Algorithms for Failure Protection in Large IP-over-Fiber and Wireless Ad Hoc Networks

Frederick Ducatelle[1], Luca Maria Gambardella[1], Maciej Kurant[2], Hung X. Nguyen[2], and Patrick Thiran[2]

[1] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Galleria 2, CH-6928 Manno-Lugano, Switzerland
Frederick@idsia.ch
[2] LCA - School of Communications and Computer Science
EPFL, CH-1015 Lausanne, Switzerland
Maciej.Kurant@epfl.ch,
Project home page: icawww.epfl.ch/kurant/hasler

Abstract. We address failure location and restoration in both optical and wireless ad hoc networks. First, we show how Maximum Likelihood inference can improve failure location algorithms in the presence of false and missing alarms. Next, we present two efficient algorithms for mapping an IP network on an optical network in such a way that it is protected against failures at the optical layer. The first algorithm offers a method to formally verify the existence of a solution, contrary to all other heuristics known to date. The second algorithm is a heuristic search that takes capacity constraints in account. Both algorithms are shown to be faster by orders of magnitude than existing solutions. Finally, we develop a new routing algorithm for wireless mobile ad hoc networks, adopting ideas from the Ant Colony Optimization metaheuristic. The routing scheme can adapt to network and traffic changes and uses multipath routing and an efficient local repair mechanism to improve failure resilience.

1 Introduction

An *IP-over-fiber network* is a typical building block of the Internet's backbone. It usually belongs to a single Internet Service Provider (ISP), and is centrally monitored and managed. The physical infrastructure of an IP-over-fiber network consists of a mesh of optical fibers usually put in the ground along roads, rails, or power-lines. The IP links are realized as end-to-end connections routed on this mesh. The topology formed by the IP links is a result of a centralized optimization process and reflects the long term user demands. Therefore the IP-over-fiber network topology rarely changes.

In contrast, a *wireless ad-hoc network* consists of a group of nodes that communicate with each other through wireless radio channels. There is no fixed infrastructure. Moreover, in some scenarios the nodes are mobile. There is no centralized control or overview. There are no designated routers: nodes serve as

routers for each other, and data packets are forwarded from node to node in a multi-hop fashion. Wireless ad-hoc networks are not yet widely deployed, but their first real-life applications are beginning to emerge.

Although the IP-over-fiber and wireless ad-hoc settings are quite different in nature, they share a number of problems and challenges. One of them is *failures* of network components. There are many possible sources of failures. In IP-over-fiber networks it might be a fiber cut, a failure of optical equipment (switch, router, amplifier), software errors, system misconfiguration, to name a few. In fact, in real IP backbones failures occur almost every day [1]. Moreover, due to huge capacities of optical fibers, even a single failure may result in a very significant disruption of the network functionality. In wireless ad-hoc networks the main source of failures is the instability of the wireless medium, which results in frequent failures of existing links and arrivals of new links[3]. This happens in terms of minutes [2]. The phenomenon is especially strong if we allow for mobility. Another typical problem is the limited battery power of nodes, eventually causing a node failure.

Failures often result from random events and thus are unavoidable. Therefore one of the crucial properties of a communication network is handling failures. It is twofold. First, a failure should be *located*. Since permanent and full network monitoring is resource inefficient, the network operators often limit it, at the cost of having only a partial knowledge of the present network state, such as a set of end-to-end measurements. In this setting, locating a failure becomes a nontrivial task. Second, once a failure is located, the traffic must be rerouted and the network operability *restored*. The mechanisms ensuring this should take into account all important failure scenarios and a number of physical constraints (e.g., link capacities).

In this paper we address both issues: failure location and restoration. In Section 2 we present the algorithms that can be used for failure location in IP-over-fiber and wireless networks. Next we give a number of various algorithms for failure restoration in Section 3 (optical networks) and in Section 4 (wireless ad-hoc networks). Finally, in Section 5 we conclude the paper.

2 Failure Location

When a failure occurs in the network, monitoring devices (passive or active) detect the failure and generate alarms to warn the management system. The management system then needs to infer the location of the failures based on the received alarms. The failure location task in communication networks is hindered not only by the existence of multiple possible explanations for some sets of alarms but also by corrupted alarms, which are those alarms that unexpectedly arrive at the management system when they should not (false alarms), or those that do not arrive at the management system when they should (missing alarms).

The nature of failures and available monitoring information differ significantly in IP-over-WDM and wireless networks. Each type of network therefore

[3] The terms *link* and *edge*, as well as *node* and *vertex*, will be used interchangeably

needs to have its own failure location methods. We present in this section two failure location algorithms that can be used to locate failures in IP-over-fiber and wireless sensor networks, respectively.

2.1 Failure Location in IP-over-WDM Networks

Failures of optical devices often manifest themselves in the degradation or loss of optical signals. Passive monitors are widely deployed in these networks to assess the signal health. When a monitor observes a significant drop in the signal quality, it sends an alarm to the management system. Alarms can be generated by devices at the optical, SDH/SONET or IP layer. A full review of the available monitoring information is provided in [3]. Failure location in IP-over-WDM networks is known to be NP-hard [4] and several algorithms have been proposed to solve this intractable problem in the literature (see [3] for a complete review of the existing failure location algorithms). Although many researchers [4] have suggested that failure location algorithms must be able to cope with alarm errors, most location algorithms today avoid this issue because of the complexity of covering all possible failures and corrupted alarms.

In optical networks, most network monitoring devices use a threshold to decide whether they should send alarms or not. For instance, an SDH device counts the number of errors it encounters in a time window and generates an alarm if the count is greater than a threshold, otherwise it remains silent [5]. Network operators have the option of trading false alarms for missing alarms and vice versa by tuning the parameters of monitoring devices. We have studied the failure location problem in an all-optical IP-over-WDM network when there are false and missing alarms in [6]. We have rigorously shown that for a network with binary alarms (alarms are either present or not), there is an asymmetry between false alarms and missing alarms. We have proven that false alarms can be corrected in polynomial time, but the correction of missing alarms is NP-hard. The correction of missing alarms is indeed equivalent to the red-blue set cover problem [7]. Because of this asymmetry, false alarms have a lesser effect on the accuracy of the diagnosis results than missing alarms do. Network operators therefore, when allowed, should set the threshold low to favor false alarms.

To handle corrupted alarms, we have proposed in [6] a polynomial time algorithm that can accurately locate failures with corrupted alarms. The algorithm takes as inputs the network topology and the positions of the monitors (this information is available in most IP-over-WDM networks). The algorithm consists of two steps. In the first step, called the Error Correction (EC) step, the algorithm uses a maximum likelihood reasoning to identify and correct the most probable set of corrupted alarms. In the second step, called the $MFAULT$ step, the algorithm then uses a set-cover heuristic to locate the faulty components with the cleaned alarms. The failure location algorithm performs well in simulated networks of real topologies as shown in Fig. 1.

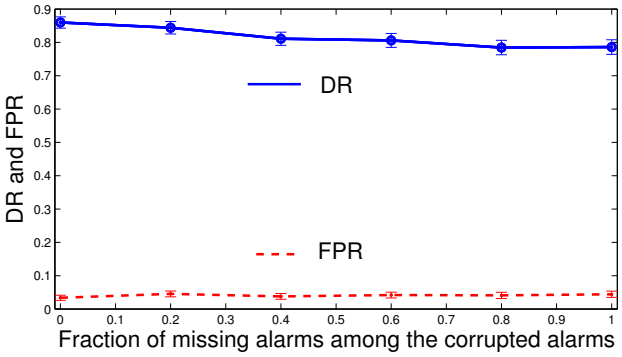

Fig. 1. Performance of the proposed failure location algorithm in the NSFNET topology [6]. The number of corrupted alarms is kept constant but the fraction of missing alarms is varied from 0 to 1, where 1 means all corrupted alarms are missing alarms. The algorithm is evaluated in terms of detection rate (DR), which is the fraction of failures that are correctly identified; and false positive rate (FPR), which is the fraction of failures that are wrongly identified. The algorithm achieves detection rates above 75% in all settings and the fewer missing alarms we have, the more accurate the algorithm is.

2.2 Failure Location in Wireless Sensor Networks

Contrary to IP-over-optical networks, qualities of wireless links vary significantly in the order of minutes [2]. In wireless sensor networks, it is not essential to monitor and locate all bad links (links with high loss rates), as the network should quickly self-organize around them. The problem occurs when all links surrounding a sensor node are lossy, because of low battery or physical obstacles. In this case, there is no other choice to access this node than to use a lossy link. The failure location task in a wireless sensor network therefore mainly concerns the identification of links that are consistently used to transport data but have bad quality. Diagnosing sensor networks is challenging because the networks cannot support much monitoring traffic and change their routing topologies frequently.

In [8] we have proposed to use only end-to-end application traffic to infer the bad performing links in sensor networks. Due to the lack of other network monitoring means, end-to-end application traffic is the most reliable source of network performance indication in wireless sensor networks. The inference of internal link properties given end-to-end observations is called network tomography. A detailed survey of the current tomography techniques is provided in [3]. In most networks end-to-end data do not provide enough information to identify the exact link loss rates but enough to identify the worst performing links.

We have introduced in [8] two inference techniques to infer lossy links in wireless sensor networks. The first algorithm (the LLIS algorithm) uses the maximum likelihood inference principle, whereas the second one (the MCMC algorithm)

adopts the Bayesian principle. Both algorithms handle well noisy end-to-end data and routing changes in wireless sensor networks.

