Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 51 (2007) 2104–2125

Computer Networks

www.elsevier.com/locate/comnet

Performance analysis of the AntNet algorithm

S.S. Dhillon *, P. Van Mieghem

Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, P.O. Box 5031, 2600 GA Delft, The Netherlands

Received 20 April 2005; received in revised form 26 September 2006; accepted 22 October 2006
Available online 27 November 2006

Responsible Editor: A. Kshemkalyani

Abstract

A number of routing algorithms based on the ant-colony metaphor have been proposed for communication networks. However, there has been little work on the performance analysis of ant-routing algorithms. In this paper, we compare the performance of AntNet, an ant-routing algorithm, with Dijkstra's shortest path algorithm. Our simulations show that the performance of AntNet is comparable to Dijkstra's shortest path algorithm. Moreover, under varying traffic loads, AntNet adapts to the changing traffic and performs better than shortest path routing.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Ant-colony metaphor; Ant-routing algorithms; AntNet

1. Introduction

A wide variety of routing protocols and algorithms exist for communication networks. In the traditional approach to routing, the routing tables are updated by exchanging routing information between the routers. Various routing protocols differ in their approaches to exchange the routing information. For example, in Open Shortest Path First (OSPF) the routers exchange link-state[1] information by flooding. A relatively new approach to routing is the use of mobile agents for updating and maintaining the routing tables. Recently, a number of routing algorithms inspired by the ant-colony metaphor and using mobile agents have been proposed for both wired and wireless networks. In this paper, we collectively refer to these algorithms as ant routing algorithms (ANTRALs).

An ANTRAL is a hop-by-hop routing algorithm based on the principle of *stigmergy* observed in real-life ant colonies. *Stigmergy* is a form of indirect communication mediated by modifications of the environment [26]. It has been shown that real ants are able to find the shortest path by following the trail of a chemical substance called *pheromone* deposited by other ants [18,19,23]. The idea behind ANTRALs is to use a form of *stigmergy* to coordinate societies of artificial agents. The artificial

* Corresponding author. Tel.: +31 152786238.
E-mail address: s.dhillon@ewi.tudelft.nl (S.S. Dhillon).

[1] Link refers to an interface on the router and each link has a cost associated with it. The state of the link is a description of the interface and the relationship of the interface to the neighboring routers. In link-state protocols, each router is assumed to be capable of finding the state of the link to its neighbors (up or down) and the cost of each link.

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.11.002

agents (mobile agents) or ants move on the network and are used to update the routing tables. The mobile agents update the routing tables in an asynchronous manner and independently of other mobile agents. Thus, in ANTRALs, the routers (nodes) do not need to directly exchange routing information for updating the routing tables. The principles of ant colony and *stigmergy* have been applied to numerous other optimization problems besides routing and have been referred to as ant colony optimization in the literature [6,20,27].

There are two critical components that determine the performance of an ANTRAL. First, the performance depends on how the mobile agents search for the shortest path, which is referred to as *exploration* [16]. The mobile agents could either use heuristics based on the routing tables or the routing table values without any modifications to move to the next node. Second, the mobile agents have to update the routing tables based on the paths that have been searched. In general, exploration and routing table update in ANTRALs are coupled since the mobile agents use the same routing table values for exploration which they also update. However, Bean and Costa [2,5] have demonstrated that it is possible to de-couple exploration from routing table updates in ANTRALs by restricting exploration to the first hop, and letting mobile agents follow the data routing policy for all subsequent hops to the destination.

In all the ANTRALs proposed so far, the algorithm parameters have been chosen heuristically. There has been little work on studying the robustness of ANTRALs to the variations in different parameters. In [2,5], Bean and Costa have examined the equilibrium load-balancing properties of ANTRALs with respect to system and user optimization criteria. However, the effect of parameter changes on the performance of ANTRALs is not examined and the analysis has been limited to a few network topologies. In this paper, we analyze the performance of AntNet, an ANTRAL proposed by Di Caro and Dorigo [4]. We compare the performance of AntNet with Dijkstra's shortest path algorithm for constant delays (non-varying traffic). We also study the effect of different parameters used for exploration and routing table updates on the performance of AntNet algorithm.

Our simulations indicate that the performance of the AntNet algorithm is comparable to the Dijkstra's shortest path algorithm under non-varying traffic. However, the performance of the AntNet algorithm is dependant on the network size and topology. One of the key insights is that the AntNet algorithm performs well for sparse random graphs. Furthermore, the variations in different parameters have a limited impact on the performance of the AntNet algorithm indicating the robustness of AntNet algorithm.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the network model and Section 4 presents the AntNet algorithm. In Section 5, we present the simulation results. Finally, Section 6 presents the conclusions.

2. Overview of ANTRAL implementations

This section describes some of the prominent ANTRALs proposed for routing in communication networks.

AntNet [4] is a routing algorithm proposed for wired datagram networks based on the principle of ant colony optimization. In AntNet, each node maintains a routing table and an additional table containing statistics about the traffic distribution over the network. The routing table maintains for each destination and for each next hop a measure of the goodness of using the next hop to forward data packets to the destination. These goodness measures, called pheromone variables, are normalized to one in order to be used by a stochastic routing policy. AntNet uses two sets of homogeneous mobile agents called forward ants and backward ants to update the routing tables. The forward ants use heuristics based on the routing table to move between a given pair of nodes and are used to collect information about the traffic distribution over the network. The backward ants retrace the paths of forward ants in the opposite direction. At each node, the backward ants update the routing table and the additional table containing statistics about the traffic distribution over the network. AntNet [4] has been shown to perform better than Bellman-Ford, OSPF etc. routing protocols under varying and near saturation traffic loads.

Ant-Based Control (ABC) is an algorithm proposed by Schoonderwoerd et al. [12,25] for load balancing in circuit-switched networks. In ABC, the calls are routed using probabilistic routing tables that consist of next hop probabilities for each destination. The link costs are assumed to be symmetric and hence, only one-directional mobile agents are used for updating and maintaining the routing tables. The mobile agents use heuristics based on the routing tables to move across the network between arbitrary pairs of nodes. At each node along the path, the mobile agents update the routing tables based on their distance from the source node and the current state of the routing table.

Another ANTRAL for wired networks has been proposed by Kuntz et al. [22]. The proposed algorithm differs from AntNet in terms of different loop detection behavior, simpler backward ant and different routing table update procedure. The authors also propose another routing protocol called Co-operative Asymmetric Forward (CAF) routing. CAF is similar to ABC but it works for asymmetric networks where the link costs are not identical in opposite directions. CAF has been shown to perform as well as AntNet and is able to cope with changing bandwidth and network topology [22].

AntHocNet is a hybrid routing protocol proposed by Ducatelle et al. [21] for mobile ad hoc networks. AntHocNet consists of both the reactive and proactive components. In AntHocNet, nodes do not maintain routes to all possible destinations at all the times, rather the nodes generate mobile agents only at the beginning of a data session. The mobile agents search for multiple paths to the destination and these paths are set up in the form of pheromone tables indicating their respective quality. During the course of the data session, the paths are continuously monitored and improved in a proactive manner. AntHocNet has been shown to outperform AODV in terms of end-to-end delay and delivery ratio [21].

Ad hoc Networking with Swarm Intelligence (ANSI) is a reactive routing protocol proposed by Rajagopalan and Shen [24] for mobile ad hoc networks. ANSI protocol uses two sets of mobile agents called forward reactive ants and backward reactive ants. The routing tables in ANSI contain an entry for each reachable node and next best hop while the ant decision tables store the pheromone values. In ANSI, the forward reactive ants are generated only when a node needs to transmit data to another node. The forward reactive ants are broadcast while the backward reactive ants retrace the path of forward reactive ants and update the pheromone values at the nodes. The data packets choose the next hop deterministically i.e., the hop which contains the largest pheromone value is chosen as the next hop. ANSI has been shown to perform either better or comparable with AODV with respect to packet delivery, end-to-end delay and delay jitter [24].

A number of routing protocols in which the mobile agents do not update the routing tables directly have also been proposed (e.g. Termite [10], Global Positioning System/Ant-Like Routing Algorithm (GPSAL) [3]). In Termite [10], the routing table entries contain pheromone values for choosing a neighbor as the next hop for each destination. The pheromone values decay exponentially with time and the corresponding entries are removed from the routing tables, if all the pheromone for a particular destination decays. Thus, the routing tables in Termite maintain entries for only the destinations from which packets have been received during recent times. Termite does not use mobile agents for updating the routing tables instead route discovery and maintenance are performed by a set of four control packets: route request packets, route reply packets, hello packets and seed packets. Both data packets and control packets (except route request packets) are used for updating the routing tables.

Ant-Colony-Based Routing Algorithm (ARA) is a routing protocol proposed by Güneş and Spaniol [7] for mobile ad hoc networks. The routing table entries in ARA contain pheromone values for choosing a neighbor as the next hop for each destination. The pheromone values in the routing tables decay with time and the nodes enter a sleep mode if the pheromone in the routing table has reached a lower threshold. Route discovery in ARA is performed by a set of two mobile agents – forward ants and backward ants. During route discovery, the forward and backward ant packets having unique sequence numbers, to prevent duplicate packets, are flooded through the network by the source and destination nodes, respectively. The forward and backward ants update the pheromone tables at the nodes along the path for the source and destination nodes respectively. Once the route discovery for a particular destination has been performed, the source node does not generate new mobile agents for the destination instead the route maintenance is performed by the data packets.

Uniform ant routing algorithms are a class of ANTRALs in which the mobile agents choose the next node uniformly among the neighbors of the node [13]. Thus, in uniform ANTRALs, the mobile agents move independently of the routing tables and perform a random walk (with memory) on the network graph while searching for the destination. This feature of uniform ANTRALs reduces the complexity of the ant routing algorithm and leads to exploration of all the paths with equal probabilities.

3. Network model

We model the network as a graph $G(N, L)$ consisting of N nodes and L links. All the links in the network are considered bidirectional and specified by a transmission capacity and a transmission delay. Each node is considered a communication end-point (host) and a forwarding unit (router).

As illustrated in Fig. 1, every node in the network maintains an input buffer composed of a single queue and an output buffer composed of a high priority queue and a low priority queue for each neighbor or outgoing link. The high priority queue is served before the low priority queue. All the packets within the network can be divided into two different classes:

- Data packets: represent the information that the end-users exchange with each other. In ant-routing, data packets do not maintain any routing information but use the information stored at routing tables for travelling from the source to the destination node.
- Mobile agents (forward ants and backward ants): are used to update the routing tables and distribute information about the traffic load in the network.

Backward ant packets have a higher priority than the data and forward ant packets and are thus stored in the high priority queue, while data and forward ant packets are stored in the low priority queue. We assume that all the packets in the low priority queue and the high priority queue in the output buffer are served in a FIFO order. Further, the maximum number of packets stored in the input buffer or output buffer is limited by the size of the buffer. We have assumed that the buffer size is sufficiently large to neglect buffer overflow.

When a node receives a packet from a neighbor, the packet is first stored in the input buffer. The packet in the input buffer is served in a FIFO order or according to a different scheduling rule. After the packet has been served, the packet is sent to the output buffer. Within the output buffer, the packet goes to a particular queue for a particular outgoing link based on the type of the packet and the next node.

