agents: *Forward_Ant* and *Backward_Ant*. A *Forward_Ant* agent is launched at regular intervals from a source to a certain destination. The authors proposed a model in [4] that enables a *Forward_Ant* agent to estimate queuing delay without waiting inside data packet queues. *Forward_Ant* agent is equipped with a stack memory on which the address and entrance time of each node on its path are pushed. Once the *Forward_Ant* agent reaches its destination it creates a *Backward_Ant* agent and transfers all information to it. *Backward_Ant* visits the same nodes as *Forward_Ant* in reverse order and modifies the entries in the routing tables based on the trip time from the nodes to the destination. At each node the average trip time, the best trip time, and the variance of the trip times for each destination are saved. The trip time values are calculated by taking the difference of entrance times of two subsequent nodes pushed onto the stack. *Backward_Ant* agent uses the system priority queues so that it quickly disseminates the information to the nodes. The interested reader may find more details in [3][4]. Later on the authors of [1] made significant improvements in the routing table initialization algorithm of *AntNet*, bounded the number of *Forward_Ant* agents during congestion, and proposed a mechanism to handle routing table entries at the neighbors of crashed routers.

The authors of [8] showed that the information needed by *AntNet* for each destination is difficult to obtain in real networks. Their idea of *global information* is that there is an entry in the routing table for each destination. This shortcoming motivated the authors to propose in [9] a *Distributed Genetic Algorithm (DGA)* that eliminates the need for having an entry for each destination node in the routing table. In this algorithm ants are asked to traverse a set of 6 nodes in a particular order, known as a *chromosome*. Once an agent visits the 6th node then it is converted into a backward agent that returns to its source node. In contrast to *AntNet* the backward agents only modify the routing tables at the source node. The source node also measures the fitness of this agent based on the trip time value, and then it generates a new population using single point cross over. New agents enter the network and evaluate the assigned paths. The routing table stores the agents' IDs, their fitness values and trip times to the visited nodes. Routing of a data packet is done through the path that has the shortest trip time to the destination. If no entry for a particular destination is found then a data packet is routed with the help of an agent that has the maximum fitness value. *DGA* was designed assuming that the routers could crash during network operations. The interested reader will find more details in [9].

In contrast to above-mentioned algorithms, we propose in this paper, an agent model in which agents need not be equipped with a stack to perform their duties. Moreover, our model requires only forward moving agents and they utilize an estimation model to calculate the trip time from their source to a given node. This model eliminates the need for global clock synchronization among routers, and it is expected that for very large networks routing information could be disseminated quickly with a small overhead as compared to *AntNet*. Our agent model does not require to store the average trip time, the variance of trip times, and the best trip time for each destination at a node to determine the goodness of a neighbor for a particular destination. Last but not the least, our algorithm works with a significantly smaller routing table as compared to *AntNet*.

Organization of the Paper. In section 2 we first develop the key ideas of the bee agent model underlying the *BeeHive* algorithm. On this basis we will present our *BeeHive* algorithm in section 2.1. Section 3 will describe the simulation and the network environment that were used to compare the performance of *BeeHive* with other state-of-the-art algorithms. In Section 4 we will discuss the results obtained from the extensive simulations. Finally we conclude on our findings, and provide an outlook to future research.

2 The *Bee Agent* Model

In this section, we will briefly describe the organizational principles of bee behavior that inspired us to transform them into an agent model. Honey bees evaluate the quality of each discovered food site and only perform *waggle dance* for it on the *dance floor* if the quality is above a certain threshold. So not each discovered site receives a reinforcement. As a result, quality flower sites are exploited quite extensively. Hence we abstract a *dance floor* into a routing table where *bee agents*, launched from the same source but arrived from different neighbors at a given node, could exchange routing information to model the network state at this node. The agent communication model of *BeeHive* could easily be realized as a *blackboard system* [10]. In comparison *AntNet* utilizes the *principle of stigmergy* [7] for communication among agents.

