Models and Techniques for Communication in Dynamic Networks

Christian Scheideler

Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
scheideler@cs.jhu.edu

Abstract. In this paper we will present various models and techniques for communication in dynamic networks. Dynamic networks are networks of dynamically changing bandwidth or topology. Situations in which dynamic networks occur are, for example: faulty networks (links go up and down), the Internet (the bandwidth of connections may vary), and wireless networks (mobile units move around). We investigate the problem of how to ensure connectivity, how to route, and how to perform admission control in these networks. Some of these problems have already been partly solved, but many problems are still wide open. The aim of this paper is to give an overview of recent results in this area, to identify some of the most interesting open problems and to suggest models and techniques that allow us to study them.

1 Introduction

In this paper we present various models and techniques for communication in dynamic networks. Generally speaking, dynamic networks are networks of dynamically changing network characteristics such as bandwidth or topology. There are many scenarios in which dynamic networks occur. For example, a fixed interconnection network might experience (adversarial or random) faults, or we may have a wireless network formed by users that move around. Also, the Internet can be seen as a dynamic network, since some virtual connections or communication links might experience large differences in their availability over time, offering a low bandwidth at one time and a high bandwidth at another time.

In the future, dynamic networks will play an increasingly important role, since the development and deployment of powerful and flexible communication systems that are accessible by large parts of the population is seen as a key component for the transition to an information society. While this has been dominated in the past 10 years by the establishment of wired connections to the Internet, this is now shifting more and more to wireless connections, since wireless connections offer much higher mobility and flexibility. Many companies and universities have already deployed wireless access points that allow employees and/or students to access their local area network or the Internet via mobile devices such as laptops and palmtops. So far, wireless communication has mostly been restricted to small areas, mostly because currently used strategies have big problems with establishing and maintaining routes in large-scale, mobile networks. In fact, the development of protocols for communication in mobile, wireless networks that are *provably* efficient and robust even under severe circumstances (for example, quickly moving devices) is still in its infancy.

There are several reasons why only a few theoretical results are known for dynamic networks so far. First of all, it is quite challenging to analyze the behavior of communication strategies in dynamic networks. Furthermore, it is quite difficult to formulate models for dynamic networks that are on the one hand abstract enough to allow a mathematical analysis and on the other hand close enough to reality so that they are of practical value. In this paper, we will present some of the models that have recently been used and also suggest some new models to stimulate fundamental research in this field. We will also describe some important, recent results in the area of dynamic networks. The results demonstrate that despite the (seemingly) enormous mathematical challenges, one can prove very strong results about communication in dynamic networks even in fairly general models.

Before we state any models and techniques, we first introduce some basic notation and describe the basic framework underlying all of these models. Afterwards, we will describe various models and approaches for faulty networks, the Internet, wireless networks, and adversarial networks. We will not try to give an exhaustive overview of results in each of these areas, since this would certainly require to write a whole book. Instead, we will rather concentrate on some, to our mind important results in these fields.

1.1 The basic model

In this paper we will only consider communication networks whose topology can be modelled as a directed graph $G = (V, E)$ with a set of nodes V representing the processing units or routers and a set of edges $E \subseteq V \times V$ representing the communication links. In some of our models we may assume that each edge $e \in E$ has a cost associated with it, which will usually represent its *capacity* (i.e. the number of packets it can forward in a step). By default, the capacity of an edge is equal to 1.

In a *static* network the communication links (and their cost values) do not change, i.e. the topology can be described by a fixed directed graph. In this paper we will concentrate on *dynamic* networks, i.e. networks in which the connections (or their properties) between the processing units are continuously changing.

In practice, network communication is performed in many different layers:

- *physical layer*: defines the physical transport of data from one node to the other
- *link layer*: defines the mechanism by which data is transmitted from one node to another
- *network layer*: defines functions for route selection and congestion control

– *transport layer*: defines functions for the end-to-end transport of data

If a shared medium is used such as in wireless communication, the link layer is also called *medium access control (MAC) layer*. We will concentrate on the upper three layers. Issues of particular importance will be connectivity, routing, and admission control. By "routing" we understand the process of moving information across a network from a source to a destination. Routing involves two basic activities:

– the determination of routing paths, and
– the transport of information groups (typically called *packets*) along the paths.

The first item is called *path (or route) selection*, and the second item is usually called *switching* (or *scheduling*). (Depending on the research community or research field, both path selection algorithms and switching algorithms have sometimes been called routing algorithms. We want to avoid these ambiguities here.) Different switching models have been defined in the past such as (virtual) circuit, wormhole, or packet switching. Here, we will only concentrate on packet switching. In packet switching models it is usually assumed that all packets are atomic (i.e. unsplittable) objects of uniform size. Time proceeds in synchronous time steps. In each time step, each link of capacity c can forward up to c packets. If too many packets contend to use the same link at the same time, then the switching algorithm has to decide which of them to prefer. Once a packet reaches its destination it is discarded.

We will concentrate on routing protocols in which the nodes locally decide which packets to move forward in each step, i.e., a decision only depends on the routing information carried by the packets and on the local history of the execution. These algorithms are called *local control* or *online* algorithms.

In several of the examples below we will study the performance of online routing protocols under an adversarial injection model. For static networks, the following model has widely been used, which is an extension by Andrews et al. [4] of a model introduced by Borodin et al. [12]:

Suppose that we have an adversary that is allowed to demand network bandwidth up to a prescribed injection rate $\lambda < 1$. For any $w \in \mathbb{N}$, an adversary is called (w, λ)-*bounded* if for all edges e and all time intervals I of length w, it injects no more than $\lambda \cdot w$ packets during I that contain edge e in their routing path. A protocol is called *stable* for a given injection rate λ and network G if for any (w, λ)-bounded adversary the number of packets stored in injection or edge buffers in G does not grow unboundedly with time. Moreover, a protocol is called *universally stable* if it is stable for any $\lambda < 1$ and any network G.

2 Communication in Faulty Networks

We start with the problem of efficient communication in faulty networks. Different fault models have been studied in the literature: faults may be permanent or temporary, nodes and/or edges may break down, and faults may happen at random or may be caused by an adversary or attacker. In the following, the former faults are called *random faults*, and the latter faults are called *worst-case faults*

Central questions in the area of faulty networks are:

- How frequently does a faulty network have to be repaired so that (either with high probability of with certainty) the size of its largest connected component is always a constant fraction of the original size?
- Or better, how frequently does a faulty network have to be repaired so that (either with high probability of with certainty) its largest connected component still offers a connectivity and throughput that is similar to the original network?
- How can an uninterrupted service be offered in faulty networks?
- Are there online routing algorithms that achieve a throughput in faulty networks that is close to a best possible throughput?

These questions will be addressed in the following subsections. We will show that for several of these questions, partial answers have already been obtained. Moreover, we will point out what parts of the questions are still wide open.