The LLIS algorithm first uses a threshold t_p to determine whether the loss rate on an end-to-end path is good or bad. After classifying all paths as good and bad, the algorithm then tries to find the smallest set of links whose badness would explain the badness of all paths in the network. The LLIS algorithm has the advantage of being simple. But, it is sensitive to estimation errors of end-to-end transmission rates and the choice of the path threshold t_p. The end-to-end transmission rates are only accurate when we have a sufficiently large number of packets. To handle the cases where there are not sufficient data to calculate the end-to-end transmission rates, we have proposed to use the second technique, namely the Bayesian inference technique [9] that is less vulnerable to end-to-end loss rates but also much more complex. The idea here is to try to generate a set of possible link loss rates that can explain the observations of end-to-end data. If the majority of the possible loss rates of a link are bad, then it is likely that the link is bad, otherwise it is good. For details of the MCMC method, please refer to [9].

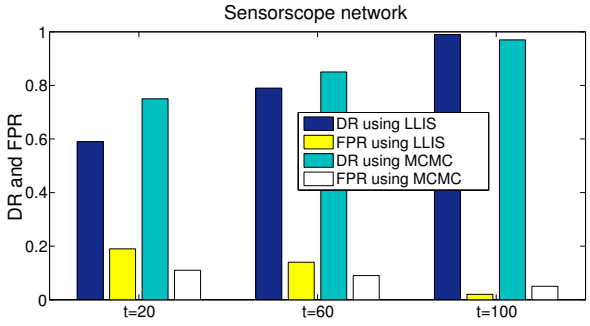

Fig. 2. Performance of the failure location algorithms (LLIS and MCMC) on the Sensorscope [10] network. We consider only links that are used to route more than t packets. As t increases, the algorithms become more accurate because of two reasons: (1) end-to-end data are more reliable, and (2) there are less errors generated by routing changes.

The performance of both inference algorithms (LLIS and MCMC) are evaluated by simulations and real network traces in [8]. Both algorithms achieve accurate failure location results with high detection and low false positive rates as shown in Fig. 2.

3 Failure Protection and Restoration in IP-over-WDM Networks

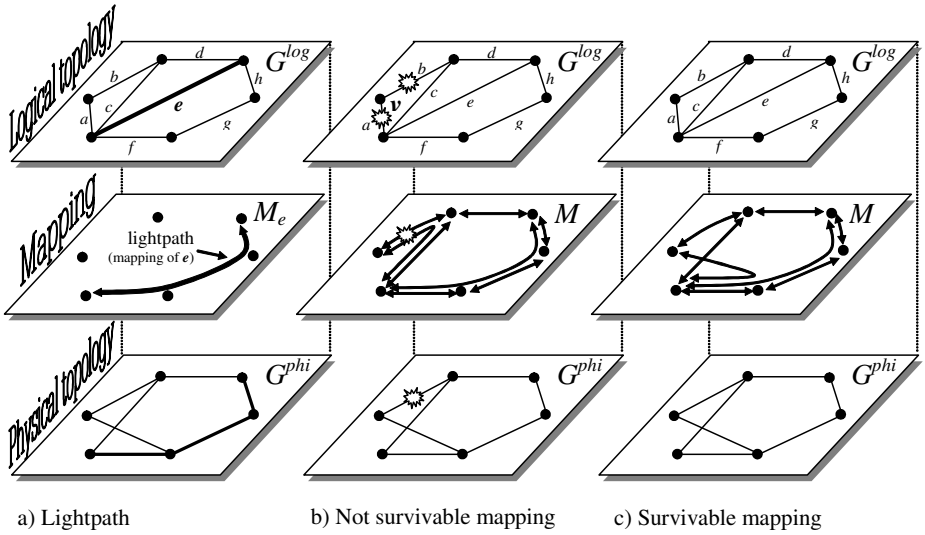

Fig. 3. IP-over-WDM network. (a) Given the physical and logical topologies, every logical edge (here e) is mapped on the physical topology as a lightpath. (b) An example of a mapping that is *not* survivable. After a failure of the indicated physical link, the logical topology becomes disconnected (the node v gets separated from the rest of the logical graph). (c) An example of a survivable mapping. One can easily check that after any single physical link failure the logical topology remains connected.

Most of the IP backbone networks are currently designed on top of a Wavelength Division Multiplexing (WDM) infrastructure. This stack is called an IP-over-WDM network. WDM allows the same optical fiber to carry many signals independently, each using different wavelengths (colors). In real networks the number of these signals is in the order of tens (in the Sprint network the maximum number is 25 [1]). Specialized labs can reach hundreds and thousands. Therefore a single failure of a physical fiber might have very significant consequences on the network, and should be carefully handled.

In an IP-over-WDM network we distinguish *two layers*: the *physical graph* is a mesh of optical fibers (edges) and optical switches (nodes). The *logical graph* is a mesh of IP connections (edges) and IP routers (nodes). Since we assume that on every optical switch lies an IP router, the sets of nodes at both layers are identical. Each logical link is mapped on the physical topology as a *lightpath* (see Fig. 3a). The set of all lightpaths defines a *mapping* of the logical graph on the physical graph. To construct a mapping, many objectives should be taken into account. One of them is the robustness to failures, or *survivability*. This issue is especially important in the IP-over-WDM architectures, where one physical link can carry many lightpaths, and thus where a single physical link failure may

bring down a large number of logical links. A survey of different approaches for providing survivability of IP-over-WDM networks can be found in [3]. In this paper we consider exclusively the *IP restoration approach* that was shown to be effective and cost–efficient (see e.g., Sprint network [11]). In IP restoration, failures are detected by IP routers, and alternative routes in the logical topology are found. In order to enable this, the logical topology should remain *connected* after a failure of a physical link; this in turn may be guaranteed by an appropriate mapping of logical links on the physical topology. We call such a mapping a *survivable mapping*. In Fig. 3 we present two examples of mappings: the first one is not survivable (b) and the second one is survivable (c).

The problem of finding a survivable mapping was first defined in [12], and many algorithms solving this problem (with different variations) have been proposed since then. In general, they can be divided into two groups: exact algorithms based on Integer Linear Programming (ILP), and heuristics. The ILP solutions can be found for example in [13, 14]. They lead to an unacceptably high complexity for networks of a non-trivially small size [15] (larger than a few tens of nodes). This is because the survivable mapping problem is NP-complete [16]. To avoid this prohibitive complexity, the second line of approach uses various heuristics, such as Tabu Search [12, 17, 18, 14], Simulated Annealing [19] and others [20, 21].

In this paper we describe two recent algorithms solving the survivable mapping problem: SMART [22, 23] and FastSurv [24–26]. These algorithms are very efficient, and somewhat complementary. SMART is the fastest and the most scalable algorithm known to date. Moreover, the formal analysis of SMART [23] has led to new applications: the formal verification of the existence of a survivable mapping, and a tool tracing and repairing the vulnerable areas of the network. SMART can be applied only if we assume unlimited capacities of the physical links. In a more realistic scenario, the FastSurv algorithm shows its strengths. FastSurv can be easily adapted to any set of real-life constraints, while still being much faster and more scalable than the other heuristics known to date.

In the following subsections we present versions of the SMART and FastSurv algorithms that solve the basic survivability problem (single physical edge failures, no capacity constraints). A number of specific properties of the algorithms and their possible extensions are described in a later section.

We will use the following notation. The physical and logical topologies are represented by undirected graphs $G^{ph} = (V, P)$ and $G^{log} = (V, L)$, respectively. V is the set of vertices (common for both layers), P and L are the sets of undirected edges. Note that according to our assumption, we take $V^{ph} \equiv V^{log} \equiv V$. The mapping is represented in a form of a $|P| \times |L|$ binary matrix $M = \{m_{p,l}\}$, where $m_{p,l} = 1$ if the logical link l uses the physical link p in its mapping. A mapping M is survivable if after the failure of any single physical link $p \in P$, the logical topology G^{log} remains connected. More formally, M is survivable, if for every physical link $p \in P$ the graph

$$G_p^{log} = G^{log} \setminus \{l : m_{p,l} = 1\} \tag{1}$$

is connected.

3.1 SMART

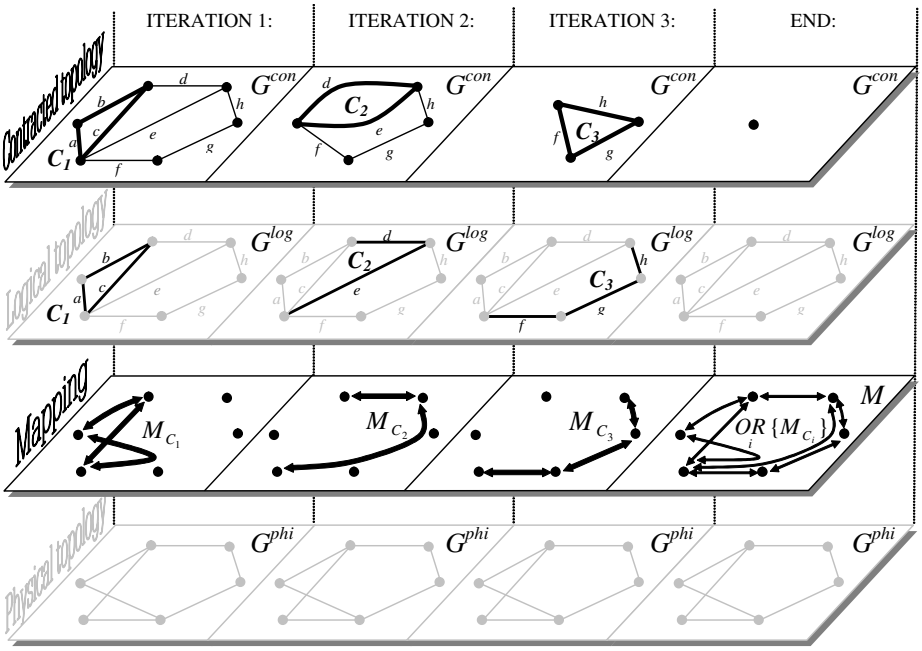

Fig. 4. Illustration of the SMART algorithm. We have four layers, from bottom to top: physical topology G^{ph}, mapping, logical topology G^{log} and contracted logical topology G^{con}. During a run of the SMART algorithm, only the contracted topology and the mapping change from one iteration to the next one. The logical and physical topologies are included only for the context; therefore they are set in grey. At each iteration a cycle C picked from the contracted topology is set in bold. This cycle is defined as a set C of logical links. (Although C is always a cycle in the contracted topology G^{con}, it does not necessarily form a cycle in the logical topology G^{log}; see e.g., Iteration 2 and 3.) Next, a disjoint mapping M_C is found for the set C. Then C is contracted in G^{con}, resulting in a new contracted logical topology G^{con} used at the subsequent iteration. Once G^{con} has converged to a single node, the underlying mapping is survivable. If there are still some unmapped logical links, they can be mapped in any way (e.g., a shortest path). Now we combine the lightpaths found in all iterations to obtain a mapping M (last column) of the entire logical topology. The mapping M is survivable.