The majority of foragers exploit the food sources in the closer vicinity of their hive while a minority among them visit food sites faraway from their hive. We transformed this observation into an agent model that has two types of agents: *short distance bee agents* and *long distance bee agents*. *Short distance bee agents* collect and disseminate routing information in the neighborhood (upto a specific number of hops) of their source node while *long distance bee agents* collect and disseminate routing information to all nodes of a network. Informally, the *BeeHive* algorithm and its main characteristics could be summarized as follows:

1. The network is organized into fixed partitions called *foraging regions*. A partition results from particularities of the network topology. Each *foraging region* has one representative node. Currently the lowest IP address node in a *foraging region* is elected as the representative node. If this node crashes then the next higher IP address node takes over the job.

2. Each node also has a node specific *foraging zone* which consists of all nodes from whom *short distance bee agents* can reach this node.
3. Each non-representative node periodically sends a *short distance bee agent*, by broadcasting replicas of it to each neighbor site.
4. When a replica of a particular *bee agent* arrives at a site it updates routing information there, and the replica will be flooded again, however, it will not be sent to the neighbor from where it arrived. This process continues until the life time of the agent has expired, or if a replica of this *bee agent* had been received already at a site, the new replica will be killed there.
5. Representative nodes only launch *long distance bee agents* that would be received by the neighbors and propagated as in 4. However, their life time (number of hops) is limited by the *long distance limit*.
6. The idea is that each agent while traveling, collects and carries path information, and that it leaves, at each node visited, the trip time estimate for reaching its source node from this node over the incoming link. *Bee agents* use priority queues for quick dissemination of routing information.
7. Thus each node maintains current routing information for reaching nodes within its *foraging zone* and for reaching the *representative nodes of foraging regions*. This mechanism enables a node to route a data packet (whose destination is beyond the *foraging zone* of the given node) along a path toward the *representative node of the foraging region* containing the destination node.
8. The next hop for a data packet is selected in a probabilistic manner according to the quality measure of the neighbors, as a result, not all packets follow the best paths. This will help in *maximizing the system performance though a data packet may not follow the best path*, a concept directly borrowed from a principle of bee behavior: *A bee could only maximize her colony's profit if she refrains from broadly monitoring the dance floor to identify the single most desirable food* [11] (In comparison *OSPF* always chooses a next hop on the shortest path).

Figure 1 provides an exemplary working of the flooding algorithm. *Short distance bee agents* can travel upto 3 hops in this example. Each replica of the shown *bee agent* (launched by Node 10) is specified with a different trail to identify its path unambiguously. The numbers on the paths show their costs. The flooding algorithm is a variant of breadth first search algorithm. Nodes 2,3,4,5,6,7,8,9,11 constitute the *foraging zone* of node 10.

Now we will briefly discuss the estimation model that *bee agents* utilize to approximate the trip time t_{is} that a packet will take in reaching their source node s from current node i (ignoring the protocol processing delays for a packet at node i and s).

$$t_{is} \approx \frac{ql_{in}}{b_{in}} + tx_{in} + pd_{in} + t_{ns} \quad (1)$$

where ql_{in} is the size of the queue (in bits) for neighbor n at node i , b_{in} is the bandwidth of the link between node i and neighbor n , tx_{in} and pd_{in} are transmission delay and propagation delay respectively of the link between node i and neighbor n , and t_{ns} is trip time from n to s . Bandwidth and propagation delays of all links of a node are approximated by transmitting *hello packets*.