2.1 Large connected components in faulty networks

Suppose that the information is distributed among the nodes of a network in such a way that as long as at least a constant fraction of the nodes stays connected, agreement can be achieved resp. no information is lost. (This can, for example, easily be reached with the help of the information dispersal approach by Rabin [37].) Then we only have to start repairing the network once the largest connected component in it is below a certain constant fraction. The question is, how long do we have to wait for that? In order to address this question, we start with considering random faults and afterwards consider worst-case faults.

Given a graph G and a probability value of p, let $G^{(p)}$ be the random graph obtained from G by keeping each edge of G independently at random with probability p (i.e. p is the *survival probability*). Given a graph G, let $\gamma(G) \in [0,1]$ be the fraction of nodes of G contained in a largest connected component.

Let $\mathcal{G} = \{G_n \mid n \in \mathbb{N}\}$ be any family of graphs with parameter n. Let p^* be the *critical probability* for the existence of a linear-size connected component in G, i.e. for every constant $\epsilon > 0$:

1. for every $p > (1+\epsilon)p^*$ there exists a constant $c > 0$ with
$$\lim_{n\to\infty} \Pr[\gamma(G_n^{(p)}) > c] = 1$$
2. for all constants $c > 0$ and for all $p < (1-\epsilon)p^*$:
$$\lim_{n\to\infty} \Pr[\gamma(G_n^{(p)}) > c] = 0$$

Of course, it is not obvious whether critical probabilities exist. However, the results by Erdős and Rényi [15] and its subsequent improvements (e.g. [11, 34]) imply that for the complete graph on n nodes, $p^* = 1/(n-1)$, and that for a random graph with $d \cdot n/2$ edges, $p^* = 1/d$. For the 2-dimensional $n \times n$-mesh, Kesten showed that $p^* = 1/2$ [28]. Ajtai, Komlós and Szemerédi proved that for the hypercube of dimension n, $p^* = 1/n$ [3]. For the n-dimensional butterfly network, Karlin, Nelson and Tamaki showed that $0.337 < p^* < 0.436$ [27].

If we define the *robustness* of a graph of average degree d as $r = p^* \cdot d$, then it certainly holds: the smaller r, the more robust is the graph (and therefore the longer repairs on it can be postponed.) Using the results for p^* above, we obtain for the

- complete graph: $r = 1$
- n-dimensional hypercube: $r = 1$
- 2-dimensional mesh: $r = 2$
- n-dimensional butterfly: $r \geq 1.348$
- random graph with $d \cdot n/2$ edges: $r = 1$

So the best graph of regular structure that is the easiest to build is the hypercube. An interesting open question is whether there is also a constant degree graph of regular structure with $r = 1$ (or at least an r that is arbitrarily close to 1).

Also worst-case fault models have been investigated. Leighton and Maggs showed that there is a so-called *indirect* constant-degree network connecting n inputs with n outputs via $\log n$ levels of n nodes each, which they named multibutterfly, that has the following property: no matter how an adversary chooses f nodes to fail, there will be a connected component left in it with at least $n - O(f)$ inputs and at least $n - O(f)$ outputs [31]. (In fact, one can even still route packets between the inputs and outputs in this component in almost the same amount of time steps as in the ideal case.) Later, Upfal showed that there is also a *direct* constant-degree network on n nodes, a so-called expander, that fulfills: no matter how an adversary chooses f nodes to fail, there will be a connected component left in it with at least $n - O(f)$ nodes [40]. Both results are optimal up to constants. However, it is not known what the best constants are that can be reached, and whether similar results can also be reached by networks of regular structure (i.e. that are easy to build). For some classes of graphs such as the class of complete graphs, the exact constant is easy to determine: a complete graph fulfills that no matter how an adversary chooses f nodes to fail, there will be a connected component left with $n - f$ nodes.

2.2 Simulation of fault-free networks by faulty networks

Next we look at the problem that we do not only demand that the remaining component is large, but that it can still be used for efficient communication (such as in the case of the multibutterfly above).

Consider the situation that there can be up to f worst-case node faults in the system at any time. One way to check whether the largest remaining component still allows efficient communication is to check whether it is possible to embed into the largest connected component of a faulty network a fault-free network of the same size and kind. An *embedding* of a graph G into a graph H maps the nodes of G to non-faulty nodes of H and the edges of G to non-faulty paths in H. An embedding is called *static* if the mapping of the nodes and edges is fixed. Both static and dynamic embeddings have been used. A good embedding is one with minimum load, congestion, and dilation, where the *load* of an embedding is the maximum number of nodes of G that are mapped to any single node of H, the *congestion* of an embedding is the maximum number of paths that pass through any edge e of H, and the *dilation* of an embedding is the length of the longest path. The load, congestion, and dilation of the embedding determine the time required to emulate each step of G on H. In fact, Leighton, Maggs, and Rao have shown [32] that if there is an embedding of G into H with load ℓ, congestion c, and dilation d, then H can emulate any communication step (and also computation step) on G with slowdown $O(\ell + c + d)$.

When demanding a constant slowdown, only a few results are known so far. In the case of worst-case faults, it was shown by Leighton, Maggs and Sitaraman (using of dynamic embedding strategies) that an n-input butterfly with $n^{1-\epsilon}$ worst-case faults (for any constant ϵ) can still emulate a fault-free butterfly of the same size with only constant slowdown [33]. Furthermore, Cole, Maggs and Sitaraman showed that an $n \times n$ mesh can sustain up to $n^{1-\epsilon}$ worst-case faults and still emulate a fault-free mesh of the same size with (amortized) constant slowdown [14]. It seems that the n-node hypercube can even achieve a constant slowdown for $n^{1-\epsilon}$ worst-case faults, but so far only partial answers have been obtained [33].

Also random faults have been studied. For example, Håstad, Leighton and Newman [24] showed that if each edge of the hypercube fails independently with any constant probability $p < 1$, then the functioning parts of the hypercube can be reconfigured to simulate the original hypercube with constant slowdown. Combining this with what we know from Section 2.1, the hypercube seems to be an extremely good candidate for a fault-tolerant network.

For a list of further references concerning embeddings of fault-free into faulty networks see the paper by Leighton, Maggs and Sitaraman [33].

The embedding results immediately imply that there are values f so that as long as there are at most f faults at any time, then for certain networks its largest component will basically have the same communication properties as the original network at any time. The problem is, how to exploit this knowledge for routing in networks of dynamically changing faults, since it may not be known where future faults occur, especially if they are under the control of an adversary. An interesting approach might be to combine the results mentioned here with the models and approaches in Section 2.4.

2.3 Fault-tolerant path selection in networks

Next we consider the problem of adapting paths selected for data streams (such as multimedia streams) to faults in networks. We first start with suitable models for online path selection in fault-free networks.

In the *edge-disjoint paths (EDP) problem* we are given an undirected graph $G = (V, E)$ and a set T of pairs of nodes (s_i, t_i) (also called *requests*), and the problem is to find a subset $T' \subseteq T$ of maximum size so that each request in T' has an edge-disjoint path.