One of the main operations in the SMART algorithm is contraction [27]. *Contracting* an edge $e \in E$ in a graph $G = (V, E)$ is deleting that edge and merging its end–nodes into one. The result is called a *contracted graph* G^{con}. We will also allow contracting a set of edges $A \subset E$. Note that the order of the edges in A does not affect the result.

Now we present the idea of the SMART algorithm. First choose from the logical topology G^{log} a cycle $C \subset L$ and map it *disjointly* (i.e., not using the

same physical edge twice). The disjoint mapping of a logical cycle ensures that this cycle will remain connected after any single physical link failure. In other words, the cycle C is already mapped in a survivable way. Now, we contract the cycle C in the logical topology G^{log} and repeat the above procedure for the resulting graph. We iterate this operation until the contracted logical topology converges to a single node, which guarantees survivability. The example run of SMART is illustrated in Fig. 4. The pseudo-code of the SMART algorithm is:

Initialization Contracted logical topology $G^{con} := G^{log}$.
Step 1 Pick a cycle C in G^{con}.
Step 2 Use DisjointMap (see below) to map disjointly the cycle C on the physical topology. Denote this mapping by M_C. (M_C is of the size of M, with non-zero elements appearing only in the rows corresponding to the logical edges in C)
Step 3 Contract C in G^{con}.
IF G^{con} consists of *one* node, THEN RETURN survivable mapping M which is a superposition of all disjoint mappings M_C found before: $M = \bigvee_i M_{C_i}$. END.
Step 4 GOTO Step 1.

DisjointMap In Step 2 of the SMART algorithm, we have to find a disjoint mapping of the set C of logical links. The problem is equivalent to the edge-disjoint paths problem [28] that is proven to be NP-complete. Therefore we apply the following heuristic we call *DisjointMap* . Let each physical edge have a weight (these weights will be used exclusively within DisjointMap) and let this weight be initially set to one. At each iteration, map the logical links from C with shortest path. If no physical link is used more than once, a disjoint solution is found. Otherwise, the weight of each physical link used more than once is increased, and a new iteration starts.
Clearly, DisjointMap does not guarantee success. Therefore after several unsuccessful iterations it fails. In this case, the SMART algorithm cannot proceed to Step 3. Instead it comes back to Step 1 and picks another cycle; a choice of a short cycle will help the heuristic to converge rapidly. After a rare event of several consecutive failures of Step 2, the SMART algorithm quits returning the partial mapping $M = \bigvee_i M_{C_i}$ (some rows of M remain empty).

It is possible that even if the contracted graph G^{con} converges to a single node, there are still some unmapped logical links. They would form self-loops in G^{con}. We can map them in any way (e.g., with shortest path), which does not affect the survivability of the resulting full mapping.

3.2 FastSurv

FastSurv is a heuristic algorithm that works in an iterative manner. It starts from an initial mapping $M(0)$ obtained with a simple method. At each iteration t, the algorithm evaluates the current solution $M(t)$ and tries to improve it by rerouting a number of logical links.

The improvement phase is based on an observation made in [13] that a mapping is survivable if and only if no physical link is shared by all logical links belonging to a cut-set of G^{log}.[4] E.g., in Fig. 3b, logical links a and b share a physical link and cause unsurvivability because $\{a, b\}$ is a cut-set of G^{log}. In Fig. 3c, however, a, e and f share a link, but this does not cause unsurvivability because $\{a, e, f\}$ is not a cut-set of G^{log}. An exact solution method based on this idea (such as the ILP method of [13]) needs to take all cut-sets into account as hard constraints when constructing a mapping, which can be difficult and inefficient. In FastSurv, we use the notion of cut-sets in a heuristic way. Specifically, the algorithm keeps track of information about which pairs of logical links l_i and l_j cause unsurvivability when they are routed together over the same physical link, and which do not. If l_i and l_j form a cut-set of size two (such as $\{a, b\}$ in Fig. 3), they cause unsurvivability each time they are routed together. If l_i and l_j are part of a larger cut-set (e.g., in Fig. 3, a and e are part of the larger cut-set $\{a, c, e, f\}$), they only cause unsurvivability if all the other logical links of the same cut-set are also routed on the same link. The routing of these other logical links depends on the specific situation (the current mapping $M(t)$), which in FastSurv changes slowly from iteration to iteration since each time a number of logical links are rerouted. FastSurv updates at each iteration according to the current mapping the information about which pairs of logical links cause unsurvivability when they share a link, and uses this information to reroute logical links. This way, FastSurv can focus on the cut-sets which are important in the current situation when trying to improve the survivability of the mapping. The pseudo code for FastSurv is as follows:

Initialization Calculate $M(0)$ and set the number of iterations t to 0.
Step 1 Evaluate $M(t)$.
Step 2 Update the information about which pairs of logical links cause unsurvivability when they share a physical link, based on the evaluation of $M(t)$.
Step 3 RETURN $M(t)$ IF (($M(t)$ is survivable) OR (t = maximum number of iterations)).
Step 4 Calculate $M(t+1)$ by rerouting logical links of $M(t)$ using the information of Step 2.
Step 5 Increase t and GOTO Step 1.

To obtain $M(0)$, the logical links are routed on G^{ph} one after the other in random order. We use shortest path routing, with the cost of a physical link p equal to the number of logical links that are already routed over p. This simple algorithm

[4] A *cut-set* of a network is defined by a cut of the network: a cut is a partition of the set of nodes V into two sets S and $V - S$, and the cut set defined by this cut is the set of edges that have one endpoint in S and one in $V - S$.

avoids that some links carry many more logical links than others, which would make them more vulnerable with respect to survivability.

In Step 1, $M(t)$ is evaluated by considering all physical links p of P individually, and investigating whether the remaining logical graph G_p^{log} (as defined in formula (1)) is connected. Physical links whose failure leaves G_p^{log} disconnected are called *unsurvivable physical links*, and the logical links that are routed over them are called *unsurvivable logical links*. The algorithm uses a binary vector $U = \{u_p\}(t)$, where $u_p(t) = 1$ if p is an unsurvivable physical link in $M(t)$ and $u_p(t) = 0$ otherwise.

The information about which pairs of logical links cause unsurvivability when they share a physical link is kept in a $|L| \times |L|$ matrix $Z = \{z_{l_i,l_j}\}$. Z is updated according to the formulas (2)-(4) below. In formula (2), $a_{l_i,l_j}(t)$ is defined as the number of times that logical links l_i and l_j share a physical link in $M(t)$, and in formula (3), $b_{l_i,l_j}(t)$ is defined as the number of times that this shared physical link is unsurvivable in $M(t)$. Dividing $b_{l_i,l_j}(t)$ by $a_{l_i,l_j}(t)$, one obtains a ratio that can be seen as an estimate (based on the experience of iteration t) of the probability that combining logical links l_i and l_j on a physical link will make that physical link unsurvivable. z_{l_i,l_j} is then defined in formula (4) as the exponential average of this probability estimate (with $\alpha = 0.2$ in the experiments).

$$a_{l_i,l_j}(t) = \sum_p m_{p,l_i}(t) m_{p,l_j}(t) \qquad \forall l_i, l_j \in L \tag{2}$$

$$b_{l_i,l_j}(t) = \sum_p u_p(t) m_{p,l_i}(t) m_{p,l_j}(t) \qquad \forall l_i, l_j \in C \tag{3}$$

$$z_{l_i,l_j} = \begin{cases} \alpha z_{l_i,l_j} + (1-\alpha)\frac{b_{l_i,l_j}(t)}{a_{l_i,l_j}(t)} & if \quad a_{l_i,l_j}(t) > 0 \\ z_{l_i,l_j} & if \quad a_{l_i,l_j}(t) = 0 \end{cases} \tag{4}$$

In Step 4, FastSurv reroutes all logical links that are unsurvivable in the current mapping (survivable logical links are left mapped as they were), using the probability estimates of Z. A shortest path algorithm is applied in which the cost of a path for a logical link l_i is the probability that l_i will be unsurvivable somewhere along its path. The probability $Prob_{path}^{l_i}$ that l_i will be unsurvivable on a path is the probability that it will be unsurvivable on at least one physical link of the path. The probability $Prob_p^{l_i}$ that l_i will be unsurvivable on a physical link p of the path is the probability that l_i will cause unsurvivability when routed together with any of the other logical links l_j which use p,[5] which is estimated in z_{l_i,l_j}. We use formula (5) to estimate $Prob_p^{l_i}$ and formula (6) to estimate $Prob_{path}^{l_i}$.

$$Prob_p^{l_i} = 1 - \prod_{l_j \text{ on } p} (1 - z_{l_i,l_j}) \tag{5}$$

$$Prob_{path}^{l_i} = 1 - \prod_{p \text{ on } path} (1 - Prob_p^{l_i}) \tag{6}$$

[5] Other logical links can already be using p either because they were not removed after the previous iteration or because they were rerouted before l_i.

