An elementary proposition on the dynamic routing problem in wireless networks of sensors

Alexandros K. Koliousis

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computing Science
University of Glasgow
United Kingdom

To my family.

Abstract

The routing problem (finding an optimal route from one point in a computer network to another) is surrounded by impossibility results. These results are usually expressed as lower and upper bounds on the set of nodes (or the set of links) of a network and represent the complexity of a solution to the routing problem (a routing function).

The routing problem dealt with here, in particular, is a dynamic one (it accounts for network dynamics) and concerns wireless networks of sensors. Sensors form wireless links of limited capacity and time-variable quality to route messages amongst themselves. It is desired that sensors self-organize *ad hoc* in order to successfully carry out a routing task, e.g. provide daily soil erosion reports for a monitored watershed, or provide immediate indications of an imminent volcanic eruption, in spite of network dynamics.

Link dynamics are the first barrier to finding an optimal route between a node x and a node y in a sensor network. The uncertainty of the outcome (the best next hop) of a routing function lies partially with the quality fluctuations of wireless links. Take, for example, a static network. It is known that, given the set of nodes and their link weights (or costs), a node can compute optimal routes by running, say, Dijkstra's algorithm. Link dynamics however suggest that costs are not static. Hence, sensors need a metric (a measurable quantity of uncertainty) to monitor for fluctuations, either improvements or degradations of quality or load; when a fluctuation is sufficiently large (say, by Δ), sensors ought to update their costs and seek another route. Therein lies the other fundamental barrier to find an optimal route – complexity.

A crude argument would suggest that sensors (and their links) have an upper bound on the number of messages they can transmit, receive and store due to resource constraints. Such messages can be application traffic, in which case it is desirable, or control traffic, in which case it should be kept minimal. The first type of traffic is demand, and a user should provision for it accordingly. The second type of traffic is overhead, and it is necessary if a routing system (or scheme) is to ensure its fidelity to the application requirements (policy). It is possible for a routing scheme to approximate optimal routes (by Δ) by reducing its message and/or memory complexity.

The common denominator of the routing problem and the desire to minimize overhead while approximating optimal routes is Δ, the deviation (or stretch) of a computed route from an optimal one, as computed by a node that has instantaneous knowledge of the set of all nodes and their interaction costs (an oracle). This dissertation deals with both problems in unison. To do so, it needs to translate the policy space (the user objectives) into a metric space (routing objectives). It does so by means of a cost function that normalizes metrics into a number of hops. Then it proceeds to devise, design, and implement a scheme that computes minimum-hop-count routes with manageable complexity.

The theory presented is founded on (well-ordered) sets with respect to an elementary proposition, that a route from a source x to a destination y can be computed either by y sending an advertisement to the set of all nodes, or by x sending a query to the set of all nodes; henceforth the *proactive* method (of y) and the *reactive* method (of x), respectively.

The debate between proactive and reactive routing protocols appears in many instances of the routing problem (e.g. routing in mobile networks, routing in delay-tolerant networks, compact routing), and it is focussed on whether nodes should know *a priori* all routes and then select the best one (with the proactive method), or each node could simply search for a (hopefully best) route on demand (with the reactive method).

The proactive method is stateful, as it requires the entire metric space – the set of nodes and their interaction costs – in memory (in a routing table). The routes computed by the proactive method are optimal and the lower and upper bounds of proactive schemes match those of an oracle. Any attempt to reduce the proactive overhead, e.g. by introducing hierarchies, will result in sub-optimal routes (of known stretch). The reactive method is stateless, as it requires no information whatsoever to compute a route. Reactive schemes – at least as they are presently understood – compute sub-optimal routes (and thus far, of unknown stretch).

This dissertation attempts to answer the following question: "what is the least amount of state required to compute an optimal route from a source to a destination?" A hybrid routing scheme is used to investigate this question, one that uses the proactive method to compute routes to near destinations and the reactive method for distant destinations.

It is shown that there are cases where hybrid schemes can converge to optimal routes, despite possessing incomplete routing state, and that the necessary and sufficient condition to compute optimal routes with local state alone is related neither to the size nor the density of a network; it is rather the circumference (the size of the largest cycle) of a network that matters. Counterexamples, where local state is insufficient, are discussed to derive the worst-case stretch.

The theory is augmented with simulation results and a small experimental testbed to motivate the discussion on how policy space (user requirements) can translate into metric spaces and how different metrics affect performance. On the debate between proactive and reactive protocols, it is shown that the two classes are equivalent. The dissertation concludes with a discussion on the applicability of its results and poses some open problems.

Acknowledgments

I am forever grateful to my supervisor, Professor Joe Sventek, for all the trust he shown me over the years. His wisdom and guidance have been a catalyst during my studies.

I would like to thank my second supervisor, Alice Miller, for her valued comments during this work. I would also like to thank all members of the DIAS-MC project for useful discussions on wireless sensor networks.

Last, but not least, I would like to thank my friends and colleagues for their continuous encouragement. Special thanks to Ross, Oliver, Stephen, Martin, Paul, and other members of the ENDS group.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sensor networks represent a natural evolution of distributed systems. Small, battery-powered, wireless devices have inspired a diverse set of monitoring applications. Environmental and habitat monitoring is one exemplar application domain: deployed in remote (usually inaccessible) areas, unattended sensors search, record, and report *in situ* rare physical phenomena which have a distinct event signature. As for any distributed system, message delivery is of the essence.

Sensors are not mere peripheral devices. On the contrary, there is a high degree of interaction amongst them, both positive and negative. Sensors interact positively as they form a wireless *ad hoc* network; routes from any source to one or more destination nodes are constructed in this network along which messages are conveyed. On the other hand, sensors interact negatively as the links from which routes are constructed exhibit variable quality. Nevertheless, sensors rarely interact with users, as they react to network or environmental stimuli autonomously.

Since sensors self-organize into linked structures (graphs) to route messages and links evolve over time, the routes should evolve as well. In a sense, any network change causes the autonomous routing system to diverge from its stable state; when the divergence is sufficiently large, the system should seek another (optimal or near-optimal) route, say, from sensor x to sensor y.

Given a graph (a representation of nodes and their communication links), the **routing problem** is to devise for each node x a function $\mathcal{F}_x(y)$ such that, whenever a message arrives at node x, the function returns the *next hop* towards which the message must be forwarded in order to reach node y. If the graph is fixed in advance, one could devise such a function off-line (prior to deployment); this is the *static* version of the routing problem. In the *dynamic* version, one should account for possible transformations of the network. The set of all routing functions, together with the data structures they require to compute the next hop to all destinations, constitute a *routing scheme* (also referred to as a routing system).

In brief, the dynamic routing problem demands an adaptive, distributed mechanism that can converge to optimal (or near-optimal) routes between arbitrary x-y pairs of nodes in light of network dynamics.

This dissertation considers dynamic, wireless networks of resource-constrained devices, such as sensors (Figure 1), with a fixed topology – nodes are static or move infrequently – but of time-varying link quality. By fixing the topology, the dynamics of networks are restricted to link connectivity. Although this precludes node mobility from the discussion (until § 8.5), the routing problem is still not trivial.

It is neither limits on energy nor limits on bandwidth that make the routing problem so difficult to solve. These limitations can be expressed as constraints (e.g. "sensors are power-constrained"), or objectives (e.g. "maximize time to first node death"), or more general expressions (e.g. "use resources efficiently") of *policy*. From another point of view, the solution to the routing problem lies with a scheme that computes optimal routes subject to an additional constraint, the size n of a network. This identifies scalability as a major issue for self-organized networks.

The reference to optimal, rather than "shortest-path", routes is because the routing problem is an optimization problem [4]. As such, a routing scheme aims to find optimal routes, those that incur minimum cost amongst the finite set of all feasible solutions. For example, minimum-hop-count (or shortest) paths are optimal when all links are equiprobable and nodes are unconstrained.

An optimal routing scheme incurs $O(n^2)$ messages start-up cost, $O(n)$

Figure 1.1: An exemplar sensor device. Such devices have limited bandwidth, battery, memory, and computational resources.

messages maintenance cost, and $O(n \log n)$ *bits memory cost.* These results can be derived as follows.[1] [2]

The message and memory complexity of the optimal routing scheme are best illustrated with an example. Consider an n-node network, with every pair of nodes connected by a link (a *mesh* network). Let the sentence "maximize lifetime" be an objective and $\mathbb{M} = [c_{ij}]_{n \times n}$ be a matrix (not necessarily symmetric) that represents the *energy cost* to traverse link $\langle i, j \rangle$. This matrix is referred to as a *metric space.* Then, nodes i and j must advertise their link to the entire network. There are $n(n-1)$ possible links (or $\frac{n(n-1)}{2}$, if links are symmetric), hence the upper bound of n^2 messages.

The moment when a node receives, and possibly stores, a complete and consistent view of the matrix $\mathbb{M}$ (the metric space), it is able to route messages to any other node over least-energy-cost paths: it can compute locally a routing table of size $n-1$ that lists the *best next hop* towards the $n-1$ possible destinations.

Now consider a change in the cost of a link $\langle i, j \rangle$. Then, node i or j must advertise it to the entire network; this requires at most n messages to

[1] Assumes that the size of a routing entry is on the order of the logarithm of the number n of nodes and that messages carry a single entry. See [63, 73] for instances of such an optimal scheme.

[2] The $O(\cdot)$ and $\Omega(\cdot)$ notations provide upper and lower complexity bounds for a routing algorithm, respectively; bounds that hold on every graph and every execution of the algorithm. They read as follows. A function $g(n) = O(f(n))$ has "order at most $f(n)$"; and $g(n) = \Omega(f(n))$ has "order at least $f(n)$" as $n \to \infty$ [48].

propagate the change and recompute the minimum-cost routes.[1]

Such overheads are prohibitive for dynamic, constrained networks if scalable solutions to the routing problem are to be obtained. It is also the case that if designers choose to compress routing tables, they have *de facto* accepted sub-optimal routes, before the network is even switched on. Furthermore, any additional change will cause the system to switch off, re-compute routes, and switch on again *ad infinitum.* Take, for example, hierarchical routing [84, 47].

In hierarchical routing, nodes decide (or users impose) on an r-subset of nodes to serve as proxies for the rest of the destinations. This way, however, they construct a hierarchical structure in which a message from x to y may first ascend before reaching (an otherwise closer) node, say y; thus the fundamental trade-off between memory cost, the number of proxies (r), and *stretch* (Δ), the maximum deviation of a computed route from an optimal one [70]. For $r = n - 1$, the computed routes are optimal. For $r < n - 1$, it is known that the stretch is bounded by 3 times the cost of an optimal route [83].

To return to the previous example, suppose that the cost of a link $\langle i, j \rangle$ changes by Δ. As before, node i or j must advertise it to the entire network, an action that requires $O(n)$ messages, and then each node must recompute the minimum-cost routes. To do so, however, nodes should decide (or users should impose) on a new r-subset of proxies to ensure, once again, an upper bound on the uncertainty of a routing scheme.

Such *proactive* solutions, where a number of routes are pre-determined, are also prohibitive for dynamic, constrained networks if optimal solutions to the routing problem are to be found. Hence, the question: "what is the minimum number r of routes that have to be pre-determined at each source so as to compute optimal routes to any destination?" The alternative to proactive routing is a *reactive* solution, one that determines routes as needed.

This dissertation discusses optimal, yet compact, routing schemes as re-

[1]One should note that such changes in the network are not just attributed to link quality, but also to network flow. For example, costs for energy or costs for latency are also functions of load and may change often.

gards the following elementary proposition.

Proposition *Let x and y be any two nodes in a connected network, and assume that x wishes to communicate with y. Then, an optimal route from x to y is computable at node x from either a proactive scheme $\mathbb{P}$ of y or a reactive scheme $\mathbb{R}$ of x.*

This proposition is further elaborated below.

1.1 Two elementary principles

The two elementary principles refer to the *proactive* and *reactive* method of route computation.

For every node x, there is a routing function $\mathcal{F}_x(y)$ that computes the next hop to a destination y. The set of all routing functions, together with their data structures, constitute a routing scheme $\mathbb{F}$. Reactive schemes are defined first, as the definition for proactive follows naturally. A routing scheme is said to be **reactive** if all unknown routes are found by some on-line, query mechanism. In a **proactive** routing scheme there are no *a priori* unknown routes.

In line with the above, $\mathbb{P}$ and $\mathbb{R}$ denote the two general classes of routing schemes that compute a route from a source x to a destination y as follows. A proactive routing scheme $\mathbb{P}$ advertises y's intent to participate in the routing task to node x; accordingly, a reactive routing scheme $\mathbb{R}$ queries y's intent to participate in the routing task.

Thus, the **proactive method** $\mathcal{P}_x(y)$ returns a node w from which x has received an advertisement on behalf of y; and the **reactive method** $\mathcal{R}_x(y)$ returns a node w from which x has received a reply to its query for destination y.

A **hybrid** routing scheme computes routes by either the proactive or reactive method (an inclusive disjunction), hence the aforementioned elementary proposition "$\mathbb{P}$ or $\mathbb{R}$".

The exact semantics of the two methods (and, subsequently, their schemes) will be determined in Chapter 3. This section describes a rather more general way to associate the routing problem with the two methods, by introducing well-order to the routing system.[1]

Well-order is used to define the r-first neighbors of a node. This dissertation explores the upper and lower bounds of the routing problem using a hybrid routing scheme $\mathbb{F}(r)$, that computes routes to the r-first neighbors using the proactive method, and uses the reactive method for the rest of the nodes. The first order to introduce is an order on the set of nodes.

Let V be the set of all nodes, numbered in some arbitrary fashion from 1 to n, and let

$$c(x, y) = \text{ the minimum cost to route a message from node } x \text{ to node } y$$

for all $x, y \in V$, with $c(x, x) = 0$, for all $x \in V$. The cost function $c(\cdot, \cdot)$ is a *metric* and, together with the set of all nodes V, it defines a *metric space* (V, c) (a matrix $\mathbb{M}$). By definition of the function c, and to distinguish later between different metrics, refer to (V, c) as an *optimal space*. For example, in geographic routing [41], an optimal space is defined by the Euclidean metric $\|x - y\|$.

With no loss of generality, select a node $x \in V$ as a reference point and order the remaining set of nodes as

$$y_1, y_2, y_3, \ldots, y_r, y_{r+1}, \ldots, y_{n-1}$$

by increasing cost from $x = y_0$, breaking ties by lexicographical order (names are the numbers 1 to n). For concreteness, the binary relation "$\prec_x$" that orders the set of nodes $Y = \{y_r\}$, for all $0 \le r < n - 1$, is

$$\begin{aligned} &c(x, y_r) < c(x, y_{r+1}), \text{ or} \\ &c(x, y_r) = c(x, y_{r+1}) \text{ and } y_r < y_{r+1}. \end{aligned}$$

[1]A total order (or order) of a set X is a transitive relation $\prec$ such that for every $x, y \in X$ exactly one of $x \prec y$ or $x = y$ or $y \prec x$ is true. The set X is also a well-ordered set if, in addition to total order, every non-empty subset of X contains a least element. As an example, the set of natural numbers is a well-ordered set (1,2,3, and so on).

The relation $\prec_x$ orders the set of all nodes V, and $Y = (V, \prec_x)$ is a well-ordered set. This first well-ordering of nodes is very useful for two reasons. Firstly, it defines the r-first neighbors of node x to be the set

$$I_x(r) = \{y | y \prec_x y_r\}.$$

Thus, a hybrid method $\mathcal{F}_x(y)$ at node $x \in V$ is

$$\mathcal{F}_x(y) = \begin{cases} \mathcal{P}_x(y) & \text{if } y \in I_x(r), \text{ or} \\ \mathcal{R}_x(y) & \text{otherwise.} \end{cases}$$

Secondly, it sets a limit (the value of r) on the amount of memory a node requires to compute an optimal (or near-optimal) route: "how many routes should be pre-determined before a routing system $\mathbb{F}(r)$ can provide any guarantees on the optimality or uncertainty of its routes?" From the definition of the hybrid method $\mathcal{F}$, it can now be seen that $\mathbb{P} = \mathbb{F}(n-1)$ and $\mathbb{R} = \mathbb{F}(0)$. Thus, the value of r also answers the *proactive vs. reactive* debate on whether all routes should be pre-determined or not.

It would be inappropriate, at this point, to conjecture that well-order implies that all routes are optimal, because so far there has been no mention as of how either proactive or reactive schemes converge to an optimal solution. It is appropriate however to discuss the relation between optimality (or uncertainty) and well-order. For this, the second order to be introduced to the system is an order on the *routing tables* of every node in the network.

Using the same reference point x, as with the first order, and a routing function $\mathcal{F}_x$, construct the sequence

$$\mathcal{F}_x(y_1), \mathcal{F}_x(y_2), \mathcal{F}_x(y_3), \ldots, \mathcal{F}_x(y_r), \mathcal{F}_x(y_{r+1}), \ldots, \mathcal{F}_x(y_{n-1}),$$

where the function $\mathcal{F}_x(y_r)$ returns the next hop, say w_r, towards the r^{th} destination y_r. The binary relation that orders the set $W = \{\mathcal{F}_x(y_r)\}$, for all $0 \leq r < n-1$, is

$$\begin{aligned} &c(x, w_r) + c(w_r, y_r) < c(x, w_{r+1}) + c(w_{r+1}, y_{r+1}), \text{ or} \\ &c(x, w_r) + c(w_r, y_r) = c(x, w_{r+1}) + c(w_{r+1}, y_{r+1}) \text{ and } y_r < y_{r+1}. \end{aligned}$$

The set W is also a well-ordered set. It is also the *routing table* at node x. Ideally, for all $x, w, y \in V$ and $w = \mathcal{F}_x(y)$,

$$c(x, y) = c(x, w) + c(w, y)$$

in which case

a) node w is the *best next hop* from a source x towards a destination y; and

b) The scheme $\mathbb{F}$ is an optimal routing scheme.[1]

The are two things to take away from this second well-ordering of nodes. First, the routing function $\mathcal{F}_x : W \longrightarrow Y$ is an *order-isomorphism*:

i) it is a bijection between W and Y (every destination $y \in Y$ corresponds to *exactly one* entry $w \in W$), and

ii) it preserves the order of Y. That is, for every $\alpha, \beta \in W$,

$$\mathcal{F}_x(\alpha) \prec_x \mathcal{F}_x(\beta) \text{ iff } \alpha \prec_x \beta.$$

Second, *if a routing scheme* $\mathbb{F}$ *is optimal, then it preserves the first ordering of nodes.*

If, on the other hand, there exists nodes $x, z, y \in V$ and $z = \mathcal{F}_x(y)$ such that

$$c(x, y) < c(x, z) + c(z, y),$$

then the computed route from x to y via node z is sub-optimal. In this case, node z does not belong on the shortest path from x to y. Such a node z may, for example, belong to a subset of nodes that form a hierarchical structure in

[1]If this is not obvious at first, select a source-destination pair $\{x, y\}$ and proceed with w, the best next hop from x towards a destination y, as your next reference point to construct the routing table at node w; continue the process, always following the best next hop w, until eventually you reach destination y. The sum of the costs to traverse each node w is $c(x, y)$.

the network. When such uncertainty is introduced into the routing system, it is necessary to bound it. Thus, for all $x, z, y \in V$ and $z = \mathcal{F}_x(y)$,

$$c(x,z) + c(z,y) \leq \Delta c(x,y).$$

The first inequality refers to the *triangle inequality*; the second inequality refers to *stretch* and it is an upper bound on the uncertainty of the system. The remainder of this section investigates the relation between well-order and the value of Δ.

If $y \prec_x z$, then the routes are sub-optimal. Or, in other words, a destination y should not precede the next hop z in the first-ordering of node x. Thus, in order to bound the stretch of a computed route, it must be

i1) $c(x,z) \leq c(x,y)$.

Using equation i1) as an invariant, an upper bound on the stretch of sub-optimal routing schemes can be derived as follows.

By the triangle inequality, the cost from node z to node y is

$$c(z,y) \leq c(z,x) + c(x,y)$$

and, by replacing $c(z,x)$ with $c(x,y)$ according to i1), the cost from node z to node y becomes

$$c(z,y) \leq c(x,y) + c(x,y).$$

Thus

i2) $c(z,y) \leq 2c(x,y)$.