A generalization of the EDP is the *unsplittable flow problem* (UFP): each edge $e \in E$ is given a capacity of c_e and each request (s_i, t_i) has a demand of d_i. The task is to choose a subset $T' \subseteq T$ such that each request (s_i, t_i) in T' can send d_i flow along a single path, all capacity constraints are kept, and the sum of the demands of the requests in T' is maximized. (More general variants of the UFP also assign a profit to each request and consider the problem of maximizing the sum of the profits of the accepted requests.)

The EDP and UFP are classical optimization problems in the routing area, and there is a substantial body of literature on these two problems (see, for example, [6, 29, 10, 30]). The EDP and UFP are useful models to study, for example, the performance of admission control algorithms for multimedia streams in fault-free networks. However, the EDP and the UFP are only of limited use for practical purposes, since they do not allow a rapid adaptation to edge faults or heavy load conditions. In fact, if an edge fault occurs, then the requests using that edge may have to be cancelled, since only a single flow path was taken for each request and alternative paths might not be available at the time of the edge fault. A straightforward solution to this problem would be to reserve multiple paths for each accepted request that can flexibly be used to ensure rapid adaptability. However, the paths should be chosen so that not too much bandwidth is wasted under normal conditions. Motivated by these thoughts, we suggest the following two optimization problems: the k edge-disjoint paths problem, and the k-splittable flow problem.

In the *k edge-disjoint paths problem* (k-EDP), we are given an undirected graph $G = (V, E)$ and a set of terminal pairs (or requests) T, and the problem is to find a subset $T' \subseteq T$ of maximum size so that each pair in T' is connected by k disjoint paths and also all paths selected for T' are disjoint.

In the *k-splittable flow problem* (k-SFP), we are given an undirected network $G = (V, E)$ with edge capacities and a set T of terminal pairs (s_i, t_i) with demands d_i. The problem is to find a subset $T' \subseteq T$ of maximum size so that each pair $(s_i, t_i) \in T'$ is connected by k disjoint paths of flow value d_i/k and the paths selected for T' satisfy the capacity constraints.

It is not difficult to imagine how the k-SFP can be used to achieve fault tolerance while not wasting too much bandwidth: suppose it is ensured that in the given network G there will be no more than f edge faults at any time. Given a request with demand d, we try to reserve $k + f$ disjoint flow paths for it, $k \geq 1$, with a flow value of d/k for each path (i.e. we raise the total demand for the k-SFP from d to $d \cdot (k + f)/k$). Then it obviously holds that as long as at most f edges break down, it will always be possible to ship a demand of d along the remaining paths. Furthermore, under fault-free conditions, only a fraction of $((d/k)(k+f)-d)/d = f/k$ of the reserved bandwidth is wasted, which can be made sufficiently small by setting k sufficiently large (which may, of course, be limited by the network resources).

Another way of achieving fault tolerance is to select for each accepted request of demand d a path of flow value d and a collection of disjoint, alternative paths of flow value d that are allowed to share their bandwidth with alternative paths of other requests. This allows a better utilization of the links than the two models above but is harder to administer. (If more than one edge is allowed to brake down, *arbitrary* overlaps of alternative paths can lead to the disconnection of a request.) Models that allow bandwidth sharing were considered in [17].

One can study the problems above under two different scenarios: all requests are given in advance, or requests arrive one after the other. In the latter case it is usually assumed that the algorithm must make a decision before seeing future requests. An algorithm is called *c-competitive* if it can accept at least a fraction of $1/c$ of the maximum number of acceptable requests.

Recently, we were able to show that there are online algorithms for the k-EDP that are $O(k^3 \cdot \Delta \cdot \alpha^{-1} \log n)$ competitive, where Δ is the maximum degree, α is the edge expansion, and n is the number of nodes of the given network [9]. To the best of our knowledge, this is the first non-trivial result for the k-EDP. Nothing non-trivial is known for the k-SFP so far.

So many interesting questions in this field are still wide open.

2.4 Routing in faulty networks

There is a large body of literature about routing around faults in specific networks (such as butterflies [13] and multibutterflies [31]. However, not many *universally applicable* approaches are known for routing packets around faults in general networks. We will look at two general approaches that are able to handle certain kinds of faults in arbitrary networks.

The information dispersal approach. Suppose that each edge will be faulty in a time step with probability p, and that this cannot be detected by the system. If p is sufficiently small (we will specify later what this means), then one approach that can be used so that with high probability no message gets lost is to use information dispersal [37]: Suppose that each message consists of k packets. Then it can be encoded in $k+f$ packets so that as long as at least k of the $k+f$ packets reach their destination, the original message can be recovered.

Suppose further that we have the following adversarial injection model: an adversary controls the injection of messages into the network, but guarantees that for each injected message a path can be selected for each of its $k+f$ packets so that for every time interval I of size at least w and every edge e, less than w paths are chosen for messages injected in I that contain e. In this case, we fulfill the conditions of the adversarial injection model introduced in Section 1.1.

It is known that if the paths are revealed to the system, then simple switching disciplines such as longest-in-system or furthest-to-go will ensure that all packets will reach their destination in a finite amount of time steps [4]. This result, however, only holds for fault-free networks. Since we allow edges to be faulty in a time step with probability p, some of the packets may get lost. Suppose now that the longest path taken by a packet has a length of L. Then the probability that any particular packet will get lost is at most $1-(1-p)^L \le L \cdot p$. Hence, if $k, f = \Theta(\log n)$ and $p \le \frac{1}{c \cdot L}$ for a sufficiently large constant c, it can easily be shown via Chernoff bounds that with high probability at least k of the $k+f$ packets of a message will reach their destination. In general, p must be in the order of $O(1/L)$ so that this property can be guaranteed for a constant amount of communication overhead (i.e. $f/k = O(1)$). If edge faults can be detected, then we can allow much higher failure rates, as demonstrated in the next subsection.

Routing via balancing. For the case that edge faults are detectable, another general approach for routing in faulty networks is to use load balancing. We assume that now simply packets are injected. As above, the injections are controlled by an adversary. Almost all of the papers using that model assume that the adversary reveals the paths to the online algorithm. In this case, the only problem that is left is to decide which packet to prefer if several packets content to use the same link at the same time step. Recently, also the case was studied that the adversary does not reveal the paths to the algorithm [2, 18, 7], i.e. the protocol has to determine the paths itself. If edge faults are not known in advance, then this is the only reasonable model.

In the case of edge faults, Aiello et al. [2] suggested the following modification in the definition of the adversary. For any edge e and any time interval I, let $u_{e,I}$ be the amount of time steps e is working in I, and let $i_{e,I}$ be the number of packets injected by an adversary during I that contain e in their path. An adversary is called a (w, ϵ)-adversary if for every edge e and every time interval I of size at least w, $u_{e,I} - i_{e,I} \ge \lceil \epsilon \cdot w \rceil$.