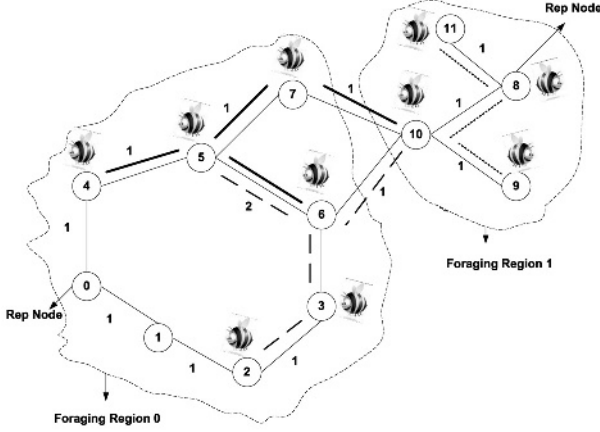


Fig. 1. Bee Agents Flooding Algorithm

Table 1. *Intra foraging zone* Routing Table

R_i	$D_1(i)$	$D_2(i)$	\dots	$D_d(i)$
$N_1(i)$	(p_{11}, q_{11})	(p_{12}, q_{12})	\dots	(p_{1d}, q_{1d})
\vdots	\vdots	\vdots	\ddots	\vdots
$N_n(i)$	(p_{n1}, q_{n1})	(p_{n2}, q_{n2})	\dots	(p_{nd}, q_{nd})

2.1 Algorithm Design of BeeHive

In *BeeHive*, each node i maintains three types of routing tables: *Intra Foraging Zone* (IFZ), *Inter Foraging Region* (IFR) and *Foraging Region Membership* (FRM). *Intra Foraging Zone* routing table R_i is organized as a vector of size $|D(i)| \times (|N(i)|)$, where $D(i)$ is the set of destinations in *foraging zone* of node i and $N(i)$ is the set of neighbors of i . Each entry P_{jk} is a pair of queuing delay and propagation delay (q_{jk}, p_{jk}) that a packet will experience in reaching destination k via neighbor j . Table 1 shows an example of R_i . In the *Inter Foraging Region* routing table, the queuing delay and propagation delay values for reaching the *representative node* of each *foraging region* through the neighbors of a node are stored. The structure of the *Inter Foraging Region* routing table is similar to the one shown in Table 1 where destination is replaced by a pair of (representative, region). The *Foraging Region Membership* routing table provides the mapping of known destinations to a *foraging region*. In this way we eliminate the need to maintain $O(N \times D)$ (where D is total number of nodes in a network) entries in a routing table as done by *AntNet* and save a considerable amount of router memory needed to store this routing table.

Goodness of a Neighbor: The goodness of a neighbor j of node l (l has n neighbors) for reaching a destination d is g_{jd} and defined as follows

$$g_{jd} = \frac{\frac{1}{p_{jd}}(e^{-\frac{q_{jd}}{p_{jd}}}) + \frac{1}{q_{jd}}(1 - e^{-\frac{q_{jd}}{p_{jd}}})}{\sum_{k=1}^n (\frac{1}{p_{kd}}(e^{-\frac{q_{kd}}{p_{kd}}}) + \frac{1}{q_{kd}}(1 - e^{-\frac{q_{kd}}{p_{kd}}}))} \quad (2)$$

The fundamental motivation behind Definition 2 is to approximate the behavior of the real network. When the network is experiencing a heavy network traffic load then the queuing delay plays the primary role in the delay of a link. In this case it is trivial to say that $q_{jd} \gg p_{jd}$ and we could see from equation (2) that $g_{jd} \approx \frac{1}{\sum_{k=1}^n \frac{q_{kd}}{p_{kd}}}$. When the network is experiencing low network traffic then the propagation delay plays an important role in defining the delay of a link. As $q_{jd} \ll p_{jd}$, from equation (2) we get $g_{jd} \approx \frac{1}{\sum_{k=1}^n \frac{p_{jd}}{p_{kd}}}$. We use *stochastic sampling with replacement* [6] for selecting a neighbor. This principle ensures that a neighbor j with goodness g_{jd} will be selected as the next hop with at least the probability $\frac{g_{jd}}{\sum_{k=1}^n g_{kd}}$. Algorithm 1 provides the pseudo code of *BeeHive*.