3.3 Time Complexity

SMART The complexity of one iteration of SMART is dominated by the DisjointMap function in Step 2 of the algorithm. Assuming a small size of the cycles C (in order of $O(1)$), this heuristic has a complexity $O(Dijkstra)$ that is at most $O(N^2)$, where N is the number of nodes in the graph. To estimate the number of iterations before SMART converges, we note that a single iteration maps one cycle, which reduces the number of nodes by at least one. So we need at most $O(N)$ iterations. It results in a total complexity of SMART equal to $O(N^3)$.
In practice, the physical graph is sparse, i.e., has got $O(N)$ edges, which reduces the complexity of $O(Dijkstra)$ to $O(N \log N)$. Consequently, the complexity of SMART drops to $O(N^2 \log N)$.

FastSurv Each iteration of FastSurv consists of an evaluation and a number of logical link reroutings. Rerouting a single logical link has a complexity $O(Dijkstra)$ (which is maximally of $O(N^2)$) and the evaluation of survivability has a maximal complexity of $O(N^4)$ [26]. Therefore, given that the maximum number of logical links to be rerouted in one iteration is the total number of logical links, which is $O(N)$ (we follow [17], where logical networks of a fixed degree are considered), a single iteration of FastSurv has a maximal complexity of $O(N^4)$. Since the maximal number of iterations is a parameter independent of N (and usually small), the overall complexity of FastSurv is $O(N^4)$ as well.

3.4 Results

ILP approach In [13] the necessary and sufficient conditions for a mapping to be survivable are specified (see Subsection 3.2 for details). These conditions are injected into the Integer Linear Programming (ILP) formulation, that is used to find a survivable mapping. Then a simple relaxation (ILP-Relax) for the ILP is introduced, which substantially reduces the processing time.

We ran the SMART and FastSurv algorithms for exactly the same topologies as in [13], namely NSFNET as the physical topology and the same 300 random graphs of degree $\bar{d} = 3, 4$ and 5 as those in [13] for the logical topologies. A survivable mapping was found in *all runs* when using ILP, ILP-Relax, SMART and FastSurv approaches. Therefore it is interesting to compare the *run–times* of the algorithms. The machines were not the same, yet comparable (Sun Sparc Ultra-10 vs. Pentium 500). However, we must stress that SMART and FastSurv were implemented in pure C++, whereas ILP required a dedicated program (CPLEX), which could significantly affect the results. The run-times from [13] are reprinted in Table 1; the last two columns show the results of SMART and FastSurv. The SMART algorithm is several orders of magnitude faster than pure ILP, and about 3 orders of magnitude faster than the relaxed version of ILP. FastSurv is about one order of magnitude slower than SMART. Note that, in contrast to ILP, the degree of the logical topology hardly affects the run-time of SMART and FastSurv.

Table 1. Run-times of ILP, SMART and FastSurv

Average degree $\bar{d}$	ILP	ILP-Relax	SMART	FastSurv
3	8.3 sec	1.3 sec	0.0028 sec	0.0117 sec
4	2 min 53 sec	1.5 sec	0.0028 sec	0.0155 sec
5	19 min 17 sec	2.0 sec	0.0029 sec	0.0166 sec

Tabu Search and Large Topologies One of the most efficient and widely used techniques to solve a survivable mapping problem is *Tabu Search*. Our implementation of Tabu Search follows the one in [12]; we will refer to it as *Tabu97*. Since Tabu Search turned out to be substantially faster than the ILP approach (described in the previous section), we carried out the simulations for relatively large graphs and studied the scalability of Tabu Search, FastSurv and SMART. To emulate larger real-life physical topologies we generate square lattices where each vertex is connected by an edge to its four closest neighbors only. Next a fraction f of these edges is deleted; we call them f-lattices. We keep only these f-lattices that are 2–edge–connected.[6] The parameter f ranges from 0 to 0.35. The maximal value 0.35 was chosen in such a way that even the smallest topologies could be 2–edge–connected. Since the logical graph is less regular (for instance, there is no reason why it should be planar), the logical topologies are 2-edge-connected random graphs of average vertex degree $\bar{d} = 4$. The number of vertices ranges from $N = 16$ to 900. In Fig. 5 we present the results obtained in simulations on a Pentium 4 machine, for the three algorithms implemented in C++. In Fig. 5a we investigate the run–times of the Tabu97, FastSurv and SMART. The observed complexities of the algorithms are polynomial, with $O(N^{3.6})$ for Tabu97, $O(N^{2.8})$ for FastSurv, and $O(N^{2.4})$ for SMART[7]. These values fit in the theoretical maximal bounds that are $O(N^5)$, $O(N^4)$ and $O(N^3)$, respectively. Note that Tabu97 took about 11 hours when solving a 900 node problem, which is a lot more than the 3 minutes measured for FastSurv and the 25 seconds for SMART.

Fig. 5b is related to the *effectiveness* of the algorithms, i.e., their ability to find a survivable mapping. Although FastSurv and SMART are comparable (with a slight advantage of the former), Tabu97 is significantly worse, especially for larger topologies. It should be noted that for every N a fraction of studied topologies is impossible to be mapped in a survivable way, which makes the upper bound of the effectiveness smaller than one.

[6] A graph G is *k-edge-connected* if G is connected and every set of edges disconnecting G has at least k edges [27]. Clearly, 2-edge-connectivity of both physical and logical graphs is a necessary condition for the existence of a survivable mapping.

[7] The measured value of SMART complexity $O(N^{2.4})$ is larger than the theoretical bound $O(N^2 \log N)$. This is probably because the DisjointMap function often takes several (not one) iterations to converge.

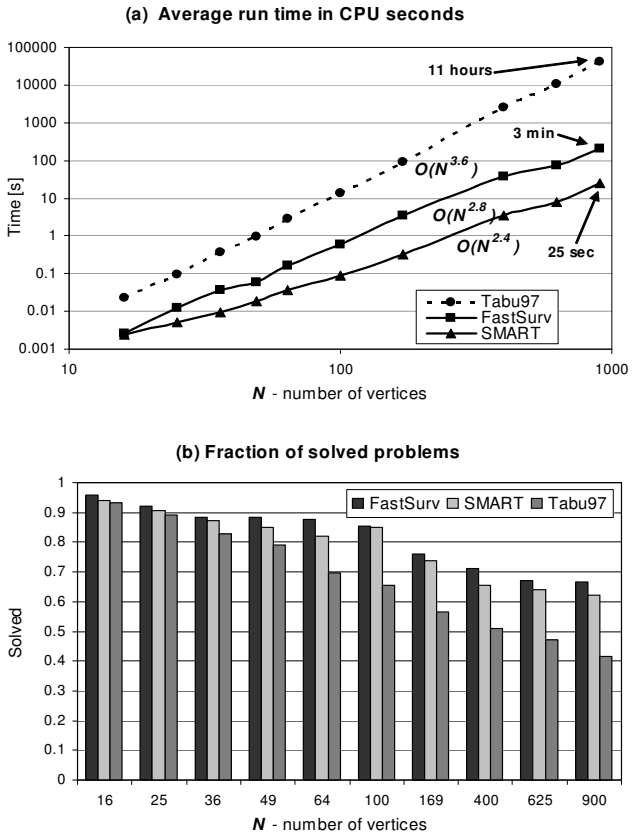

Fig. 5. Test results for Tabu97, FastSurv and SMART using logical and physical networks of increasing number of nodes ($16 - 900$). The logical networks are random graphs, whereas the physical networks are f-lattices, with f ranging from 0 till 0.35 with a step size of 0.05. Each data point represents an average over 1000 different test problems and over all values for f. (a) shows the run time in CPU seconds as a function of the number of nodes in a log-log scale. For each curve we indicate the exponent of its power-law fit. (b) shows the fraction of successfully mapped topologies as a function of the number of nodes.

3.5 Extensions

In the previous sections we have defined the survivability problem by taking into account single physical edge failures only, and assuming no capacity or other real-life constraints. We have described and compared the versions of the SMART and FastSurv algorithms, solving this basic survivability problem. However, these algorithms have a number of useful properties that can be exploited. In that regard they turn out to differ substantially. In particular, a proper appli-

cation of the SMART algorithm gives us a valuable insight into the survivability problem, whereas FastSurv can be easily extended to a setting with any set of real-life constraints (e.g., limited fiber capacities). We briefly describe some of the possible applications below.

Capacity Constraints An optical fiber connection can only carry a limited number of different lightpaths, which is a capacity constraint for each physical link. Here we present an extended version of the FastSurv algorithm that can find a survivable mapping while considering the limited physical link capacities. For SMART such an extension is less straightforward due to its particular approach.

Like the basic FastSurv survivable routing algorithm, the extended FastSurv algorithm starts from an initial solution that it tries to improve in subsequent iterations. The algorithm uses two different types of iterations. *Survivability iterations* are identical to the iterations of the basic FastSurv algorithm and aim at improving survivability while relaxing the capacity constraint. *Capacity iterations* reduce the number of capacity constraint violations while relaxing the survivability goal. The algorithm alternates between a number of survivability iterations and a number of capacity iterations, and stops when it finds a mapping that is survivable and satisfies all link capacity constraints (or when a maximum number of iterations is reached).

In each capacity iteration, the algorithm tries to improve the current mapping by rerouting logical links that were routed on overfull physical links. For the rerouting, we use the shortest path routing where the cost of a physical link is equal to the number of logical links already routed over this physical link divided by the maximum capacity of the physical link, except when the number of logical links is higher than or equal to the capacity, in which case the number is not divided by the capacity. This way physical links that are full are avoided.

In a large series of tests, we have shown that the extended FastSurv algorithm outperforms tabu search in terms of solution quality and time [26]. Moreover, it is much more scalable.