Then, the cost of a sub-optimal route from a source x to a destination y via a next hop z is

$$c(x,z) + c(z,y) \leq c(x,y) + c(z,y),$$

because $c(x,z)$ is less than or equal to $c(x,y)$ according to i1), and

$$c(x,y) + c(z,y) \leq c(x,y) + 2c(x,y),$$

because $c(z,y)$ is less than or equal to $2c(x,y)$ according to i2). Thus,

$$c(x,z) + c(z,y) \leq 3c(x,y).$$

This implies that if a routing scheme computes next hops that satisfy invariant i1), then the computed routes are at most 3 times longer from the optimal ones. It also implies that any attempt to reduce the size of routing tables by means of aggregation (e.g. by introducing a hierarchical structure in the network) will result in sub-optimal routes.

1.1.1 The proactive method

A *purely proactive* routing scheme $\mathbb{P}$ (a scheme where all nodes $y \in V$ follow the proactive method) is an optimal routing scheme.

i) The memory cost is $O(n \log n)$ bits.

For all $y \in Y$, a proactive scheme of y will add an entry in the routing table of node x; there are $n-1$ possible destinations, hence $n-1$ entries (of size $O(\log n)$ bits[1]) in the routing table of x.

ii) The start-up overhead of a purely proactive scheme is $O(n^2)$ messages.

For all $y \in Y$, a proactive scheme of y sends an advertisement to node x. There are $n-1$ possible destinations y, and each of them willing to participate in the routing task from x to y, hence the cost to propagate an advertisement is $O(n)$ messages. The total message overhead for $n-1$ proactive schemes is $O(n^2)$ messages.

Subsequent advertisements for a destination y require $O(n)$ messages as well. Ultimately, all routes to destination y converge to an optimal solution.[2]

1.1.2 The reactive method

The reactive method is for node x to query an optimal (or near-optimal) route to node y, instead of computing it locally. Clearly, this would diffuse

[1]A flat address space of a n-node network can be represented by $O(\log n)$ bits; the base of the logarithm is 2.

[2]To see this, select any destination y and use a distance vector algorithm to propagate advertisements. That is, when a node x receives an advertisement from some node w on behalf of y, node x uses node w as the next hop towards y if and only if destination y is previously unknown to x, or destination y is closer to x through node w than it was through the previous next hop. Eventually, $c(x,y) = c(x,w) + c(w,y)$.

the computational burden and, more importantly, avoid the requirement to store *a priori* large amounts of topological state.

If $\mathbb{R}$ is a *purely reactive* routing scheme (all nodes x follow the reactive method), then the routing tables it computes should also preserve the order $\prec_x$, as in the case of a purely proactive scheme, since there is a one-to-one correspondence between proactive and reactive schemes. However, reactive schemes – as they are presently understood – are not optimal. In particular, reactive protocols (e.g. the Ad hoc On demand Distance Vector protocol (AODV) [72]) are not known to converge. This is because, by construction, it is the destination (or some r-neighbor of the destination) that decides on a route; nodes do not collaborate to converge to some better solution [71]. This is to be contrasted with the proactive method where route decisions are made by node x, as it collects state from other nodes. Experimental results have also shown reactive routing protocols to diverge from optimal solutions [7] (also cf. Figure 6.7 in § 6.4.3).

1.1.3 A hybrid approach

It is possible to construct hybrid routing schemes that maintain partial routing state by ways of omission, rather than aggregation. For example, a node x can omit a route to y and query it instead, whenever required. This is because a reactive scheme of x can complement any omitted proactive scheme of y. Let us now reconsider the proactive vs. reactive debate.

The proactive method is an isomorphism $\mathcal{P} : W \longrightarrow Y$ well-ordered by minimum-cost routes. In essence, the proactive method is a function that maps the second ordering of nodes to the first ordering of nodes, for all nodes in a network. In a similar manner, it should be $\mathcal{R} : W \longrightarrow Y$ as well. However, it is not known whether reactive protocols converge to optimal routes. Thus, it is not known whether function $\mathcal{R}$ preserves the first ordering of nodes. Nonetheless, $\mathcal{R}$ will return a well-ordered set. It is known that there is an isomorphism between two well-ordered sets that preserves the order between one and an initial segment of the other (e.g. [32]).

This isomorphism is the function $\mathcal{F}$, and the scheme $\mathbb{F}(r)$ is an initial segment of the proactive scheme $\mathbb{P}$ (by definition of the hybrid method, the routes to the r-first neighbors are computed proactively). If $\mathcal{F}$ is optimal, then it would preserve the order $\prec_x$ of Y, for all $x \in V$. Then, the second ordering of nodes that is computed by the proactive method (i.e. the ordered set of best next hops W) would be isomorphic to the one computed by the reactive method. Or, reactive schemes would be equivalent to proactive schemes:

$$\mathbb{P} = \mathbb{R}.$$

Let us investigate this equality a bit further. The equivalence relation to look for between proactive and reactive routing schemes is indeed an order-isomorphism. This is because, when speaking of infinitely scalable networks (i.e. very large values of n), if $\mathbb{P}$ and $\mathbb{R}$ are said to be equivalent then they are of the same order type: $\mathcal{P}$ and $\mathcal{R}$ are both isomorphisms that map to the same unique ordinal number (the order $\prec_x$ of Y, for all $x \in V$).

From another perspective, when a routing function is applied to all source-destination pairs, it computes a finite metric space (a set with n^2 elements) which, ideally, it should map to the optimal space (V, c). One way to visualize a routing scheme is to take the set of all routing tables (which are themselves sets). For example, $\mathbb{P}$ is the set of all routing tables as computed by the proactive method – similar for $\mathbb{R}$. If both routing methods are optimal, then they both compute the same set (of sets). Hence the relation $\mathbb{P} = \mathbb{R}$. This dissertation investigates the following conjecture.

Conjecture. *In a weighted network (V, c) and for every node $x \in V$, there is a positive integer r, such that $\mathcal{F}(r)$ preserves the order $\prec_x$ of Y.*

This dissertation attempts to establish a lower bound on the value of r.

1.2 Policy spaces

The previous section has defined the metric space (V, c) – the set of all nodes V, together with a cost function

$$c(x, y) = \text{ the minimum cost to route a message from node } x \text{ to node } y$$

for all $x, y \in V$ – as an optimal metric space. This section delves further into the nature of this cost function (or metric). Sensors are opportune candidates for the application of metrics for two reasons.

The first reason is an intuitive argument. It suggests that sensor networks are not general-purpose networks. They rather have stringent requirements and a routing system ought to meet them. For example, consider the following three user goals, common in a sensor monitoring system:

1) network longevity,

2) successful message delivery, and

3) message delivery within delay bounds.

A routing scheme of choice must satisfy one or more of these goals. The input to a routing scheme is a cost function and the objective is to minimize it (compute minimum-cost routes). Thus, from a pragmatic point of view, the cost $c(x, y)$ should be 1) the *energy spent* to route a message from x to y, or 2) the *probability of success* to route a message from x to y, or 3) the *average time* to route a message from x to y. These quantities (energy spent, probability of success, time) are not beyond dispute as to whether they satisfy their intended goals or not; for their appropriateness, the dissertation revisits them in Chapter 4. These quantities are however *measurable*, and users (resp., sensors) ought to devise (resp., measure) such metrics if the application (resp., system) is to achieve its goals in a dynamic environment.

The second reason is an imperative argument. Static systems (i.e. time-invariant) cannot solve the dynamic routing problem in sensor networks because sensors form variable-quality links that vary with time (see [13, 79, 89]). In other words, the metric space of a wireless sensor network is continuous.

An immediate consequence of the above two arguments is the following. Let $\overline{c}(x, y)$ be such a measurable metric and consider two successive measurements. When the divergence between them is sufficiently large, say by $\epsilon > 0$, the system should seek another route, say, from sensor x to sensor y. Then, for all $x, y \in V$ and $\epsilon > 0$,

$$c(x, y) \leq (1 + \epsilon)\overline{c}(x, y).$$

and an optimal routing scheme is an ϵ-optimal scheme. The value of ϵ controls the frequency of updates. The more frequent the updates, the less likely is the system to converge to an optimal solution. Moreover, every update message should be propagated to the entire network, resulting in $O(n)$ messages.

It should be obvious by now that simply using any cost function $\overline{c}$ as an input to an optimal routing scheme does not necessarily return optimal routes, simply because the mapping from $(V, \overline{c})$ to (V, c) does not necessarily preserve order.

The problem again is to find an isomorphism from $(V, \overline{c})$, the measured metric space that is ϵ-optimal, to the policy space (V, c). Thus, the dissertation proposes a mapping of costs to hops (discrete-valued weights), where

$$\text{a hop} = \text{ the cost of a successful transmission.}$$

Use node x as a reference point. What is required is a continuous map from $(V, \overline{c})$ to (V, c) such that if, for all y,

$$\overline{c}(x, y) < \epsilon,$$

then

$$c(x, y) < k.$$

This defines the k-neighborhood of a node x. This is also a measure of the deviation of a computed solution from an optimal solution. This also suggests that one could control the distortion of routes within a k-neighborhood.

An improvement within a k-neighborhood will converge to an optimal solution. It remains to be seen if a degradation will bound the divergence

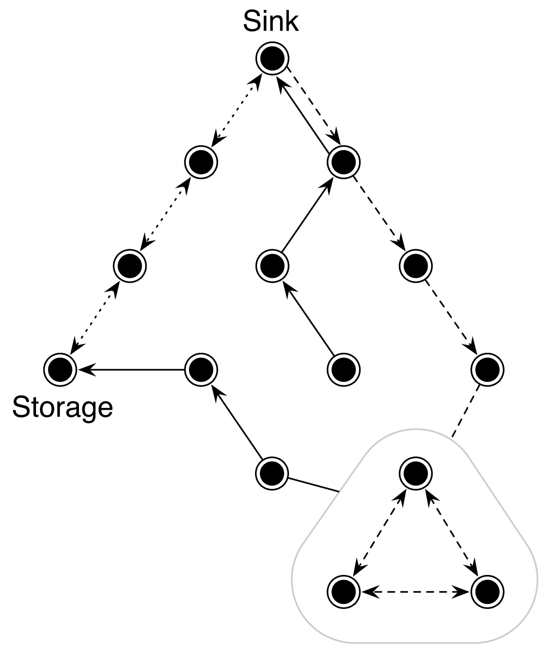


Figure 1.2: An example wireless sensor network application. A sink, besides collecting data (solid links), can query individual nodes (dotted links) or disseminate a task to a group of nodes (dashed links).

from an optimal solution by a factor of k. The value of k is associated with the value of r as follows:

$$I_x(r) = \{y|c(x, y_r) < k\}.$$

Typical deployments of sensor networks may free an optimal routing system from the burden of computational complexity (due to simple traffic patterns), but they have put forth link dynamics as a major barrier to converge to optimal routes. This is further discussed below.

1.3 State of the art

There are two basic networking tasks in a wireless sensor network, collection and dissemination [53]. Figure 1.2 illustrates some examples.

Many sensor networks exhibit a unique gradient communication model where *many* sensors route their measurements via multiple wireless links to a *few* collection points, or sinks. Often, sensor monitoring systems have a single sink.

Dissemination tasks, e.g. management tasks, are also performed in the aggregate. There, applications select a *few* access points to configure (or query) *many* sensors. Again, applications often use a single access point to configure (or query) the network, which usually coincides with the sink (e.g. [62]).

In the absence of any pre-constructed state in the network, flooding (and its variants) is a natural way to disseminate messages to a set of nodes. One should note that the basic mechanics for dissemination and collection are complementary: a message flood from a sink y (dissemination) also suffices to construct a tree $T(y)$ (collection). This reduces the general routing problem to routing in a tree.

At the heart of collection there is a distance vector routing algorithm. Suppose that x receives a distance vector[1] from some neighbor w. Then, x will set w as the next hop towards the sink y if and only if $c(x, y) > c(x, w) + c(w, y)$.

In situations like this, e.g. where x's successor to y changes, node x must issue an update message to notify its predecessors; in turn, they must notify their children and so on, until leaf nodes have re-computed their least-cost path to the sink y. There are two major problems with this: the scope of update messages, and their frequency. Also, during this process of convergence, routing loops are likely to appear.

1.4 Challenges

As new applications emerge, e.g. urban sensing [11], there is a growing need for efficient point-to-point routing support in sensor systems. Sensors must

[1]A distance vector is a message that contains a vector (or list) of distances (or costs) to known destinations [63]. For the case of a single destination, as in collection, the vector consists of one entry.

be able to address[1], locate and communicate with one another. At the same time, it must minimize the complexity of the optimal routing scheme.

In summary, the following issues arise.

1) The setup overhead is $O(n^2)$ messages. All known routing algorithms, in theory or in practice, require a complete topological view (a complete metric space) of the network.

2) The update overhead is $O(n)$ messages. The aforementioned topological view must be promptly updated at all nodes; network-wide updates (or network-wide queries) due to network dynamics are a limiting factor as they make convergence to a solution difficult.

3) The memory overhead is $O(n)$. The result is a n-size routing table, even though only a few of the destinations are required. Any attempt to truncate this table results in sub-optimal routes.

Routing schemes that incur these overheads are expensive, although they can guarantee optimality. This limits their applicability at large scale. These quantities cannot be minimized all at once. Take, as an example, hierarchical routing. Hierarchical routing protocols trade-off memory (and, subsequently, message) overhead but settle for sub-optimal routes.

An intuitive approach is the following. Network state is stored in the nodes (memory) and, subsequently, in their in-between links (as messages). Network state is necessary to compute an optimal route. Hence, if a scheme is to reduce the amount of network state a node stores in memory, then it must increase the number of messages in order to distribute it. Similarly, if a scheme is to reduce the number of messages, then it must store more network state in memory.

The locality of routing state plays a key role in dynamic sensor networks. For example, in the presence of multiple sinks it is advantageous to collect data at nearby locations (subject to resource availability or load balancing).

[1]A destination's address does not necessarily coincide with its identifier. For example, sensors can route packets based on geographic (absolute or relative) coordinates, data attributes, or roles.

Thus, in this dissertation, local state of a node, say x, refers to those nodes that are within distance at most k from of x. Here, an inexpensive routing algorithm utilizes only local state to converge to optimal (or near-optimal) routes between arbitrary pairs of nodes.

This leads to the following thesis statement.

1.5 Thesis statement

Most of the time, there is no need to apply the most expensive routing algorithm, but there are circumstances in which an inexpensive routing algorithm would be inefficient.

I assert that there is a family $\mathcal{F}(k)$ of routing schemes that trade off size of routing tables and communication overhead to produce paths that k-satisfy different application requirements.

I will demonstrate the efficacy of $\mathcal{F}(k)$ in terms of memory cost, message overhead, stretch, and resilience to faults. I will first show that proactive and reactive routing schemes are equivalent, up to an isomorphism. Next, I show that the scheme $\mathbb{F}(r)$ converges to optimal routes between arbitrary source-destination pairs for values of $r \leq n-1$. The value of r, the number of pre-computed routes, depends neither upon the size nor the density of a network; it rather depends upon the circumference, the size of the largest cycle, of a network.

1.6 Contributions

1) The proactive and reactive setup overheads are complementary.

2) In the absence of a sense of direction, the setup overhead is $O(n^2)$.

3) Given a network with girth $g \leq 2k+1$, in the presence of sense of direction, $\mathcal{F}(k)$ converges to optimal routes.

4) Every subsequent failure stretches the path by $O(2k-1)$ hops.

The above results are derived based on the algorithm $\mathcal{F}(k)$ and hold for every network. They are derived by normalizing the cost to hops.

If a cost function $c(x, y)$ is "a proper" cost function – i.e. $c(.,.)$ reflects a user objective – the resulting paths are optimal with respect to that objective. (Subsequent paths are also optimal, by the principle of optimality).

By normalizing the link and node costs to hops, the results for F(k) hold for dynamic networks. Specifically,

a) the hop-count metric achieves the best possible results when all links are equiprobable and have unlimited capacity.

b) the ETX metric (by definition, a normalization of link quality into hops) achieves the best possible efficiency – the ratio "No. of transmissions / path length" – when all links have unlimited capacity.

c) the congestion-degree (or service-time) metric achieves load balancing when all links are equiprobable.

d) the energy metric achieves similar results to congestion degree. This is because nodes report an ever-decreasing linear cost (the battery level), which is related to the number of transmissions.

1.7 Outline

The remainder of this dissertation consists of the following chapters.

Chapter 2 is an in-depth review of routing metrics and routing schemes that are related to the research questions of this work, the values of k and (subsequently) r, in one way or the other. First, it reviews link quality measurement techniques to better understand the notion of the best next hop towards a destination and how such a node can be computed by an on-line mechanism. With this in mind, the chapter proceeds with an enquiry into approximate routing schemes, starting from the proactive method (which is known to converge to optimal routes), and finishing with existing hybrid routing schemes. The approximate routing schemes under consideration are:

1) *Interval routing.* Interval routing schemes reduce the size of routing tables by assigning names to nodes using intervals. For example, one could assign the numbers 1 to n to nodes in such a way that, at source $x = 0$ and for destinations 1 to r there is a unique next hop w, and for destinations r to $n - 1$ another node z, resulting in a routing table of size 2.

2) *Geographic routing.* Geographic routing schemes reduce the size of routing tables by assigning names to nodes using a coordinate system. For example, one could assign the system of r-coordinates $\{y_1, y_2, \ldots, y_r\}$ to all destinations y, reducing the number of pre-computed routes to r (or if the coordinate system is known, to 0.)

3) *Compact routing.* Compact routing schemes reduce the size of routing tables by constructing a system of k hierarchies $\{y_1, y_2, \ldots, y_k\}$ in such a way that the uncertainty of the system is bounded by k.

These schemes are discussed with proactive and reactive schemes in mind and, of course, wireless networks. For this reason, it also reviews the cost of broadcast. The chapter ends with a principled classification of existing routing protocols according to the proactive or the reactive method.

Chapter 3 presents the family of protocols $\mathcal{F}(k)$. It describes the semantics of the proactive and reactive method. It shows that the proactive and reactive schemes are equivalent; and that their overheads are complementary. It also shows that the hybrid method $\mathcal{F}$ can converge to optimal routes. This means, as I have discussed previously, that $\mathbb{R} = \mathbb{P}$.

Chapter 4 discusses metric spaces for wireless sensor networks. It discusses different metrics, and whether they satisfy a number of goals (or objectives) set for the system. The discussion on metrics, and their impact on the performance of a routing system, is augmented with simulation results and a small experimental testbed.

Chapter 5 gives a functional overview of $\mathcal{F}(k)$ in terms of its operations on the basic protocol data structures. It describes the implementation of such a scheme in the TinyOS framework.

Chapter 6 discusses the value of r, with respect to the growth of networks. It related the family of protocols $\mathcal{F}(k)$ with the routing scheme $\mathbb{F}(r)$ and attempts to establish a lower bound on the value of r.

Chapter 7 summarizes the contributions of this work. Chapter 8 summarizes any questions left open by this dissertation. It discusses the special case of $\mathbb{F}(0)$. The scheme $\mathbb{F}(0)$ is optimal, yet the policy space is not known. Or better, if measurements are not required, then $\mathbb{F}(0)$ can explore the metric space. It enumerates instances of the cost function $c(x, y)$ where it could include policy that is related to duty-cycles; it generalizes to self-managed systems; and, finally, it discusses mobility.

Chapter 2

Related work

Sensors form a multi-hop *ad hoc* wireless network. Therefore, in addition to their sensing and processing tasks, they also act as routers. As such, sensors are equipped with a mechanism to relay messages along optimal, or near-optimal, routes. This mechanism is part of a *routing scheme*, and it defines those protocols and data structures that are necessary to compute and maintain routes between any source-destination pair.

This chapter discusses existing optimal and approximate (Δ-optimal) routing schemes for wireless *ad hoc* networks. Recent work on the Internet's routing system and, in particular, its scaling limits [50] has shifted the focus to compact routing schemes, the theory of which also applies to networks of sensors [61, 37]. For that reason, the discussion may sometimes extend to general networks, as well.