Algorithm 1 (BeeHive)

```

t := current_time, t_end := time to end simulation
// Short_Limit := 7, Long_Limit := 40, Bee_Generation_Interval := 1
// i = current node, d = destination node, s = source node
// n = successor node of i, p = predecessor node of i
// z = Representative node of the foraging region containing s
// w = Representative node of the foraging region containing d
// q is queuing delay estimate of a bee agent from node p to s
// p is propagation delay estimate of a bee agent from node p to s
Δt := Bee_Generation_Interval, Δh := hello packet generation interval
bip := estimated_link_band_width_to_neighbor_p
pip := estimated_propagation_delay_to_neighbor_p
hi := hop limit for bees of i, lip := size_normal_queue_i_to_p (bits)
foreach Node // concurrent activity over the network
  while (t ≤ t_end)
    if (t mod Δt = 0)
      if (i is representative node of the foraging region)
        set hi := Long_Limit, bi is long distance bee agent
      else
        set hi := Short_Limit, bi is short distance bee agent
      endif
      launch a bee bi to all neighbors of i
    endif
  foreach bee bs received at i from p
    if (bs was launched by i or its hop limit reached)
      kill bee bs
    elseif (bs is inside foraging zone of node s)
      q := q +  $\frac{l_{ip}}{b_{ip}}$  and p := p + pip

```

```

    update IFZ routing table entries  $q_{ps} = q$  and  $p_{ps} = p$ 
    update  $q$  ( $q := \sum_{k \in N(i)} (q_{ks} \times g_{ks})$ ) and  $p$  ( $p := \sum_{k \in N(i)} (p_{ks} \times g_{ks})$ )
else
     $q := q + \frac{l_{ip}}{b_{ip}}$  and  $p := p + p_{ip}$ 
    update IFR routing table entries  $q_{pz} = q$  and  $p_{pz} = p$ 
    update  $q$  ( $q := \sum_{k \in N(i)} (q_{kz} \times g_{kz})$ ) and  $p$  ( $p := \sum_{k \in N(i)} (p_{kz} \times g_{kz})$ )
endif
if(  $b_s$  already visited node  $i$  )
    kill bee  $b_s$ 
else
    use priority queues to forward  $b_s$  to all neighbors of  $i$  except  $p$ 
endif
endfor
foreach data packet  $d_{sd}$  received at  $i$  from  $p$ 
    if ( node  $d$  is within foraging zone of node  $i$  )
        consult IFZ routing table of node  $i$  to find delays to node  $d$ 
        calculate goodness of all neighbors for reaching  $d$  using equation 2
    else
        consult FRM routing table of node  $i$  to find node  $w$ 
        consult IFR routing table of node  $i$  to find delays to node  $w$ 
        calculate goodness of all neighbors for reaching  $w$  using equation 2
    endif
    probabilistically select a neighbor  $n$  ( $n \neq p$ ) as per goodness
    enqueue data packet  $d_{sd}$  in normal queue for neighbor  $n$ 
endfor
if (  $t \bmod \Delta h = 0$  )
    send a hello packet to all neighbors
    if (time out before a response from neighbor) (4th time)
        neighbor is down
        update the routing table and launch bees to inform other nodes
    endif
endif
endwhile
endfor

```

3 Simulation Environment for BeeHive

In order to evaluate our algorithm *BeeHive* in comparison with *AntNet*, *DGA* and *OSPF*, we implemented all of them in the OMNeT++ simulator [12]. For *OSFP* we implemented a static link state routing that implements the deterministic Dijkstra Algorithm [5] which selects the next hop according to the shortest path from a source to a destination. For *AntNet* and *DGA* we used the same parameters that were reported by the authors in [3] and [9], respectively. The

network instance that we used in our simulation framework is the Japanese Internet Backbone (NTTNET)(see Figure 2). It is a 57 node, 162 bidirectional links network. The link bandwidth is 6 Mbits/sec and propagation delay is from 2 to 5 milliseconds. Traffic is defined in terms of open sessions between two different nodes. Each session is characterized completely by sessionSize (2 Mbits), inter-arrival time between two sessions, source, destination, and packet inter-arrival time during a session. The size of data packet is 512 bytes, the size of a *bee agent* is 48 bytes. The queue size is limited to 500 Kbytes in all experiments. To inject dynamically changing data traffic patterns we have defined two states: uniform and weighted. Each state lasts 10 seconds and then a state transition to another state occurs. In *Uniform* state (U) a destination is selected from a uniform distribution. While in *Weighted* state (W), a destination selected in *Uniform* state is favored over other destinations. This approach provides a more challenging experimental environment than the one in which *AntNet* was evaluated.