Verification of the Existence of a Survivable Solution. If a heuristic fails, nothing can be claimed about the existence of a survivable mapping. To date, the only general method verifying the existence of a survivable mapping was an exhaustive search run for the *entire* logical topology G^{log}. Due to NP-completeness of the survivable mapping problem, the exhaustive approach is not realizable in practice for topologies larger than a few nodes. The SMART algorithm can substantially simplify this procedure. It turns out that it is sufficient to verify only the resulting contracted graph G^{con} (from a terminated run), instead of G^{log}. This makes the verification of the existence of a survivable mapping often possible for moderate and large topologies [23].

Tracing and Repairing the Vulnerable Areas in the Network. A second novel application of SMART is tracing the vulnerable areas in the network and pointing where new link(s) should be added to enable a survivable mapping.

Once we know that a particular pair of physical and logical topologies cannot (or can difficultly) be mapped in a survivable way, a natural question is to modify the topologies to enable such a mapping. Where should a new logical link l^{new} be added? The SMART algorithm helps us answer this question. Run SMART and wait until it terminates. Since a survivable mapping does not exist, the contracted topology G^{con} will not converge to a single node. Most probably G^{con} will shrink to a small structure (in comparison with the original logical graph G^{log}) and the algorithm will give up. Consider the graph G^{con}. Addition of the new logical link l^{new} to the logical topology results in addition of l^{new} also to G^{con}. In [23] it was shown that if l^{new} forms a self-loop in G^{con}, then its introduction will never help survivability. In other words, to enable a survivable mapping we should locate l^{new} in such a way, that it connects two different vertices in G^{con}.

The simulation results in [23] have shown that the SMART-aided introduction of a new logical link greatly helps, contrary to a completely random choice of location of this link.

Node, Span and Double Link Failures So far we have only considered survivability with respect to single physical link failures, which are the most common type of failures in WDM networks. Here, we describe how FastSurv and SMART can be adapted to deal with more complicated failures such as span, node and double link failures. A span is a bundle of physical links that have been placed together for cost reasons (e.g., along railway and electricity lines). A single cut can break all of these physical links at once, in which case we speak of *span failure* [29]. We can also encounter *node failures* [30]; they are the consequence of a failure of equipment at nodes, such as switches. In our context a node failure is equivalent to a failure of all physical links neighboring the node. Finally, we consider *double–link failures*, i.e., independent failures of any two physical links [31]. Usually such a situation takes place when the second failure occurs before the first one is repaired.

Adapting FastSurv to deal with span failures is straightforward. Although the basic algorithm described in Subsection 3.2 kept track of which pairs of logical links cause unsurvivability when they share a link, it should now consider which pairs of logical links cause unsurvivability when they share a span. To adapt SMART we have to modify only the DisjointMap function used in Step 2 to produce *span-disjoint* mappings (instead of only link-disjoint).

The adaptation of FastSurv to deal with node failures is again not difficult: the algorithm should consider which pairs of logical links cause unsurvivability when they are routed together over the same node. Since a logical link can never be routed survivably with respect to its end nodes, logical links incident on a node should not be considered to share that node. For SMART, we make DisjointMap generate *node-disjoint* mappings.

To deal with double link failures, FastSurv should investigate survivability with respect to all pairs of physical links, and register which pairs of logical links cause unsurvivability when routed over these pairs of physical links. In the

case of SMART, any logical cycle C processed by the algorithm can clearly be disconnected by a double failure. In order to enable protection against double failures we take small 3-edge-connected structures instead of this cycle, as shown in Fig. 6. Note that the contracted logical graph can have multi-edges, and so do these structures. The rest of the SMART algorithm remains unchanged. In particular the DisjointMap heuristic searches for a link-disjoint mapping, as in its original version.

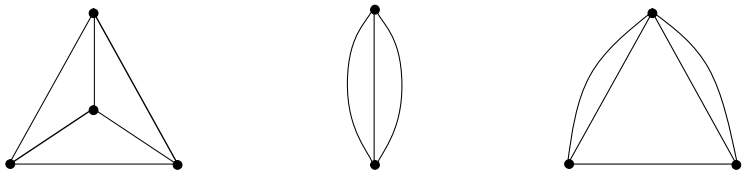

Fig. 6. Examples of 3-edge-connected structures that might be used by SMART to handle double failures.

3.6 Conclusion

Table 2 summarizes the efficiency and functionality of SMART and FastSurv, and compares them with ILP and Tabu97. Both algorithms are much faster and more scalable than any solution proposed to date. As for the possible extensions and particular properties, the two algorithms can be regarded as being complementary. SMART provides us with a method of formal verification of the existence of a survivable mapping and a tool tracing and repairing the vulnerable areas of the network, whereas FastSurv can be easily adapted to any set of real-life requirements such as capacity constraints.

Table 2. Comparison of the efficiency and functionalities of SMART, FastSurv, ILP and Tabu97. The question mark "?" means that the option might be possible to realize, but that, to the best of our knowledge, no one has yet done it to date.

Functionality	SMART	FastSurv	ILP	Tabu
fast and scalable	✓✓	✓	×	×
capacity and other constraints	×	✓	✓	✓
verification of a solution existence	✓	×	✓	×
node failures	✓	✓	?	?
span failures	✓	✓	?	?
multiple failures	✓	✓	?	?
tracing and repairing the vulnerable areas	✓	×	×	×

4 Failure Restoration in Mobile Ad Hoc Networks by Rerouting

Mobile Ad Hoc Networks (MANETs) [32] are wireless ad hoc networks in which all nodes are mobile. Due to this mobility, the network topology, which is formed by the wireless links established between nodes that are in each other's vicinity, is dynamic, with regular failures of existing links and arrivals of new links. Dealing with these constant changes is made more difficult by other challenges, such as the low bandwidth of the shared wireless channel, which is mainly due to the need to use inefficient decentralized mechanisms for medium access control, the limited resources of mobile devices (battery power and memory), the high error rates and signal interference in wireless communication, the lack of central control, etc.. In this setting a failure restoration problem boils down to constructing a scalable and highly adaptive routing algorithm. It should also be robust and efficient, and work in a distributed way. The abilities to deal with link failures and to take advantage of new opportunities arising from the appearance of new links are crucial.

In this section, we describe a novel routing algorithm for MANETs which is adaptive and failure resilient. It takes inspiration from Ant Colony Optimization (ACO) [33] and the related class of ACO routing algorithms [34], and uses both reactive and proactive strategies to deal with the dynamic MANET topology. In what follows, we first give a short overview of the current state of the art in MANET routing. Next, we give an introduction to the field of ACO and ACO routing. Then we describe the working of our algorithm, and finally we provide some results from simulation tests.

4.1 MANET Routing

Over the course of the last 10 years a large number of different MANET routing protocols have been proposed (see [32, 35] for overviews). All these algorithms deal with the dynamic aspects of MANETs in their own way, using reactive or proactive behavior, or a combination of both. *Reactive behavior* means that an algorithm gathers routing information in response to an event, such as the start of a data session or the failure of a link on an existing route. *Proactive behavior* means that the algorithm also gathers routing information at other times, so that it is readily available when the event happens.

In the MANET literature, the classical distinction is between table-driven, on-demand and hybrid algorithms. *Table-driven* algorithms, such as e.g. Destination-Sequenced Distance-Vector Routing (DSDV) [36], are purely proactive: all nodes try to maintain routes to all other nodes at all times. This means that they need to keep track of all topology changes, which can become difficult if there are a lot of nodes or if they are very mobile. *On-demand* algorithms, such as Ad-Hoc On-Demand Distance Vector Routing (AODV) [37] and Dynamic Source Routing (DSR) [38], are purely reactive: nodes only gather routing information when a data session to a new destination starts, or when a route that is in use fails. Reactive algorithms are generally more scalable since they greatly reduce

the overhead [39], but they can suffer from oscillations in performance because they are never prepared for disruptive events. In practice, many algorithms are *hybrid algorithms* (e.g. Zone Routing Protocol (ZRP) [40]), using both proactive and reactive components in order to combine the best of both worlds.

The traditional distinction between table-driven, on-demand and hybrid protocols tells only part of the story. One can classify MANET routing algorithms along a wide range of other dimensions. An important classification with respect to the work presented here is the difference between single path and *multi-path algorithms*. Many algorithms that use more than one path between each source and destination have been proposed (see [41] for an overview). They differ in the way multiple paths are set up, maintained and used. Multiple paths can serve as a way to enhance throughput, or as a way to increase robustness to link failures by providing backup paths. A disadvantage is that more overhead is needed because more than one path needs to be maintained.

4.2 ACO and ACO routing

ACO (Ant Colony Optimization) is a framework for optimization inspired by the mechanisms used by ant colonies to find the shortest path between their nest and a food source [33]. Ants leave behind a trail of a volatile chemical substance called *pheromone*; they also move preferentially in the direction of a higher pheromone intensity [42]. Since shorter paths can be completed quicker and more frequently by the ants, they get marked with higher pheromone intensity. These paths therefore attract more ants, which in turn increases the pheromone level. Finally, there is convergence of the majority of the ants onto the shortest path, with only a few ants continuing exploration of other paths. In ACO, artificial ants build solutions to an optimization problem guided by an artificial pheromone matrix, and update the matrix according to the quality of the solution they have constructed. ACO was first developed as a meta-heuristic for combinatorial optimization, and its first applications were for the travelling salesman problem [43]. Later, it has been applied to a whole range of different problems (see [33] for an overview).

The application of the ACO ideas to the problem of routing in wired networks led to the development of ACO routing algorithms [34], such as *Ant Based Control* (ABC) [44] and *AntNet* [45]. The main idea behind ACO routing is the acquisition of routing information through path sampling using ant agents. These lightweight agents are generated concurrently and independently by the nodes, with the task to try out a path to an assigned destination. An ant going from source s to destination d collects information about the cost of its path (e.g. end-to-end delay) and, tracing its way back from d to s, uses this information to update routing tables at intermediate nodes. The routing tables contain values indicating the relative goodness of each routing decision. The routing tables are updated by the ants and they are also used by the ants to find their way to their destination: at each node ants stochastically choose a next hop, giving higher probability to those next hops that are associated with higher goodness values. This way, the routing table entries play the role of artificial pheromone values in

the ant learning process. The routing tables are therefore also called *pheromone tables*, and their entries *pheromone values*. The continuous generation of ants results in the availability at each node of a bundle of paths, each with an estimated measure of quality. These paths are used to route data packets. Like the ants, data packets are routed *stochastically*, choosing with a higher probability those links associated with higher pheromone values. This way data for a same destination are adaptively spread over *multiple paths* (but with a preference for the best paths), resulting in *load balancing*.