Aiello et al. [2] showed that the following protocol called BASIC will remain stable under this model even if the adversary does not reveal the paths. The protocol maintains several buffers in each node v. For each edge $e = (v, u) \in E$ adjacent to v and for every destination $d \in V$, the protocol maintains a buffer for packets bound for d. Thus, there is a buffer of packets for each triple (v, u, d), which is called $Q_{v,u,d}$. Denote the set of packets in $Q_{v,u,d}$ at time t by $Q^t_{v,u,d}$ and the size of the set $Q^t_{v,u,d}$ as $q^t_{v,u,d}$. Each $q^t_{d,u,d}$ is always assumed to be 0 (as packets that arrive at their destination are immediately removed). Initially, all $q^t_{v,u,d}$ are also assumed to be 0. The following protocol is performed by each node v at each time step t (it is assumed here that edge faults are detectable):

1. For each working $e = (v, u) \in E$, let $d \in V$ be chosen such that $q^t_{v,u,d} - q^t_{u,v,d}$ is maximal over all $d \in V$ (break ties arbitrarily). If $q^t_{v,u,d} - q^t_{u,v,d} > 0$ then send one packet over the edge e from $Q_{v,u,d}$ to $Q_{u,v,d}$.
2. Accept all packets injected by the adversary to the node v.
3. Remove any packets that arrive at their destination.

4. For every d, distribute the packets with destination d among the buffers $Q_{v,u,d}$ as evenly as possible.

This protocol has the following performance.

Theorem 1 ([2]). *For any (w,ϵ)-adversary with $\epsilon > 0$, the number of packets stored at any given time in any of the buffers of* BASIC *is at most $O(m^{3/2}n^{3/2}w/\epsilon')$, where $\epsilon' = \min[\epsilon, 1/w]$.*

The model in [2] appears to be too restrictive, since for every interval I of size at least w, an edge e has to be up more times *in* I than the number of packets with path through e injected *in* I, which can certainly be relaxed. A more general model that avoids the use of intervals is the following:

Suppose that the adversary just has to guarantee that for every injected packet there is a schedule for delivering it over non-faulty edges to its destination. A *schedule* $S = ((e_0,t_0),(e_1,t_1),\ldots,(e_\ell,t_\ell))$ consists of a sequence of movements by which the injected packet P can be sent from its source node to its destination node. For this the adversary must ensure that

– P is injected at the starting point of e_1 at time step t_0 (this is represented by the imaginary *injection edge* e_0),
– the edges $e_1,\ldots,e_\ell$ form a connected path, with the endpoint of e_ℓ being the destination of P,
– the time steps have the ordering $t_0 < t_1 < \ldots < t_\ell$, and
– edge e_i is active at time t_i for all $1 \le i \le \ell$.

Another necessary requirement is that two schedules are not allowed to *intersect*, that is, they are not allowed to have a pair (e,t) in common. This ensures that a schedule represents a valid routing strategy for a packet.

Note that this model allows $(w,0)$-adversaries. Hence, the result by Aiello et al. does not hold any longer in this model.

In order to measure the performance of protocols in this model, we need some parameters. A schedule $S = ((e_0,t_0),(e_1,t_1),\ldots,(e_\ell,t_\ell))$ is called *active* at time t for every t with $t_0 \le t \le t_\ell$. Let A be the maximum number of schedules that are allowed to be active at any time step.

For the special case that all packets must have the same destination, Awerbuch et al. [7] proposed the following algorithm.

Suppose that every node has a single buffer. For any node v and any time step t let $h_{v,t}$ denote the amount of packets in node v at the beginning of time step t. For the destination d, we set $h_{d,t} = -T$ for some parameter T determined later. Initially, all buffers are empty. In each step t, each node v performs the following so-called *T-balancing protocol*:

1. For every edge $e = (v,w)$, check whether $h_{v,t} - h_{w,t} > T$. If so, send a packet from v to w (otherwise do nothing).
2. Receive incoming packets and newly injected packets, and absorb those that reached the destination.

Let Δ be the maximum degree of a node and n be the number of nodes in the system. Then the following result can be shown.

Theorem 2 ([7]). *For any* $A, \Delta \in \mathbb{N}$ *and any* $T \geq 2(\Delta - 1)$*, the* T*-balancing protocol ensures that there are at most* $\max[A, A+T-1] \cdot \binom{n-1}{2} + A$ *packets in the system at any time. Furthermore, the maximum number of packets in a node at any time is at most* $\max[A, A+T-1](n-2) + A$*.*

It is still an interesting open question whether a similar result can also be shown for multiple destinations.

3 Communication in the Internet

Routing algorithms for the internet must fulfill several properties:

- *Optimality*: Optimality refers to the ability of the routing algorithm to select the "best" route. The best route depends on the metrics and metric weights used to make the calculation. For example, one routing algorithm might use the number of hops and the delay, but might weigh delay more heavily in the calculation. Naturally, routing protocols must strictly define their metric calculation procedure.
- *Simplicity and low overhead*: The routing algorithm must offer its functionality efficiently, with a minimum of software and utilization overhead. Efficiency is particularly important when the software implementing the routing algorithm must run on a computer with limited resources.
- *Robustness and stability*: The routing algorithm should perform correctly in the face of unusual or unforeseen circumstances such as hardware failures, high load conditions, and incorrect implementations. Because routers are located at network junction points, they can cause considerable problems when they fail to be robust.
- *Flexibility and rapid convergence*: Routing algorithms should quickly and accurately adapt to a variety of network circumstances. This is important, since algorithms that converge too slowly can cause routing loops or network outages.

Internet routing protocols can be separated into two different classes: link state algorithms and distance vector algorithms. Link state algorithms (also known as *shortest path first* algorithms) flood routing information to all nodes in the network. However, each router sends only that portion of the routing table that describes the state of its own links. The most prominent link state algorithm used in the Internet is OSPF (*Open Shortest Path First*). Distance vector algorithms (also known as *Bellman-Ford* algorithms) call for each router to send all or some portion of its routing table, but only to its neighbors. The most prominent distance vector algorithm is RIP (*Routing Information Protocol*). In essence, link state algorithms send small updates everywhere, while distance vector algorithms send larger updates only to neighboring routers.

Because they converge more quickly, link state algorithms are somewhat less prone to routing loops than distance vector algorithms. On the other hand, link state algorithms require more CPU power and memory than distance vector algorithms. Link state algorithms can therefore be more expensive to implement and support. Despite their differences, both algorithm types perform in an acceptable way in most circumstances (otherwise, they would not have been chosen for use in the Internet!).

As already indicated above, both link state algorithms and distance vector algorithms use routing tables. Routing tables contain information about the status of links that is used by the switching software to select the best route. But what kind of information should they contain? How do routing algorithms determine that one route is preferable to others?