2.1 Background

A routing scheme *implicitly* specifies a path

$$\Pi(x, y) = \{x = x_0,\ x_1,\ x_2,\ \ldots,\ x_j = y\}$$

between a source node x and a destination node y that lists the $j - 1$ intermediate nodes a message must traverse from x to reach y. Node x_i is called a **successor** of x_{i-1} and, respectively, x_{i-1} is a **predecessor** of x_i.

A routing scheme *explicitly* distributes and stores this information at each of the j nodes in a way that a routing table look-up at some node x_i, $0 \leq i < j$, for destination y will return its successor x_{i+1}.

Each node in the path $\Pi(x, y)$ executes a distributed routing algorithm to calculate a common routing table. Routing tables must be consistent to guarantee that a message from node x will eventually reach node y. In essence, if a message starts at node $x = x_0$, then successive look-ups at x_i for destination y will return path $\Pi(x, y)$. Any table inconsistency will otherwise cause the message to be dropped or get caught in a routing loop.[1]

Routing protocols distribute network state (topological information) in a *link-state* or a *distance vector* manner. Link-state protocols require each node to first store the entire network state in a data structure (a topological database) that contains the cost to traverse every link in the network; then, each node runs a shortest path algorithm, e.g. Dijkstra's algorithm, over this topological database to construct a routing table. The size of the database on each node is in the order of n^2 [83]. On the other hand, distance vector protocols construct routing tables incrementally, while network state is in transit, without the aid of a database. The memory complexity is $O(n)$ [72, 63].

In § 1.2, a distance vector routing protocol has been described as the basis of data collection to a single destination, the sink. This requires a more elaborate definition, as it will serve as a reference point for what follows. Recall the definition of r-first neighbors and $c(x, y)$ from § 1.1.

Let $\{w_i\}$, $1 \leq i \leq r$, denote the r-first neighbors of node x, breaking ties according to labels. Then, the **best next hop** to a destination y is an r-neighbor, namely w^*, such that

$$c(x, w^*) + c(w^*, y) = \min_i \{c(x, w_i) + c(w_i, y)\}. \tag{2.1}$$

Node x requires two pieces of information to compute w^*: $c(x, w_i)$, the cost to reach each of its r-neighbors; and $c(w_i, y)$, the cost from each of its r-neighbors to destination y. It is in the interest of this dissertation to

[1]A message is caught in a routing loop whenever it traverse routes of the form $a \rightarrow b \rightarrow a$, $a \rightarrow b \rightarrow c \rightarrow a$, and so on. A routing loop can be detected, and possibly resolved, once the message returns to a.

determine which of these costs are either local (stored in memory) or in transit (stored in messages), since it has conjectured on the value of r. This is also a way to characterize the relative complexity of different schemes in terms of

(a) routing state, the amount of topological information stored in memory, and

(b) message overhead, the amount of topological information stored in transit.

Furthermore, routing schemes are characterized in terms of

(c) routing stretch, the maximum factor by which the cost of the computed route from x to y deviates from an optimal one (e.g. as computed by an oracle that runs Dijkstra's algorithm over an up-to-date topological database).

In order to argue about the locality of $c(x, w_i)$, and subsequently $c(w_i, y)$, first it is necessary to introduce routing metrics.

2.2 Routing metrics

A cost function in this dissertation is an expression, not always known, of a routing objective. As networks change over time (e.g. quality degrades, load increases, or nodes' batteries deplete), an adaptive routing scheme must be able to sense such changes and re-compute routes accordingly. Every adaptive routing strategy includes a method to monitor or measure the network's dynamics.

Monitoring is important to diagnose and repair faults in the network. A fault signifies an incoherence in some node's routing state and it must be reported to the network. Every fault has a magnitude, or a *cost*. A fatal fault has a cost of infinity.[1] Lesser magnitudes are measurable quantities

[1]In practice, an infinite cost is represented by a large, integer value relative to the diameter of the network, e.g. `0xff`, `0xffff`, etc.

(e.g. the probability of loss, the size of the message queue, or the remaining battery life).

Nodes exchange messages periodically to estimate their costs to direct neighbors, those who are within the scope of one broadcast (e.g. within radio range), or to more distant neighbors, those who are two or more broadcasts away. Approximation methods are tightly coupled with the absence or presence of traffic between any two neighbors.

2.2.1 The importance of neighbor monitoring

The simplest routing metric is the hop-count metric. The hop-count metric assumes that the cost to all direct neighbors is constant, say 1, and all neighbors are equally willing to participate in the routing task. Then, the best route from x to y is the shortest path $\Pi(x, y)$, one that traverses the minimum number of hops. The hop-count metric is a function of the cardinality of a path $\Pi(x, y)$.

Once nodes converge to stable routes, they should be able to detect link or node failures. A link failure to some direct neighbor y (respectively, a node failure of y) is detected at node x by the absence of control traffic from y. Typically, node x may assume that the link to neighbor y is lost if it receives no traffic from direct neighbor y for a pre-specified period of time.

Node x considers the link $\langle x, y\rangle$ a hop if y is a direct neighbor – i.e., it receives y's updates directly, within the scope of one broadcast, and periodically, within an activity period. Such short updates are broadcast packets, usually termed hello messages, and they are sent every τ seconds, the hello interval. This periodic message exchange, from x to y and vice versa, is termed a hello protocol [65, 71].

The inactivity period is usually a multiple, say 3, of the hello interval τ. It signifies how many of y's hello packets must be missed before node x announces a link or node failure to the network. The value of the inactivity period, i.e. the hello interval multiplied by the allowed hello packet loss, is a user-specified parameter.

An alternative interpretation of the hello protocol is the following. Node

x should receive one hello message from node y during the interval τ, two hello messages during the interval 2τ, and so on. Suppose that the i^{th} hello message arrived at time t. Then, node x should have received approximately $\frac{t}{\tau}$ hello messages during the time interval $[0, t]$. The hello protocol also can provide an approximation of the delivery ratio from y to x: node x received i out of $\frac{t}{\tau}$ messages from node y.

Estimators of quality require that node x retain some memory about previous transmissions from node y.

2.2.2 Link quality estimators

Shortest paths are not good enough [13] because the presence of a link does not necessarily guarantee its quality. Indeed, the hop-count metric is agnostic to both quality and load. On the other hand, the Expected Transmission Count (ETX) metric [13, 10] (cf. § 4.2.1.1) continuously approximates the delivery ratio between two neighbors x and y as the ratio

$$p = \frac{i}{(\frac{t}{\tau})},$$

where i is the number of hello messages received from y during the last t seconds (stored in node x's memory), and $\frac{t}{\tau}$ is the number of messages that should have been received from y. Node y approximates the delivery ratio q from x to y in a similar manner. The delivery ratio p (respectively, q) from y to x (respectively, from x to y) is piggybacked onto hello messages. The expected number of transmissions to traverse link $\langle x, y\rangle$ is then

$$\frac{1}{pq}.$$

Since its inception, the ETX metric has been augmented with additional measurements from the data plane [18]. In wireless networks, nodes await a link-layer acknowledgment to ensure that a data packet has been successfully received at the next hop along the route.[1] Otherwise, they retransmit the data packet (or drop it, after a maximum number of retransmissions). It is

[1]Assumes a CSMA/CA Medium Access Control (MAC) protocol.

advantageous to use link-layer acknowledgments (unicast traffic), as well as hello messages (broadcast traffic), to derive more accurate estimates of link quality. This modified ETX metric has been the metric of choice in TinyOS sensor networks [89, 24].

Wireless link quality is associated with the quality of the received signal. The IEEE 802.15.4 physical layer provides two measures of signal quality, the Link Quality Indicator (LQI) and the Received Signal Strength Indicator (RSSI) [34]. These measures attempt to prevent nodes from having to actively send hello messages to assert neighbor connectivity. However, since LQI or RSSI are computed upon reception, they do not take into account packet loss. There has been substantial effort to correlate signal quality with packet reception probability pq, as discussed in ETX (cf. § 4.2.1.2. See also [79]).

The cost of a wireless link is also a function of its load. With ETX, link cost has been associated with the average number of transmissions a message requires to traverse it, rather than the number of messages that actually traverse it. High load can lead to buffer drops or even congestion collapse. To avoid congestion, wireless nodes employ a binary feedback scheme, similar to [74]. In particular, each router monitors its average message queue growth and whenever it becomes large, it sets a congestion avoidance bit to notify its predecessors to limit their sending rate.[1] The congestion bit can also serve as a measure of link quality during route (re-)computation.

In a similar manner, the four-bit wireless link estimator [18] uses four bits to fuse different indicators of quality into a single metric: the *pin* and *compare* bits are set from the network layer to prevent or allow the eviction of a particular neighbor from the routing table, respectively; the *ack* bit is set from the data-link layer to indicate a successful transmission of a data packet to that neighbor; and the *white* bit is set from the physical layer to indicate high signal quality between the two neighbors.

Given these measures of quality, it may be asked whether routing state

[1]An Explicit Congestion Notification (ECN) bit can be carried either in a hello message or in an acknowledgment.

should expire or not. Since a fault is denoted by a cost increase, nodes may simply switch to a better neighbor whenever it is available. However, one can not simply ignore the aging of routing state: it is the presence of traffic that enables a node to estimate link quality; in the absence of traffic, routing table entries should expire.

It turns out that link quality estimators are scattered and tangled (and usually duplicated) across the networking stack. Starting from coarser approximations based on the presence or absence of hello messages, nodes can derive more accurate estimates of costs.

Combinations of different metrics are, in essence, successive applications of different filters to the set of r-neighbors. Let us now return to the definition of the best next hop w^* (Equation 2.1).

Suppose that $W_0 = \{w_1, w_2, ...w_r\}$ is the initial set of the r-first neighbors. Then, by filtering out neighbors successively (e.g. based on the congestion bit, the ack bit, and so on) one can derive

$$W_0 \supseteq W_1 \supseteq \cdots \supseteq W^* = \{w^*\},$$

where W^* is the subset of neighbors that contains the best next hop to a destination y.

The set W^* is non-empty because node x requires at least one table entry to route to a destination y, one that points towards a neighbor w^*.[1] This is further discussed in the following section.

2.3 The proactive method

With no loss of generality, this section assumes the hop-count metric, where all link costs are constant and equal. Thus, $c(x, y) = 1$ for all x and y. Using a hop-count metric, the best next hop to a destination y is a direct neighbor w^* such that

$$1 + c(w^*, y) = \min_i \{1 + c(w_i, y)\}. \tag{2.2}$$

[1]In the case of wired networks, this is the output port to w^*. For wireless networks, as in this case, it is a unique node identifier in the message header.

The costs $c(w_i, y)$ arrive in messages. This follows from the proactive method, according to which the route to a destination y is computed at node x by the proactive method $\mathcal{P}$ of y. More specifically, node y floods an advertisement to the entire network which will arrive at node x via some neighbor w.

Node x re-computes w^* if one of the possible routes from x to y changes by some Δ, a confidence interval. Specifically, there are two distinct cases:

c1) $c(w_i, y)$ decreases by a user-chosen Δ; or

c2) the current route to node y expires.

Case c1) represents the static case, in which a proactive routing scheme will eventually converge to shortest paths after a finite number of steps; the resulting paths will be loop-free [38, 73].

For c1), node x will possibly switch to another neighbor only if some neighbor w_i reports a newer, lower cost. This means either that (i) node x receives a message from w_i for the first time; or (ii) node w_i has found a better route than the one it had previously reported. This is because the hop-count metric is an ever-decreasing cost; a cost increase only signifies a fatal fault, in which case the cost increases from a constant, in this case 1, to infinity.

Case c2) represents the dynamic case, in which a proactive routing scheme may construct a routing loop. If the table entry for node w^* expires, then node x must solicit costs $c(w_i, y)$ again, since they have been previously discarded. This requires more careful consideration.

Since the costs $c(w_i, y)$ are in transit, the memory complexity is $O(1)$ for a single destination (a single entry suffices). For $n - 1$ possible destinations, the memory complexity is $O(n)$.

If costs $c(w_i, y)$ are in transit however, then node x must be able to distinguish whether the newer cost $c(w_i, y)$ is up-to-date. The Destination-Sequenced Distance-Vector (DSDV) protocol for mobile computers [73] uses sequence numbers to distinguish between new and stale advertisements. It always uses the most recent state to compute the best next hop.

If, on the other hand, costs $c(w_i, y)$ are local, then node x can make an informed decision on the best next hop locally, without any delay. The size of local storage required at node x is determined by the number of its direct neighbors w_i (or node degree). Let $\varrho(x)$ be the degree of node x. The memory complexity for a single destination is then $O(\rho)$, where

$$\rho = \max_{x \in V}\{\varrho(x)\}$$

is the maximum node degree in the network. For $n-1$ possible destinations the memory complexity has order $O(\rho n)$, a function of both the density ρ and the cardinality n of the network. This is prohibitive for resource-constrained devices, such as sensors.

2.4 Names, locations, and attributes

So far, node names have been chosen arbitrarily from the set $\{1, 2, 3, \ldots n\}$. A way to reduce the size of the routing tables is to assign meaningful names to nodes (or meaningful labels to links). These node names and link labels reflect a measure of "closeness", and thus they can establish a sense of direction amongst distant nodes. (For a survey on sense of direction, see [17].)

First consider an interval routing scheme [85], and more specifically an interval routing scheme for rings (as in [23]) that uses the hop-count metric. An interval routing scheme routes messages from x to y as follows: a table look-up at node x for destination y will return the link with label $[a, b)$ (an interval) such that $a \leq y < b$. This is illustrated in the n-node ring network of Figure 2.1.

If node names $x_0, x_1, x_2, \ldots, x_{n-1}$ are assigned randomly, then node x must store at least one bit for every possible destination (there are two possible routes for every pair of nodes, which can be decided by a coin flip); this results in an $\Omega(n)$-bit routing table per node. If, on the other hand, renaming of nodes is allowed, then one can compact the routing table as follows.

(i) Assign names to nodes as $x_0 = 0$, $x_1 = 1, \ldots, x_{n-1} = n-1$; and

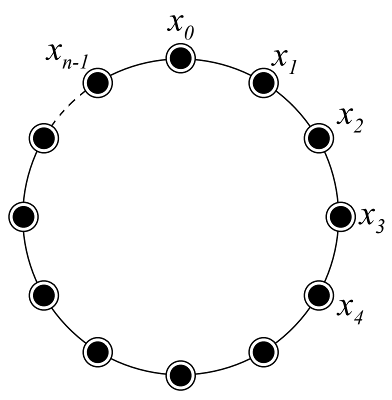


Figure 2.1: A ring graph.

(ii) Assign labels to links as $L(i) = ((i+1) \mod n)$ and $R(i) = ((\lceil \frac{n}{2} \rceil + i) \mod n)$ for the "Left" and "Right" links of each node $x_i = i$, respectively.

Then, for any source-destination pair (x, y), the best next hop from a node named x towards a node named y is located "Left" if $L(x) \leq y < R(x)$, or else it is "Right".

The above routing scheme uses only two table entries to establish a sense of direction (left or right) from which it computes the shortest route for every source-destination pair.[1] One may also establish a sense of direction in n-dimensional space.

The spatiotemporal relationship of events in sensor networks has led to labeled (or name-dependent) routing schemes. In contrast to unlabeled (or name-independent) schemes, sensors may route messages based on their data attributes or location coordinates, rather than their node identifiers.

Directed Diffusion [35] is an example of a collection protocol that uses data attributes, rather than node names, to initiate one or more flows towards a sink. In Directed Diffusion, a sink floods an interest expressed as a data predicate (e.g. an interest for "temperatures greater than 0^oC"); only the

[1]All networks have an interval routing scheme, but it is not necessarily an optimal one, as it is in the case of rings or trees [85].

nodes that have sensed data which satisfy this predicate will establish and maintain a path to the sink. Directed Diffusion borrows heavily from the reactive method[1] which is discussed in § 2.6. For now, the focus is turned to geographic routing.

Geographic routing uses location coordinates in order to make routing tables more compact. For a destination y with coordinates $\{y_1, y_2, \ldots y_n\}$ and a neighbor w of x with coordinates $\{w_1, w_2, \ldots w_n\}$, node w is the best next hop to y if

$$\sqrt{(y_1 - w_1)^2 + (y_2 - w_2)^2 + \cdots + (y_n - w_n)^2} \tag{2.3}$$

is the minimum Euclidean distance amongst all neighbors.

Greedy forwarding based on geographic coordinates, as a local heuristic, may converge to local minima. A local minimum is reached whenever the current next best hop towards a destination y is unable to forward messages any further because there are no neighbors closer to y than the current one and y is not within range. Consider for example the network of Figure 2.2, where node x wants to forward a message to node y. The Euclidean distance $\|x - y\|$ is smaller than $\|w - y\|$ and the routing algorithm can not propagate messages any further, although there is a valid route to y via node w. In order to route around these "voids" (or local minima), geographic routing schemes construct a planar graph of the network.[2] This requires a more elaborate discussion.

The Greedy Perimeter Stateless Routing (GPSR) protocol [42, 41] uses a right-hand rule to resolve cases of local minima (route around them). According to the right-hand rule, the successor of x_i along the path from x to y is a neighbor x_{i+1} that is located right (counter-clockwise) from x_i with respect to its predecessor x_{i-1}. Figure 2.3(a) shows a case where the right-hand rule succeeds. The right-hand rule fails in Figure 2.3(b) where two links cross each other, and therefore GPSR must construct a planar graph.

To generate a planar graph, a link $\langle x, y \rangle$ is removed from the network if

[1] Although Directed Diffusion [35] changes the address semantics, its underlying routing mechanism is reactive as it floods the network.

[2] A planar graph is a graph where no two links cross each other (e.g., a triangle).

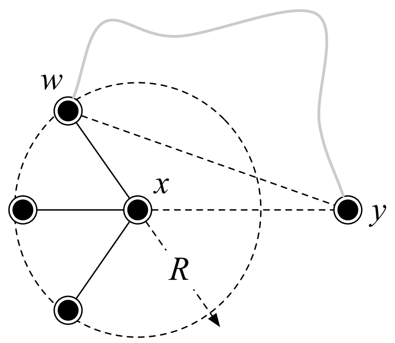


Figure 2.2: A local minima in geographic routing.

there is a node w within the transmission range of both x and y. Node w is called a mutual witness of x and y. The Cross Link Detection Protocol (CLDP) [45] uses periodic messages (a hello protocol) to eliminate links that cross based on mutual witnesses. It also filters out (as described in § 2.2) those links with poor quality. Yet, the right hand rule can result in routes that may traverse the entire network (thus the stretch is on the order of n). For example, apply the right-hand rule to the network of Figure 2.1 to route a message from x_0 to x_{n-1}.

Geographic routing schemes assume that the coordinate system is an accurate one. Accurate coordinates come at a cost, either a monetary one (e.g., spent on specialized hardware) or an algorithmic one (e.g., spent on transmissions). The following introduces relative coordinates.

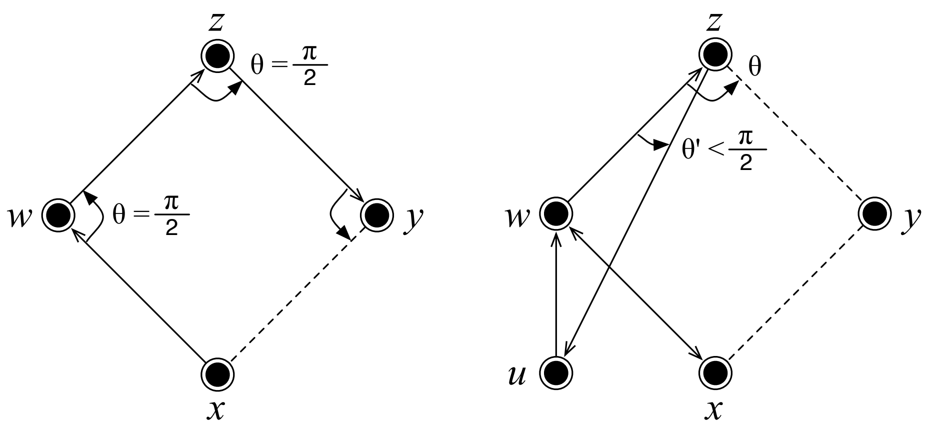
The Euclidean distance defines a metric space which is not necessarily an optimal one. This is because geographic coordinates do not reflect true link costs.[1]

One can perform geographic routing without true location information, but rather using relative coordinates. Relative coordinates are derived based on the proximity of nodes to some landmark, or anchor, nodes [76]. This is

[1]Although a signal decays with distance, it is **not safe** to assume that

$$\|x - z\| \leq \|x - y\| \Rightarrow c(x, z) \leq c(x, y)$$

due to the other vagaries of wireless links, e.g. reflected signals etc. (cf. Figure 6.2.)