4.3 A Novel ACO Routing Algorithm for MANETs

ACO routing algorithms have properties that are useful for MANETs. First of all, the continuous exploration of paths provides adaptivity, which is crucial in the dynamic MANET environment. Although ACO routing algorithms for wired networks were mainly designed to provide adaptivity with respect to data load changes, the same techniques can be extended to provide adaptivity with respect to topology changes, allowing for the use of new links and adjusting routing information after link failures. Second, the use of multiple paths provides a way to both increase throughput via data load spreading and to proactively deal with link failures by providing backup paths. Finally, the fact that routing information is learned from the accumulated experience of agents that sample full paths offers robustness, in two different ways. First of all, loss of agents is not a problem: it leads to slower updates, but not to wrong information. Second, route cost estimates are based on real experiences. This is in contrast with information bootstrapping techniques used in traditional distance vector routing algorithms [46], where nodes calculate route cost estimates based on the estimates reported by neighboring nodes. Although information bootstrapping is an efficient process, it is slow to converge after changes and can easily lead to errors in dynamic environments. Sampling full paths provides extra guarantees with respect to the correctness of the information.

There are however also disadvantages for MANETs in ACO routing. Firstly, ACO routing algorithms are normally purely proactive, maintaining routing information between all pairs of nodes at all times. Following this approach, a lot of unnecessary overhead is created, making the algorithm less efficient. Also, the lack of reactive components decreases adaptivity, because no specific reactions are triggered after a disruptive event. A second important disadvantage of ACO routing algorithms is the fact that the repeated path sampling using ant agents can come into conflict with the limited bandwidth in MANETs. Moreover, the high change rate of MANETs commands a higher sampling rate to keep routing information up to date, further aggravating the problem.

Here we present *AntHocNet*, an attempt to build an efficient, adaptive and robust routing algorithm for MANETs using the design principles from ACO routing. The algorithm has a hybrid architecture, combining a reactive path setup phase, which is typical for purely reactive algorithms such as AODV [37], with proactive monitoring and exploration using sampling with ant agents. Further reactive elements are introduced for dealing with link failures: while proactive path

sampling allows for the updating of information about current paths and for the finding of new paths, the use of mechanisms to reactively deal with link failures enhances direct adaptivity. A last important feature of AntHocNet is the fact that the process of path sampling using ant agents is supported by a lightweight information bootstrapping mechanism: the routing information learned by the ant agents is spread over the networks in a process we call pheromone diffusion. Although information bootstrapping is more efficient, it is less reliable and robust than ant sampling, and it is therefore used as a secondary process to guide and speed up the learning by the ants.

In what follows, we first give a general overview of the AntHocNet algorithm, and then discuss each of its components in more detail. For other, more detailed descriptions of the algorithm, we refer the interested reader to [47–51].

Overview of the Algorithm In AntHocNet nodes only actively gather and maintain routing information for destinations they are currently communicating with. At the start of a communication session, the source node gathers initial routing information in a *reactive path setup* phase. During the course of the session, the source node engages in *proactive route maintenance and exploration*. To this end, it periodically sends out ant-like agents, to sample paths to the destination, very much like in ACO routing algorithms for wired networks. This basic mechanism is supported by the previously discussed *pheromone diffusion* process: the routing information obtained via repeated ant sampling is spread between the nodes of the MANET via information bootstrapping to provide secondary guidance. When a link failure is detected during the course of a session, this is dealt with using *reactive link failure mechanisms*, such as a local route repair mechanism and the spreading of failure notification messages. The use of all of these mechanisms together results in the availability of a set of multiple paths for each communication session. *Data packets are spread stochastically* over these different paths, according to the learned pheromone tables.

Pheromone Routing Tables Like other ACO routing algorithms, AntHocNet uses the datagram model of IP networks, where paths are expressed in the form of tables kept locally at each node. A pheromone table $\mathcal{T}^i$ at node i is a matrix, where each entry $\mathcal{T}^i_{nd} \in \mathbb{R}$ of the table is an artificial pheromone value indicating the estimated goodness of going from i over neighbor n to reach destination d. Goodness is a combined measure of path end-to-end delay and number of hops. Since AntHocNet only maintains information about destinations that are active in a communication session, and since the neighbors of a node change continually, the filling of the pheromone tables is sparse and dynamic. The learned tables are used to route data packets in a stochastic forwarding process (see further).

Reactive Path Setup When a source node s starts a communication session with a destination node d, and it does not have routing information for d available, it broadcasts a *reactive forward ant* to obtain initial information. At each

node, the ant is either unicast or broadcast, depending on whether the current node has or has not routing information for d. If pheromone information is available, the ant chooses its next hop n with the probability P_{nd} which depends on the relative goodness of n as a next hop, expressed in the pheromone variable $\mathcal{T}^i_{nd}$:

$$P_{nd} = \frac{(\mathcal{T}^i_{nd})^\beta}{\sum_{j \in \mathcal{N}^i_d} (\mathcal{T}^i_{jd})^\beta}, \quad \beta \geq 1, \tag{7}$$

where $\mathcal{N}^i_d$ is the set of neighbors of i over which a path to d is known, and β is a parameter value that controls the exploratory behavior of the ants. If no pheromone information is available, the ant is broadcast. Due to subsequent broadcasts, many duplicate copies of the same ant travel to the destination. A node that receives multiple copies only accepts the first and discards the others. This way, only one path is set up initially. During the course of the communication session more paths are added via the proactive path exploration and maintenance mechanism to provide a *mesh of multiple paths* for data forwarding.

Each forward ant keeps a list $\mathcal{P} = [1, 2, \ldots, d]$ of the nodes it has visited. Upon arrival at the destination d, it is converted into a *backward ant* that travels back to the source retracing $\mathcal{P}$. At each intermediate node $i \in \mathcal{P}$ $(i < d)$, the backward ant reads a locally maintained estimate $\hat{T}^i_{i+1}$ of the time it takes to reach the neighbor $i+1$ the ant is coming from. The time $\hat{T}^i_d$ it would take a data packet to reach d from i over $\mathcal{P}$ is calculated incrementally as the sum of the local estimates $\hat{T}^j_{j+1}$ gathered by the ant between i and d. A pheromone value is a goodness measure, expressed as an inverted cost, which takes into account both end-to-end delay and number of hops. It has the dimension of an inverted time. Therefore, to calculate the pheromone update value τ^i_d, we combine the estimated delay $\hat{T}^i_d$ with the number of hops to the destination h as follows:

$$\tau^i_d = \left(\frac{\hat{T}^i_d + hT_{hop}}{2} \right)^{-1}, \tag{8}$$

where T_{hop} is a fixed value representing the time to take one hop in unloaded conditions. Defining τ^i_d like this is a way to avoid possibly large oscillations in the time estimates gathered by the ants (e.g., due to local bursts of traffic) and to take into account both end-to-end delay and number of hops. The pheromone value $\mathcal{T}^i_{nd}$ is updated as follows:

$$\mathcal{T}^i_{nd} = \gamma \mathcal{T}^i_{nd} + (1 - \gamma)\tau^i_d, \quad \gamma \in [0, 1]. \tag{9}$$

Once the backward ant makes it back to the source, a full path is set up and the source can start sending data. If the backward ant does not arrive for some reason, a timer will run out at the source, and the whole process is started again.

Proactive Path Maintenance and Exploration During the course of a session, source nodes send out *proactive forward ants* to update the information

about currently used paths and to try to find new paths. They follow pheromone and update routing tables in the same way as reactive forward ants. As pointed out previously, the ant sending rate needed to keep up with the constant changes of a MANET environment is quite high, so that the process comes into conflict with the typically limited bandwidth in such networks. Moreover, to find entirely new paths, blind exploration through random walks or broadcasts would be needed, again leading to excessive bandwidth consumption. Therefore, we use a supporting process, called *pheromone diffusion*. It provides a second way of updating pheromone information about existing paths and can give information to guide exploratory behavior.

Pheromone diffusion is implemented using *hello messages*, broadcast periodically and asynchronously by the nodes. In these messages, the sending node n places a list of destinations it has information about, including for each of these destinations d the best pheromone value $\mathcal{T}^n_{m^* d}, m^* \in \mathcal{N}^n_d$, which n has available for d. A node i receiving the hello message from n first registers that n is its neighbor. Then, for each destination d listed in the message, it can derive an estimate of the goodness of going from i to d over n, combining the cost of hopping from i to n with the reported pheromone value $\mathcal{T}^n_{m^* d}$. We call the obtained estimate the *bootstrapped pheromone* variable $\mathcal{B}^i_{nd}$, since it is built up using an estimate which is non-local to i. This bootstrapped pheromone variable can in turn be forwarded in the next hello message sent out by i, giving rise to a bootstrapped pheromone field over the MANET. This way of spreading information over a network is based on information bootstrapping techniques used in dynamic programming and it is often used in traditional routing algorithms for wired networks [46]. It is an efficient process, but can be slow to converge.

For the *maintenance* of existing paths, a bootstrapped pheromone is used directly. If i already has a pheromone entry $\mathcal{T}^i_{nd}$ in its routing table for destination d going over neighbor n, $\mathcal{B}^i_{nd}$ is treated as an update of the goodness estimate of this path, and is used directly to replace $\mathcal{T}^i_{nd}$. Due to the slow multi-step forwarding of bootstrapped pheromone in hello messages, this information does not provide the most accurate view of the current situation. However, the information is obtained via a lightweight, efficient process, and is complemented by the explicit path updating done by the ants. In this way we have two updating frequencies in the path maintenance process.