3.1. Data structures at nodes

Mobile agents communicate in an indirect way, through the information they concurrently read and write in two data structures stored at each network node k:

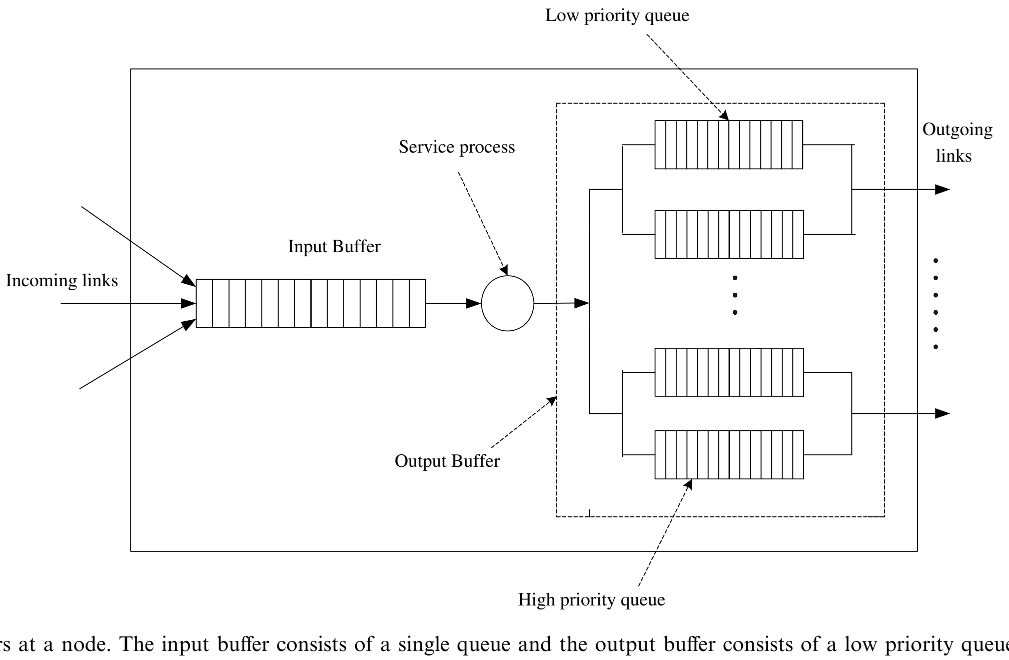

Fig. 1. Buffers at a node. The input buffer consists of a single queue and the output buffer consists of a low priority queue and a high priority queue for each outgoing link.

1. A routing table T_k, organized as a matrix with probabilistic entries as shown in Fig. 2. Each row in the routing table corresponds to one destination in the network and each column corresponds to a neighbor of the current node. The routing table T_k defines the probabilistic routing policy currently adopted at node k: for each possible destination d and for each neighbor node n, T_k stores a probability value p_{nd} expressing the probability of choosing n as the next node when the destination is d such that:

$$\sum_{n \in N_k} p_{nd} = 1,$$

where $d \in [1,N]$ and $N_k = \{\text{neighbors}(k)\}$.

2. A table $M_k(\mu_d, \sigma_d^2, W_d)$ containing statistics about the network topology and the traffic distribution over the network as seen by the local node k. For each destination d in the network, the table M_k contains a moving observation window W_d, an estimated mean μ_d and an estimated variance σ_d^2. The moving observation window W_d, of size $W_{\max}$, represents an array containing the trip times of last $W_{\max}$ forward ants that travel from the node k to the destination d. The moving observation window W_d is used to compute the best trip time t_{best_d} i.e., the best trip time experienced by a forward ant travelling from the node k to the destination d among the last $W_{\max}$ forward ants that travel from the node k to the destination d. The mean μ_d and variance σ_d^2 represent the mean and variance of the trip times experienced by the forward ants to move from the node k to the destination node d and are calculated using the exponential model:

$$\mu_d \leftarrow \mu_d + \eta(t_{k\to d} - \mu_d), \tag{1}$$

$$\sigma_d^2 \leftarrow \sigma_d^2 + \eta((t_{k\to d} - \mu_d)^2 - \sigma_d^2). \tag{2}$$

In (1) and (2), $t_{k\to d}$ represents the newly observed forward ant's trip time to travel from the node k to the destination node d and $\eta \in (0,1]$ is a factor that weighs the number of recent samples that will affect the mean μ_d and the variance σ_d^2. Di Caro and Dorigo [4] relate η to the maximum size of the observation window $W_{\max}$ by

$$W_{\max} = \frac{5c}{\eta} \text{ where } c < 1. \tag{3}$$

Di Caro and Dorigo [4] calculated that $\frac{5}{\eta}$ samples affect the mean so (3) has been used to ensure that the mean and the best trip time are calculated over the same moving observation window. We choose the maximum size of the moving observation window $W_{\max}$ to be sufficiently large i.e., $W_{\max} > \frac{5}{\eta}$ but independently of the parameter η.

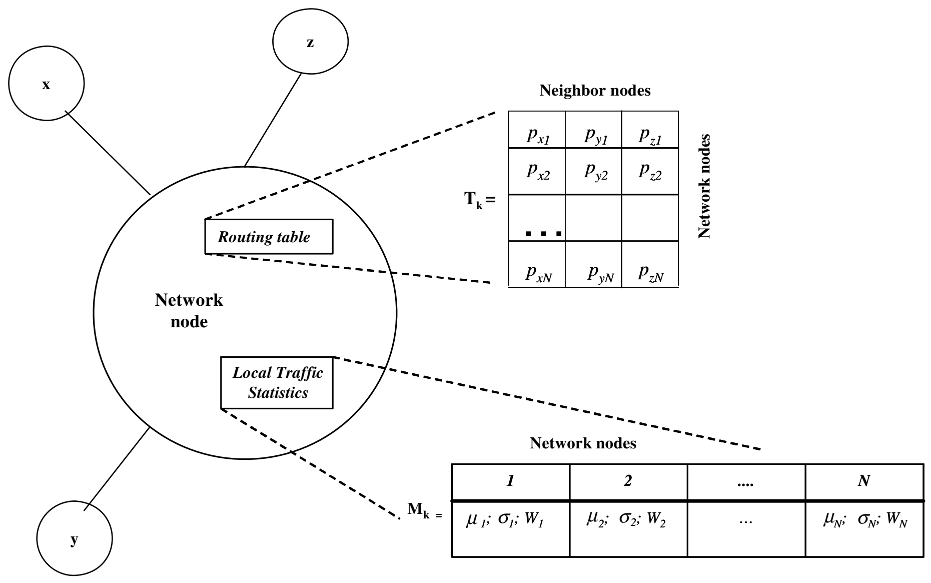

Fig. 2. The data structures for a node with neighbors x, y and z and a network with N nodes: routing table (T_k) and statistics table (M_k).

4. AntNet algorithm

4.1. Description of the AntNet algorithm

The AntNet algorithm [4] can be described as follows:

1. At regular intervals, from every network node s, a forward ant $F_{s\to d}$ is launched with a randomly selected destination node d. Destinations are chosen to match the current traffic patterns i.e., if f_{sd} is a measure (in bits or in the number of packets) of the data flow $s \to d$, then the probability y_d of creating at node s a forward ant with node d as destination is:
$$y_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}. \tag{4}$$
2. While travelling towards their destination nodes, the forward ants store their paths and the traffic conditions. The identifier of every visited node k and the time elapsed since the launching time of the forward ant to arrive at this kth node are pushed onto a memory stack $S_{s\to d}$ stored in the data field of the forward ant. Forward ants share the same queues as data packets, so they experience the same traffic delays as data packets.
3. At each node k, each forward ant chooses the next node as follows:
   - If all the neighboring nodes have not been visited, then the next neighbor is chosen among the nodes that have not been visited as:
$$p'_{nd} = \frac{p_{nd} + \alpha l_n}{1 + \alpha(|N_k| - 1)}. \tag{5}$$
     In (5), N_k represents the set of neighbors of the current node k and $|N_k|$ the cardinality of that set, i.e., the number of neighbors while the heuristic correction l_n is a normalized value [0,1] such that $1 - l_n$ is proportional to the length q_n of the queue of the link connecting the node k with its neighbor n:
$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{|N_k|} q_{n'}}. \tag{6}$$
     The value of α in (5) weighs the importance of the instantaneous state of the node's queue with respect to the probability values stored in the routing table.
   - If all the neighboring nodes have been visited previously, then the next node is chosen uniformly among all the neighbors. In this case, since all the neighbors have been visited previously the forward ant is forced to return to a previously visited node. Thus, irrespective of which neighbor is chosen as the next node, the forward ant is in a loop (cycle).
   - With a small probability ε, the next node may be chosen uniformly among all the neighboring nodes. The parameter ε is deliberately incorporated in the ANTRAL to overcome the problem where one of the entries in the routing table is almost unity, while the other are vanishingly small. In such a situation, the forward ants always choose the same link and thus stop exploring the network for other routes. The parameter ε ensures that the network is being constantly explored, though it introduces an element of inefficiency in the algorithm. The use of parameter ε to introduce randomness in the algorithm is referred to as ε -*greedy* policy in reinforcement learning literature [14]. Thus, due to the ε-*greedy* policy there are no restrictions on the routing table values and some of the routing table entries may be zero. In AntNet implementation [4,11], the value of parameter ε is chosen as zero. In the special case, when $\varepsilon = 1$, the AntNet algorithm is a uniform ANTRAL (unfNet).
4. If a cycle is detected, that is, if the ant is forced to return to an already visited node, the cycle's nodes are popped from the ant's stack and all memory about the cycle is destroyed. If the cycle lasted longer than the lifetime of the forward ant before entering the cycle, the ant is destroyed (Fig. 3). The lifetime of a forward ant is defined as the total time since the forward ant was generated.
5. When the destination node d is reached, the forward ant $F_{s\to d}$ generates a backward ant $B_{d\to s}$. The forward ant transfers all the memory contained in the stack $S_{s\to d}$ to the backward ant, and dies.
6. The backward ant takes the same path as the corresponding forward ant, but in the opposite direction. At each node k, the backward ant pops the stack $S_{s\to d}$ to move to the next node. Backward ants do not share the same queues as data packets and forward ants; they use high priority queues to quickly propagate to the routing tables the information collected by the forward ants.
7. Arriving at a node k coming from a neighbor node h, the backward ant updates the two main

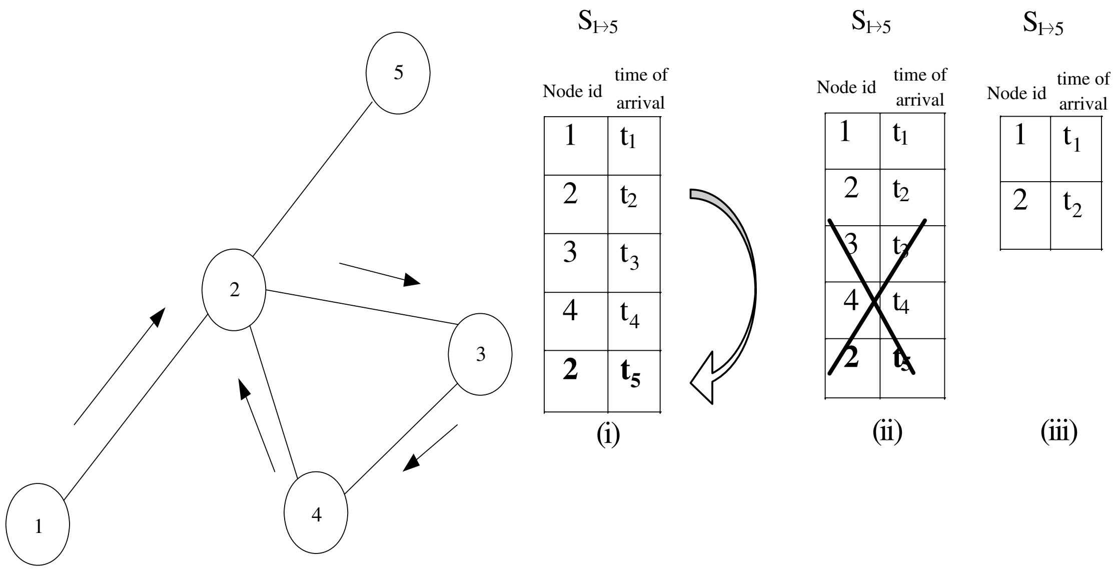

Fig. 3. (i) A cycle ($2 \to 3 \to 4 \to 2$) in the forward ant's $F_{1\to5}$ path is detected. (ii) All the memory about the cycle ($2 \to 3 \to 4 \to 2$) is destroyed. (iii) The stack $S_{1\to5}$ maintained by forward ant $F_{1\to5}$ after the removal of cycle. Further if the time spent in cycle ($t_5 - t_2$) is greater than the lifetime of the ant before the cycle ($t_2 - t_1$), the forward ant $F_{1\to5}$ is destroyed.

data structures of the node, the local model of the traffic M_k and the routing table T_k, for all the entries corresponding to the destination node d. The update of routing tables at each node along the path as the backward ant travels from the destination to the source node is known as sub-path update method.[2]

- The mean μ_d and variance σ_d^2 entries in the local model of traffic M_k are modified using (1) and (2). The best value t_{best_d} of the forward ants trip time from node k to the destination d stored in the moving observation window W_d is also updated by the backward ant. If the newly observed forward ant's trip time $t_{k\to d}$ from the node k to the destination d is less then t_{best_d}, then t_{best_d} is replaced by $t_{k\to d}$.
- The routing table T_k is changed by incrementing the probability $p_{hd'}$ (i.e., the probability of choosing neighbor h when destination is d') and decrementing, by normalization, the other probabilities $p_{nd'}$. The probability $p_{hd'}$ is increased by the reinforcement value r as:

$$p_{hd'} \leftarrow p_{hd'} + r(1 - p_{hd'}). \tag{7}$$

The probabilities $p_{nd'}$ of the other neighboring nodes n for destination d' are decreased by the negative reinforcement as:

$$p_{nd'} \leftarrow p_{nd'} - rp_{nd'}, \quad \forall n \neq h,\ n \in N_k. \tag{8}$$

Thus, in AntNet, every path found by the forward ants receives a positive reinforcement.