(a) $x \to w \to z \to y$ (b) $x \to w \to z \to u \to w \to x$

Figure 2.3: The right-hand rule.

also the case for localization algorithms that use a number of anchor nodes to construct an absolute or relative coordinate space [78, 29].

2.5 Routing hierarchies

A routing scheme can compact routing tables by grouping nodes together into sets. By recursion, these sets form different levels of hierarchies. This is done by selecting a set of designated nodes (or landmarks) to represent each set and assigning names accordingly. For example, consider the network of Figure 2.4.

All paths to y_1, y_2, and y_3 have a common predecessor, namely node y. Thus, one can designate node y as a landmark and assign names as $y.1$, $y.2$, and so on. By way of construction, all nodes x can route to all y nodes using a single table entry, one that points at node y.

2.5.1 The landmark hierarchy

Hierarchical routing schemes organize a network into areas, which in turn are grouped recursively into levels [64]. A node's name is an indication of the

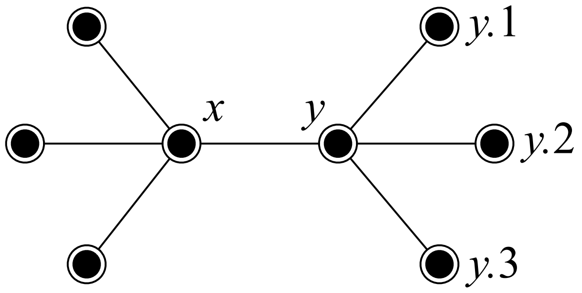


Figure 2.4: A routing hierarchy.

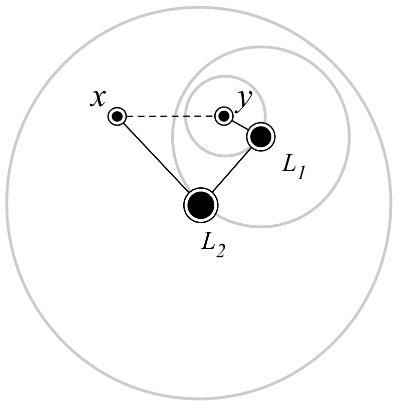


Figure 2.5: A landmark hierarchy.

area in which that node resides. At the lowest level (namely, level-0), routing is flat and a level-0 hierarchy represents a flat address space. The following describes how to construct a level-i hierarchy using landmarks.

Let A_i denote the set of landmarks at level-i, for $i \geq 0$. At level-0, every node is a landmark and initially $|A_0| = n$. A subset A_1 of A_0 forms a level-1 hierarchy, a subset A_2 of A_1 forms a level-2 hierarchy, and so on. Eventually, $A_{i+1} = \emptyset$.

The radius of a landmark is the scope, in number of hops, of its advertisements. For example, a landmark with radius 1 is known to all nodes that are one-hop away, a landmark with radius 2 to nodes that are two-hops away, and so on. Eventually, the radius equals the diameter of the network, in which case a landmark is known to every node in the network. There should be at least one landmark of A_i within the radius of every landmark in A_{i-1}.

Consider the destination y of Figure 2.5 and the landmarks L_1 and L_2 such that $L_1 \in A_1$ and $L_2 \in A_2$, respectively; the name of node y is $L_2.L_1.y$. Then, a source x routes messages to destination y as follows. Since neither L_1 or y are known to node x (it is outside its scope), it will forward messages towards L_2. Intermediate relays continue to forward x's messages towards L_2, until they reach a node (not necessarily L_2) within the scope of L_1. The next hop now points towards L_1 and the process iterates. Eventually, messages will arrive at a node within the scope of destination y, at which time they are routed directly to y.

The landmark hierarchy [84, 37], as well as the area hierarchy that forms today's Internet routing system [47], follows the "dot" naming convention. Hierarchical routing schemes limit the growth of routing tables to the order of the logarithm of the number of nodes, namely $O(\log n)$, but they suffer from boundary conditions [19]. That is, two nodes that are close in the original network may have to traverse a longer path in the overlay network imposed by the hierarchy.

It is possible to construct a k-level hierarchical structure in such a way that the deviation from the optimal routes (the routing stretch) is bounded by some function of k [70].

2.5.2 Compact routing schemes

Compact routing schemes choose a set of landmarks so as to better approximate $c(x, y)$, the optimal cost of routing between any pair of nodes x and y. Compact routing schemes are willing to settle for sub-optimal routes in order to minimize the memory overhead.

Peleg and Upfal [70] have shown that there is a family of hierarchical routing protocols that can trade-off size of routing tables for stretch. While they were the first to introduce a universal routing scheme,[1] they considered only unit weights [3]. Nonetheless, they enabled research for more compact

[1]A *universal* routing scheme is a scheme that applies to **all** networks.

hierarchical schemes that can reduce the gap between the upper and lower bound for memory complexity. They have derived a lower bound on the amount of memory required and stretch based on *girth*.[1] In particular, a routing scheme cannot k-satisfy (i.e. ensure that the stretch is less than or equal to k) networks of girth greater than $2k+2$. They have shown this with a simple example: given two nodes x and y connected via a node w, the only route from x to y that k-satisfies the network is the route via w; any other route has length at least $2k+1$. Thus, approximate distance oracles return routes of stretch less than $2k+1$ [83].

The previous section has discussed how to construct a landmark hierarchy, but not the selection process for landmarks. Given a level-k hierarchy, the best stretch is always achieved when k is small. In particular, a single hierarchy suffices to find good approximations of the cost.

The best known schemes achieves stretch ≤ 3 whenever there is a single hierarchy [82, 83]. In particular, if one selects approximately $O(\sqrt{n})$ nodes as landmarks, then the stretch is bounded by 3 (as in [61]).

For a single hierarchy, compact routing schemes construct a network similar to Figure 2.6. That is, they select a set of landmarks that serve as relays for far away destinations.

In Eq. 2.3, the best next hop has been placed in an n-dimensional space. A set of landmarks with cardinality r, can represent a r-dimensional space (Figure 2.6).

Equation 2.1 changes as follows. Let L_i, $1 \leq i \leq r$, be the r-set of landmarks, breaking ties according to labels. Then, the best next hop to a destination y is a landmark, namely L^*, such that

$$c(x, L^*) + c(L^*, y) = \min_i \{c(x, L_i) + c(L_i, y)\}. \tag{2.4}$$

In essence, a landmark is an r-neighbor.

A node x selects either a landmark L closer to y or a landmark L closer to itself. This distinguishes between a name-dependent and a name-independent scheme.

[1]The girth of a network is the length of the shortest cycle in the network.

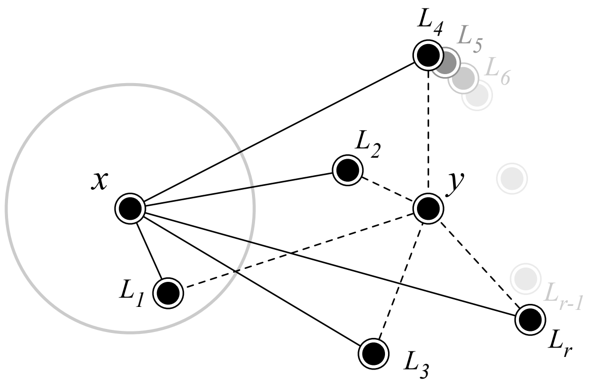


Figure 2.6: Landmarks.

A name-dependent scheme assumes that the virtual coordinates of y are known at node x. This is realized with a name service based on Distributed Hash Tables [76, 19, 61]. But because these network structures are constructed probabilistically and they are static, one may have to the re-assign labels as the network changes.

There is an equivalent name-independent approach that guarantees a worst-case stretch of 3. In [1] a network is constructed by assigning one of $\sqrt{n}$ colors, where n is the cardinality of the network, to every node such that

(i) every color-set has at most $2\sqrt{n}$ nodes; and

(ii) every node contains in its neighborhood at least one node from every color-set.

Compact routing schemes are rather complex algorithms that result in best worst-case routing for a static network. Distributed implementations are possible, but under network dynamics they require the complete topological view of the network to re-compute their data structures.

A common theme amongst compact routing schemes is the local neighborhood (or the vicinity ball) of every node in the network. Let $C(x)$ represent the neighborhood of a node x. Then, the two approaches for routing from x to y can summarized as follows. A routing scheme returns either (i) a landmark $L \in C(y)$, or (ii) a landmark $L \in C(x)$, such that L knows the shortest path to the destination.

The Small State Small Stretch (S4) protocol [61] is an application of the first approach to networks of sensors. It is a name-dependent scheme; as such it assumes a location service on top, such as the Beacon Location Service (BLS) [67].

It becomes evident that in order to efficiently implement and deploy compact routing schemes, it is required to have a method to query the location of distant destinations. This is realized by either a distributed hash table (BLS), or some other name service (DNS).

This dissertation takes a different approach. Since hierarchies impose a structure that negates self-organization, this dissertation rather assumes a flat network and uses the reactive method to query for a landmark L. That is, instead of constructing a set of landmarks, it rather queries them on-line.

In the absence of sense of direction, a query must propagate to the entire network.

2.6 The reactive method

There is no need for some source node x to know its best next hop towards all possible destinations $y \in Y$ in a network *a priori*, but possess rather the ability to query them on-line, whenever required [83].

The reactive method has been traditionally associated with flow demand. By way of example, an on-demand protocol will request a route whenever a data packet arrives in the message queue for some unknown destination.

This dissertation decouples route requests from data packet arrivals. A delay prior to the transmission of the first data packet is possible in either the proactive or the reactive method. That is, given a message queue of length Q at node x and a flow demand of f packets per second for node y, the queue will overflow after $\frac{Q}{f}$ seconds if $\Pi(x, y)$ hasn't been distributed by either a proactive protocol of y or a reactive protocol of x.

Reactive schemes establish paths in response to a query. Nodes rebroadcast this query until it arrives at a node that is either the intended destination or it holds state for the intended destination; this node will then reply over the inverse of the route that the query has followed.

An exemplar reactive scheme is the Ad hoc On-demand Distance Vector (AODV) protocol [72, 71]. AODV populates routing tables based on flow demand. Nodes that participate in the routing task (active nodes) use stateless hello messages to detect a failure, while the rest remain silent. Upon detecting a failure, an active node generates an error message to notify its predecessors that the current route is broken. The following discusses the benefits and drawbacks of the reactive method with respect to route discovery, route establishment, and the route monitoring process of the AODV protocol.

The route discovery for a x-y pair of nodes is as follows. Node x originates a route request packet if it possesses outstanding data for destination y. Route requests are broadcast packets, and their propagation is usually constrained by the diameter of the network. As a node forwards x's route request, it increments an additive metric (e.g. hop count); meanwhile, it caches the sender as the next hop to the originator x (the inverse route). Every node that retransmits a request is a possible next hop to the destination y.

Upon receipt of a request, destination y establishes the route to source x in the reverse order: it generates a reply packet for node x using the previous sender as the next hop towards x. It is possible for destination y to receive more than one request per source x. Usually, it answers the first request (that request has traversed the fastest path), and ignores the rest. Because reply packets are unicast, rather than broadcast, the cost of discovery is

$$O(n) + e$$

messages, where e represents a constant number of replies. An intermediate node w can generate a gratuitous reply for node x if it already maintains a route to destination y. This does not guarantee optimality (as in the case of landmarks), but improves the responsiveness of the algorithm.

Active nodes (i.e. successors and predecessors) assert connectivity by exchanging hello messages periodically. In AODV, these hello messages are stateless because the protocol uses the hop-count metric. As described in § 2.2.1, if a node does not receive a hello message (or a data packet) from its active neighbors for some period of time, is assumes a failure and generates an error packet. The error packet contains a list of those destinations that have been affected by the failure (a node can participate in more than one flow). Active nodes iterate the error packet until all affected sources re-establish paths to their destinations. Given the resource constraints of sensors, it is best to repair paths locally.

The AODV protocol was derived from the Dynamic Source Routing (DSR) protocol [40]. DSR, in contrast to AODV, constructs a routing label that consists of the entire path from x to y (as defined by node y's reply message).

Since the AODV protocol constructs routes on demand, routing tables scale with the number of destinations, which is independent of both the density and size of the network – a much desired property for compact routing in resource-constrained sensor networks [58]. In practice, AODV is a truly stateless protocol as it can discover a route with no routing state whatsoever.

The drawback of the AODV protocol, and subsequently of the reactive method, is that routes are static, once they are computed. More specifically, the reactive method does not provide any stretch guarantees: the length of the computed route is upper bounded by n. This is an inherent limitation of broadcast protocols (such is the discovery process), as described in the following section.

One of the contributions of this dissertation is an update process that enables stateless protocols, such as the AODV protocol, to converge to shortest paths in the presence of network dynamics.

2.7 Broadcast forwarding

Let $G(n, m)$ be an arbitrary network of n nodes and m links and consider the proactive and reactive method of route computation. Both methods rely on flooding to disseminate updates or queries, respectively. The main focus of this section is the cost of broadcast protocols.

Broadcast packets, as opposed to unicast packets, are not acknowledged. Therefore, it is difficult to ensure a broadcast packet eventually reaches its intended destination.

Dijkstra and Scholten [14] have presented a deterministic signaling scheme that uses counters to detect the termination of a diffusing computation. In their algorithm, they consider a sink (or an "environment") which sends a message (e.g. an update or a query) to its neighboring nodes and they, in turn, propagate the message to their neighbors, and so on. Signals (e.g. replies) are sent back along an incoming edge. Eventually, a node reaches a state in which it neither sends nor receives any more messages. Whenever the environment reaches this neutral state, the computation has terminated. Yet, neither the update nor the query mechanism of proactive and reactive protocols, respectively, return signals along all nodes.

It is clear that a broadcast algorithm terminates after n transmissions (every node transmits a message at least once). Thus, for broadcast protocols, only soft timers can ensure the termination of diffusing computation. For example, if a timer expires and a node has not received a reply, it will issue another request.

The cost of a broadcast protocol has been associated with the number of nodes in a network. A simple broadcast algorithm requires $O(n)$ messages. This requires a more elaborate discussion. It is not a requirement to traverse all nodes, but rather all edges. So given a network $G(n, m)$, it is m, rather than n, that determines the cost of broadcast.

The route discovery problem is equivalent to a search game where a searcher searches for a hider in a graph. In [22], it has been proved that the cost c of finding the hider is $\frac{\mu}{2} \leq c \leq \mu$, where μ is the sum of the lengths of all links $\langle x, y\rangle \in E$:

$$\mu = \sum_{\langle x,y\rangle \in E} c(x, y).$$

For unit-cost links, $\mu = m$. This is in agreement with the upper bound on the number of messages a broadcast protocol requires.

It is possible to use a local neighborhood to reduce the number of transmissions. Once again, there is a trade-off between information and communication cost. Suppose that every node knows its k-neighbors, where a k-neighborhood of x is the set of neighbors that are k-hops away from x.

For $k = 1$, Trickle timers [53, 57] use a counter-based technique to reduce the number of transmissions, based on the density of nodes in the vicinity of x, in a sensor network. They focus on eventual consistency.

In gossip-based flooding techniques [27], node x re-broadcasts a message with a probability p; and it discards it with probability $1 - p$. Gossiping techniques can be augmented with an additional parameter k, the radius of the local neighborhood: within a local neighborhood, all nodes re-transmit the broadcast with probability $p = 1$. Such gossiping techniques, although they can significantly reduce the number of transmissions, are not always successful in finding a destination.

For $k = 2$, the Optimized Link State Routing (OLSR) protocol [9] uses 2-hop neighbor information to heuristically construct a set of Multi-Point Relays (MPR). Only nodes that belong to the MPR set rebroadcast packets.

For arbitrary k-neighbors, the Zone Routing Protocol (ZRP) protocol [28, 26] uses a technique termed *bordercasting* to propagate broadcast messages (route requests) using only the border nodes of each k-neighborhood. However, when neighborhoods overlap, it might result in a larger number of messages than simple flooding [28]

The k-FLOOD algorithm [68] is a graph-theoretic framework that explores the impact of k-neighbors in flooding. First, it sets an upper bound of $O(|E|)$, the number of edges in the network. Then it states that a broadcast can be performed with at most $O(\min\{|E|, n^{1+\frac{c}{k}}\})$, for some constant c. This is because it can eliminate any redundant edges by detecting all cycles within a k-neighborhood.

The k-FLOOD algorithm introduces an important concept of this dissertation, cycles. The girth of the network is the size of the smallest cycle. The circumference of the network is the size of the longest cycle. The upper bound on girth for the k-FLOOD algorithm is $2k + 1$. In particular, the algorithm detects all cycles of length less or equal to $2k$. Then, it can elliminate one edge from each cycle (hence the reduced message overhead).

2.8 Hybrid routing schemes

The family of protocols studied in this dissertation is most similar to hybrid routing protocols proposed for wireless *ad hoc* networks. Such hybrid routing protocols usually consist of two components, a proactive and a reactive component. Hybrid routing protocols use the proactive method to improve the operations of the reactive method. A hybrid routing protocol is defined as follows.

Every destination sends an advertisement with a scope (radius) k. This means that every node that is k-hops away from a destination y will maintain a route to it proactively. The result is a tree subgraph, rooted at destination y.

The rest of the network is managed reactively. Source nodes that are outside the scope of a y's advertisements establish routes to destination y by sending query messages, as in the reactive case. As was discussed in § 2.7, the first obvious benefit of local state is a reduction on the cost of flooding. Consider, for example, the Zone Routing Protocol (ZRP) [26, 28].

Rather than broadcasting query messages to all nodes, ZRP *bordercasts* query messages to peripheral nodes, i.e., those that are at distance k from a source node x. To do so, ZRP constructs a multicast tree, rooted at the source x, of depth k. This operation is similar to the k-FLOOD algorithm (cf. § 2.7). This construction, however, requires each node to extend its local neighborhood from k to $2k-1$ hops [28]. The ZRP protocol determines the (best) value for k based solely on the number of messages proactive and reactive components generate.

The Sharp Hybrid Adaptive Routing Protocol (SHARP) [75] enables certain nodes to vary the value of k dynamically, in order to adapt to current traffic conditions (e.g. delay jitter, or packet loss). For example, heavily-used nodes (e.g. sinks) have larger values of k to efficiently distribute resource utilization. Once again, the SHARP protocol determines the (best) value of k

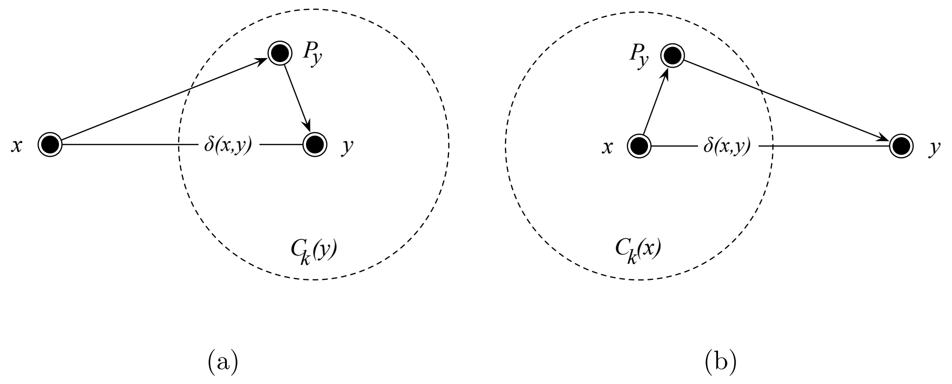


Figure 2.7: The route discovery and maintenance problem.

based on approximate estimates of the proactive and reactive message overhead.