For path *exploration*, a bootstrapped pheromone is used indirectly. If i does not yet have a value for $\mathcal{T}^i_{nd}$ in its routing table, $\mathcal{B}^i_{nd}$ could indicate a possible new path from i to d over n. However, this path has never been sampled explicitly by an ant, and due to the slow convergence of the multi-step pheromone bootstrapping process it could be inaccurate, containing undetected loops or dangling links. It is therefore not used directly for data forwarding. It is seen as a sort of virtual pheromone, which needs to be tested. Proactive forward ants will use both the regular and the virtual pheromone to find their way to the destination, so that they can test the proposed new paths. This way, promising virtual pheromone is investigated, and if the investigation is successful it is turned into a regular path that can be used for data.

Reactively Dealing with Link Failures Nodes can detect link failures when unicast transmissions fail, or when expected periodic hello messages were not received from a neighbor. When a neighbor disappears, the node takes a number of actions. First, it removes the neighbor from its neighbor table and all associated entries from its pheromone table. Next, the node broadcasts a link failure message to notify other nodes of the changed situation. The message contains a list of the destinations to which the node lost its best path, and the new best pheromone to this destination (if it still has entries for the destination). All its neighbors receive the message and update their pheromone table using the new estimates. If they in turn lost their best only path to a destination, they will broadcast a message on to their neighbors, until all concerned nodes are notified of the new situation.

If after the link failure an intermediate node is left with data packets to send but without paths to their destination, the node will start a *local route repair*. The node broadcasts a *route repair ant* that travels to the involved destination like a reactive forward ant: it follows available routing information when it can and is broadcast otherwise. One important difference is that it has a restricted number of broadcasts so that its proliferation is limited. If the local repair fails, the node broadcasts a link failure message to notify other nodes. If the source of a communication session is left with no paths to the destination, it starts a new path setup phase.

Stochastic Data Routing Data are forwarded according to the values of the pheromone entries. Like in other ACO routing algorithms, nodes in AntHocNet *forward data stochastically*. When a node has multiple next hops for the destination d of the data, it randomly selects one of them, with probability P_{nd}. P_{nd} is calculated in the same way as for reactive forward ants, using equation (7). However, a higher value for the exponent β is used in order to avoid the least good paths. The probabilistic routing strategy leads to data load spreading according to the estimated quality of the paths. If estimates are kept up-to-date, this leads to *automatic load balancing*. When a path is clearly worse than others, it is avoided, and its congestion is relieved. Other paths get more traffic, leading to higher congestion, which makes their end-to-end delay increase. By adapting the data traffic, the nodes spread the data load evenly over the network.

4.4 Simulation Results

AntHocNet's performance was evaluated in an extensive set of simulation tests using QualNet [52]. We have studied the behavior of the algorithm under different conditions for network size, connectivity and change rate, radio channel capacity, data traffic patterns, and node mobility. Performance was measured in terms of data delivery ratio, end-to-end packet delay and delay jitter as *measures of effectiveness*, and routing overhead in number of control packets per successfully delivered data packet as *measure of efficiency*. To assess the performance of our algorithm relative to the state of the art in the field, we compare to AODV [37],

which is a de facto standard algorithm and is commonly used in comparative studies. Due to space limitations, we only present a small subset of the results of these simulation tests. For the full set of experiments we again refer to [47–51].

For the tests reported on here, we used MANET scenarios in which 100 nodes are randomly placed in an area of $3000 \times 1000\ m^2$. Each test lasts 900 seconds. Data traffic is generated by 20 constant bit rate (CBR) sources sending one 64-byte packet per second. Each source starts sending at a random time between 0 and 180 seconds after the start of the simulation, and keeps sending until the end. The radio range of the nodes is 300 meters, and the data rate is 2 Mbit/s. At the MAC layer we use the IEEE 802.11b DCF protocol as is common in MANET research. The nodes move according to the *random waypoint* (RWP) mobility model [38]: they choose a random destination point and a random speed, move to the chosen point with the chosen speed, and then rest at that point for a fixed amount of pause time before they choose a new destination and speed. The speed is chosen between 0 and 20 m/s. The pause time is the variable over which we compare the algorithms.

We created experiments using pause times from 0 up to 480 seconds. Higher pause times lead to slower changing environments (so less link failures), but also to sparser scenarios and hence to lower connectivity. This is because moving nodes tend to cluster around the middle of the MANET area, whereas nodes that pause are spread out randomly (see [53] for properties of the node distribution under RWP mobility). For each pause time we made 10 different test runs. The results of the tests are presented in Figs. 7 (average delay and delivery ratio) and 8 (average jitter and overhead). AntHocNet shows much better effectiveness than AODV, in terms of average delay, delivery ratio and jitter. AODV has better efficiency, measured as routing overhead, but the difference is rather small. The bad efficiency for high pause times is due to the reduced connectivity: AntHocNet has a high frequency to retry failed path setups, leading to high overhead in case source and destination are not connected.

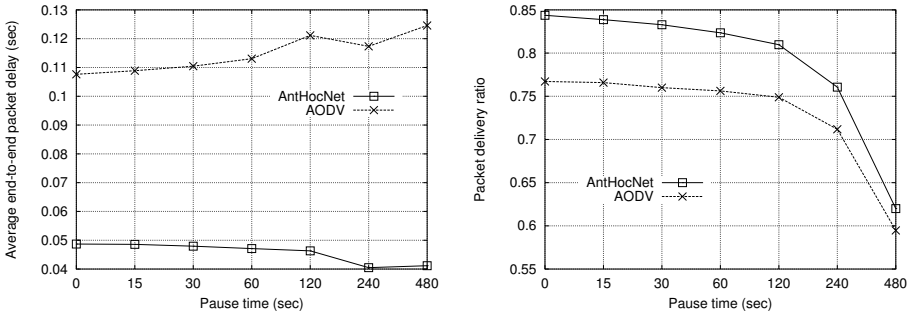

Fig. 7. Average delay and delivery ratio for increasing pause times

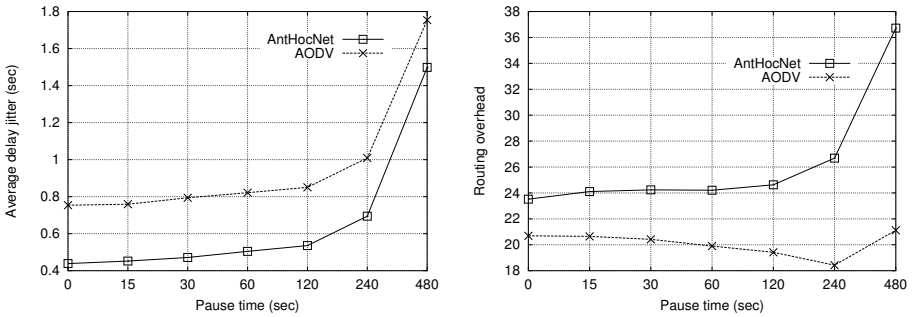

Fig. 8. Average jitter and overhead for increasing pause times

In order to illustrate the difference in adaptivity between AntHocNet and AODV, we show the detailed evolution of the end-to-end delay (rather than showing average values) over the course of a test run in which some important events take place. We use the same setup as before, but keep the pause time constant at 30 seconds, and use different data traffic patterns in order to introduce disruptive events. 10 randomly chosen sources start to send to one single destination between 100 and 110 seconds after the simulation begins, and they keep on sending until the end. After 300 seconds, 20 new sources start to send to a different single destination. 200 seconds later these stop again. All sources send four 64 byte packets per second. Fig. 9 shows, for one communication session, how the end-to-end delay, averaged per 10 seconds, evolves throughout the simulation. The arrival of 20 new sessions after 300 seconds leads to a long period of unstable behavior. The congestion caused by the increased data traffic not only leads to longer queueing times, but also to higher interference that can cause transmissions to fail. Since failed transmissions are usually treated as link failures by routing algorithms, they often trigger strong reactions. As can be seen in the figure, AntHocNet deals with this challenge in a much smoother way than AODV. After the end of the 20 sessions, at second 500, the situation stabilizes again, but faster for AntHocNet than for AODV.

4.5 Conclusions

In this section we have described AntHocNet, a novel ACO routing algorithm for MANETs. It is a hybrid algorithm that combines reactive route setup with proactive route maintenance and exploration. The main learning mechanism based on path sampling using ant agents is supported by a lightweight information bootstrapping process. Link failures are also dealt with in a hybrid way: proactive protection is provided by the use of multiple paths, whereas reactive mechanisms like route repair and link failure notification messages are used to enhance the adaptivity in the highly mobile MANET environments. In simulation tests, we show that AntHocNet can outperform AODV in different environments, and for different evaluation measures. A detailed examination of the evolution of

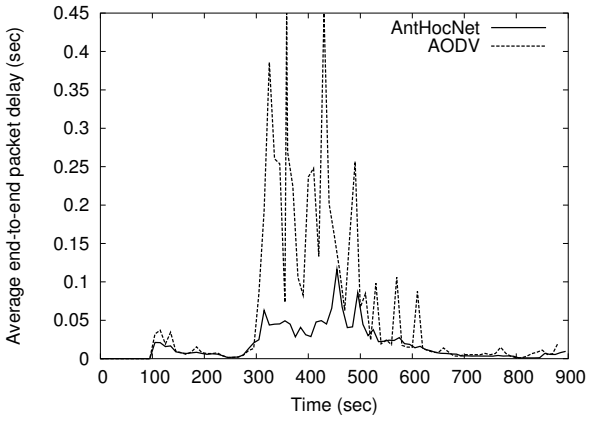

Fig. 9. Evolution of the end-to-end delay over the course of a test run

the algorithm's performance during the course of an experiment illustrates its adaptivity.