- The reinforcement value r used in (7) and (8) is a dimensionless constant (0,1] and is calculated as:

$$r = c_1 \frac{t_{\text{best}_d}}{t_{k\to d}} + c_2 \frac{t_{\sup} - t_{\text{best}_d}}{(t_{\sup} - t_{\text{best}_d}) + (t_{k\to d} - t_{\text{best}_d})}. \tag{9}$$

In (9), $t_{k\to d}$ is the newly observed forward ant's trip time from node k to the destination d and t_{best_d} is the best trip time experienced by the forward ants traveling towards the destination d over the observation window W_d. The value of $t_{\sup}$ is calculated as:

$$t_{\sup} = \mu_d + \frac{\sigma_d}{\sqrt{1-\gamma}\sqrt{|W_{\max}|}}, \tag{10}$$

where γ is the confidence level.[3] Eq. (10) represents the upper limit of the confidence interval

[2] If the cycles are not removed from the forward ant's path, then the sub-path update will lead to statistical bias [5]. Thus, the sub-path update should be used only if the cycles are removed from the forward ant's path.

[3] Consider a sample of observations $\mathbf{X} = (X_1, \ldots, X_n)$. For any parameter δ defined for $\mathbf{X}$, find an interval $[l(\mathbf{X}), u(\mathbf{X})]$ such that $P[l(\mathbf{X}) \leqslant \delta \leqslant u(\mathbf{X})] = 1 - \phi$ i.e., the interval contains the true value of the parameter δ with probability $1 - \phi$. In such a case the interval $[l(\mathbf{X}), u(\mathbf{X})]$ is a $(1 - \phi) \times 100\%$ confidence interval and $1 - \phi$ is the confidence level [17].

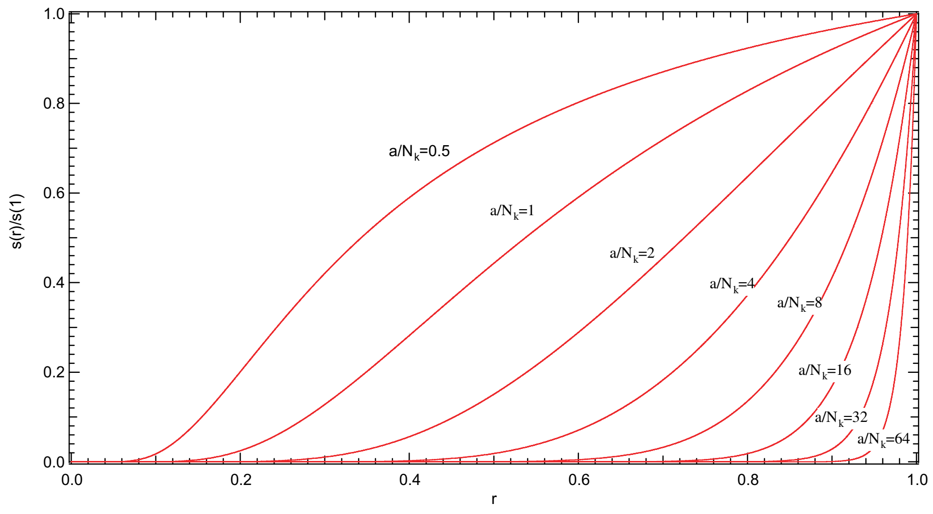

Fig. 4. The squash function $s(x)$ for different values of the coefficient $\frac{a}{N_k}$.

for the mean μ_d, assuming that the mean μ_d and the variance σ_d^2 are estimated over $W_{\max}$ samples [17]. There is some level of arbitrariness in choosing the confidence interval in (10) since the confidence interval is asymmetric and the mean μ_d and the variance σ_d are not arithmetic estimates [4]. The first term in (9) evaluates the ratio between the current trip time and the best trip time observed over the moving observation window. The second term is a correction factor and indicates how far the value of $t_{k\to d}$ is from t_{best_d} in relation to the extension of the confidence interval [4]. The values of c_1 and c_2 indicate the relative importance of each term. It is logical to assume that the first term in (9) is more important than the second term. Hence, the value of c_1 should be chosen larger than the value of c_2. The value r calculated in (9) is finally transformed by means of a squash function $s(x)$ defined by:

$$s(x) = \frac{1}{1+\exp\left(\frac{a}{x|N_k|}\right)}, \quad \text{where } x \in (0,1],$$

$$a \in \mathbb{R}^+, \tag{11}$$

$$r \leftarrow \frac{s(r)}{s(1)}. \tag{12}$$

The squash function $s(x)$ is introduced in the AntNet algorithm so that small values of r would have negligible effect in updating the routing tables [4]. Due to the squash function $s(x)$, the low values of r are reduced further, and therefore do not contribute in the update of routing tables.[4] The coefficient $\frac{a}{N_k}$ determines the dependence of squash function $s(x)$ on the number of neighbors N_k of the node k. Fig. 4 shows the effect of coefficient $\frac{a}{N_k}$ on the squash function $s(x)$. Fig. 4 shows that if the value of coefficient $\frac{a}{N_k}$ is less than 1, then even low values of r get incremented due to the squash function $s(x)$. Thus, the value of parameter a should be chosen such that the coefficient $\frac{a}{N_k}$ is greater than 1.

Data packets use different routing tables than the forward ants for travelling from the source node to the destination node. The routing table values for data packets are obtained by re-mapping the routing table entries used by forward ants by means of a power function $g(v)$ and re-normalizing these entries.

$$g(v) = v^\beta, \quad \beta > 1. \tag{13}$$

The power function $g(v)$ emphasizes the high probability values and reduces lower ones, and thus prevents the data packets from choosing links with very low probability. The data packets have a fixed time to live (TTL); if the data packets do not arrive at the destination within the TTL, they are dropped.

[4] Low values of r indicate sub-optimal paths.

4.2. Complexity analysis of the AntNet algorithm

We first calculate the complexity, defined as number of elementary operations, of a single forward ant to travel between a given source node and a given destination node in the AntNet algorithm. At every node along the path between a given source and destination node, the forward ant needs to search through the stack it maintains in the memory to find whether to use (5) for choosing the next node or to choose the next node uniformly among the neighbors. The worst-case complexity of searching through the stack is O(1), if the stack is implemented as a combination of linked list and an additional array or an hash table. Further, the complexity of (5) is $O(N_k)$ since the probability values have to be calculated for each of the N_k neighbors. In unfNet, the forward ants choose the next node uniformly among the neighbors of the node, and therefore the above operations are not required. There are other computations that a forward ant performs at each node: the forward ant needs to push the identifier of the current node and the time at which it arrived into the stack which is O(1), the forward ant goes to the queue for one of the outgoing links which is O(1). Let the maximum hopcount of the forward ant be M. In the worst case, the forward ant has to do all the computations at each of the M nodes. Thus, the worst-case complexity for a single forward ant to travel between a given source node and a destination node in AntNet is $O(MN_k)$, while in unfNet it is $O(M)$.

The worst-case complexity for a single backward ant to travel between a given source node and a destination node is also $O(MN_k)$. This can be calculated as follows. The backward ant performs three operations at each node along the path it travels between a given source node and a given destination node. First, the backward ant needs to pop the stack to find out the next node to travel. The pop operation in the stack is O(1). Second, the backward ant goes to the queue for one of the outgoing links which is O(1). The backward ant updates the routing table for each of the neighbors N_k for the destination d which is $O(N_k)$. Furthermore, in the worst case the backward ant has to do all the computations at each of the M nodes.

We calculate the worst-case complexity of AntNet when the total number of forward or backward ants generated is given by $\rho > 1$. Since the worst-case complexity of a single forward or backward ant to travel between a given pair of nodes is $O(MN_k)$, the worst-case complexity of AntNet when ρ forward or backward ants are generated is $O(\rho MN_k)$. Furthermore, we know that the worst-case complexity for Dijkstra's shortest path algorithm using a Fibonacci heap is $O(N\log N + L)$. This analysis shows that the complexity of AntNet algorithm is comparable to Dijkstra's' shortest path algorithm when M and N_k are small as compared to N.

We now calculate the worst-case complexity of AntNet for finding the shortest path. Let us assume that one forward ant searches for one distinct path and one forward/backward ant pair updates only the routing tables at the source node for the given destination node. In the worst case, at any given node there is an equal probability of creating a forward ant with any one of the $N-1$ nodes as the destination. Thus, only one out of $N-1$ forward ants is used to search for the shortest path between a given source and destination. Let us denote the number of paths between a source and a destination in $G(N,L)$ by m. Adding the contributions yields a worst-case complexity to search for the shortest path in AntNet (C_{AntNet}) with $m = m_{\max}$ of

$$C_{\text{AntNet}} = O(mMN_kN), \tag{14}$$

where $m_{\max}$ is the upper bound on the maximum number of paths between a source and destination node in $G(N,L)$. The upper bound $m_{\max}$ on the total number of paths between a source and destination node in $G(N,L)$ is [9]

$$m_{\max} = [e(N-2)!]. \tag{15}$$

In case of a random graph $G_p(N)$, with a fixed link density p and N nodes, the following upper-bound for the average number of paths applies [15],

$$m_{\max} = \left[p^{N-1}e^{\frac{1}{p}}(N-2)!\right], \tag{16}$$

Eq. (16) shows that the number of paths between a given pair of nodes decreases as the link density p in the random graph $G_p(N)$ is decreased.

4.3. AntNet implementation

The time $T_{i\to j}$ for a data packet or ant packet (forward ant or backward ant) to travel from a node i to a node j is calculated as:

$$T_{i\to j} = \frac{q_j + \text{size}_{\text{Packet}}}{C_{i\to j}} + D_{i\to j}. \tag{17}$$

where q_j is the length of the low priority queue at node i for the link $i \to j$, $\text{size}_{\text{Packet}}$ is the size of the packet, $C_{i\to j}$ is the available capacity of the link between node i and node j and $D_{i\to j}$ is the propagation delay of the link $i \to j$. The term $\frac{\text{size}_{\text{Packet}}}{C_{i\to j}}$ represents the transmission delay for the packet and the term $\frac{q_j}{C_{i\to j}}$ represents the queuing delay experienced by the packet while waiting at node i for the link $i \to j$. The term propagation delay and link weight have been used interchangeably in the text.