The results for ZRP and SHARP have shown that there is a value of k such that a hybrid approach can perform at least as well (if not better) than a purely proactive or a purely reactive routing protocol. What is missing, however, is a strong theoretical argument with regards to the value of k. That is, all things considered, what are benefits (or limits) of a k-neighborhood?

This enquiry on the benefits and limits of local state (k-neighborhoods) becomes more clear when either transient or permanent failures are introduced to a routing system. Ideally, a routing algorithm should repair such failures locally [58]. Local repair has been widely associated with neighbor monitoring (cf. § 2.2.1) and, subsequently, the proactive method.

It becomes now evident that existing hybrid routing protocols, such as ZRP and SHARP, expand and contract their k-neighborhoods according to network conditions for the proactive method to always be able to repair faults locally. This is to be contrasted with the work in Chapter 3, where a hybrid route recovery method (as opposed to a purely proactive recovery method) is used to establish the lower and upper bound on the value of k.

Note the notion of a k-neighborhood is also a common theme in geographic, hierarchical, or compact routing schemes (cf. § 2.5). This work considers k-neighborhoods to be a basic networking abstraction, and the next chapter describes their properties in more detail.

In line with the above, one can reformulate the routing problem as follows (Figure 2.7).

Route discovery. *For a route from node x to node y, and given the k-neighborhood $C_k(y)$ of node y, find a node w such that $w \in C_k(y)$.* The discovery process returns a path $x \longrightarrow w \longrightarrow y$, with the path from w to y being an optimal one.

Route maintenance. *For a route from node x to node y, and given the k-neighborhood $C_k(x)$ of node x, find a node w such that $w \in C_k(x)$.* The recovery process returns a path $x \longrightarrow w \longrightarrow y$, with the path from w to y being an optimal one.

The route discovery and route recovery methods do not contradict the previous formulations of the routing problem (Equations 2.1, 2.2, 2.3, and 2.4). It enables the separation of the routing problem into discovery and recovery and confines the number of r-neighbors within a subgraph of the network (a k-neighborhood).

2.9 A principled classification of routing protocols

Figure 2.8 shows that the classification of proactive and reactive routing schemes is orthogonal to address-centric (unlabeled) and attribute-based (labeled) schemes. Although attribute-based routing schemes change the routing semantics, the underlying routing mechanisms – i.e. route discovery or route recovery – remain the same. Some of the protocols considered in Figure 2.8 are not considered for discussion in Table 2.1, and vice versa.

Table 2.1 summarizes the upper bound complexities of some previously discussed routing protocols.

The only protocols that provide worst-case stretch guarantees are purely proactive routing schemes (those that do not have a reactive component)

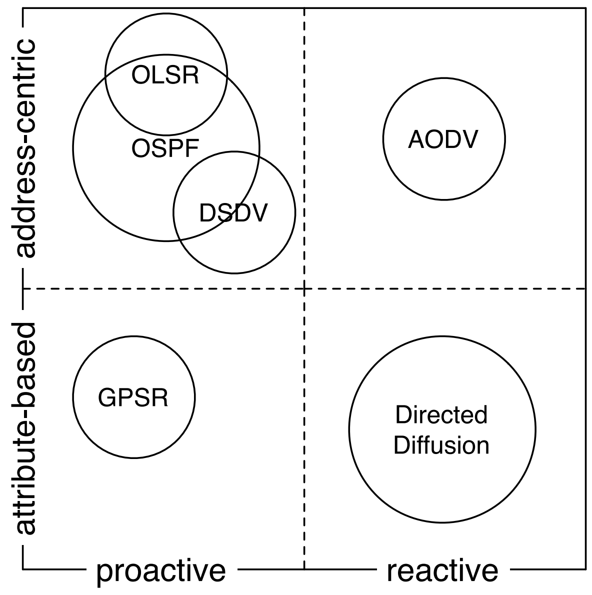


Figure 2.8: A taxonomy of wireless network routing protocols.

or compact routing schemes (those whose reactive component is based on a hierarchy of landmarks). Table 2.1 contains the examples of OSPF [63] and DSDV [73] protocols to show the differences between link-state and a distance vector routing (also see [73]); the examples of BVR [19] and S4 [61] protocols to show the difference between querying a random set of landmarks and querying a pre-computed set of landmarks in order to determine the (virtual) coordinates of a given destination; and finally, the examples of AODV [72] and ZRP [28] protocols to show that whenever an existing scheme uses the reactive method to compute a route, it provided no stretch guarantees.

The list of the routing protocols presented in Figure 2.8 and Table 2.1 is by no means complete, and it is subject to continuous refinement. The aim of this chapter was to navigate the reader through the design space and present routing mechanisms along different axes. The basic notions to take away are:

1) for a source x, the next hop w to a destination y is an r-neighbor (or a k-neighbor) of x; and

Protocol	*Stretch*	*State*	*Reactive overhead*	*Proactive overhead*
BVR	–	$O(r+d)$	–	$O(rn)$
S4	$2k+1$	$O(r)$	–	$O(rn)$
ZRP	–	$O(r+d)$	$O(dn)$	$O(rn)$
AODV	–	$O(d)$	$O(\frac{n(n-1)}{2}n)$	0
DSDV	1	$O(n)$	0	$O(n^2)$
OSPF	1	$O(n^2)$	0	$O(n^2)$

Table 2.1: Characteristics of previously discussed routing protocols. In the table, n is the number of nodes; r denotes the r-first neighbor of a node; k is the scope of an r-neighborhood; d is the number of destinations outside an r-neighborhood (respectively, k-neighborhood).

2) an algorithm can detect short cycles of length $2k$ or less within a k-neighborhood.

Chapter 3

Family ties – $\mathcal{F}(k)$

This chapter incorporates proactive and reactive schemes into a single routing framework. It describes a family $\mathcal{F}(k)$ of protocols that trades off size of routing tables and communication overhead to compute optimal (or near-optimal) routes between arbitrary source-destination pairs.

Recall from § 1.1 that a hybrid routing scheme $\mathbb{F}(r)$ uses the proactive method to find routes to the r-first neighbors, and the reactive method for the rest of the nodes. The r-first neighbors of a node x define its local neighborhood of radius k, where k is the cost from node x to its r^{th} neighbor.

The $\mathcal{F}(k)$ family is defined using a parameter k, $k \geq 0$, that controls the radius (or depth) of the local neighborhood $C_k(x)$ maintained by every node x in the network. Within this family, a hybrid routing scheme $\mathbb{F}$ consists of two basic functions (or protocols), $\textsc{Query}(x, y)$ and $\textsc{Update}(x, y)$, for the discovery and maintenance of a route from source x to destination y, respectively.

This chapter proves that the $\mathcal{F}(k)$ family finds routes between *any* source-destination pair using only local state, without using hierarchies and without rewriting addresses. In particular, it shows that $\mathcal{F}(k)$ satisfies every network G of circumference[1] $c(G) \leq 2k + 1$.

[1]The circumference $c(G)$ of a graph G is the maximum length of a cycle in G; similarly, the girth $g(G)$ of G is the minimum length of a cycle. If G is an acyclic graph (i.e. a tree), the circumference and girth are not defined and $c(G)$ in the equation above is replaced by the depth of the tree.

Then,

$$k \geq \frac{c(G) - 1}{2}.$$

It is known from § 1.1 that if $\mathbb{F}$ is proven to be optimal, then it preserves the first ordering of nodes and, subsequently, $\mathcal{F}$ is an order-preserving isomorphism between proactive and reactive schemes. The value of k will also implicitly determine the maximum number r of pre-computed routes necessary to guarantee optimality.

Three conflicting goals limit the applicability of optimal routing schemes at scale: a node should route packets over minimum-cost routes; it should keep minimal routing state; and finally, it should incur minimal communication overhead [70]. These quantities are not mutually exclusive. In particular,

1) Shortest paths matter and Shortest Path First (SPF) algorithms – either link-state or distance vector – are used exclusively to find them. For example, the capacity of a path in a wireless *ad hoc* network decreases as the distance between the source-destination pair increases [59].

2) To do so, they require to store and maintain a consistent view of the entire network locally: the size of the routing tables increases linearly at best with the number of nodes n in the network.

3) This limits scalability not just due to the memory constraints of sensor devices but also due to the number of update messages required to maintain them: upon a change in an n-sensor network (e.g. a node failure), nodes must exchange $O(n)$ messages to converge to the new routing state and recalculate their routing tables [49].

3.1 Mathematical preliminaries

This section describes in more detail the basic network model used throughout this dissertation and the different measures of routing efficiency.

3.1.1 The network model

Let $G = (V, E)$ be an undirected network with $n = |V|$ vertices (or nodes) and $m = |E|$ edges (or links). A node's address is a unique integer from 1 to n.

Every link $\langle x, y\rangle$ is assigned a positive integer weight, or *cost*. The cost function $c(\cdot, \cdot)$ denotes the cost to traverse a link. The nodes are stationary (not mobile), although link weights can either increase or decrease over time (thus, c is also a function of time). Let $c(x, y)$ be the cost between nodes x and y in the network. In an unweighted network – which is equivalent to a weighted network in which all links are assigned unit weights (i.e. $c(x, y) = 1$ for all links $\langle x, y\rangle \in E$) – this cost is measured in hops.

The following properties hold for a weighted network:

$w.1$ $c(x, y) = c(y, x)$,

$w.2$ $c(x, y) \geq 0$, and

$w.3$ $c(x, y) \leq c(x, z) + c(z, y)$ (triangle inequality).

Property $w.1$ may appear unrealistic at first, since wireless links are not bidirectional [79]. It is the method of wireless transmission however that turns property $w.1$ into a requirement. In brief, a unicast transmission from a sender x to a receiver y over a wireless link $\langle x, y\rangle$ is a complex handshake that involves messages being sent in both directions (e.g. an acknowledgment from y to x) for the transmission to be successful [21]. Hence, weights must be bidirectional. Chapter 4 describes later on different bidirectional measures of link quality.

The second property states that link weights are positive. The equality holds when $x = y$.

Property $w.3$ is the triangle inequality and it expresses the uncertainty of the outcome (the next hop z) of a routing function $\mathcal{F}$. The equality holds only when z is on a minimum-cost route from x to y.

The diameter of a weighted network is the maximum cost between any

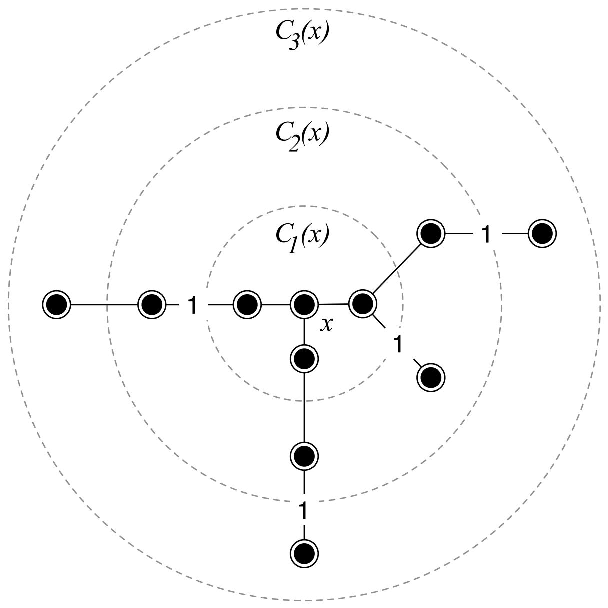


Figure 3.1: The k-neighborhoods of node x on an unweighted graph, for k=1, 2, and 3.

two nodes in the network. Formally,

$$\mathrm{Diam}(G) = \max_{x,y \in V}\{c(x,y)\}.$$

The k-neighborhood of a node x is the subset of nodes at cost k or less from it (Figure 3.1):

$$C_k(x) = \{y | c(x,y) \leq k\}.$$

The cardinality of a k-neighborhood is the value of r. The following associate the value of k with r.

The upper bound of k is the network's diameter. Therefore, $k \leq \mathrm{Diam}(G)$. A node that maintains that large a neighborhood knows the entire network topology and the resulting routing scheme is *purely proactive.* For the special case of $k = 0$, a routing scheme is *purely reactive.*

The cardinality of a k-neighborhood is bounded by n. A k-neighborhood of x with cardinality r contains the r-first neighbors of x (termed as $I_x(r)$), breaking ties by lexicographical order: node a is closer to x than b if $c(x,a) <$

$c(x,b)$, or $c(x,a) = c(x,b)$ and $a < b$. The value of k is then

$$k = \max_{y \in I_x(r)} \{c(x,y)\}.$$

The following property holds for the r-first neighbors:

k.1 (cf. Lemma 14.4.2 [69]) If w is on some shortest path from x to y and $y \in I_x(r)$, then also $y \in I_w(r)$. (Proof omitted.)

3.1.2 Quality measures

The following quality measures are of interest.

3.1.2.1 Memory overhead

The memory overhead (or memory cost) is the total number of routing table entries (or memory bytes) stored at each node by the routing algorithm to establish and maintain a route. Of interest is also the maximum memory overhead. It is desired that the memory cost is shared equally amongst sensors, as opposed to having a small subset of nodes that serve as oracles.

At a minimum, a node maintains at least one routing table entry for each destination [73]. Therefore, $\Omega(d)$, where d is the number of destinations, is a lower bound on the space complexity of any routing algorithm.[1]

A routing entry consists at a minimum of (i) the address of the destination, (ii) the length (or cost) of the shortest (least-cost) path, and (iii) the address of the next node on the route.

3.1.2.2 Message overhead

The message overhead (or message cost) is the number of messages sent by the routing algorithm to establish and maintain a route. Of interest is also the sizes of message headers: a request, a reply, or an update message carries state for at least one destination, therefore it is $O(\log n)$.

Figure 3.2 shows a simple flooding algorithm. The scope of broadcast messages is bounded by k. When k equals the diameter of the network,

[1] Or $O(d \cdot \log n)$ bits, since each address uses $O(\log n)$ bits.

```
At node $x$, do:
    send message $m$ to all neighbors – i.e. broadcast – with scope $k$.

At node $y \neq x$, upon receiving message $m$, do:
    if $m$ exists in the buffer then
        discard it.
    Else,
        1) store in buffer;
        2) if $c(x, y) < k$, rebroadcast it.
```

Figure 3.2: A simple flooding algorithm.

the message cost of the simple flooding algorithm is $O(n)$. That is, each node must transmit the message exactly once, in the worst case. This is the algorithm that is used in this chapter to disseminate message to the entire network because, although there are more efficient algorithms (cf. § 2.7), it provides an exact bound.

The message overhead is also a function of time, since route discovery and route maintenance are continuous processes in a dynamic network (but at different granularities). For example, hello messages, periodic messages sent by active nodes to assert connectivity (cf. § 2.2.1), introduce a constant overhead of $O(n)$ messages per hello time interval t.

The chosen hello interval depends on the quality of the links (and nodes): the more frequently the network changes, the more quickly such changes should be detected, otherwise the routing tables become inconsistent. In an error-free network (where nodes and links never fail), $t = \infty$.

Besides broadcast overhead (due to control messages), there is also the sub-optimal routing overhead [77]. This is discussed together with routing stretch.

3.1.2.3 Routing stretch

If the cost of transmitting a message over a link (or hop) is 1, then the cheapest cost to transmit a message from node x to node y is $c(x, y)$. Any additional transmission adds to the sub-optimal routing overhead.

One can speak of additive and multiplicative routing stretch. Given a cost estimate $d(\cdot,\cdot)$ of $c(\cdot,\cdot)$, the multiplicative stretch is $\Delta_\mu = \frac{d}{c}$. The additive stretch is $\Delta_\alpha = d - c$. For all $d, c > 0$, $\Delta_\mu \geq 1$ and $\Delta_\alpha \geq 0$. The multiplicative/aditive stretch of a routing scheme is the maximum multiplicative/additive stretch for all node pairs in the network.

The multiplicative stretch can sometimes be misleading. For example, a $\Delta_\mu = 1.5$ approximation could account for one additional hop in the route ($d = 3$ and $c = 2$) or more ($d = 15$ and $c = 10$). On the other hand, additive stretch reflects more accurately the efficiency of a path: an $(1+\Delta_\alpha)$-approximate route from x to y accounts for Δ_α extra transmissions per data packet along the route from x to y. For example, if $\Delta_\alpha = 1$ for all routes, then the sub-optimal routing overhead equals the total number of data packets transmitted: there is one extra transmission per packet. This would imply that half of the transmissions during a network's lifetime are unnecessary (sub-optimal overhead).

Routing stretch is used to compare the performance, or efficiency, of different routing schemes. The term *efficient route* can capture a wide range of desired properties (e.g. high message delivery speed or low message loss) [23]; but it usually reflects a user-defined cost, one that depends on meaning given to link weights (or link costs) at any given time. Overall, routing stretch actually captures something far more fundamental in network theory: it describes the worst case routing behavior – and, subsequently, the limits – of a particular routing algorithm when applied to a class of networks. This work focuses on all networks G. Specifically, for all $x, y \in V$ and a cost estimate $d(x, y)$ of $c(x, y)$, one of the following inequalities must hold:

a) $c(x, y) \leq d(x, y) \leq \Delta_\mu c(x, y)$, or

b) $c(x, y) \leq d(x, y) \leq \Delta_\alpha + c(x, y)$.

Otherwise, the stretch is unbounded (with the theoretical maximum being the diameter of a network, which in the worst case graph – a ring – is bounded by n).

3.1.2.4 Query and convergence time

Every message transmission incurs a delay of one time unit (or one round). It takes at least k rounds for a message to travel a distance of k hops.

The CPU time (computational complexity) is not taken into account here. Time is rather associated with message exchanges, which are more important in sensor networks since the transmission of 1 bit of data consumes the same amount of energy as the execution of approximately 1000 instructions [54].

The query and convergence times are associated with the time complexity of the $\textsc{Query}(x, y)$ and $\textsc{Update}(x, y)$ protocols, respectively. Estimating these time complexities does not imply that one can actually calculate *a priori* the delay of their execution, because both protocols are asynchronous and asynchronous executions are nondeterministic. They are, however, a measure of quality relative to k, also bounded by the diameter of the network. In general,

1) the query (or discovery) time of a path is the number of time units (or rounds) from the moment a node sends a request to the rest of the network until it receives a reply; and

2) the convergence (or recovery) time is the number of time units (or rounds) from the moment a node signals a change in its own local state until all nodes converge to their new routing state.

3.2 Route discovery

The discovery of a route from source x to destination y is based on the $\textsc{Query}(x, y)$ protocol of Figure 3.3.

Every source-destination pair is associated with two sets, the *reactive set* $\mathtt{R}(x)$ of the source x, and the *proactive set* $\mathtt{P}(y)$ of the destination y: the source x queries the former and the destination y updates the latter (Figure 3.4). For a source-destination pair $\{x, y\}$,

$$\mathtt{R}(x) = V - C_k(y) \text{ and } \mathtt{P}(y) = C_k(y).$$

```
At node $x$, do:
    If $y \in C_k(x)$ then
        return the next hop $w$ towards $y$.
    Else,
        broadcast a query message $m$ for $y$ with scope $\mathrm{Diam}(G)$.

At node $w \neq x$, upon receiving query message $m$, do:
    if $m$ exists in the buffer then
        discard it.
    If $y \in C_k(w)$ then
        generate a reply.
    Else,
        1) store in buffer;
        2) rebroadcast it.
```

Figure 3.3: The $\textsc{Query}(x, y)$ protocol.

Any decentralized algorithm armed only with local state, i.e. a k-neighborhood, can not construct all (shortest) paths. Short chains may exist, but nodes can not choose from their set of neighbors an appropriate next hop to guide messages towards a distant target [46].[1] In absence of a sense of direction, e.g. location coordinates, a protocol must perform an exhaustive search in the network, and hence it must resort to flooding.

In view of the above, the discovery overhead is not associated with the k-neighbors of the source but with those of the destination: a query message must reach at least one node $w \in C_k(y)$ to generate a reply. It is also the case that $y \notin C_k(x)$.