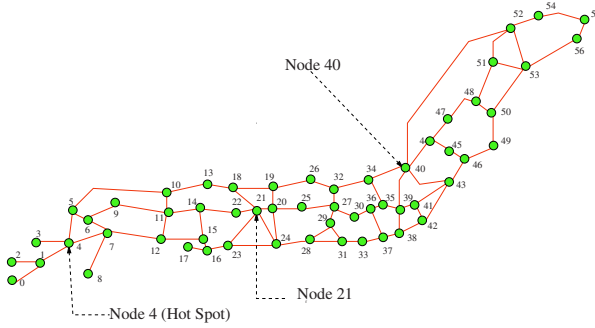


Fig. 2. Japanese Backbone NTTNet

4 Experimental Findings

Now we will report our results obtained from the extensive simulations. MSIA is the mean of session inter-arrival times and MPIA is the mean of packet inter-arrival times during a session. The session inter-arrival and packet inter-arrival times are taken from negative exponential distributions with mean MSIA and MPIA, respectively. All reported values are an average of the values obtained from ten independent runs. % Packets Delivered (on top of bars) report the percentage of deliverable packets that were actually delivered. By deliverable packet we mean a packet whose destination router is up.

Saturating Loads. The purpose of the experiments was to study the behavior of the algorithms by gradually increasing the traffic load, through decreasing MSIA from 4.7 to 1.7 seconds. MPIA is 0.005 seconds during these experiments. Figure 3 shows the average throughput and 90th percentile of the packet delay distribution. It is obvious from Figure 3 that *BeeHive* delivered approximately

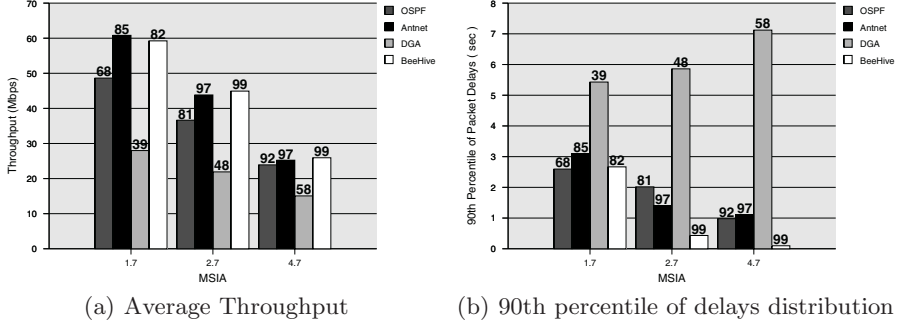


Fig. 3. Behavior under saturating traffic loads

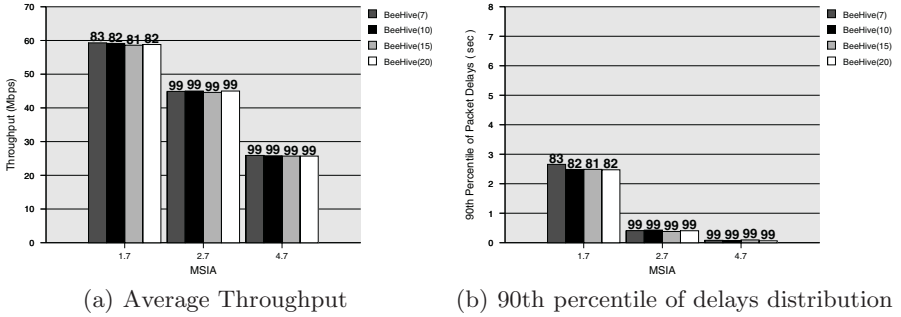


Fig. 4. Effect of foraging zones sizes

the same number of data packets as that of *AntNet* but with lesser packet delay. Both *OSFP* and *DGA* are unable to cope with a saturated load yet the performance of *DGA* is the poorest.

Size of Foraging Zones. Next we analyzed the effect of the size of a *foraging zone* in which a *bee agent* updates the routing table. We report the results for sizes of 7, 10, 15 and 20 hops in Figure 4. Figure 4 shows that increasing the size of *foraging zone* after 10 does not bring significant performance gains. This shows the power of *BeeHive* that it converges to an optimum solution with a size of just 7 hops.