5 Conclusion

In this paper we have presented a number of recent algorithms for failure location and restoration in both IP-over-fiber and wireless ad-hoc networks.

For failure location we use a Maximum Likelihood inference to correct error alarms. This approach, together with a standard set-cover heuristic, turned out to handle false and missing alarms far better than the techniques proposed to date.

For failure restoration in IP-over-fiber networks we described two different algorithms, SMART and FastSurv. Both are much faster and more scalable than any solution proposed to date. Moreover, each approach has a number of possible extensions. SMART provides us with a novel method of formal verification of the existence of a survivable mapping and a tool tracing and repairing the vulnerable areas of the network, whereas FastSurv can be easily adapted to any set of real-life requirements such as capacity constraints.

In wireless ad-hoc networks the connectivity changes so quickly, that failure restoration is in fact equivalent to a highly adaptive routing algorithm. Therefore we addressed this problem separately, and proposed AntHocNet - a routing algorithm inspired by Ant Colony Optimization. AntHocNet outperforms standard algorithms in terms of efficiency and scalability.

References

1. Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.N., Diot, C.: Characterization of Failures in an IP Backbone. Proc. of IEEE INFOCOM'04 (2004)
2. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multhop routing in sensor networks. In: The First ACM Conference on Embedded Networked Sensor Systems, Los Angeles, CA, USA (2003)
3. Kurant, M., Nguyen, H.X., Thiran, P.: Survey on Dependable Networks. In: Dependable Systems: Software, Computing, Networks. Springer Lecture Notes in Computer Science (2006)
4. Mas, C., Nguyen, H., Thiran, P.: Failure location in WDM networks. In: Optical WDM Networks: Past Lessons and Path Ahead. Kluwer Academic Publishers (2004)
5. ITU-T Rec. G.783. Characteristics of SDH equipment functional blocks (1997)
6. Nguyen, H.X., Thiran, P.: Failure location in all optical networks: the assymetry between false and missing alarms. In: Proceedings of ITC 19. (2005)
7. Carr, R.D., Doddi, S., Konjevod, G., Marathe, M.: On the red-blue set cover problem. In: Proceedings of Symposium on Discrete Algorithms. (2000)
8. Nguyen, H.X., Thiran, P.: Using end-to-end data to infer lossy links in sensor networks. In: To appear in the proceedings of INFOCOM 2006. (2006)
9. Padmanabhan, V.N., Qiu, L., Wang, H.J.: Server-based inference of internet performance. In: Proceedings of the IEEE INFOCOM'03, San Francisco, CA (2003)
10. Schmid, T., Dubois-Ferriere, H., Vetterli, M.: Sensorscope: Experiences with a wireless building monitoring. In: Proceedings of Workshop on Real-World Wireless Sensor Networks (REALWSN'05). (2005)
11. Iannaccone, G., Chuah, C.N., Bhattacharyya, S., Diot, C.: Feasibility of IP restoration in a tier-1 backbone. Sprint ATL Research Report Nr. RR03-ATL-030666 (2003)
12. Armitage, J., Crochat, O., Boudec, J.Y.L.: Design of a Survivable WDM Photonic Network. Proceedings of IEEE INFOCOM 97 (1997)
13. Modiano, E., Narula-Tam, A.: Survivable lightpath routing: a new approach to the design of WDM-based networks. IEEE Journal on Selected Areas in Communications **20** (2002) 800–809
14. Giroire, F., Nucci, A., Taft, N., Diot, C.: Increasing the Robustness of IP Backbones in the Absence of Optical Level Protection. Proc. of IEEE INFOCOM 2003 (2003)
15. Leonardi, E., Mellia, M., Marsan, M.A.: Algorithms for the Logical Topology Design in WDM All-Optical Networks. Optical Networks Magazine (2000)
16. Sen, A., Hao, B., Shen, B.: Survivable routing in WDM networks. In: Proceedings of the 7^{th} International Symposium on Computers and Communications (ISCC). (2002)
17. Crochat, O., Boudec, J.Y.L.: Design Protection for WDM Optical Networks. IEEE Journal of Selected Areas in Communication **16** (1998) 1158–1165
18. Nucci, A., Sansò, B., Crainic, T., Leonardi, E., Marsan, M.A.: Design of Fault-Tolerant Logical Topologies in Wavelength-Routed Optical IP Networks. Proc. of IEEE Globecom 2001 (2001)
19. Fumagalli, A., Valcarenghi, L.: IP Restoration vs. WDM Protection: Is There an Optimal Choice? IEEE Network (2000)
20. Sahasrabuddhe, L., Ramamurthy, S., Mukherjee, B.: Fault management in IP-Over-WDM Networks: WDM Protection vs. IP Restoration. IEEE Journal on Selected Areas in Communications **20** (2002)

21. Sasaki, G.H., Su, C.F., Blight, D.: Simple layout algorithms to maintain network connectivity under faults. In Proceedings of the 2000 Annual Allerton Conference (2000)
22. Kurant, M., Thiran, P.: Survivable Mapping Algorithm by Ring Trimming (SMART) for large IP-over-WDM networks. Proc. of BroadNets 2004 (2004)
23. Kurant, M., Thiran, P.: On Survivable Routing of Mesh Topologies in IP-over-WDM Networks. Proc. of Infocom'05 (2005)
24. Ducatelle, F., Gambardella, L.: A scalable algorithm for survivable routing in IP-over-WDM networks. In: Proceedings of the First Annual International Conference on Broadband Networks (BroadNets). (2004)
25. Ducatelle, F., Gambardella, L.: Fastsurv: A new efficient local search algorithm for survivable routing in WDM networks. In: Proceedings of Annual IEEE Global Telecommunications Conference (Globecom). (2004)
26. Ducatelle, F., Gambardella, L.: Survivable routing in IP-over-WDM networks: An efficient and scalable local search algorithm. Optical Switching and Networking (2005) To appear.
27. Gross, J., Yellen, J.: Graph Theory and its Applications. CRC Press, 1999 (1999)
28. Frank, A.: Packing paths, circuits and cuts - a survey (in Paths, Flows and VLSI-Layout). Springer, Berlin, 1990 (1990)
29. Li, G., Doverspike, B., Kalmanek, C.: Fiber Span Failure Protection in Mesh Optical Networks. Optical Networks Magazine **3** (2002) 21–31
30. Kim, S., Lumetta, S.: Addressing node failures in all-optical networks. Journal of Optical Networking **1** (2002) 154–163
31. Choi, H., Subramaniam, S., Choi, H.A.: On Double-Link Failure Recovery in WDM Optical Networks. Proc. of IEEE INFOCOM'02 (2002)
32. Royer, E., Toh, C.K.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications (1999)
33. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization. Artificial Life **5** (1999) 137–172
34. Di Caro, G.: Ant Colony Optimization and its application to adaptive routing in telecommunication networks. PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Brussels, Belgium (2004)
35. Abolhasan, M., Wysocki, T., Dutkiewicz, E.: A review of routing protocols for mobile ad hoc networks. Ad Hoc Networks **2** (2004) 1–22
36. Perkins, C., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications. (1994)
37. Perkins, C., Royer, E.: Ad-hoc on-demand distance vector routing. In: Proc. of the 2^{nd} IEEE Workshop on Mobile Computing Systems and Applications. (1999)
38. Johnson, D., Maltz, D.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: Mobile Computing. Kluwer (1996) 153–181
39. Broch, J., Maltz, D., Johnson, D., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proc. of the 4^{th} Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom98). (1998)
40. Haas, Z.: A new routing protocol for the reconfigurable wireless networks. In: Proc. of the IEEE Int. Conf. on Universal Personal Communications. (1997)
41. Mueller, S., Tsang, R., Ghosal, D.: Multipath routing in mobile ad hoc networks: Issues and challenges. In: Performance Tools and Applications to Networked Systems. Volume 2965 of LNCS. Springer-Verlag (2004)

42. Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraulaz, G., Bonabeau, E.: Self-Organization in Biological Systems. Princeton University Press (2001)
43. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics–Part B **26** (1996) 29–41
44. Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: Ant-based load balancing in telecommunications networks. Adaptive Behavior **5** (1996) 169–207
45. Di Caro, G., Dorigo, M.: AntNet: Distributed stigmergetic control for communications networks. J. of Artificial Intelligence Research **9** (1998) 317–365
46. Bertsekas, D., Gallager, R.: Data Networks. Prentice–Hall, Englewood Cliffs, NJ, USA (1992)
47. Di Caro, G., Ducatelle, F., Gambardella, L.: AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In: Proceedings of Parallel Problem Solving from Nature (PPSN VIII). LNCS, Springer-Verlag (2004)
48. Ducatelle, F., Di Caro, G., Gambardella, L.: Ant agents for hybrid multipath routing in mobile ad hoc networks. In: Proceedings of The Second Annual Conference on Wireless On demand Network Systems and Services (WONSS). (2005)
49. Di Caro, G., Ducatelle, F., Gambardella, L.: Swarm intelligence for routing in mobile ad hoc networks. In: Proceedings of the 2005 IEEE Swarm Intelligence Symposium (SIS). (2005)
50. Di Caro, G., Ducatelle, F., Gambardella, L.: AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications **16** (2005)
51. Ducatelle, F., Di Caro, G., Gambardella, L.: Using ant agents to combine reactive and proactive strategies for routing in mobile ad hoc networks. International Journal of Computational Intelligence and Applications (IJCIA) (2005) To appear.
52. Scalable Network Technologies, Inc. Culver City, CA, USA: Qualnet Simulator, Version 3.6. (2003) `http://stargate.ornl.gov/trb/tft.html`.
53. Bettstetter, C., Resta, G., Santi, P.: The node distribution of the random waypoint mobility model for wireless ad hoc networks. IEEE Transactions on Mobile Computing **2** (2003) 257–269