[1]Unless the network is a small world, in which case there is a neighbor w of x that can direct packets to distant nodes with probability $|d(x, w)|^{-r}$, for $r > 0$. Such networks can be constructed [88] or emerge [5]. In the context of this thesis, such a neighbor exists within a k-neighborhood and the routing scheme $\mathbb{F}$ attempts to find it.

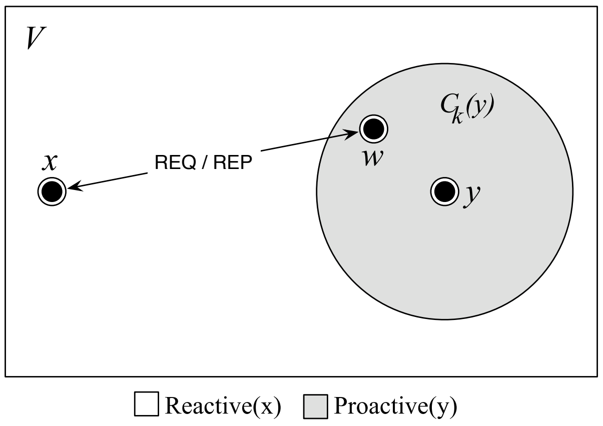


Figure 3.4: The proactive and reactive sets of a source-destination pair $\{x, y\}$.

Remark 1. *The proactive and reactive sets are complementary.*
By definitions of the proactive set of a destination y and the reactive set of a source x, for all $x, y \in V$,

$$\mathtt{R}(x) = \overline{\mathtt{P}(y)}.$$

This follows from the fact that

$$\mathtt{R}(x) \cup \mathtt{P}(y) = V,$$

and

$$\mathtt{R}(x) \cap \mathtt{P}(y) = \emptyset.$$

In more detail, the set $V - \mathtt{P}(y)$ is the subset of all nodes that are not k-neighbors of y (or, the subset of nodes that do not follow the proactive method of y). Formally,

$$V - \mathtt{P}(y) = \{x | x \in V \text{ and } x \notin \mathtt{P}(y)\}.$$

Thus,

$$\mathtt{R}(x) = V - \mathtt{P}(y) = V \cap \overline{\mathtt{P}(y)}.$$

Then,

$$\mathtt{R}(x) \cup \mathtt{P}(y) = V \cap \overline{\mathtt{P}(y)} \cup \mathtt{P}(y) = V$$

Similarly,

$$\mathtt{R}(x) \cap \mathtt{P}(y) = V \cap \overline{\mathtt{P}(y)} \cap \mathtt{P}(y) = \emptyset.$$

Thus,

$$\mathtt{R}(x) = \overline{\mathtt{P}(y)}.$$

It has been remarked that the proactive and reactive sets are complementary. This implies that the message cost of discovery for a source-destination pair is always $O(n)$, and it is independent of k. Indeed, an advertisement from destination y requires $O(r)$ messages to reach its r-first neighbors (recall that the cardinality of a k-neighborhood is r); and a route request from source x for destination y requires $O(n-r)$ messages to reach an r-neighbor of y.

The first trade-off that becomes evident at this point is on the query time: an intermediate reply from a neighbor of y improves the query time by k rounds. In general, the query time is $O(\mathrm{Diam}(G) - k)$. The following revisits the discovery process of proactive and reactive routing protocols in more detail.

3.2.1 The proactive case

For $k = \mathrm{Diam}(G)$ and a source-destination pair $\{x, y\}$,

$$\mathtt{R}(x) = \{x\} \text{ and } \mathtt{P}(y) = V.$$

A purely proactive routing scheme generates a forest of d shortest path trees, where d is the number of destinations, one for each destination. A shortest path tree $T(y)$, rooted at a destination y, has the property that the distance from y to every node x of the tree is optimal ($\Delta_m = 1$ or $\Delta_\alpha = 0$) [69]. Since the construction of each tree requires $O(n)$ messages, the total message overhead is $O(d \cdot n)$. For a complete network view (i.e., $d = n$), a purely proactive routing scheme requires $O(n^2)$ messages and can answer a query in $O(1)$ time, since $y \in C_k(x)$ by the definition of k-neighborhoods and the fact that $k = \mathrm{Diam}(G)$.

3.2.2 The reactive case

For $k = 0$ and a source-destination pair $\{x, y\}$,

$$\mathtt{R}(x) = V \text{ and } \mathtt{P}(y) = \{y\}.$$

Then, the discovery of a path from x to y requires $O(n) + e$ messages, where e is a small constant that denotes the number of reply messages sent by nodes that are on some path from x to y, not necessarily the shortest one. A fully reactive scheme does not guarantee that there exists some intermediate node to reply on behalf of y (because $k = 0$); the query time is bounded by $O(\mathrm{Diam}(G))$ (because a flooding algorithm requires $\mathrm{Diam}(G)$ time steps to forward a message to all links [69]).

In contrast to fully proactive schemes, where the message overhead is a function of the number of destinations d, the message overhead of fully reactive schemes is a function of the number of source-destination pairs. For a single source-destination pair, the message overhead is $O(n)$ and as long as the number of source-destination pairs is smaller than the number of destinations, reactive protocols are preferable. Unfortunately, there are $\frac{n(n-1)}{2}$ distinct source-destination pairs in a n-sensor network and a reactive routing scheme would require $O(n^3)$ messages to discover routes between any source-destination pair.

Because reactive schemes scale with the number of source-destination pairs (as it was experimentally shown in [30]), purely proactive schemes have been favored in sensor networks where all nodes send their measurements to a single destination (a sink): it would require $O(n^2)$ messages (a message flood for each of the $n - 1$ pairs) to discover routes to a sink reactively.[1]

[1]However, there exists an execution of the algorithm that requires $O(n)$ messages to establish $n - 1$ paths to a single destination v reactively. The execution of the algorithm is split into phases such that at phase k, the k-neighbors of v request a path to it. At phase i, the $C_i(v)$ neighbors of v send a request and $C_{i-1}(v)$ will reply, since they have already established a path at phase $i-1$. This way, every request will be transmitted once, hence the message overhead of $O(n)$. This execution however requires that all phases are spaced enough so that phase k starts after phase $k-1$ has started, since it will suppress any subsequent requests to the same destination. This would require someone to design

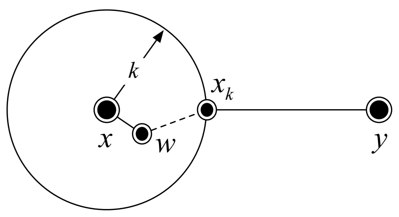


Figure 3.5: A shortest path from x to y.

3.3 Updates and failures

Since the proactive and reactive message overheads for discovery are complementary, this section turns the focus to update messages. It attempts to answer the question whether a local neighborhood can bound the update message overhead (and, subsequently, the convergence time), given a change in the network.

It is also an opportunity to establish an upper-bound on the routing stretch of the $\mathcal{F}(k)$ family of protocols. Both proactive and reactive schemes are capable of finding optimal routes, therefore a lower bound on their (multiplicative) stretch is $\Omega(1)$. However, an upper bound on the stretch of reactive protocols is unknown, whereas the stretch of proactive protocols is a well-studied problem.

A change in the network is denoted by either a cost (or weight) decrease or increase. The effect of weight decrease is positive as it strictly improves the stretch of a routing system; a weight decrease is associated with the process of convergence (of a protocol) to optimal routes. Weight increases, however, have a negative effect on stretch. In particular, upon a weight increase, stretch can only increase. When either change occurs, the node that detects it generates an update message.

For concreteness, consider the local neighborhood $C_k(x)$ of a node x and a node w that is on a shortest path to some destination y (as in Figure 3.5). Node x will change its routing table whenever one of the following hap-

such a network and program it (i.e. assign addresses) manually, or a transmission from the destination, which is the proactive case.

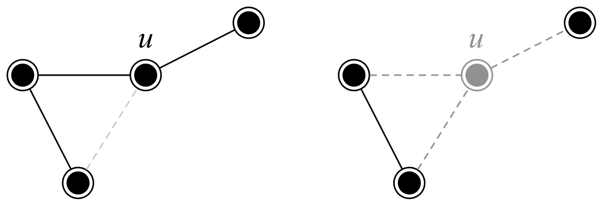


Figure 3.6: A link and a node failure in a network.

pens [38, 89]:

1) The distance $d(x, w)$ to neighbor w (that is the *best next hop* to destination y) changes by Δ_α (or by Δ_m times).

2) Node x receives an update message m from neighbor w about destination y.

The second case depends upon the action taken in the first case. For example, in case of a proactive protocol, a weight decrease or increase will cause node x to generate a new advertisement for node y; in a similar manner, node w has generated an update message because either it has detected a change or it has received an update message (a new advertisement) from its next hop, and so on. This chain can lead up to destination y, in which case consider an update message from y as a new self-advertisement.[1] If the originator of the update message is y, then $y \in C_k(x)$ (i.e. $c(x, y) \leq k$), because advertisements are bounded in scope by k. Otherwise, it is an advertisement from node w on behalf of y.

In case of a reactive protocol, on the other hand, update messages are either query, reply, or error messages. An error (a fatal failure) is denoted by an infinite weight increase. Thus, an error message is an update message carrying an infinite cost and it is considered a weight increase. In view of the above, the following focus on case 1), where node x detects a weight increase – a failure.

[1]Indeed, node y might have to send more than one advertisement for some node x to converge to an optimal route.

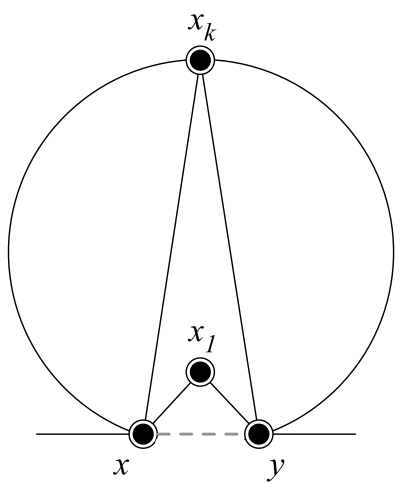


Figure 3.7: A ring graph.

There are two types of failures in a network: link and node failures (Figure 3.6) [19]. A node failure can cause more that one link failure. Link failures are difficult to characterize in wireless networks because a wireless link is a shared medium. Nonetheless, links can have zero reception, therefore links can fail. In order to bound the update overhead, failures should be repaired locally, within a k-neighborhood. This would also yield a convergence time of $O(k)$, since the propagation of an update message is now bounded by the value of k.

Consider the network of Figure 3.5. Upon a weight increase, node x can safely assume the existence of a k-neighbor x_k, for $k > 1$, along the path

$$x, w, \ldots, x_k, \cdots, y$$

that has its routing table entry to destination y intact. In particular, there must be a k-neighbor x_k on some shortest path from x to y, otherwise the network is partitioned. The problem now is to find that k-neighbor without the protocol having to resort to flooding.

Take, for example, the network of Figure 3.7 and suppose that the link from x to y fails. It is possible that there exists a direct neighbor x_1 that can recover the path or, in general, there is some k-neighbor x_k that can repair

```
At node $x$, do:
    broadcast an update message $m$ with scope $k$.

At node $w \neq x$, upon receiving update message $m$, do:
    if $x \in C(w)$ then
        store $y$ in buffer.
    If $c(x, w) + c(w, y) < c(x, y)$ then
        generate a reply message to $x$.
    Else,
        rebroadcast the update message $m$ with scope $k - 1$.
```

Figure 3.8: The UPDATE(x, y) protocol.

the path locally. There are two noteworthy observations. First, the neighbor x_k ought to exist on a cycle of length less of equal to $2k+1$. This is according to property k.1: node y should also be a neighbor of x_k. Second, the new length of the route from x to y is $2k$. Hence, the new route stretches $c(x, y)$ by $(2k - 1)$ hops.

The process that returns node x_k is the UPDATE(x, y) protocol of Figure 3.8.

3.3.1 The proactive case

In the following, $k = \text{Diam}(G)$.

Suppose that $d'(x, w) = d(x, w) + \Delta_\alpha$ and w is currently the best next hop to the destination y. If there is a neighbor $z \in C_k(x)$ such that

$$d(x, z) + d(z, y) < d(x, w) + \Delta_\alpha + d(w, y),$$

then node x must select z as its best next hop to y. This assertion however assumes that the distance $d(z, y)$ is known to node x. Thus, the distance $d(z, y)$ is either stored in memory, or it is in transit. If it is stored in memory – a routing table entry now consists of (i) the address of the destination y, (ii) the length (or cost) of the shortest (least-cost) path, (iii) the address of the next node towards y, and (iv) the length (or cost) of the shortest (least-cost) path from each neighbor to the destination y – the per node routing state is $O(r \cdot d)$, as it would require to know the distance from every k-neighbor z to each of the d possible destinations. For $d = n$, the routing state is $O(r \cdot n)$. For $r = n - 1$ (since $k = Diam(G)$), the routing state is $O(n^2)$ (see for example [63]).

If the distance $d(z, y)$ is in transit, the distance $d(z, y)$ should be known at least at node z, which means that every node $z \in V$ in the network must know its distance to all d possible destinations. For $d = n$, the result is a routing table with at least $n - 1$ entries on every node in the network (see for example [72]).

In addition to the aforementioned memory overhead, it would require $O(n)$ messages to propagate a link change to the entire network. This latter overhead can not be bounded as an update message is flooded to the entire network: every node follows the proactive method of destination y. Their convergence time is $O(\mathrm{Diam}(G))$.

The limits of local state. The limits of local state are derived from compact routing schemes [70]. It has been proven by Peleg and Upfal (Theorem 3.1 [70]) that for a routing scheme to k-satisfy[1] all n-vertex networks, it requires $\Omega(n^{1+\frac{1}{2k+4}})$ bits of memory, because there exist n-vertex graphs with girth (the length of the minimum cycle) larger than $2k+2$, and such networks cannot be k-satisfied because on a cycle of length $2k+2$, an alternative path for a route (x, w, y) has length longer than $2k$. This has spurred a number of k-hierarchical routing schemes that attempt to match this lower memory bound with an upper bound (cf. Section 2.5).[2]

[1]A routing scheme k-satisfies a network if its maximum routing stretch is less than or equal to k. That is, $\Delta_\mu \leq k$ or $\Delta_\alpha \leq k - 1$, for every $k \geq 1$.

[2]For $k = 1$, the lower bound is matched by a universal routing scheme with $O(n \log n)$ memory bits per node [23]. A universal scheme is a scheme that satisfies all networks.

3.3.2 The reactive case

In a reactive routing scheme, a node does not acquire the routing state of its neighbors proactively but, instead, it selectively queries it. In the following, the scope of update messages is bounded by k.

3.3.2.1 $k = 1$

Suppose that $k = 1$ and node x has detected an infinite weight increase (a failure) on the link to the next hop w for some destination y. Node x can not recalculate the path locally, therefore it will broadcast a new query (an update) message in search of a k-neighbor x_k on a cycle of length $2k + 1$.

Node x can not safely assume the existence of such a neighbor. What it can safely assume is that it belongs to a cycle of length $g \geq 3$, because otherwise there is no alternative route to destination y.

If $g = 3$, then the resulting subgraph is a tritree (a triangle) and there is a node z on some shortest path from x to w. It is also the case that $w \in C_1(z)$ (according to property k.1). If $g > 3$, node x can not repair the path locally (because $k = 1$).

3.3.2.2 $\mathcal{F}(k)$

There is a k-neighbor x_k that can locally repair a path from x to y if x and x_k are part of cycle of length (girth) $g \leq 2k + 1$. The resulting (additive) stretch is $\Omega(2k - 1)$ in the worst case. The stretch is less than $2k+1$ because in a worst-case network (a ring) it will *add* $2k$ hops.

3.3.2.3 $k = \mathbf{Diam}(G)$

When k is large enough to cover the entire network, an update message inevitably travels through the entire network. However, it reduces the query time considerably: there *is* a k-neighbor on a shortest path to the destination y (since every node belongs to $n - 1$ trees) and it can provide an alternate path to a destination y, otherwise the network is disconnected.

The benefits of local state. There is an unproved conjecture that there are n-vertex graphs with $\Omega(n^{1+\frac{1}{k}})$ edges that are of girth $2k+2$ and these graphs have no proper $(2k+1)$-spanner [83]:

> "The conjecture also implies that $\Omega(n^{1+\frac{1}{k}})$ are needed, in the worst case, by any oracle giving estimates of stretch smaller than $2k+1$."

This dissertation does not wish to rely on unproven theorems; it refers to this conjecture however for the following reasons. First, the approximate distance oracles of [83] are the best possible results one could obtain by building hierarchies. Second, the $\mathcal{F}(k)$ family does not build oracles per se, but it rather enables nodes to consult nearby oracles (via queries), if they exist. Finally, the $\mathcal{F}(k)$ family is a dynamic routing system that matches the complexity of the best hierarchical scheme.

Claim 1. *Given $c(G) \leq 2k+1$, $\mathcal{F}(k)$ converges to optimal routes.*

The proof is by induction on k. For $k=1$, $c(G) \leq 3$. Consider such a cycle (x, w, y) and assume that a route uses node w as the next hop from x to y. However, node x will receive an advertisement from node y (because advertisements have scope $k=1$ and the distance from x to y is 1), hence it will route its messages directly to y.

Assume that the claim is true for k. Then, for $k+1$, the circumference is

$$c(G) \leq 2k+3$$

and $\mathcal{F}(k+1)$ ought to converge to optimal routes. Assume a sub-optimal route from x to y. The scope of the advertisements from node x are $k+1$. Thus, the distance from x to y should be greater than $k+1$; otherwise, y would be one of the $(k+1)$-neighbors of x. Say that $d(x,y) \geq k+2$. In the worst case, x and y reside on the longest cycle, of length less than or equal to $2k+3$. Then, the alternative path from x to y has length at most $2k+3-(k+2)=k+1$, and node x should have received y's advertisements.

This concludes the proof.

Remark 2. *Every execution of the update method reduces the length of a route by at most $O(2k-1)$ hops.*

As node x receives an advertisement from a direct neighbor y, and nodes x and y belong on a cycle of length $2k+1$. In the worst case, the route from x to y has length $2k$.

Remark 3. *Every link failure stretches the length of a route by at most $O(2k-1)$ hops.*

As node x receives a reply from a k-neighbor of y, and nodes x and y belong on a cycle of length $2k+1$. In the worst case, a failure on the link from x to y will cause node x to discover a new route of length $2k$.

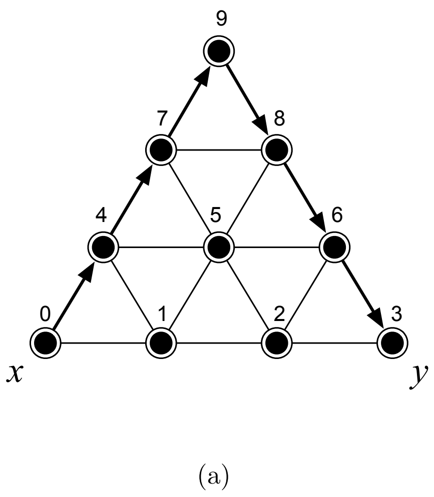
An example. Figure 3.9 shows an example of a network that $\mathcal{F}(1)$ can converge to an optimal route from node $x = 0$ to node $y = 3$.

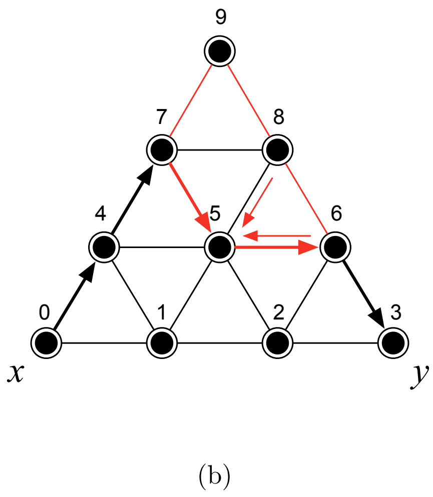
Initially, and because node y is outside the 1-neighborhood of x, the reactive method has returned a sub-optimal route of 6 hops (Fig. 3.9(a)) – this is also the longest path in the network. The following show how nodes close all cycles of length 3 using the update process.