Routing algorithms for the Internet have used many different metrics to determine the best route. Sophisticated routing algorithms can base route selection on multiple metrics, combining them in a single (hybrid) metric. All of the following metrics have been used: path length, reliability, delay, bandwidth, load, and communication cost. The path length is the most common routing metric. The easiest way to define the path length is simply by the number of routers and switches a packet has to pass. However, some routing protocols also allow to assign an arbitrary cost to each network link (which can be any one of the above parameters, such as delay, available bandwidth, or the price of using a link). In this case, the path length is the sum of the costs associated with each link traversed.

It is not difficult to prove that in the ideal situation (all information is up-to-date), both RIP and OSPF will select the path of smallest available cost, i.e. for each individual request an optimal path will be chosen. However, minimizing individual costs does *not* imply minimizing the global cost.

In order to study the global impact of cost functions, Andrews et al. [5] recently considered the following model.

Again, we assume that we have an adversary that chooses the injection time, source, destination, and route for each packet. A sequence of injections is called (w,λ)-admissible for a window size w and injection rate $\lambda < 1$ if in any time interval I of size at least w the total number of packets injected into the network whose paths pass through any link e is at most $|I| \cdot \lambda$. In this case, the corresponding paths are also called (w,λ)-*admissible*. We say that a set of packet paths is *weakly* (w,λ)-*admissible* if we can partition time into windows of length w such that for each window *in the partition* and each link e, the number of paths that pass through e and correspond to packets injected during the window is at most $w\lambda$.

Suppose now that the adversary does not reveal the paths to the system, but that instead always a best possible route is selected according to some cost function. Is it then possible to ensure that if the injected packets have paths that are (w,λ)-admissible, then the selection process ensures that the selected paths are (W,Λ)-admissible, possibly for a different window size W and a different $\Lambda < 1$?

Based on the ϵ-approximation algorithm by Garg and Könemann [19] for concurrent multicommodity flow problems, which in turn builds upon the work of Plotkin, Shmoys, and Tardos [36], Andrews et al. [5] use the following algorithm:

Suppose that control information is communicated instantly. Whenever a source node chooses a route for a packet, this information is immediately transmitted to all the links on the route. Whenever the congestion on a link changes, this fact is instantaneously transmitted to all the source nodes. (Andrews et al. also show how to relax these assumptions, but for simplicity we will stick to them here.)

Every injected packet is routed along the path whose total congestion is the smallest under the current congestion function $c(\cdot)$, i.e. we route along shortest paths with respect to $c(\cdot)$. Initially, the congestion along every link is set to δ where δ is defined in (2). Each time a route is selected for an injected packet, the congestion $c(e)$ of every link e along the chosen route is updated to $c(e)(1+\mu/w)$ where μ is defined in (1). We reset the congestion of every link to its initial value of δ at the beginning of each *phase*. A phase terminates in t windows of w steps, where t is an integer defined in (3).

The values of μ, δ and t are defined as follows. Let m be the number of links in the network. For any $R \in (r, 1)$ of our choice, let

$$\mu = 1 - \left(\frac{\lambda}{\Lambda}\right)^{1/3} \tag{1}$$

$$\delta = \left(\frac{1-\lambda\mu}{m}\right)^{1/r\mu} \tag{2}$$

$$t = \lfloor \frac{1-\lambda\mu}{\lambda\mu} \ln \frac{1-\lambda\mu}{m\delta} \rfloor + 1 \tag{3}$$

Andrews et al. prove the following theorem.

Theorem 3 ([5]). *For all packets injected during one phase, at most $tw\Lambda$ of their routes chosen by the above procedure go through the same link for some $\Lambda < 1$. In other words, these routes are weakly (tw, Λ)-admissible.*

Furthermore, they show the following lemma that implies that the selected paths are not only weakly (tw, Λ)-admissible, but also (w', λ')-admissible for some $w' \geq w$ and $\lambda' \in [\Lambda, 1)$.

Lemma 1 ([5]). *If a set of paths is weakly (w, λ)-admissible then it is also (w', λ')-admissible for some $w' \geq w$ and $\lambda' \in [\lambda, 1)$.*

Hence, using a universally stable switching protocol on top of their route selection scheme (such as longest-in-system or furthest-to-go [4]) would result in a routing protocol that is universally stable.

As we saw, the protocol by Andrews et al. [5] allows to admit all packets if, as a precondition, all of them are admissible. However, in the Internet one may have the situation that much more packets are injected than are admissible.

Hence, an interesting question is what to do if not all packets are admissible, and therefore some packets may have to be dropped. In this case, one would like to drop the packets in a way so as to optimize a certain cost function. The simplest cost function would certainly be to maximize the number of admissible packets. Here, the following strategy would work (if the λ_e below is known in advance): If one would use the same scheme as above and then would assign to each packet P a survival probability that is equal to the minimum $\min[1, tw\Lambda/\lambda_e]$ over all edges e along its path (where λ_e is the number of paths crossing e in the phase in which P is injected), then the result should be that each surviving packet is admissible with high probability and the number of surviving packets is close to the maximum number of packets any scheme could deliver. The question, however, is whether a knowledge of λ_e is necessary for reaching this goal. Certain results seem to indicate that this is the case [6]. Other cost functions could be to maximize the profit when selecting the admissible packets. Certainly, more work has to be done to come up with a protocol that would potentially be able to extend (or replace) existing strategies in the Internet.

4 Communication in Wireless Networks

Communication in wireless networks is currently a rapidly evolving, highly active research field, though dominated by experimental studies. A particularly hot field has been in recent years mobile ad-hoc networks, which are wireless networks that do not rely on any fixed infrastructure. Applications of ad hoc networks have mostly been limited to the military communication environment. However, recently also the commercial sector is increasingly interested in this technology. Evidence of this is the establishment of a MANET working group in the Internet Engineering Task Force (IETF). Central questions in the area of wireless ad hoc networks are:

- How to ensure connectivity?
- How to select and maintain routes between the mobile units?
- How to send packets along the selected routes?

The first project in which a collection of computers formed a wireless network was the ALOHA project, which was conducted around 1970 at the University of Hawaii. In this project all communication was performed via a single master. Hence, it was not necessary to search for suitable routes but only to know the addresses of the computers in the network. Of course, in order to ensure that the network is connected, all computers had to be within the range of the master. If a larger distance has to be bridged with a limited transmission range, then several intermediate transmissions (often called *hops*) are necessary to send a packet from a source to a destination. In this case, finding a suitable route becomes a serious problem. One of the first that suggested strategies that solve the route selection problem in this case were Gitman et al. [20].

Today, route selection protocols for wireless networks are usually divided into proactive and reactive protocols. In a *proactive* protocol every node tries to keep routing information for all nodes in the network up to date. This is usually achieved by updating the information automatically in regular time intervals. If a message has to be sent to a certain destination, then the stored information allows to immediately determine a suitable route to the destination (if such a route exists). The latency for the transmission of a message can therefore be kept small. The disadvantage of this method is that the maintenance of the routing information requires a lot of extra communication and that space even has to be allocated for those nodes that are rarely used as a destination. All traditional link-state and distance-vector routing protocols are proactive. Examples of proactive protocols that have been suggested for wireless networks are OLSR, DSDV, and WRP.