We have simplified the model shown in Fig. 1 for our implementation of the AntNet algorithm. We assume that the packets go directly to the outgoing buffer and the outgoing buffer consists of a low priority and a high priority queue for all the outgoing links. But the queuing delay in (17) and parameter l_n in (5) are still calculated using the number of packets waiting in the low priority queue for a particular link.[5] Further, we assume that each node is able to remove one packet from its high priority queue and low priority queue in the output buffer (for any outgoing link) at a rate of 0.01 milliseconds (ms). This assumption makes the queueing delays negligible in most of our simulations.

We implement the AntNet algorithm under static and dynamic conditions. In the static implementation of AntNet, we assume that the forward ants use only the propagation delays for network exploration and routing table updates. Thus, no queueing or transmission delays that depend on the packet size and capacity of the links are taken into account during the static implementation of AntNet. In this case, the Eq. (5) reduces to $p'_{nd} = p_{nd}$ and Eq. (17) for forward ant packets reduces to $T_{i\to j} = D_{i\to j}$. In the dynamic implementation of AntNet, both the queueing and transmission delays as well as the propagation delays are used for choosing the next node and updating the routing tables.

We compare the results of AntNet with Dijkstra's shortest path algorithm. However, under dynamic implementation, there is an important distinction for comparison of AntNet to Dijkstra's shortest path algorithm. In Dijkstra's shortest path algorithm, the cost of the shortest path is the sum of individual link weights (propagation delays) along the path. However in AntNet, the cost of the shortest path used for updating the routing tables is assumed to be the sum of link weights along the path and the transmission and queuing delays. The delay in the queues is included in the AntNet algorithm to account for the traffic conditions in the network. Thus, there is no static shortest path in the AntNet algorithm. However, for the comparison of AntNet to Dijkstra's shortest path algorithm, we plot the weight of the paths in the AntNet algorithm excluding the transmission and queuing delays.

To validate our simulations, in addition to plotting the pdf of the hopcount for the shortest path and the paths in the AntNet algorithm, we also plot the values for pdf of hopcount of the shortest path obtained by using theory. It has been demonstrated by van der Hofstad et al. [8] that for a fixed link density p and sufficiently large N, the shortest path tree in a random graph $G_p(N)$ with uniformly distributed link weights is an uniform recursive tree (URT). The probability density function of the hopcount in the URT with N nodes is:

$$\Pr[H_N = k] = \frac{(-1)^{N-(k+1)} S_N^{(k+1)}}{(N-1)(N-1)!}, \qquad (18)$$

where $S_N^{(k)}$ is the Stirling number of the first kind [1].

5. Results

5.1. Simulation parameters

The simulations are performed on random graphs of the class $G_p(N)$ consisting of N nodes and independently chosen links with probability p. The link weights reflecting the propagation delays are uniformly distributed in (0,1] ms and the capacity of each link is 8.192 Mbit/s for all our simulations. For each random graph of the type $G_p(N)$, we used Dijkstra's algorithm to compute the shortest path between the source node and the destination node based on the link weights. On the other hand, in AntNet, the data packets travel between a given pair of nodes using the probabilistic routing tables. Indeed, there could be a number of paths that the data packets choose which might be the shortest path, the second shortest path etc. For AntNet, we find the average link weight (or end-to-end delay) and the average hopcount of the paths that the data packets use to travel between the source and the destination node over the entire simulation period. Furthermore, each simulation consists of large number (10^5 or 10^4) of random graphs of the

[5] The packets in the high priority queue and low priority queue are served in FIFO order.

type $G_p(N)$. The source (node 1) and destination (node N) are chosen to be fixed in our simulations.

In our implementation of the AntNet algorithm, the simulation period (SP) consists of the training period (TP) and the test period (TEP). During TP, only ant packets are generated while during TEP, both the data and ant packets are generated. We have chosen the simulation period (SP) to be 10^4 ms. The training period (TP) has been chosen to be 10^3 ms. Each node generates data packets according to a Poisson process with mean interarrival time of 12.5 ms. The destination for data packets is chosen randomly. The size of data packets generated follows a negative exponential distribution with mean of 4096 bits. The size of forward ant packet is assumed to be 192 bits at the time the forward ant packet is generated and the size increases by 64 bits for each hop the forward ant travels. The size of backward ant is assumed to be 500 bits. The average number of neighbors of a node in a random graph $G_p(N)$ is $p(N-1)$, the default value of parameter a is chosen as $p*N$ in our simulations. The maximum size of the output buffer, which is assumed to be the sum of the sizes of all low priority queues in the output buffer, is 10^9 bits. Table 1 lists the various parameters and their default values that we have chosen for our simulations. The values of parameters α, c_1, c_2, a, confidence interval (γ) and the size of forward ant packet have been replicated from [4]. The values of the parameter η, the maximum size of the observation window $W_{\max}$ and the ant generation rate have been varied extensively and therefore are not listed in Table 1.

Table 1
Various parameters and their default values used in the simulations

Name, symbol	Value
Link capacity $C_{i\to j}$	8.192 Mbit/s
Link weight (propagation delay)	Uniformly distributed (0,1] ms
Simulation period (SP)	10^4 ms
Test period (TEP)	9.10^3 ms
Training period (TP)	10^3 ms
Backward ant size	500 bits
Initial forward ant size	192 bits
α used in (5)	0.2
ε	0.1
c_1 used in (9)	0.7
c_2 used in (9)	0.3
a used in (11)	$p*N$
β used in (13)	3
Confidence interval (γ) used in (10)	0.95
Output buffer size	10^9 bits
Mean interarrival time for data packets	12.5 ms
Data packets TTL	10 ms
Mean data packet size	4096 bits

5.2. Static implementation of the AntNet algorithm

In this Section, we study the AntNet algorithm under static conditions i.e., the forward ants use only the propagation delays for network exploration and routing table updates. Under these conditions, a direct comparison between the static shortest path calculated by using Dijkstra's algorithm and the AntNet algorithm paths can be made. The ant generation interval is 4 ms during TP and 40 ms during TEP. The TTL of data packets is assumed to be 20.0 ms. The value of parameters $\eta = 0.1$ and $W_{\max} = 50$. Fig. 5 shows the simulation results comparing the pdf of the hopcount and the weight for the shortest path and the AntNet algorithm paths[6] for $N = 25$, 50 and $p = 0.2$ (The default value of parameter ε is chosen as 0.).

Fig. 5 shows that the AntNet algorithm converges to a good solution. Furthermore, the performance of AntNet improves when the value of parameter ε is chosen greater than 0. Since no queuing delays are considered, the network exploration in AntNet is restricted by the routing tables and many paths are not explored (The value of p'_{nd} in Eq. (5) is equal to p_{nd}). With $\varepsilon = 0.1$, the probability of exploring different paths increases leading to an improvement in the performance of AntNet. Indeed for static implementation, unfNet performs better than AntNet using $\varepsilon < 1$ since all the paths are explored with equal probabilities in unfNet.

5.3. Dynamic implementation of the AntNet algorithm

In this section, the AntNet algorithm implementation is dynamic but we compare the results with Dijkstra's shortest path algorithm. We also study the effect of different parameters, such as ant gener-

[6] The probability density function (pdf) of the hopcount and the weight for the AntNet algorithm paths and the shortest path are computed over (10^4) random graphs of the type $G_p(N)$. In AntNet, for each random graph of the type $G_p(N)$, the path weight and hopcount represent the "average path weight" and "average hopcount" for all the packets to travel between the source and destination node during the entire simulation period.

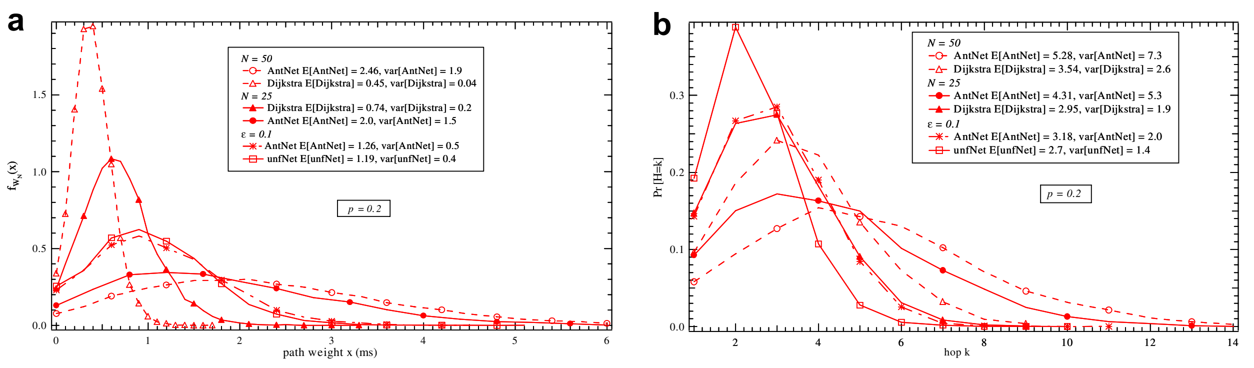

Fig. 5. The pdf of the (a) weight and the (b) hopcount of the Dijkstra's shortest path, unfNet algorithm paths and the AntNet algorithm paths for $N = 25$ and $N = 50$ for $p = 0.2$. The ant generation interval during the TP is 4 ms. Each simulation consists of 10^4 iterations ($\eta = 0.1$, $W_{\max} = 50$).

ation rate, squash function $s(x)$, parameter β etc., on the performance of AntNet. The performance comparison between AntNet and Dijkstra's algorithm and the optimization of various parameters for AntNet is still valid since the transmission delays are very small for forward ant packets. Furthermore, the queueing delays are also small since packets are removed from nodes at a very fast rate as compared to the rate at which packets are generated. Thus, the total delay experienced by forward ant packets is still dominated by the propagation delays. The TTL of data packets is assumed to be 10.0 ms and the value of parameter ε is 0.1 in all the simulations in this section.[7]

5.3.1. The effect of the ant-generation rate and the link density p

We first compare the performance of AntNet with Dijkstra's shortest path algorithm for different ant-generation rates and different values of the link density p for a 25 node network.

The values of TP and TEP are assumed to be constant in our simulations as shown in Table 1. To study the effect of ant generation rate on the AntNet algorithm, we assume a fixed ant generation rate during the TEP only varying the ant generation rates during the TP. We consider three different cases of the ant generation during the TP, namely 40, 4, 0.4 ms.[8] The corresponding results for the pdf of the hopcount and the weight for $N = 25$ and different values of the link density p are shown in Figs. 6–8.

Figs. 6–8 show that the AntNet algorithm gives a near optimal solution for a 25 node network at low values of the link density p ($p = 0.2$ and $p = 0.1$). The performance of AntNet algorithm decreases as the value of the link density p in the random graph $G_p(N)$ is increased. Additional simulations for $N = 50$ show similar trends as the link density p is varied. Moreover, the comparison of AntNet for $N = 25$ and $N = 50$ shows that the performance of AntNet degrades as the network size N is increased. Fig. 7 shows the validity of our comparison between AntNet and the Dijkstra's algorithm since the results for AntNet are similar to the results of AntNet under static implementation (Fig. 5).

Comparison of Fig. 6 with Figs. 7 and 8 show that the AntNet algorithm converges to a good solution even at low ant generation rates. In Fig. 6, the ant generation interval is 40 ms during both the TP and the TEP. Thus, only 25 forward ants are generated by each node during the TP to search for the shortest paths to all other nodes in the network. The number of forward ants searching for the shortest path increases as the ant generation interval is decreased from 40 ms to 4 ms during the TP. This leads to an improvement in the performance of AntNet. As the ant generation interval is decreased from 4 ms to 0.4 ms, the performance of AntNet remains same. This can be attributed to the fact that the variations in the ant generation rate are related to the size of the moving observation window $W_{\max}$ and the parameter η. The size of the moving observation window $W_{\max}$ in above simulations is large enough to distinguish between ant

[7] In dynamic implementation of AntNet, there is negligible difference between the performance of AntNet for $\varepsilon = 0$ and $\varepsilon = 0.1$.