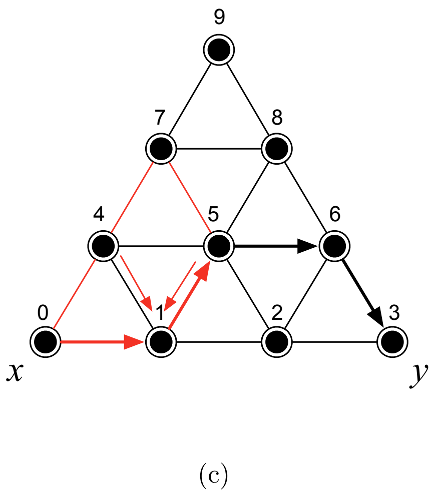
Node 5 receives update messages from both nodes 6 and 8. So, upon receipt of an advertisement from node 7 (advertising a cost of 4 hops to destination y), node 5 will reply to node 7 with a better path to y via node 6 (and of length 2). Thus, the update process further improves the path from 7 to the destination by 1 hop (Fig. 3.9(b)). The cost to route from x to y is now 5 hops.

The update process continues, with node 1 receiving update messages from nodes 4 and 5. So, when it receives an advertisement from node x about its current route to node y (of length 5), it will reply with a better path to y via node 5 (and of length 3). Thus, the new path from x to y has now length 4 hops (Fig. 3.9(c)).

The process continues (with node 1 switching to node 2 as the next hop to destination y) until eventually the protocol will converge to the shortest







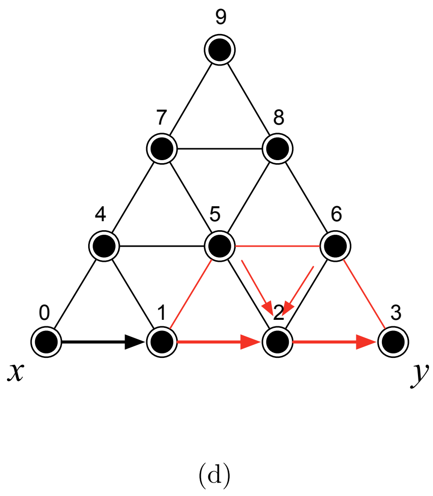


Figure 3.9: A solution of a tri-graph, for $k = 1$.

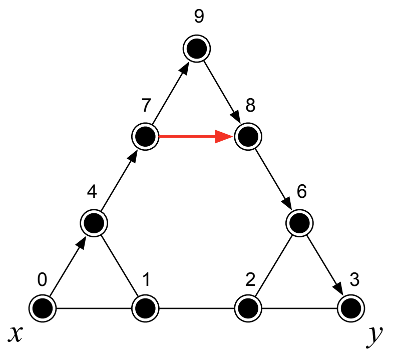


Figure 3.10: Failure to solve a network, for $k = 1$.

route of cost 3 (Fig. 3.9(d)). Notice that in every step of the process, the path length improved by 1 hop.

Figure 3.10 shows a counter-example, where $F(1)$ fails to converge to shortest paths because there is a cycle of length greater than 3. There, node 7 will switch to node 8 for a better path to destination y, yet advertisements do not travel more than 1 hops ($k = 1$). Hence, node 1 will not propagate node x's subsequent advertisements to node 2, thus it will never close the cycle $1 - 4 - 7 - 8 - 2 - 1$.

3.4 Loop freedom

In light of failures or updates, careful consideration must be given to the formation of loops.

A routing (table) loop is a result of uncoordinated propagation of updates and it is independent of the amount of routing state stored in memory – note that transient loops also appear in link-state algorithms, where nodes know the entire topology ($O(n^2)$ state).

The structure theorem by Jaffe and Moss [38] states that only a weight increase can cause a loop: a weight decrease results in a loop-free network. Unfortunately, the weights of wireless links can increase as well as decrease.

Distance vector protocols that use destination sequence numbers (i.e., DSDV [73] and AODV [72]) prevent loops by sending coordinated updates:

a weight increase for a destination y is accompanied by an increase on the destination's sequence number, making all previous entries for node y stale.

The $\mathcal{F}(k)$ protocol is loop free. The proof is according to Perkins and Royer (AODV [72]).

Intuitively, an update message from a node x_i for a destination y is also a query (a request for a better route), to which some node w will reply only if its distance to the destination y is strictly less than $c(x, y)$.

To see this, consider a routing loop of length n of the form

$$x_1, x_2, \ldots, x_n, x_1$$

that has been created on a route from x_1 to destination y. By definition, $c(x_i, y) > c(x_{i+1}, y)$, for all $1 \leq i < n$. With no loss of generality, assume the hop-count metric. Then, the cost to destination y from each node x_i is

$$c(x_i, y) = c(x_{i+1}, y) + 1,$$

thus

$$c(x_1, y) = c(x_n, y) + (n - 1). \tag{3.1}$$

But the next hop from node x_n to y is x_1, thus

$$c(x_n, y) = c(x_1, y) + 1. \tag{3.2}$$

From Equations 3.1 and 3.2, one can derive that

$$n = 0$$

and, therefore, the length of the loop is 0.

3.5 Summary

This chapter has shown that the hybrid method $\mathcal{F}(k)$ can converge to optimal routes when applied to networks of circumference $2k + 1$. This work is in position now to revisit the conjecture of § 1.1.3, that *there is a positive integer r, such that $\mathbb{F}(r)$ preserves the first ordering of nodes.*

Complexity	Reactive	Proactive
State	$O(d)$	$O(r \cdot d)$
Discovery overhead	$O(n^2)$	$O(d \cdot n)$
Update overhead	$O(n)$	$O(n)$
Routing stretch	$\Omega(1)$	$\Omega(1)$
Query time	$O(\text{Diam}(G))$	$O(1)$
Convergence time	$O(\text{Diam}(G))$	$O(\text{Diam}(G))$

Table 3.1: Summary of complexities for proactive and reactive protocols.

In § 1.1, it has been established that when a routing scheme is optimal, it preserves the first ordering of nodes; and this chapter has shown that the routing function $\mathcal{F}$ converges to optimal routes when

$$k \geq \frac{c(G) - 1}{2}.$$

Thus, the positive integer r that satisfies the aforementioned conjecture is the cardinality of the set

$$I_x(r) = \{y | c(x, y_r) < k\},$$

for all $x, y \in V$.

This chapter has contrasted the behavior of proactive and reactive protocols in terms of their discovery and recovery process. Table 3.1 summarizes the results.

The combination of proactive and reactive protocols provides exact bounds on the family $\mathcal{F}(k)$ of protocols. Clearly, it is better to ask in order to maintain minimal routing state, and since the overhead of both protocols is bounded by k, reactive protocols are preferable to approximate routes between any source-destination pair.

When the destinations are known *a priori*, proactive discovery is preferable. For arbitrary routes, reactive protocols are preferable. For example, when all sensors must send their measurements to a single sink, then proactive route discovery is preferable.

Complexity	$\Omega(\cdot)$	$O(\cdot)$
State	$r+d$	n
Discovery overhead	n^2	n^2
Update overhead	r	n
Routing stretch	1	$2k-1$
Query time	1	k
Convergence time	k	Diam(G)

Table 3.2: Summary of complexities for the $\mathcal{F}(k)$ family.

This chapter has also revisited the limits of local state. The $\mathcal{F}(k)$ family satisfies networks of circumference $2k+1$. For networks with cycles longer than $2k+2$ hops there is no proper $(2k+1)$-spanner [83], therefore either the complete view is required or a node must flood the entire network. It is of interest to note that these bounds match those of compact (i.e. hierarchical) routing schemes. Thus, even in the absence of hierarchies, the network can bound the uncertainty (sub-optimality) of route computation.

Table 3.2 summarizes the results for $\mathcal{F}(k)$. In the table, r is the cardinality of a k-neighborhood.

Finally, it is important to see how often a path will stretch by at most $2k-1$ hops. How often does a node x switches to another next hop w to reach destination y? A node will route around a link only when its currect route fails, that is $d(x,y) > d(x,w) + d(w,y)$, or when $d(x,y) > 2k+1$.

So far, the discussion has assumed arbitrary distances. It is interesting to see how to measure the distance $d(x,y)$ between nodes x and y given a set of user requirements. The next chapter explores possible ways to do so.

Chapter 4

Metric spaces

This chapter proposes a method of mapping user policy (user requirements) into discrete values (hops). Indeed, if one is routing over perfect links of unlimited capacity, then fewer hops are preferable. However, this is not always the case, since shortest paths are not always the best [13].

There are often longer paths in the network with better service guarantees that the hop-count metric cannot identify. In reality, the hop-count metric accounts for neither link nor node quality. By default, a static routing system relies on two mechanisms to deliver packets: a broadcast mechanism to assert connectivity, and a re-transmission mechanism – an Automatic Repeat Request (ARQ) – to deliver data packets over intermediate-quality links. Hence, the hop-count metric is best-effort.

A least-cost path metric associates a weight (or cost) with every link and node in the network, depending on their quality. The quality of a route has been associated with routing stretch. This chapter discusses the relation between least-cost path metrics and routing stretch. In reality, quality varies with time. Hence, sensors must take periodic measurements to estimate a link or node cost.

It is more instructive to discuss routing stretch on a packet level. A message from a source node x requires at least $d(x, y)$ transmissions to reach a target node y. But transmissions can fail – e.g. due to a lost acknowledgment – and an extra transmission should account for an extra hop in the shortest

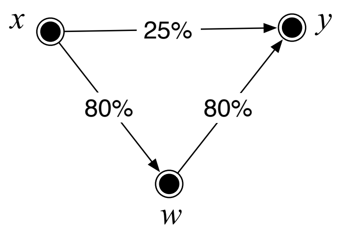


Figure 4.1: Shortest paths are not always the best.

path. The (multiplicative) routing stretch of the message is the ratio

$$\Delta = \frac{\text{Actual number of transmissions}}{d(x,y)}.$$

Figure 4.1 illustrates an example. Suppose that node x wants to send a packet to node y. There are two possible paths, the shortest path $\{x, y\}$ of length 1 and the longer path $\{x, w, y\}$ of length 2. The probability of successful delivery over the link $\langle x, y\rangle$, p_{xy}, is very low (25%). This means that, on average, path $\{x, y\}$ requires $\frac{1}{p_{xy}} = 4$ transmissions before a packet reaches node y, as opposed to $\frac{1}{p_{xw}} + \frac{1}{p_{wy}} = 2.5$ transmissions for path $\{x, w, y\}$. Hence, the routing stretch of the path $\{x, y\}$ adds a relative weight of three additional hops to the link $\langle x, y\rangle$.

In view of the above, a least-cost path metric must artificially inflate the weights of links to ensure a routing stretch of 1 on the computed paths. The key observation from the aforementioned example is to normalize link weights in terms of hops [44].

4.1 Normalizing cost to hops

Let Φ be a mapping from a metric space (V, c) to another metric space (V, d), where the cost $d(\cdot, \cdot)$ is measured in hops and it is the weight (or cost) associated with each link $\langle x, y\rangle$ of the network. Formally,

$$\Phi : c(x, y) \longrightarrow d(x, y).$$

The function Φ is continuous and depends on the quality (or utilization) $u \in [0, 1]$ of the link or of its edge nodes x and y [20]. For example, the static function $\Phi = 1$ defines the hop-count metric – it is quality-agnostic. A cost function should limit both the *magnitude* and *frequency* of link cost changes.

There are often outliers of quality – a heavy tail in the quality function – that will cause a node to oscillate between two or more paths. Furthermore, a cost change causes a node to send an update message that will flood to the entire network. This introduces an $O(n)$ overhead, which becomes prohibitive if changes are too frequent. The exact definition of such a cost function is not always known. For example, in [44], changes have been limited to half a hop. In [20], a change in utilization could inflate a link cost up to 4500. The rules for normalizing utility to hops are different in both cases (different step size). The general consensus is that as quality degrades (or utilization increases), costs should increase progressively.

The mapping used in this dissertation for experimentation is depicted in Figure 4.2 and has the following form:

$$\Phi(u) = \begin{cases} 1 & \text{if } 0 \leq u < \frac{1}{8} \\ 2 & \text{if } \frac{1}{8} \leq u < \frac{1}{4} \\ 4 & \text{if } \frac{1}{4} \leq u < \frac{1}{2} \\ 16 & \text{if } \frac{1}{2} \leq u < \frac{3}{4} \\ 64 & \text{if } \frac{3}{4} \leq u < 1 \\ 128 & \text{if } 1 \leq u < \frac{9}{8} \\ 255 & \text{if } u > \frac{9}{8} \end{cases}$$

Such a mapping represents a derivative to a (yet unknown) cost function and it is based on the work of [20]. The step sizes have changed for two intuitive reasons. First, the maximum value was set to 255 to keep path inflation relative to the size of the networks considered in this chapter. This larger value was also used to represent an infinite link cost.

Second, a cost function of the form $\frac{1}{u}$ (a similar form to transmission stretch) would be inappropriate: a cost of 2 hops for a link with utilization 50%, would misinform the routing algorithm and bias its decisions towards shortest paths that traverse links with utilization $[0, 0.5)$. Thus, additional intermediate steps were introduced to represent links with utilization $[0, 0.5)$ and $[0.5, 1.1)$.

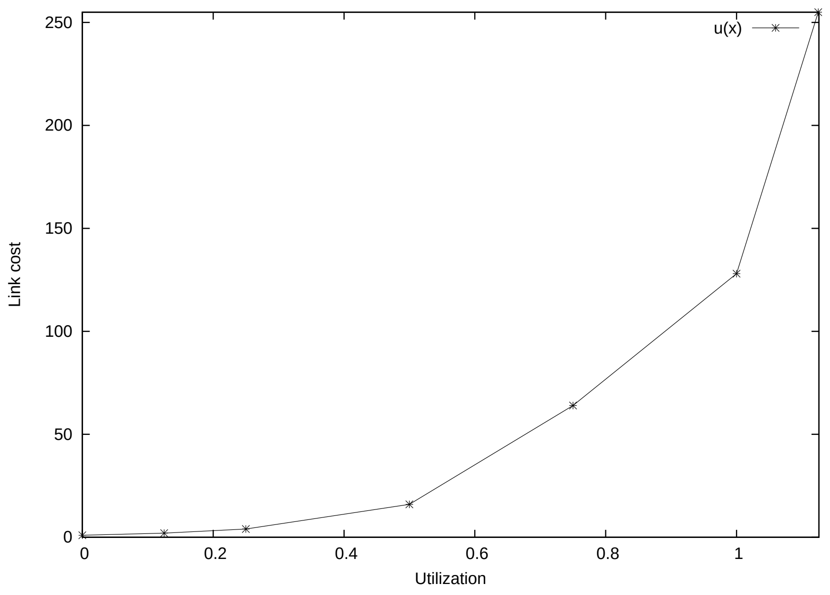


Figure 4.2: Link cost as a function of quality.

4.2 Routing metrics

Routing metrics reflect system objectives, e.g. network longevity, successful message delivery, or message delivery within delay bounds.

Routing metrics are simply estimates of quality based on periodic, active measurements. For each metric m, there is an Exponentially Weighted Moving Average (EWMA) estimator that calculates the cost to each neighbor i. At time $t_2 > t_1$, the value of $\overline{m}_i$ is

$$\overline{m}_i(t_2) = \alpha \cdot m_i(t_2) + (1 - \alpha) \cdot \overline{m}_i(t_1)$$

The value of m is normalized in the range of $[0, 1]$ and passed as an argument in the cost function: the cost of using neighbor i as the next hop to a destination at time t is then $\Phi(\overline{m}_i(t))$.

A routing algorithm operates under the assumption that the estimated path quality is a good indicator of the actual path quality **after** all nodes have converged to the best path – this way, it avoids oscillatory behavior.

This is always true in the case of static weight assignment: the algorithm, either link-state or distance vector, will eventually converge to the least-cost paths. Unfortunately, the dynamics of wireless links – asymmetry, path loss, interference, noise [79] – mean that the routing algorithm is in the process of constant optimization.

There are two types of least-cost path metrics: link and node metrics. Below is a list of desired properties that metrics ought to satisfy.

Property 1. *(Eventual delivery)* A routing should direct traffic over paths that require the least number of transmissions.

Property 2. *(Stability)* A routing should direct traffic over stable paths to avoid oscillatory behavior.

Property 3. *(Energy awareness)* A routing should direct traffic over energy-efficient paths.

Property 4. *(Load awareness)* A routing should direct traffic over paths with small utilization.

4.2.1 Link metrics

A good link metric ensures low transmission stretch: every transmission must be a useful transmission, i.e. it should move a packet closer to its intended destination, because most packets are lost due to an upper bound on re-transmissions, rather than queue overflows [33].

The mechanism to estimate link metrics is broadcast. Their distribution over time is dependent upon the packet delivery success or failure. Packets are lost due to interference from other networks, background traffic, multiple flows. Some metrics inflate to meaningless values and hence routing decisions are made on random values [12].

Therefore, link metrics must be either independent of other flows, or otherwise reflect the effects other traffic has on a desirable property (e.g. throughput).

Parameter	*Remark*
τ	Periodic time interval length
t	Timer value in range $[\frac{\tau}{2}, \tau]$
w	Estimation window size
c	Packet counter

Table 4.1: Summary of ETX parameters and variables.

4.2.1.1 Expected Transmission Count (ETX)

The ETX metric returns an estimate of the number of transmissions (and re-transmissions) that a packet requires to traverse a link [10].

Given a link $\langle x, y \rangle$ with forward success probability p and backward success probability q, the probability that a data packet is successfully transmitted and acknowledged is $p \cdot q$. Each transmission is a Bernoulli trial. Therefore, the Expected Transmission Count (ETX) for the link $\langle x, y \rangle$ is

$$ETX(x, y) = \frac{1}{p \cdot q}$$

A link estimator is the logic used to estimate p and q. Such an estimator uses a broadcast mechanism to approximate the values of p and q. For a link $\langle x, y \rangle$, the following properties hold:

1. $p_{x \to y} = q_{y \to x}$

2. $q_{x \to y} = p_{y \to x}$

The forward probability $p_{a \to b}$ is calculated at node b as follows (Figure 4.3). Node a broadcasts packets periodically, within a time interval of length τ. Its timer fires at $t \in [\frac{\tau}{2}, \tau]$ to avoid collisions – broadcast packets are neither acknowledged nor retransmitted. Node b maintains a counter c of received packets from a, which it resets to 0 after a window of w expected packets. Every time node b receives a packet from a, it increments c. The packet window w is calculated based on the packet's sequence number. The forward probability $p_{a \to b}$ is then the delivery ratio $\frac{c}{w}$.

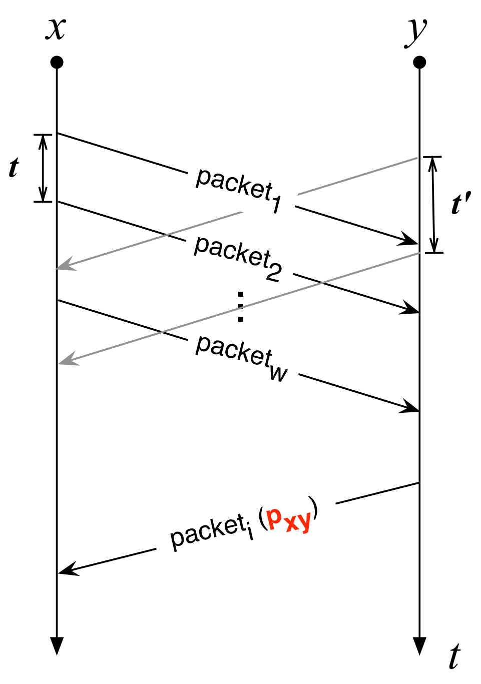


Figure 4.3: The ETX estimator.

In a similar manner, node a calculates $p_{b \to a}$, and hence $q_{a \to b}$. Node b piggybacks $p_{a \to b}$ on its broadcast packets. Therefore, node a can now calculate $ETX(\langle a, b \rangle)$. Table 4.1 summarizes the mechanism's parameters.