Also reactive protocols store routing information. However, this information is only updated when needed. This has the effect that parts of the information may be outdated. Thus, it might happen that in order to send a message to a destination, first a suitable route has to found in the network. This may cause a high latency for the transmission of the message. However, the advantage of this route-on-demand strategy is that it usually causes a much lower communication overhead compared to proactive protocols, which allows to use reactive protocols in much larger wireless networks. Examples of reactive protocols are DSR, AODV, and TORA.

Also algorithms are known that are a combination of both concepts. The usual approach is to keep the routing information for a certain area around a node up to date and to collect information about distant nodes only in the case that a message has to be sent there. Examples are ZRP and LANMAR.

Detailed descriptions of many of the protocols mentioned above can be found at

`http://www.ietf.org/html.charters/manet-charter.html`.

All of the protocols above have only been studied via simulations or experiments. Theoretical work in the area of wireless networks has mostly concentrated on the packet radio network (PRN) model. In this model, a wireless network is usually seen as an undirected graph $G = (V, E)$, where two nodes i and j are connected if and only if i is within the transmission radius of j and vice versa. One time step is defined as the time it takes to transmit a packet from one node to another. Since it is usually assumed that all nodes can only transmit at one frequency, a *transmission conflict* occurs at node i if two of its neighboring nodes transmit a packet at the same time. Communication problems have been studied by many authors for both the case that a conflict can be detected and the case that it cannot be detected.

The advantage of the PRN model is that it is reasonably simple to allow theoretical investigations. However, its drawback is that interference problems are modelled in a highly simplified way. More realistic wireless network models have been considered, for example, in [1, 21, 22, 39]. Another drawback of the PRN model is that in its original form it only allows to study wireless communication with a fixed transmission range. Both in cellular and ad-hoc wireless networks, being able to use a variable transmission power has a noticeable advantage over a fixed transmission power. This has been demonstrated experimentally both for cellular networks [25] and ad-hoc networks [35]. One primary motivation for using a variable transmission power is that energy is a scarce resource in mobile units, and therefore the possibility of being able to adjust the signal strength to the given situation can significantly help to save energy. Another important aspect is that contention among the mobile units can be significantly reduced. Furthermore, it allows the units to have more time to adjust to a changing constellation and therefore to have a much smoother transition from one interconnection scenario to another.

In order to model wireless networks of units with variable transmission power, one can use a complete graph with a cost function $c : V \times V \to \mathbb{R}_+$ that determines the signal strength (resp. energy consumption) necessary to send a packet from one node to another. Changing the distance between these two nodes or moving an obstacle between these will then change the cost function accordingly. Furthermore, in order to model interference more accurately, one can use a parameter $\alpha > 1$ with the property that if a node v transmits a message with signal strength s, then all nodes requiring a signal strength of at most $\alpha \cdot s$ to receive a message from v are blocked. Such a model has, for example, been used in [1].

4.1 Connectivity

Allowing a variable transmission power in the wireless model above adds much more flexibility but also creates new problems. For example, the connectivity problem is trivial in the fixed transmission power case (nodes either can or cannot reach each other), whereas it is highly non-trivial to select the right transmission power to ensure connectivity for the case of variable transmission powers (unless the nodes always communicate with maximum power, which would be too wasteful).

Recently, an elegant, rigorous approach was found to solve this problem under certain circumstances. Consider some fixed dimension D and some norm $|\cdot|$ on $\mathbb{R}^D$. Consider a set of points $V = \{v_1, \ldots, v_n\}$ in $\mathbb{R}^D$ and let $G = (V, E)$ be an undirected graph connecting these points. For every path $p = (v_{i_1}, v_{i_2}, \ldots, v_{i_\ell})$ in G we define its *stretch factor* as

$$f(p) = \frac{\sum_{j=1}^{\ell-1} |v_{i_j} - v_{i_{j+1}}|}{|v_{i_1} - v_{i_\ell}|} .$$

G is called an *f-spanner* for V if every pair $s, t \in V$ has a path in G of stretch factor at most f. Yao [41] and subsequent papers (e.g., [38, 16]) showed that spanners for a given set of points V can be obtained by a generalization of *proximity graphs* [26]: partition $\mathbb{R}^D$ into a collection $\mathcal{C}$ of $k \in \mathbb{N}$ convex cones $C_0, \ldots, C_{k-1}$. Then, from every point $p \in V$ and every $j \in \{1, \ldots, k\}$, draw directed edges to the closest point of V lying in the translated cone $p + C_j$. The resulting graph is called a *partitioned neighborhood graph* (PNG). For the special case that all cones have an angular diameter of at most Θ, the graph is also called a Θ-graph [38].

Suppose that $|\cdot|$ represents the Euclidean distance. In this case, Ruppert and Seidel [38] showed that if every $C \in \mathcal{C}$ has an angular diameter of at most $\Theta < \pi/3$, then the resulting PNG is an f-spanner with

$$f = \frac{1}{1 - 2\sin(\Theta/2)} .$$

This implies that in the 2-dimensional case 7 cones suffice for the PNG to be a 7.57-spanner. Combining the bound for f above with a result by Hardin, Sloane and Smith [23], one can also show that in the 3-dimensional case 20 cones suffice for the PNG to be an 88.1-spanner. Further improvements, partly using different norms and so-called weak spanner properties, can be found in [16].

Note that a spanner is always strongly connected. Hence, we can always ensure connectivity among nodes with variable transmission ranges by simply choosing the transmission range in a way so that it can transmit messages to all nearest neighbors in all cones (if possible), where the cones are selected so that the resulting PNG is guaranteed to be a spanner. This approach even works if there is a limit on the maximum possible transmission range, unless that limit does not suffice to achieve connectivity by any means.

A limitation of the PNG approach seems to be that it does not seem to work well for higher dimensions (too many cones are needed) and that it may only work in (certain) metric spaces. Furthermore, there are problems realizing the PNG approach in practice, since it might be necessary to use devices such as GPS in order to know the location of a node and therefore being able to decide in which cone it is. It is an interesting open problem whether there are alternative approaches. For example, in situations in which units randomly move around, it might suffice just to connect to some constant number of nearest neighbors to ensure connectivity with high probability.

4.2 Path updates in wireless networks

Suppose that we want to use a proactive scheme for refreshing paths between different nodes in a wireless network. The usual approach is that for every pair of nodes its route is updated in regular time intervals. But how large should these intervals be? And is it really necessary to have a route for *every* pair of nodes to ensure a good connectivity?

In order to address these questions, suppose we keep information in a redundant form in the network so that as long as a constant fraction of the nodes is connected, no information is lost. Then we are in the situation already studied in Section 2.1.