[8] $\eta = 0.1$, $W_{\max} = 50$ for all the simulations in this sub-section.

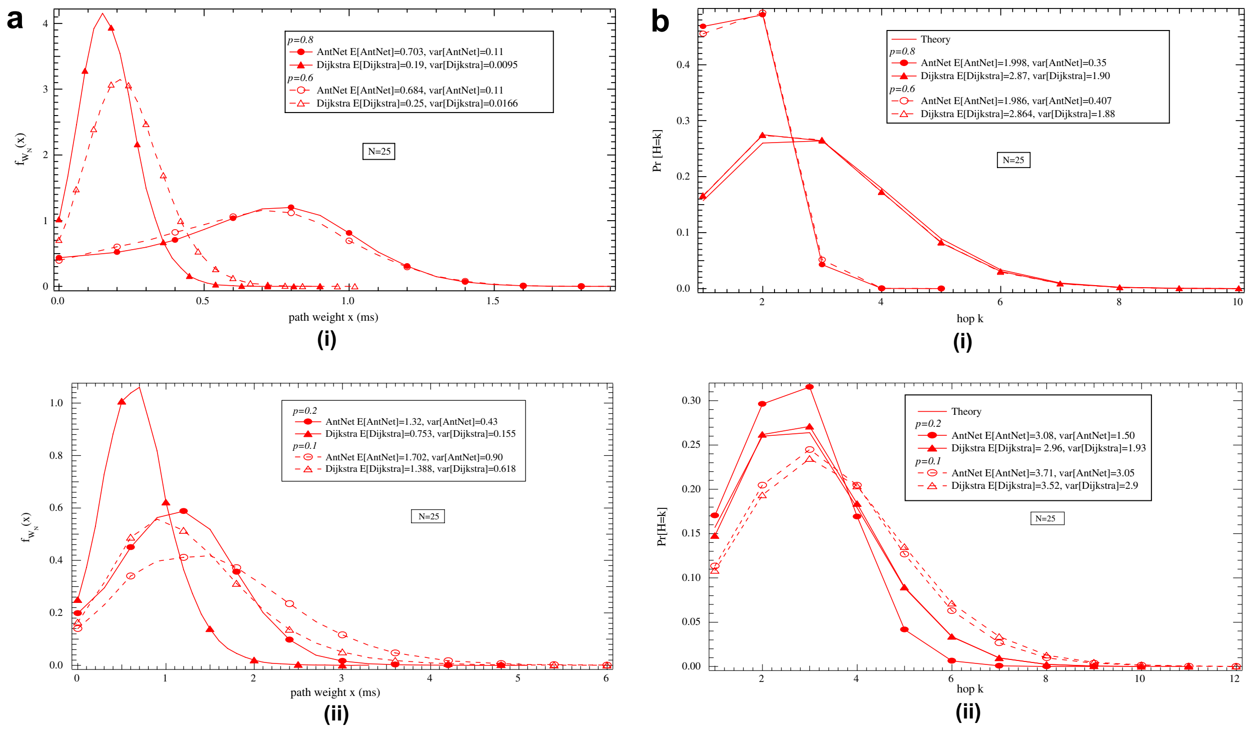

Fig. 6. (a) The pdf of the weight of the Dijkstra's shortest path and the AntNet algorithm paths (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. (b) The pdf of the hopcount of the Dijsktra's shortest path, AntNet algorithm paths and obtained by theory (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. The ant generation interval during TP is 40 ms. (Each simulation consists of 10^5 iterations.)

generation intervals of 40 ms and 4 ms during TP. But since the size of the moving observation window is small, it can store only the times of last 50 forward ants generated from a given source to a given destination. This indicates that even if the ant generation rate is increased with a small window size, the performance of AntNet remains same or may even go down. Additional simulations show that the AntNet algorithm performs better at $\eta = 0.02$ and $W_{\max} = 200$ than at $\eta = 0.1$ and $W_{\max} = 50$ for different ant generation rates and different values of the link density p.

We also performed simulations for the unfNet algorithm under identical conditions, as above. Table 2 lists the simulation results for the unfNet algorithm for $N = 25$ and different values of the link density p ($p = 0.2$ and 0.8). The results show that unfNet performs worse than AntNet for low ant generation rates (or large values of the ant generation interval). In unfNet, all the paths are continuously searched independently of the routing tables. While in AntNet, the probability that future forward ants choose paths with large delays is decreased by each forward ant. Therefore, in AntNet, mainly low delay paths contribute to routing table updates and the AntNet algorithm performs well even with a low ant generation rate.[9] When the ant generation rate is high, the performance of unfNet is comparable to the AntNet algorithm. This can be attributed to the fact that the data traffic is small and the variations in delays along different paths is negligible.

5.3.2. Lattice topologies

In this Section, we investigate the performance of AntNet for lattice topologies. We compare the performance of AntNet with Dijkstra's shortest path algorithm for the lattice topology shown in Fig. 9. The source and destination nodes D1 and D2 are assumed to be fixed. The ant generation interval is 4 ms during TP and 40 ms during TEP ($\eta = 0.1$, $W_{\max} = 50$. The value of parameter a is chosen as 5.) Fig. 10 shows the simulation results comparing the pdf of the hopcount and the weight for the

[9] All paths searched by the forward ants lead to positive reinforcement.

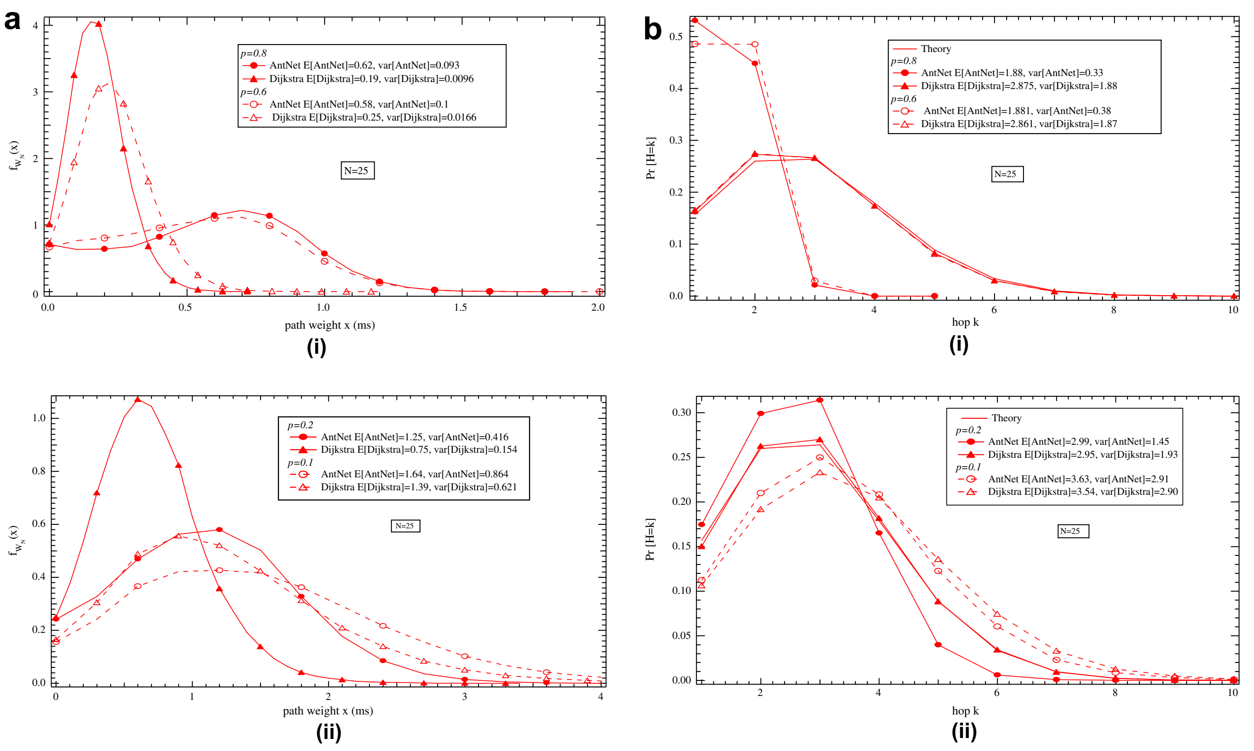

Fig. 7. (a) The pdf of the weight of the Dijkstra's shortest path and the AntNet algorithm paths (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. (b) The pdf of the hopcount of the Dijsktra's shortest path, AntNet algorithm paths and obtained by theory (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. Same scenario as in Fig. 5 but with an ant generation interval of 4 ms during the TP.

shortest path and the AntNet algorithm paths. Fig. 10 shows that the AntNet algorithm performs well for lattice topologies. Also, the performance of AntNet degrades as the number of nodes in the network is increased.

5.3.3. The effect of the moving observation window size W_{max} and the parameter η

In AntNet, the moving observation window is used to store the cost of the shortest path from a given source node to a given destination node. Furthermore, the moving observation window is also used to improve the accuracy of mean μ_d as shown in (10). The optimal size of the moving observation window is hard to determine because it is linked to the ant generation rate and the parameter η. For determining the optimal value of $W_{\max}$ it is not sufficient to calculate the number of forward/backward ants alone. This is due to two reasons. First, each forward ant does not search for a distinct path. Second, the backward ants update the routing tables at each of the nodes along the path for the given destination as they travel from the destination node to the source node. Under ideal conditions, if each ant searches for one distinct path, the size of the moving observation window should be equal to or greater than the number of paths between any pair of nodes in the network. Thus, at any instant of time the moving observation window would contain the shortest path. Indeed, the size of the moving observation window can be effectively chosen in small networks where the number of paths between a pair of nodes in the network is small. Our simulations show that increasing the value of $W_{\max}$ without changing other parameters such as the ant generation rate and η leads to a small improvement in the performance of AntNet.

The parameter η is used to estimate the mean μ_d and the variance σ_d^2 by using the exponential model as shown in (1) and (2). The parameter η represents how many of the previous forward ant's trip times effect the mean or average value. In the exponential model, the weight of the current forward ant's trip time to a destination t_z after m forward ants for the particular destination have been received is $\eta(1-\eta)^{m-z}$. The smaller values of η indicate that the mean value is calculated over a large number of forward ant trip time samples. Our simulations

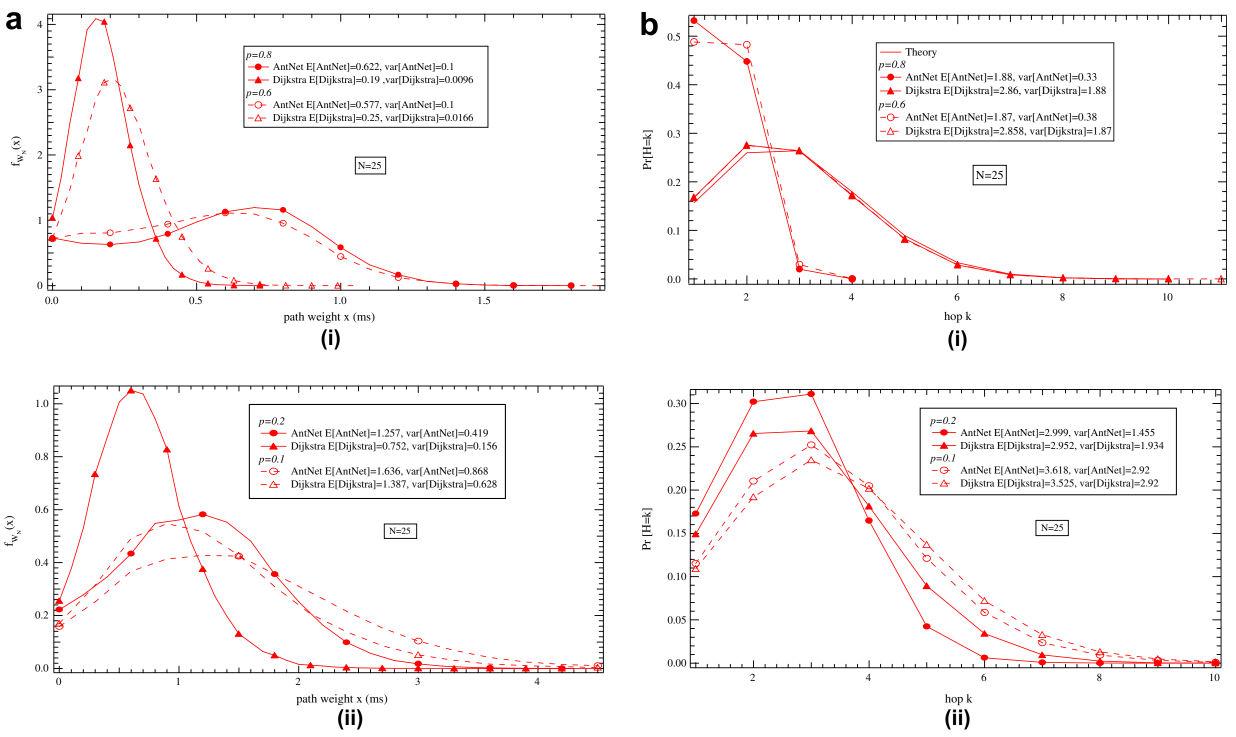

Fig. 8. (a) The pdf of the weight of the Dijkstra's shortest path and the AntNet algorithm paths (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. (b) The pdf of the hopcount of the Dijsktra's shortest path, AntNet algorithm paths and obtained by theory (i) $p = 0.8$ and $p = 0.6$ (ii) $p = 0.2$ and $p = 0.1$. Same scenario as in Fig. 5 but with an ant generation interval of 0.40 ms during the TP.