Size of the Routing Table. The fundamental motivation of the *foraging zone* concept was not only to eliminate the requirement for global knowledge but also to reduce memory needed to store a routing table. *BeeHive* requires 88, 94 and 104 entries, on the average, in the routing tables for *foraging zone* sizes of 7, 10 and 20 hops respectively. *OSFP* needs just 57 entries while *AntNet* needs 162 entries on the average. Hence *BeeHive* achieves similar performance as that of *AntNet* but size of the routing table is of the order of *OSFP*.

Hot Spot. The purpose of this experiment is to study the effect of transient overloads in a network. We selected node 4 (see Figure 2) as hot spot. The hot spot was active from 500 seconds to 1000 seconds and all nodes sent data to node 4 with $MPIA=0.05$ seconds. This transient overload was superimposed on a normal load of $MSIA=2.7$ seconds and $MPIA=0.3$ seconds. Figure 5 shows that both *BeeHive* and *AntNet* are able to cope with the transient overload, however the average packet delay for *BeeHive* is less than 100 milliseconds as compared to 500 milliseconds for *AntNet*. Again *DGA* shows the poorest performance.

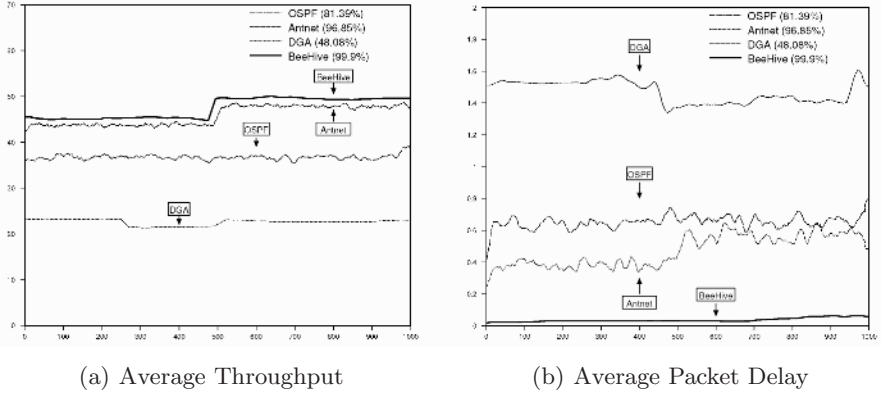


Fig. 5. Node 4 acted as hot spot from 500 to 1000 seconds

Router Crash. The purpose of this experiment was to analyze the fault tolerant behavior of *BeeHive* so we took $MSIA = 4.7$ seconds and $MPIA = 0.005$ seconds to ensure that no packets are dropped because of the congestion. We simulated a scenario in which Router 21 crashed at 300 seconds, and Router 40 crashed at 500 seconds and then both were repaired at 800 seconds. Figure 6 shows the results. *BeeHive* is able to deliver 97% of deliverable packets as compared to 89% by *AntNet*. Please observe that from 300 to 500 seconds (just Router 21 is down), *BeeHive* has a superior throughput and lesser packet delay but once Router 40 crashes, the packet delay of *BeeHive* increases because of higher load at Router 43. From Figure 2 it is obvious that the only path to the upper part of the network is via Router 43 once Router 40 crashed. Since *BeeHive* is able to deliver more packets the queue length at Router 43 increased and this led to relatively poorer packet delay as compared to *AntNet*. Please also observe that Router 21 is critical but in case of its crash still multiple paths exist to the middle and upper part of the topology via 15,18,19,20,24.

Overhead of BeeHive. Routing overhead is defined as the ratio between the bandwidth occupied by the routing packets and the total available network bandwidth [3]. The overhead of *BeeHive* is 0.00633 as compared to 0.00285 of *AntNet* [3]. *BeeHive* has a bit more overhead as compared to *AntNet*, but this overhead remains constant for a given topology.