So a reasonable strategy could be that we simply embed a graph G that is as robust as possible in the wireless network H. Communication would only be performed along the paths simulating edges in G. In this case, H is connected if and only if G is still connected with regard to those edges that still have working paths in H. Each time a node walks away from its position, a certain number of paths in H representing edges in G may get disconnected. If this exceeds some threshold (for example, a constant fraction of the nodes in G, and therefore a constant fraction of the nodes in H, is not connected any more), then one has to refresh the embedding.

If movements of nodes happen at random and we had the situation that each node is only crossed by one path representing an edge in G, then we would be back to the model in Section 2.1. So the question is, how to transfer the results obtained there to wireless networks if this is not the case? In particular, how should the paths be selected in the wireless network so that one can prove lower bounds on the time intervals needed for refreshing routes in order to ensure connectivity of a constant fraction of the nodes (with high probability)?

4.3 Routing in wireless networks

There are two important issues when routing packets in wireless networks: the energy necessary to perform the routing should be kept as low as possible while keeping the throughput as high as possible (i.e. allowing many packets to reach their destination).

The issue of achieving a high throughput has already been addressed in [21, 22]. However, to the best of our knowledge, no results are known so far about achieving a high throughput with a minimum amount of energy. Consider, for example, the following model.

Recall the adversarial routing model considered for faulty networks in Section 2.4: suppose that the adversary has to guarantee that for every injected packet there is a schedule (i.e. a sequence of hops over the time) for delivering it to its destination. Would it then be possible to design an online routing protocol that can achieve (nearly) the same amount of deliveries with an energy consumption that is close to the optimum? Concerning the optimum, two alternatives would be interesting: we want the total amount of consumed energy to be as low as possible, or we want the maximum amount of energy spent in a node to be as low as possible.

5 Communication in Adversarial Networks

All previous scenarios have in common that we have dynamic networks with specific characteristics. In this section, we choose a high-level approach that is not tied to any particular application. Suppose that we have a dynamic network that can form *any* directed network at *any* point of time, and the way it changes is under adversarial control. In order to ensure a high throughput efficiency such kind of dynamic networks, several challenging tasks have to be solved:

- *Routing*: What is the next edge to be traversed by a packet?
- *Switching*: What is the next packet to be transmitted on an edge?
  In particular, which destination should be preferred?

– *Admission control*: What is the packet to be dropped?

This seems to be impossible to decide without knowing the future, especially if both the topology and the packet injections are under adversarial control: choosing the wrong edge may lead a packet further away from its destination, and preferring packets with a destination that will be isolated in the future or dropping the wrong packets in case of overfull buffers may tremendously decrease the number of successfully delivered packets. However, we will show that the seemingly impossible task is possible, and this not only for unicasting but even for multicasting.

First, we specify our model. Each edge can transport one packet and can be used independently from the other edges. Each node may have a certain number of buffers to store different types of packets (for example, packets with different destinations). To ensure that our model is realistic, we assume that the size of every buffer (i.e. the number of packets it can hold) is finite. Furthermore, each node can only have a finite number of incoming and outgoing edges at a time. As a parameter for the former case we will use the *maximum buffer height* H, and as a parameter for the latter case we will use the *maximum in/out degree* Δ.

The adversary does not only control the topology of the network but also the injection of packets. We distinguish between two adversarial injection models: one for unicast injections and one for multicast injections.

The adversarial unicast model. Each unicast packet has a fixed destination, determined at the time of its injection. The adversary can inject an arbitrary number of packets and can activate an arbitrary number of edges in each time step as long as the number of incoming or outgoing edges at a node does not exceed Δ. In this case, only some of the injected packets may be able to reach their destination, even when using a best possible strategy. For each of the successful packets a schedule of the form $S = ((e_0, t_0), (e_1, t_1), \ldots, (e_\ell, t_\ell))$ can be specified. Of course, schedules are not allowed to intersect. When speaking about schedules in the following, we always mean a delivery strategy chosen by a best possible routing algorithm.

The adversarial multicast model. In the multicast model, a packet is allowed to have several destinations. In order to allow a higher efficiency than simply sending out one packet for each destination, we adopt the standard multicast model, which assumes that a multicast packet only takes one time step to cross an edge and that a multicast packet can split itself while in transit. Requiring an efficient use of this property make the multicasting problem considerably harder than the unicast problem. Especially when using dynamic networks, it seems to be formidable problem to decide when and where to split a packet.

As in the unicast model, the adversary is basically allowed to inject an unlimited number of multicast packets. However, only some of these packets may be able to reach their destinations, even when using a best possible strategy.

For each multicast packet P that can reach k of its destinations, a schedule can be identified. Any such schedule can be reduced to a directed tree with k leaves representing the destinations. Hence, we can view all schedules to be of the form $S = ((e_0, t_0), (e_1, t_1), \ldots, (e_\ell, t_\ell))$, where

- P is injected at the endpoint of e_0 at time step t_0,
- the edges $e_1, \ldots, e_\ell$ form an directed tree with the source as the root and the k destinations as leaves,
- for every directed path $((e_{i_1}, t_{i_1}), \ldots, (e_{i_k}, t_{i_k}))$, the time steps have the ordering $t_{i_1} < t_{i_2} \ldots < t_{i_k}$, and
- edge e_i is active at time t_i for all $1 \leq i \leq \ell$.

As in the unicast model, the schedules must fulfill the property that they do not overlap.

Analytical approach. Each time a multicast (or unicast) packet reaches one of its destinations, we count it as one *delivery*. The number of deliveries that is achieved by an algorithm is called its *throughput*. Since the adversary is allowed to inject an unbounded number of packets, we will allow routing algorithms to drop packets so that a high throughput can be achieved with a buffer size that is as small as possible.

In order to compare the performance of a best possible strategy with our online strategies, one can use competitive analysis. Given any sequence of edge activations and packet injections σ, let $\mathrm{OPT}_B(\sigma)$ be the maximum possible throughput (i.e. the maximum number of deliveries) when using a buffer size of B, and let $A_{B'}(\sigma)$ be the throughput achieved by some given online algorithm A with buffer size B'. "Maximum possible" means here maximum possible given the same type of resources as used by the online algorithm. Otherwise, we would not have a fair comparison. The only difference between the resources of the online algorithm and a best possible algorithm we allow is the buffer size. (However, we place *no* restrictions on how the packets are selected and moved by an optimal strategy.) We call A (c, s)-*competitive* if for all σ and all B,

$$A_{s \cdot B}(\sigma) \geq c \cdot \mathrm{OPT}_B(\sigma) - r$$

for some value $r \geq 0$ that is independent of $\mathrm{OPT}_B(\sigma)$. Note that $c \in [0, 1]$. Usually, the competitive ratio is defined so that $c \geq 1$. However, in our case we choose the inverse definition, since it is more intuitive: c represents the fraction of the optimal throughput that can be achieved by A.