Table 2
The mean hopcount and weight for the unfNet algorithm for $N = 25$

Link density p	Ant generation interval during TP (ms)	E [path weight] (ms)	E [Hopcount]
0.8	40	0.75	1.82
0.8	4	0.63	1.62
0.8	0.4	0.59	1.55
0.2	40	1.38	2.96
0.2	4	1.2	2.67
0.2	0.4	1.16	2.61

$p = 0.2$ and 0.8. $\eta = 0.1$ and $W_{\max} = 50$.

show that changing η from 0.1 to 0.02 does not effect the performance of AntNet significantly. This indicates that at light traffic loads, the paths between pair of nodes in the AntNet algorithm quickly converge to the mean value for that path making the second term in (1) redundant. The variations in parameter η might effect the performance of AntNet, if the traffic conditions or topology of the network varies. Under such conditions, the path between a pair of nodes will deviate significantly from the mean value. Thus, for a static topology

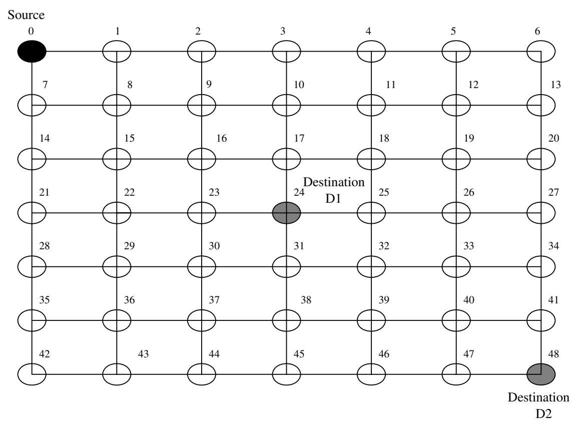

Fig. 9. The 49-node lattice topology used for our simulations. The source node and the destination nodes D1 and D2 are assumed to be fixed as indicated.

with low traffic loads, the AntNet algorithm is very robust to changes in the parameter η.

Finally, to study the combined effect of the parameters η, $W_{\max}$ and the ant generation rates on the performance of AntNet, we set the ant generation to a very high value i.e., the ant generation

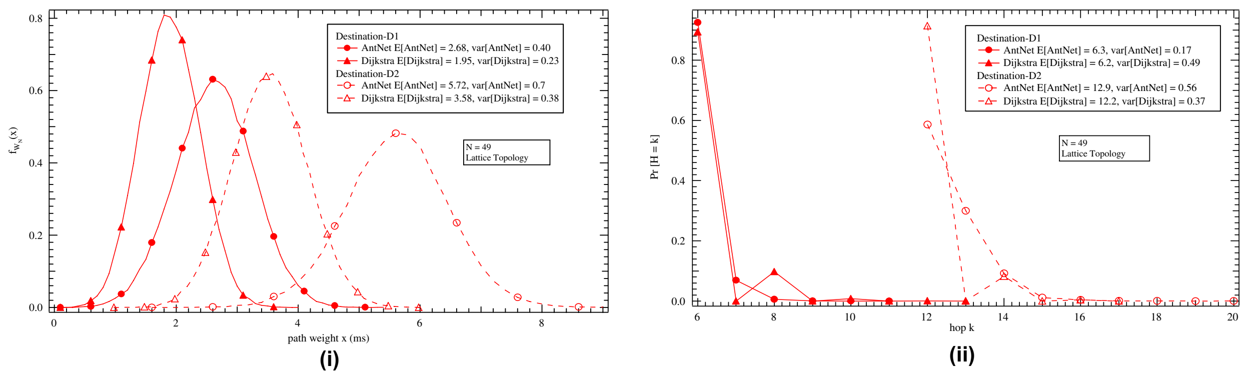

Fig. 10. The pdf of the (i) weight and the (ii) hopcount of the Dijkstra's shortest path and the AntNet algorithm paths for a 49 node lattice topology. The source node and destination nodes D1 and D2 are fixed as shown in Fig. 9. Each simulation consists of 10^5 iterations.

interval is 0.4 ms during the TP and 4 ms during the TEP. We also choose a very large size of the moving observation window i.e., $W_{\max} = 16{,}000$. The value of $W_{\max}$ is sufficient to store the trip times of all forward ants generated by the given node as well as the trip times of forward ants received due to the subupdate method. Furthermore, the value of parameter η is chosen as very small i.e., $\eta = 0.001$ so that large number of sample trip times of forward ants are used to calculate the mean and variance in (1) and (2). Fig. 11 presents the simulation results comparing the pdf of the hopcount and the weight for the shortest path, and the unfNet and AntNet algorithm paths for different values of $W_{\max}$ for $N = 25$ and $p = 0.8$.

Fig. 11 shows that the performance of AntNet can be improved by varying the parameters η, $W_{\max}$ and the ant generation rates in conjunction with each other. Thus, the inherent coupling between ant generation rate and the parameters η and $W_{\max}$ contributes to the complexity and robustness of the AntNet algorithm. Fig. 11 also shows that the performance of unfNet is worse than AntNet, even though the data traffic is small. In unfNet, the forward ants move independently of the routing tables and all the paths are searched with equal probabilities. In AntNet, if the path is found to incur a large delay, the probability of choosing the particular path by the future forward ants becomes less. But since every path found receives a positive reinforcement (in both unfNet and AntNet), the number of non-optimal paths updating the routing tables in unfNet is more.

5.3.4. The effect of the confidence interval, squash function $s(x)$ and the parameter β

We first study the effect of confidence interval on the performance of AntNet. The second term in (10) i.e., $\frac{\sigma_d}{\sqrt{1-\gamma}\sqrt{|W_{\max}|}}$ is used to improve the accuracy in estimation of the mean μ_d but introduces additional complexity in the AntNet algorithm. Fig. 12 shows the simulation results comparing the pdf of the hopcount and the weight for the Dijkstra's shortest path

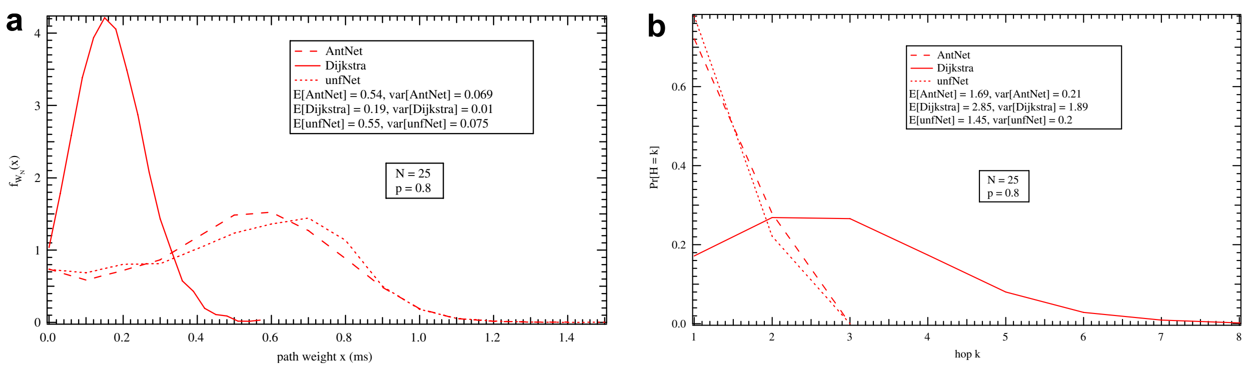

Fig. 11. The pdf of the (a) weight and the (b) hopcount of the Dijkstra's shortest path, unfNet algorithm paths and the AntNet algorithm paths for $N = 25$ and $p = 0.8$ ($\eta = 0.001$, $W_{\max} = 16{,}000$). Each simulation consists of 10^4 iterations.

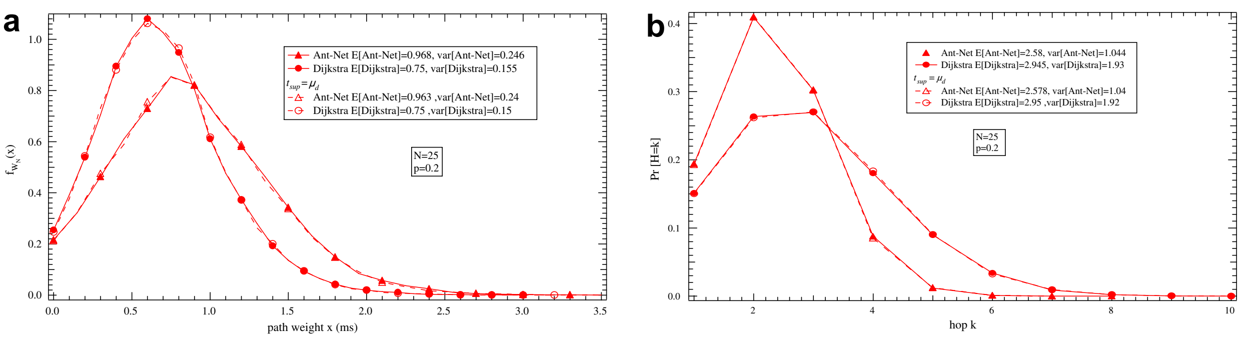

Fig. 12. The pdf of the (a) weight and the (b) hopcount of the Dijkstra's shortest path and the AntNet algorithm paths for $N = 25$ and $p = 0.2$ for $t_{\text{sup}} = \mu_d$ and t_{sup} calculated using (11) ($\eta = 0.1$, $W_{\max} = 50$, $a = 20$). The ant generation interval is 40 ms during the TP and TEP. Each simulation consists of 10^5 iterations.

and the AntNet algorithm paths, when the term $\frac{\sigma_d}{\sqrt{1-\gamma}\sqrt{|W_{\max}|}}$ is not used in the estimation of t_{sup}. This case is shown as $t_{\text{sup}} = \mu_d$ in Fig. 12. The simulations are performed for $N = 25$ and $p = 0.2$. Also, the value of parameter a is chosen as 20 such that the average value of the coefficient $\frac{a}{N_k}$ is 4. (The ant generation interval is 40 ms during TP and TEP. The value of parameters $\eta = 0.1$ and $W_{\max} = 50$.)