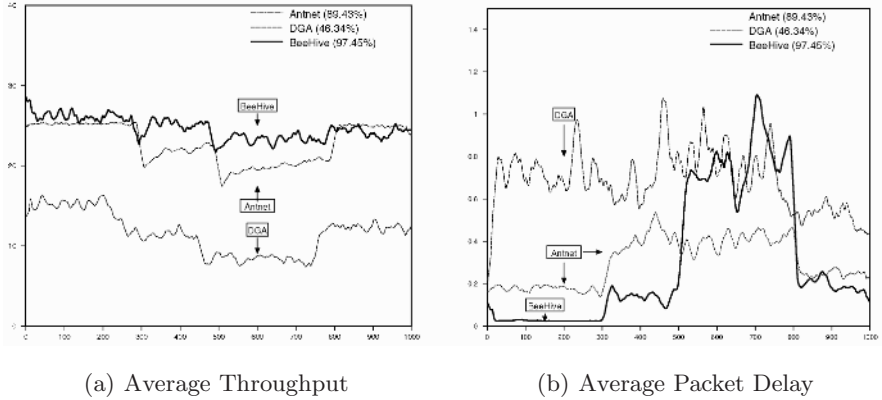


Fig. 6. Router 21 is down at 300 and Router 40 at 500 and both repaired at 800

5 Conclusion and Future Work

Swarm intelligence is evolving as a new discipline that is transforming the traditional algorithmic design paradigm of the developers by enabling them to contemplate the working principles of systems in Nature and then use them in design and development of the algorithms. In this paper we introduced a fault-tolerant, adaptive and robust routing protocol inspired from dance language and foraging behavior of honey bees. The algorithm does not need any global information such as the structure of the topology and cost of links among routers, rather it works with the local information that a *short distance bee agent* collects in a *foraging zone*. It works without the need of global clock synchronization which not only simplifies its installation on real routers but also enhances fault tolerance. In contrast to *AntNet* our algorithm utilizes only forward moving *bee agents* that help in disseminating the state of the network to the routers in real-time. The *bee agents* take less than 1% of the available bandwidth but provide significant enhancements in throughput and packet delay.

We implemented two state-of-the-art adaptive algorithms (*AntNet* and *DGA*) for the OMNeT++ simulator and then compared our *BeeHive* algorithm with them. Through extensive simulations representing dynamically changing operating network environments we have demonstrated that *BeeHive* achieves a better or similar performance as compared to *AntNet*. However, this enhancement in performance is achieved with a routing table whose size is of the order of *OSPF*. In the near future we will have implemented *BeeHive* inside the network stack of the Linux kernel in order to then test the algorithm on real network topologies. Our work during this phase will be subject of forthcoming publications.

Acknowledgments

We pay special thanks to Gianni Di Caro who provided valuable feedback on our implementation of *AntNet* on the OMNeT++ simulator. We would also like to thank Suihong Liang who helped us in understanding *DGA*.

References

1. B. Barán and R. Sosa. A new approach for antnet routing. In *Proceedings of the Ninth International Conference on Computer, Communications and Networks*, 2000.
2. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.
3. G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communication networks. *Journal of Artificial Intelligence*, 9:317–365, December 1998.
4. G. Di Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 541–546. IASTED/ACTA Press, 1998.
5. E. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271, 1959.
6. D. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989.
7. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis* et *cubitermes* sp. la théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.
8. S. Liang, A. Zincir-Heywood, and M. Heywood. The effect of routing under local information using a social insect metaphor. In *Proceedings of IEEE Congress on Evolutionary Computing*, May 2002.
9. S. Liang, A. Zincir-Heywood, and M. Heywood. Intelligent packets for dynamic network routing using distributed genetic algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference*. GECCO, July 2002.
10. P. Nii. The blackboard model of problem solving. *AI Mag*, 7(2):38–53, 1986.
11. T. Seeley. *The Wisdom of the Hive*. Harvard University Press, London, 1995.
12. A. Varga. OMNeT++: Discrete event simulation system: User manual. <http://www.omnetpp.org>.
13. K. von Frisch. *The Dance Language and Orientation of Bees*. Harvard University Press, Cambridge, 1967.