Using these models, we recently proposed different variants of the T-balancing algorithm presented in Section 2.4 for unicasting and multicasting [8]. In the unicast situation, we assume that every node has a buffer for every destination. Let $q_{v,d}^t$ denote the size of the buffer for destination d in node v at step t. In every time step t the unicast T-balancing algorithm performs the following operations in every node v:

1. For every working edge $e = (v, w)$, determine the destination d with maximum $q^t_{v,d} - q^t_{w,d}$ and check whether $q^t_{v,d} - q^t_{w,d} > T$. If so, send a packet with destination d from v to w (otherwise do nothing).
2. Receive incoming packets and absorb all packets that reached the destination.
3. Receive all newly injected packets. If a packet cannot be stored in a node because its height is already H, then delete the new packet.

A weighted form of the protocol can also be formulated for multicasting. We obtained the following main results [8]:

– *Unicasting*: For every $T \geq B + 2(\Delta - 1)$, the unicast variant of the T-balancing algorithm is $(1-\epsilon, 1+(1+(T+\Delta)/B)L/\epsilon)$-competitive, where L is the average path length used by successful packets in an optimal solution. The result is sharp up to a constant factor in the space overhead.
– *Multicasting*: For every $T \geq B + D(3\Delta - 1)$, the multicast variant of the T-balancing algorithm is $(1-\epsilon, 1+(1+(T+D\Delta)/B)D \cdot L/\epsilon)$-competitive, where L is defined as above and D is the maximum number of destinations a packet can have.

Many open questions remain. Although the space overhead is already reasonably low (essentially, $O(L/\epsilon)$ for unicasting), the question is whether it can still be reduced. For example, could knowledge about the location of a destination or structural properties of the network (for instance, it has to form a planar graph) help to get better bounds? Or are there approaches completely different from the balancing approach used above that can achieve a lower space overhead? In the worst case, the multicasting result stated above may need an exponentially large number of buffers per node, since for every possible set of destinations a buffer has to be reserved in each node. Can this be reduced to a polynomial number without restricting a best possible strategy? We suppose that this is not possible in general. But under certain circumstances (such as certain restrictions on the network topology and the movement of packets), this should be possible.

6 Conclusion

We presented existing and suggested new models and techniques for the study of dynamic networks. Although in some of the areas covered in this paper already a significant progress has been made, many problems are still wide open, and a long way still has to be gone to ensure that the methods developed for dynamic networks are not only (close to) optimal in theory but also work well in practice.

7 Acknowledgements

We would like to thank Matthew Andrews for giving valuable comments on an early version of the paper.

References

1. M. Adler and C. Scheideler. Efficient communication strategies for ad hoc wireless networks. *Theory Comput. Systems*, 33:337–391, 2000.
2. W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptive packet routing for bursty adversarial traffic. In *Proc. of the 30th ACM Symp. on Theory of Computing (STOC)*, pages 359–368, 1998.
3. M. Ajtai, J. Komlós, and E. Szemerédi. Largest random component of a k-cube. *Combinatorica*, 2(1):1–7, 1982.
4. M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. of the 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 380–389, 1996.
5. M. Andrews, A. Fernández, A. Goel, and L. Zhang. Source routing and scheduling in packet networks. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 168–177, 2001.
6. B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proc. of the 34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.
7. B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 158–167, 2001.
8. B. Awerbuch, A. Brinkmann, and C. Scheideler. Unicasting and multicasting in adversarial systems. Unpublished manuscript, 2001.
9. A. Bagchi, A. Chaudhary, P. Kolman, and C. Scheideler. Algorithms for the maximum disjoint k-paths problem. Unpublished manuscript, November 2001.
10. A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25, 2000.
11. B. Bollobas. The evolution of random graphs. *Transactions of the AMS*, 286:257–274, 1984.
12. A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proc. of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 376–385, 1996.
13. R. Cole, B. Maggs, and R. Sitaraman. Routing on butterfly networks with random faults. In *Proc. of the 36th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 558–570, 1995.
14. R. Cole, B. Maggs, and R. Sitaraman. Reconfiguring arrays with faults part I: worst-case faults. *SIAM Journal on Computing*, 26(6):1581–1611, 1997.
15. P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5:17–61, 1960.
16. M. Fischer, T. Lukovszki, and M. Ziegler. Partitioned neighborhood spanners of minimal outdegree. In *Proc. of the 11th Canadian Conference on Computational Geometry*, 1999.
17. L. Fleischer, A. Meyerson, I. Saniee, and A. Srinivasan. Fast and efficient bandwidth reservation for robust routing. In *DIMACS Mini-Workshop on Quality of Service Issues in the Internet*, February 2001.
18. D. Gamarnik. Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 206–214, 1999.

19. N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. of the 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.
20. I. Gitman, R. Van Slyke, and H. Frank. Routing in packet-switching broadcast radio networks. *IEEE Transactions on Communication*, 24:926–930, 1976.
21. P. Gupta and P. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, 2000.
22. P. Gupta and P. Kumar. Internets in the sky: The capacity of three dimensional wireless networks. *Communications in Information and Systems*, 1(1):33–50, 2001.
23. R. Hardin, N. Sloane, and W. Smith. *A library of putatively optimal coverings of the sphere with n equal caps.* `http://www.research.att.com/~njas/coverings/`.
24. J. Håstad, T. Leighton, and M. Newman. Reconfiguring a hypercubein the presence of faults. In *Proc. of the 19th ACM Symp. on Theory of Computing (STOC)*, pages 274–284, 1987.
25. J. Jacobsmeyer. Congestion relief on power-controlled CDMA networks. *IEEE Journal on Selected Areas in Communications*, 14(9):1758–1761, 1996.
26. J. Jaromczyk and G. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80:1502–1517, 1992.
27. A. Karlin, G. Nelson, and H. Tamaki. On the fault tolerance of the butterfly. In *Proc. of the 26th ACM Symp. on Theory of Computing (STOC)*, pages 125–133, 1994.
28. H. Kesten. The critical probability of bond percolation on the square lattice equals 1/2. *Communication in Mathematical Physics*, 74:41–59, 1980.
29. J. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.
30. P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proc. of the 13th ACM Symp. on Discrete Algorithms (SODA)*, 2002.
31. F. Leighton and B. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, 41(5):578–587, 1992.
32. F. Leighton, B. Maggs, and S. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.
33. F. Leighton, B. Maggs, and R. Sitaraman. On the fault tolerance of some popular bounded-degree networks. *SIAM Journal on Computing*, 27(5):1303–1333, 1998.
34. T. Luczak, B. Pittel, and J. Wierman. The structure of a random graph at the point of the phase transition. *Transactions of the AMS*, 341:721–748, 1994.
35. J. Monks. *Transmission Power Control for Enhancing The Performance of Wireless Packet Data Networks.* PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, March 2001.
36. S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proc. of the 32nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 495–504, 1991.
37. M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.
38. J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proc. of the 3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991.
39. S. Ulukus and R. Yates. Stochastic power control for cellular radio systems. *IEEE Transactions on Communication*, 46:784–798, 1998.

40. E. Upfal. Tolerating a linear number of faults in networks of bounded degree. *Information and Computation*, 115(2):312–320, 1994.
41. A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.