Comparison of Figs. 12 and 6 shows that choosing a large value of parameter a improves the performance of AntNet. Furthermore, under the simulation parameters considered, the removal of the term $\frac{\sigma_d}{\sqrt{1-\gamma}\sqrt{|W_{\max}|}}$ does not change the performance of AntNet. Thus, Eq. (10) could be simplified to reduce the complexity of AntNet.

The parameter β determines whether single or multi-path routing is followed by the data packets. The value of parameter β needs to be greater than 1 to prevent the data packets from choosing links with very low probabilities. A large value of β ($\beta \gg 1$) indicates that the data packets follow only single-path routing. On the other hand $\beta = 1$ indicates that the data packets follow the routing tables and may even choose links with very low probabilities. Thus, $\beta = 1$ corresponds to multi-path routing. We compare the performance of AntNet algorithm for $\beta = 1$ and $\beta = 100$. When $\beta = 100$, the data packets are effectively following a single-path routing. Fig. 13 shows the simulation results comparing the pdf of the hopcount and the weight for the shortest path and the AntNet algorithm paths for $p = 0.1$ for $\beta = 1$ and $\beta = 100$ for a 25 node network.[10]

Fig. 13 shows that the performance of AntNet improves as β is increased for $p = 0.1$. This shows that there is a greater probability that the shortest path has been correctly identified in the AntNet algorithm at lower values of p. Thus, making $\beta = 1$ prevents the data packets from choosing the path with the highest probability and leads to data packets following sub-optimal paths. On the other hand, $\beta = 100$ improves the performance of AntNet since the data packets follow only a single-path routing which has a greater probability of being the shortest path also.

5.4. Traffic measurements

In this Section, we compare the end-to-end delays for AntNet and Dijkstra's shortest path algorithm. A set of randomly chosen nodes are congested during each iteration and have an additional queueing delay of 10 ms for every packet. To make a fair comparison between the algorithms, we assume that the size of all data packets is same i.e., 4096 bits and the TTL for data packets is set to infinity. In addition to the normal data, we send a small number of data packets along the shortest path from the source to the destination node[11] (source routing or src_rt). Fig. 14 shows the simulation results comparing the pdf of the hopcount and the end-to-end delay (weight) for the Dijkstra's shortest path and the AntNet algorithm paths for $N = 50$ and $p = 0.1$. (The ant generation interval is 4 ms during TP and 40 ms during TEP. The value of parameter $\eta = 0.1$ and $W_{\max} = 50$.)

[10] The ant generation interval is 40 ms during the TP and the TEP. The value of parameter $\eta = 0.1$ and $W_{\max} = 50$.

[11] Node 1 is the source node and node 25 is the destination node. In AntNet, the data packets are generated at a uniform interval of 5 ms, while in src_rt, the data packets are generated at a uniform interval of 100 ms.

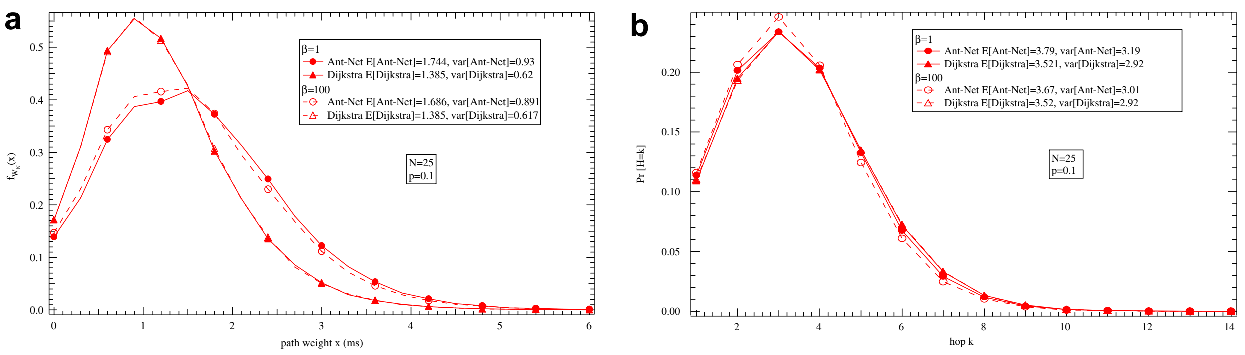

Fig. 13. The pdf of the (a) weight and the (b) hopcount of the Dijkstra's shortest path and the AntNet algorithm paths for $\beta = 1$ and $\beta = 100$ for $N = 25$ and $p = 0.1$. The ant generation interval is 40 ms during the TP and TEP. Each simulation consists of 10^5 iterations ($\eta = 0.1$ and $W_{\max} = 50$).

Fig. 14 shows that the AntNet algorithm performs better than single shortest path routing in terms of end-to-end delay. In src_rt, since Dijkstra's algorithm is used to compute the shortest path from the source to the destination, the queueing delays are not considered. During the iterations when there are no congested nodes along the shortest path, the end-to-end delays for the data packets using src_rt and AntNet are comparable. However, when there are one or more congested nodes along the shortest path, the data packets using src_rt incur queueing delays along the congested nodes. On the other hand, the AntNet algorithm performs load balancing and reduces the probability of data packets choosing the paths with congested nodes.

Table 3 lists the simulation results for the AntNet, unfNet and the src_rt algorithms for $N = 50$ and $N = 25$ ($p = 0.2, 0.1$). We consider different number of congested nodes and ant generation rates. Table 3 shows that unfNet generally performs worse than AntNet under varying traffic loads. The performance of unfNet becomes worse as compared to the AntNet algorithm, when the ant generation rate is low and the size of the network is increased. In AntNet, the network exploration is restricted to paths that incur low delays and only these paths are used to update the routing tables. In unfNet, the forward ants move independently of the routing tables. Thus, even the large delay paths are used to update the routing tables leading to a poor performance of the unfNet algorithm. This shows that the unfNet algorithm reduces the complexity but, in general, leads to a decrease in the performance.

To further study the load balancing capabilities of AntNet, we assume that during each iteration, a set of five randomly chosen nodes are congested

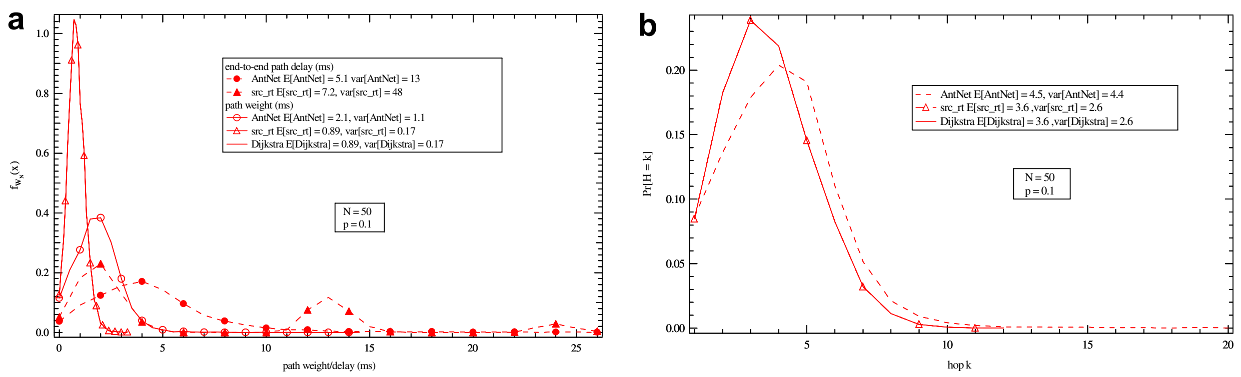

Fig. 14. The pdf of the (a) end-to-delay/weight and the (b) hopcount of the Dijkstra's shortest path, AntNet algorithm paths and the source routing path for $N = 50$ and $p = 0.1$. The ant generation interval is 4 ms during the TP and 40 ms during the TEP ($\eta = 0.1$, $W_{\max} = 50$). Each simulation consists of 10^4 iterations.

Table 3
The expected hopcount/weight for the unfNet, AntNet, and the src_rt algorithms for $N = 25$, 50 ($p = 0.2$,0.1) with different number of congested nodes and different ant generation rates ($\eta = 0.1$ and $W_{\max} = 50$)

Routing protocol	Network size (N)	Link density p	Ant generation interval (ms)		Congested nodes	E [path delay] (ms)	E [hopcount]
			TP	TEP			
unfNet	50	0.1	4	40	8	5.6	3.8
AntNet	50	0.1	40	40	8	5.8	4.7
unfNet	50	0.1	40	40	8	12.7	6.2
AntNet	25	0.2	4	40	4	3.4	3.2
src_rt	25	0.2	4	40	4	5.8	2.9
unfNet	25	0.2	4	40	4	3.5	2.8
AntNet	25	0.2	40	40	4	3.7	3.4
unfNet	25	0.2	40	40	4	5.5	3.3
AntNet	25	0.2	4	4	4	3.36	3.3
unfNet	25	0.2	4	4	4	3.1	2.8
AntNet	50	0.2	4	40	4	3.1	3.3
src_rt	50	0.2	4	40	4	4.3	3.5

after 5000 ms of the SP. For each iteration, we compute the packet delay and the hopcount averaged over 250 ms moving windows. Fig. 15 shows the simulation results for the packet delay and hopcount for $N = 50$ and $p = 0.1$ averaged over 10^4 iterations (The ant generation interval is 4 ms during TP and TEP. The value of parameters $\eta = 0.02$ and $W_{\max} = 200$). Fig. 15 shows how AntNet adapts to the introduction of congested nodes. Before 5000 ms there is small data traffic, and the average end-to-end delay and hopcount for the AntNet algorithm are constant. After the nodes become congested, there is a sudden increase in the end-to-end delay. However, the AntNet algorithm adjusts to the changing traffic leading to a decrease in the end-to-to-end delay. On the other hand in src_rt, a single shortest path is used for routing throughout the SP. As a result when the nodes become congested, the average end-to-end delay increases.

6. Conclusions

The AntNet algorithm performs well for random graphs and lattice topologies. The AntNet algorithm gives a near optimal solution at $p = 0.2$ and $p = 0.1$ for a 25 node network. This can be attributed to the fact that the number of paths between any given pair of nodes in the network at $p = 0.2$ and $p = 0.1$ is small. Since the AntNet algorithm starts searching for the shortest path in a random fashion,

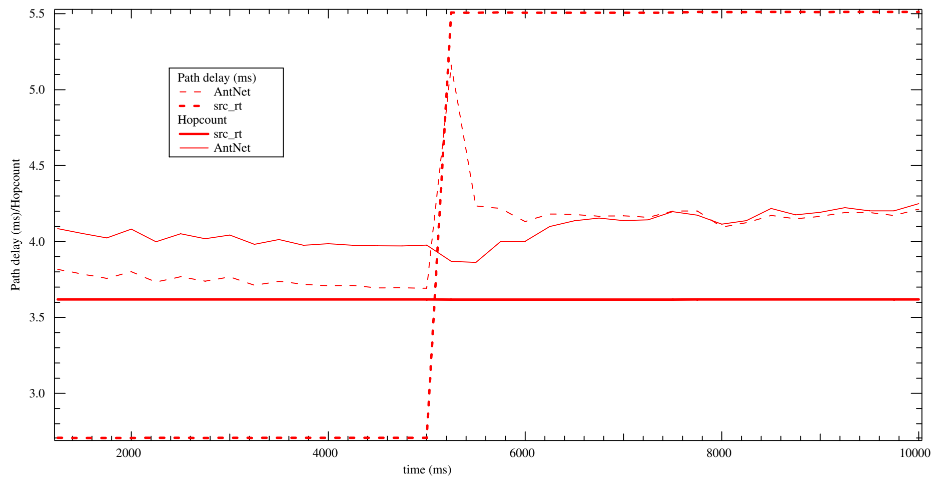

Fig. 15. Transient analysis of AntNet for $N = 50$ and $p = 0.1$. The path weight and delay are averaged over 250 ms moving windows and 10^4 iterations. The ant generation interval is 4 ms during the TP and TEP ($\eta = 0.02$, $W_{\max} = 200$).

the smaller the number of paths between a pair of nodes the greater is the probability that the algorithm converges to a near optimal solution. The performance of AntNet algorithm degrades as the network size or the link density p is increased.

The AntNet algorithm is robust to changes in the training of the network and converges to a good solution even at low ant generation rates. Further increasing the ant generation rates leads to an improvement in the performance but this is related to choice of other parameters such as $W_{\max}$ and η. Indeed, the coupling of various parameters is the inherent cause of complexity in the AntNet algorithm and changing one parameter favorably may not lead to an improvement in performance until the other parameters are also changed. The performance of AntNet can be improved by using a large value of the parameter a in the squash function $s(x)$. In this case, only the near optimal paths update the routing tables. The complexity of AntNet can be reduced by a number of methods, such as choosing parameter $\beta = 1$ and simplifying the calculation of parameter r, but this generally comes at the expense of performance.

Due to inherent load balancing, AntNet performs better than shortest path routing under varying traffic loads. For small networks, when the traffic loads are small and the ant generation is sufficiently high, a modified version of the AntNet algorithm (unfNet) can be used. This reduces the complexity of the AntNet algorithm. The robustness and near optimal performance of the AntNet algorithm makes it an attractive solution for routing in communication networks.

In future work, we will study the AntNet algorithm for dynamic networks such as ad hoc wireless networks. The scalability of AntNet would also be investigated. Furthermore, other modifications for the AntNet algorithm such as using both data and ant packets for updating the routing tables would also be considered. The AntNet algorithm would be considered a truly distributed and scalable algorithm, if the nodes are able to learn and adjust the various parameters to be used in the algorithm. This would also be investigated in future work.

Appendix

We include additional results for the AntNet algorithm under static conditions i.e., the forward ants use only the propagation delays for network exploration and routing table updates. The simulation parameters are assumed to be the same as Section 5.2 but the link weights reflecting the propagation delays are assumed to be 1 ms for all the links. Fig. 16 shows the simulation results comparing the pdf of the hopcount and the weight for the shortest path and the AntNet algorithm paths for $N = 25$, 50 and $N = 100$ for $p = 0.2$ and $p = 0.8$.

We show additional results for the AntNet algorithm under dynamic conditions (Section 5.3). Fig. 17 shows the simulation results comparing the pdf of the hopcount and the weight for the shortest path and the AntNet algorithm paths for $N = 50$ and $N = 100$ for $p = 0.1$. The ant generation interval is 40 ms during TP and TEP. (The value of parameters $\eta = 0.02$ and $W_{\max} = 200$.)

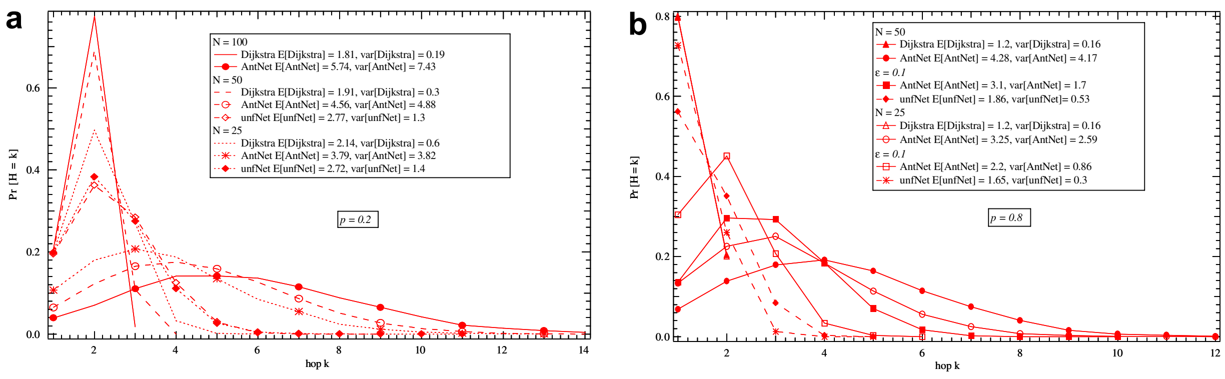

Fig. 16. (a) The pdf of the hopcount/weight of the Dijkstra's shortest path and the AntNet algorithm paths for $N = 25$, 50 and $N = 100$ and $p = 0.2$. (b) The pdf of the hopcount/weight of the Dijsktra's shortest path and the AntNet algorithm paths for $N = 25$, 50 and $p = 0.8$. The ant generation interval is 4 ms during TP and 40 ms during TEP ($\eta = 0.1$, $W_{\max} = 50$). Each simulation consists of 10^4 iterations.

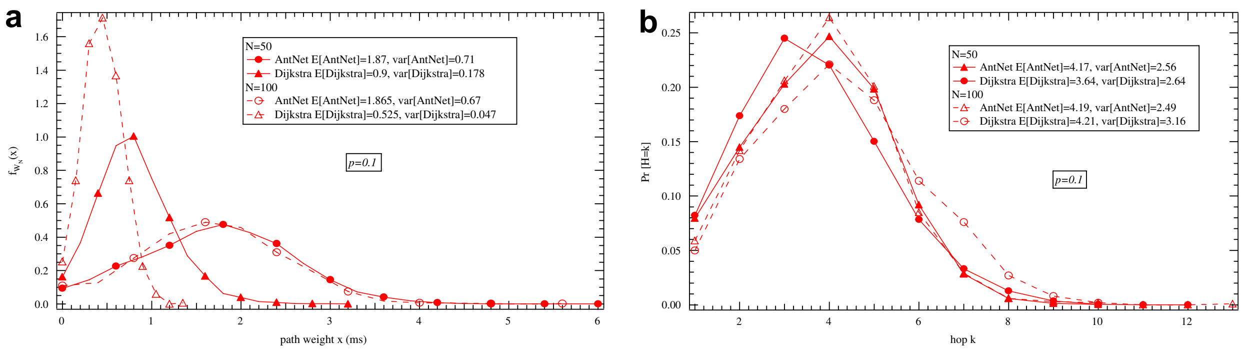

Fig. 17. The pdf of the (a) weight and the (b) hopcount of the Dijkstra's shortest path and the AntNet algorithm paths for $N = 50$, 100 and $p = 0.1$. The ant generation interval is 40 ms during TP and TEP ($\eta = 0.02$, $W_{\max} = 200$). Each simulation consists of 10^4 iterations for $N = 50$ and 10^3 iterations for $N = 100$.

References

[1] M. Abramowitz, I.A. Stegun (Eds.), A Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables, Dover, New York, 1974, ISBN 0486612724, June.

[2] Nigel Bean, Andre Costa, An analytic modelling approach for network routing algorithms that use "ant-like" mobile agents, Computer Networks 49 (2005) 243–268.

[3] Daniel Camara, Antonia A.F. Loureiro, Ants A novel routing algorithm for ad hoc networks, in: Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[4] Gianni Di Caro, Marco Dorigo, AntNet: distributed stigmergetic control for communication networks, Journal of Artificial Intelligence Research 9 (1998) 317–365.

[5] Andre Costa, Analytic Modelling of Agent-based Network Routing Algorithms, Ph.D. thesis, School of Applied Mathematics, University of Adelaide, 2003.

[6] Marco Dorigo, Thomas Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.

[7] Mesut Güneş, Otto Spaniel, Routing algorithms for mobile multi-hop ad-hoc networks, network control and engineering for QoS, security and mobility II, IFIP TC6/WG6.2 & WG6.7 Conference on Network Control and Engineering for QoS (Net-Con 2003), 2003. pp. 120–138.

[8] R. van der Hofstad, G. Hooghiemstra, P. Van Mieghem, First passage percolation on the random graph, Probability in the Engineering and Informational Sciences (PIES) vol. 15 (2001) 225–237.

[9] P. Van Mieghem, Performance Analysis of Computer Systems and Networks, Cambridge University Press, 2005.

[10] Martin Roth, Stephen Wicker, Termite: emergent ad-hoc networking, in: Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networking, June 2003.

[11] Gianni Di Caro, Marco Dorigo, Two ant colony algorithms for best-effort routing in datagram networks, in: Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems, 1998.

[12] Ruud Schoonderwoerd, Owen Holland, Janet Bruten, Ant-like agents for load balancing in telecommunication networks, in: Proceedings of the First International Conference on Autonomous agents, 1997.

[13] Devika Subramanian, Peter Druschel, Johnny Chen, Ants and reinforcement learning: a case study in routing in dynamic networks, in: Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networking, June 2003.

[14] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[15] P. Van Mieghem, Paths in the simple random graph and the Waxman graph, Probability in the Engineering and Informational Sciences (PIES) 15 (2001) 535–555.

[16] L.P. Kaeilbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.

[17] Alberto Leon-Garcia, Probability and Random Processes for Electrical Engineering, Addison-Wesley Publishing Company, 1994, ISBN 0-201-50037-X. 1996.

[18] R. Beckers, J.L. Deneubourg, S. Goss, Trails and U -turns in the selection of the shortest paths by the ant Lasius Niger, Journal of Theoretical Biology 159 (1992) 397–415.

[19] J.-L. Deneubourg, S. Aron, S. Goss, J.-M. Pasteels, The self-organizing exploratory pattern of the argenttine ant, Journal of Insect Behavior 3 (1990) 159–168.

[20] Marco Dorigo, Gianni Di Caro, L.M. Gambardella, Ant algorithms for discrete optimization, Artificial Life 5 (2) (1999) 137–172.

[21] F. Ducatelle, Gianni Di Caro, L.M. Gambardella, Ant agents for hybrid multipath routing in mobile ad hoc networks, in: Proceedings of the 2nd Annual Conference on Wireless On demand Network Systems and Services (WONS), 2005.

[22] M. Heusse, D. Snyers, S. Guerin, P. Kuntz, Adaptive agent-driven routing and load balancing in communication networks, Rapport technique de l'ENST de Brestagne, RR-98001-iasc, 1998.

[23] J.M. Pasteels, J.L. Deneubourg, S. Goss, Self-organization mechanisms in ant societies (I): trail recruitment to newly discovered food sources, Experientia Supplementum 54 (1987) 155–175.

[24] S. Rajagopalan, C. Shen, ANSI: a unicast routing protocol for mobile ad hoc networks using swarm intelligence, in: Proceedings of the International Conference on Artificial Intelligence, 2005. pp. 24–27.

[25] Ruud Schoonderwoerd, Owen Holland, Janet Bruten, Leon Rothkrantz, Ant-based load balancing in telecommunica-

tions networks, Journal of Adaptive Behavior 5 (2) (1996) 169–207.

[26] G. Theraulaz, E. Bonabeau, A brief history of stigmergy, Artificial Life (Special Issue on Stigmergy) (1999) 97–116.

[27] Marco Dorigo, Gianni Di Caro, The Ant Colony Optimization Meta-Heuristic, McGraw-Hill Press, 1999, pp. 11–32.

Piet Van Mieghem is professor at the Delft University of Technology with a chair in telecommunication networks and chairman of the basic unit Network Architectures and Services (NAS). His main research interests lie in new Internet-like architectures for future, broadband and QoS-aware networks and in the modelling and performance analysis of network behavior and complex infrastructures. He received a Master's and Ph.D. in Electrical Engineering from the K.U. Leuven (Belgium) in 1987 and 1991, respectively. Before joining Delft, he worked at the Interuniversity Micro Electronic Center (IMEC) from 1987 to 1991. From 1992 to 1993, he was a visiting scientist at MIT in the department of Electrical Engineering. During 1993–1998, he was a member of the Alcatel Corporate Research Center in Antwerp where he was engaged in performance analysis of ATM systems and in network architectural concepts of both ATM networks (PNNI) and the Internet. Currently, he is member of the editorial board of the journal Computer Networks.

Santpal S. Dhillon received the M.S. degree in Electrical and Computer Engineering from Duke University in May, 2003. He is currently pursuing his Ph.D. degree in the NAS group at Delft University of Technology. His work mainly focuses on routing protocols and algorithms.