The ETX metric satisfies eventual message delivery because it estimates the total number of transmissions required for a packet to reach its target, only a subset of which were "useful". However, it is not an energy-aware metric, although minimizing the number of transmissions directly translates into energy savings. Furthermore, it does not measure delays – it can select congested links – therefore it not load-aware, either. The ETX mechanism is implemented using broadcasts, and broadcast transmissions can fail. By design, ETX should not oscillate. However, experiments on 802.11 networks [12] indicates large variance due to background traffic.

A variant of ETX is the Packet-Pair metric [43]. The Packet-Pair metric estimates the delay from node x to node y (and vice versa) in a similar manner to the ETX metric – actually the two mechanisms are complementary. The difference is that when the timer t fires, node x broadcasts two consecutive packets to eliminate queuing delays. If the first packet arrives at time t_1 and the second at time t_2, then the delay is $\Delta t = t_2 - t_1$. This way, the mechanism does not require clock synchronization. Node y communicates the value Δt back to x by piggybacking it onto its own broadcast packet.

Section 4.2.2.2 will later discuss a different delay metric that is load-dependent, the packet service time.

4.2.1.2 Received Signal Strength Indicator (RSSI) & Link Quality Indicator (LQI)

Prior studies have indicated that the reception probability of wireless links can vary at different time scales [80]. This means that a link estimator cannot simply classify links as good or bad. That is, a routing should avoid bad links, but not to the extent that they are not used by any active route.

Sensor radios, e.g. the Chipcon CC2420 radio component, provide two measurements of link quality: the Received Signal Strength Indicator (RSSI) and the Link Quality Indicator (LQI) – an estimate of the signal strength and the error rate for an incoming packet, respectively.

How do these two metrics correlate with the packet reception probability of a link? To answer this question a small experiment was performed on a small testbed (cf. Figure 4.7 in § 4.4).

For this experiment, every sensor broadcasts 200 packets to every other node in the network with a transmission power of -25dBm. Receivers log the RSSI and LQI values. Upon sending 200 packets, another node takes over to avoid losses due to collisions. The experiment cycles through all nodes and was repeated three times. Figure 4.4 plots the packet reception probability of all the links in the experimental testbed against the average RSSI and LQI values.

The small variance of RSSI indicates that when this metric is used, routes will be stable. However, Figure 4.4(a) shows a “grey” region [81] in the range of RSSI values (from -95dBm to -85dBm) where the packet reception probability varies greatly while the received signals have comparable strength.

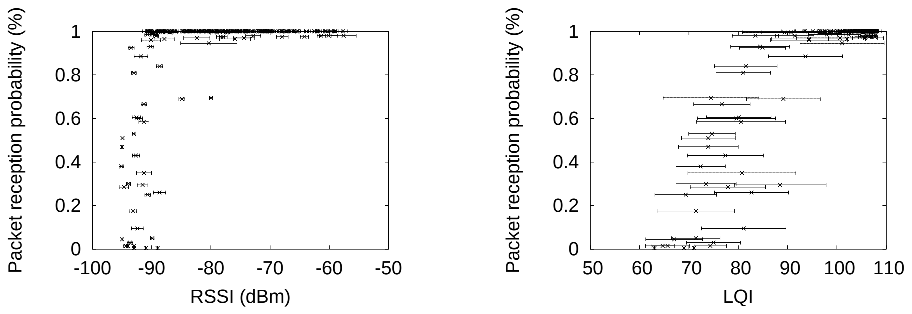


Figure 4.4: RSSI and LQI are not enough.

Recent research in wireless metric spaces (e.g. [80, 79]) have also identified this behavior. As regards the LQI metric, the large variance of its values (Figure 4.4(b)) indicates that this metric will also not determine accurately the quality of a link.

The main disadvantage of the RSSI and LQI metrics is packet bias: they rely only on successfully received packets. This way, a link will always appear good, even after long periods of disconnection. It cannot guarantee eventual delivery. Furthermore, selecting only good paths will create hot-spots in the network – even partitions.

4.2.2 Node metrics

Node metrics quantify node attributes, such as remaining energy or queue occupancy. As such, they can express the willingness or ability of a node to participate in the routing task.

Static node weights are equivalent to roles: an administrator can assign node weights so as to direct traffic towards aggregators or, on the contrary, to direct traffic away from vulnerable nodes.

Static weight assignment is an off-line task which is beyond the scope of this dissertation. The following considers only dynamic node metrics that require on-line estimators.

4.2.2.1 Energy budgets

A sensor that runs low on energy should gracefully degrade its service as a router. This way, a router should shed traffic to alternate, possibly longer, paths and evenly distribute the available energy budget. Of course, certain pathologies of the network (e.g. a single router that bridges two regions) will eventually partition it when its energy supply is exhausted.

The residual energy of a node – unless it scavenges energy – is an ever decreasing measure from E_0, the initial node's energy budget, to zero. An energy measurement can reflect either the percentage of the energy budget spent since start-up, or the remaining energy budget. The physical layer of most sensor platforms provides the latter.

The MSP430 microprocessor has an internal voltage sensor that enables one to monitor the battery voltage at discharge. Hardware components operate within a supply voltage range (typically, from 2.1 to 3.6V). Figure 4.5 shows an example of a discharge curve over time taken from a TMote Sky device.

There is a correlation between the current battery voltage and the capacity left in the battery. Therefore, it suffices to measure the voltage V_t at time t as an indicator of the remaining energy. The energy metric at time t is

$$e(t) = 1 - \frac{V_t - V_{min}}{V_{max} - V_{min}}$$

with an initial energy budget $E_0 = 1$ (thus, at $t = 0$, $e(0) = 0$).

The energy metric is highly unstable – every operation is costly, even the energy measurement itself. However, because it is ever decreasing, it does not oscillate on random values.

Similar to hop-count, the energy metric is useful only if all links are equal. It does not take into account link quality therefore it does not guarantee delivery. Another drawback of the energy metric is that it may shed traffic to links that already experience congestion – it is load-agnostic.

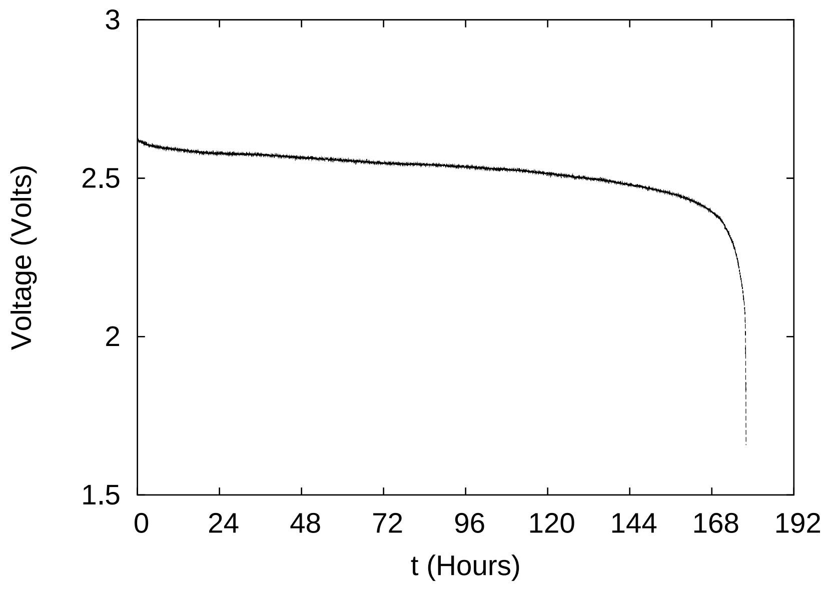


Figure 4.5: Battery discharge of a TMote Sky during the course of a week.

4.2.2.2 Congestion degree

Wang et al. [87] proposed a congestion index for rate limiting in the presence of congestion.

A sensor should send packets over links with a small utilization. That is, a routing should divert some flows to alternate, possibly longer, paths to mitigate congestion and evenly distribute the available bandwidth.

A node detects congestion by monitoring the build-up of its forwarding queue. Let t_a be the average packet inter-arrival time and t_s the average packet service time. The congestion degree $d(i)$ at node i is

$$d(i) = \frac{t_s^i}{t_a^i}$$

If $d > 1$, that is $t_s > t_a$, then a node experiences congestion.

The packet inter-arrival time is calculated at node i as follows. A node starts a timer t when the first data packet arrives at the queue, which it resets to 0 after a window of w packets. Hence, upon arrival of the w^{th} packet, the average inter-arrival time is $\frac{t}{w}$.

The packet service time is calculated on a per packet basis. It measures the time a packet "lives" in the queue until it has been successfully acknowledged by the receiver or dropped after a number of retries.

A single bit suffices to signal congestion [16, 33, 86]: when a node receives (or overhears) a packet from its next hop with the congestion bit set, it stops forwarding packets to avoid overflows at the next hop's queue. However, this can lead to a backlog at the node itself, *ad infinitum.*

In contrast to these rate limiting schemes, the congestion degree metric is continuous and ranges from $[0, 1]$. It uses seven more bits to signal its current congestion level that will cause its neighbors to switch to another, less congested, successor if necessary. For example, consider the scenario where, upon an event (e.g. a volcanic eruption or a fire alarm), a sensor is instructed to limit its transmission rate (successor sets the congestion bit) exactly at the moment when packets are needed the most.

Buffer drops are not as frequent as re-transmission drops. An increase in the service time implicitly accounts for poor delivery. However, the congestion degree metric cannot distinguish delays due to congestion or poor quality. The congestion degree will rise upon packet arrivals and fall as packets are serviced. Therefore, it is not stable.

4.3 k-neighborhoods revisited

The notion of k-neighborhoods (Section 3.2) also applies to weighted graphs. Figure 4.6 illustrates an example. By normalizing the cost to hops, direct neighbors of node u now appear in its $k > 1$ neighborhoods. Thus, the cardinality of a k-neighborhood decreases.

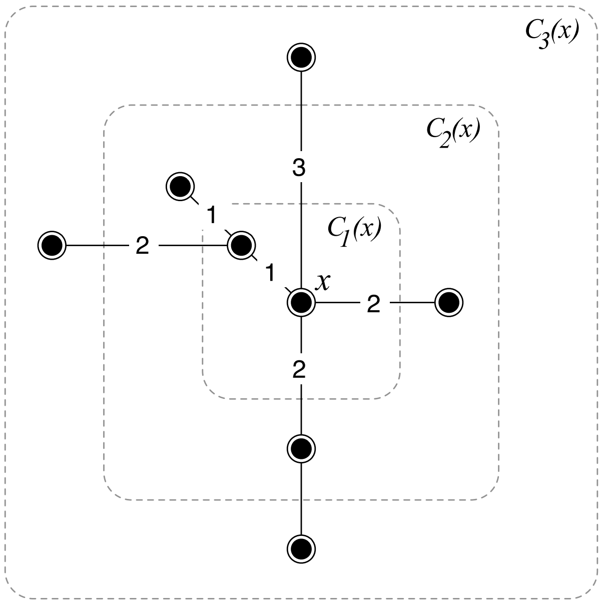


Figure 4.6: The k-neighborhoods of node u on a weighted graph, for k=1, 2, and 3.

4.4 Experimental design

The following set of simulations and experiments aim to evaluate what impact routing metrics have on the efficiency of routing. The $\mathcal{F}(1)$ protocol suffices to establish a metric of choice. Therefore the evaluation has been done in terms of the Collection Tree Protocol (CTP) [24], with certain modifications.

The metrics under consideration for the experiments are:

- ETX (`etx`),
- congestion degree (`cgd`),
- service time (`del`),
- energy left (`energy`), and
- hop-count (`hop`).

For the experiments, results are also shown for the default implementation of the ETX metric used in the TinyOS networking stack, and in particular by the Collection Tree Protocol (CTP) [24] (labelled as `etx-ctp`). All other metrics are normalized to hops.

Results on the energy metric are derived from simulations on random topologies of 49 nodes. The results are averaged over 5 runs.

4.4.1 Workload model

The simulations consider a periodic workload model, typical of many environmental monitoring systems [33]. Sensors generate readings at fixed time intervals, $f_{data} = \{0.05, 0.1, 0.25, 0.5\}$ packets/sec. Each sensor sources, but also relays, packets towards one sink.

4.4.2 Performance measures

Chapter 1 introduced three user objectives: (i) network longevity, (ii) successful message delivery, and (iii) message delivery within delay bounds. These objectives are discussed as regards to the following end-to-end performance measures: distribution of traffic; packet loss and, its complement, delivery ratio; and hop-by-hop latency. Finally, the experiments track the *stretch* of paths, or its equivalent, *path efficiency*. Path efficiency is the fraction of transmissions that have been useful (i.e. transmissions that have moved a message closer to the sink) out of the total number of transmissions. In a similar manner, node efficiency is the fraction of transmissions of a node that are successful.

4.4.3 An experimental testbed

For the experiments, a small sensor network of TMoteSky devices has been deployed on the 5^{th} floor of the Department of Computing Science at the University of Glasgow (also refered to as the Level-5 testbed). Nodes have been placed approximately 5 meters apart. Figure 4.7 shows the topology of the network. All nodes transmit their measurements to node 0. Table 4.2 enumerates the configuration parameters for the system.

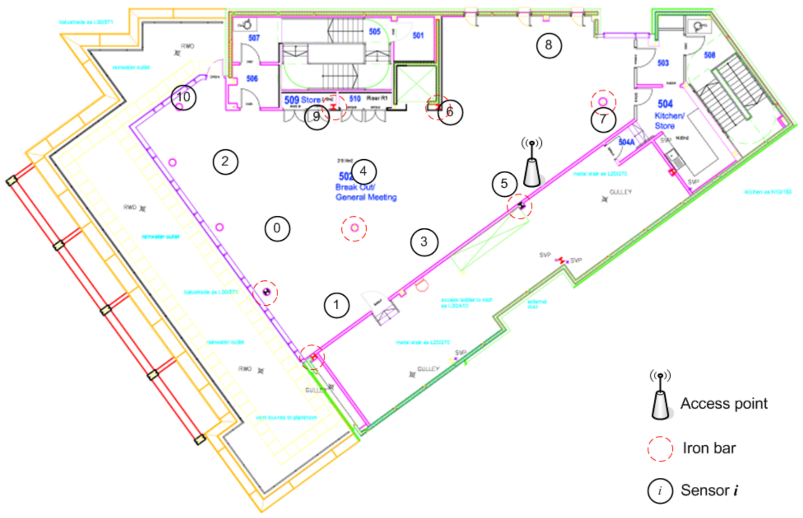


Figure 4.7: The Level-5 testbed.

Parameter	*Value*
Buffer size	13
Tx attempts	30
Packet size	36 bytes (+22 bytes for debugging)
Tx Power	-25dBm
Data rate	0.1 packets/sec

Table 4.2: Configuration parameters for the experiments.

4.5 Analysis

Figure 4.8 shows the distribution of traffic across all nodes, using an energy metric and a hop-count metric. The energy metric, since it accounts for the energy left at a node in the network, oscillates amongst alternative routes: every transmission is expensive. The hop-count metric is energy-agnostic thus it will deplete a node's energy before switching to a different path. Using an energy metric, besides conserving energy, also achieves load-balancing. This agrees with the intuition that an energy metric can satisfy more than one of the desired properties. What if load is increased? Figure 4.9 shows path

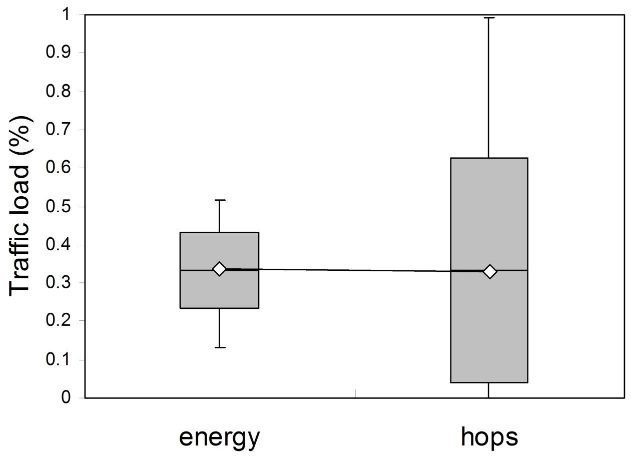


Figure 4.8: Energy metric.

efficiency as the per-node offered load increases from 0.05 packets per second to 0.5 packets per second. Efficiency decreases as the offered load increases. Notice however that efficiency decreases substantially when a load-agnostic metric, such as hop-count, is used for higher loads.

In the following experiments, very few packets were lost (Figure 4.10). This is due to choosing a high re-transmission count (see Table 4.2). It was intended to ensure eventual delivery, to enable the comparison of the performance of different metrics. The hop-count metric incurs the smallest delay. This comes at the cost of lost packets. That is, the successful packets are delivered quickly, but not all packets are delivered. This is best illustrated in Figure 4.11. There, service time and ETX are the best metrics as they inflate the path length yet they do it so as to avoid the sudden increase in latency, as observed with hop-count or congestion degree. (The latter is due to oscillations.) Table 4.3 show the percentage of traffic that has traversed paths of length 1 to 5.

Figure 4.12(a) and 4.12(b) show the hop stretch and efficiency of the computed set of paths in Level-5. The hop-count metric does indeed return the "shortest" paths, this however does not mean that shortest paths are the best. Indeed, the path and node efficiency of the hop-count metric is the worst amongst all of the metrics (30%). The rest of the metrics have

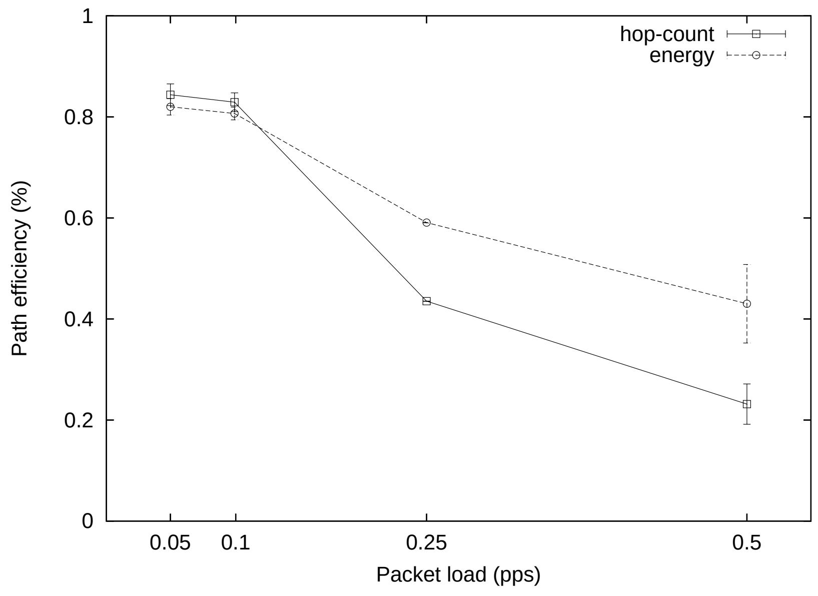


Figure 4.9: Path efficiency as a function of load.

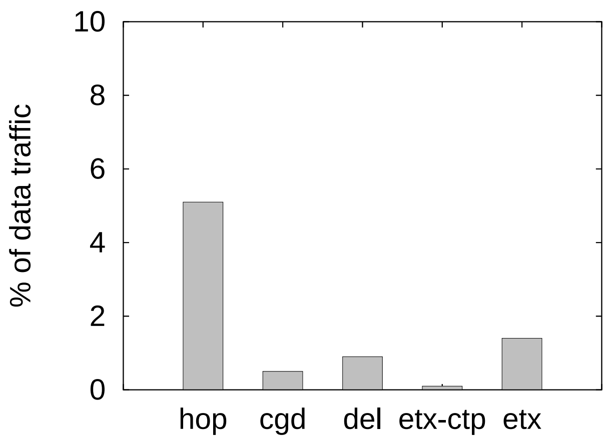


Figure 4.10: Packet loss.

	% of traffic				
length	1	2	3	4	5
hop	46.5	**50.6**	2.7	0.2	0
cgd	39.6	**60.2**	0.2	0	0
del	20.25	**31.3**	28.15	20	0.3
etx-ctp	26	**44**	24	6	0
etx	12.9	**31.1**	25.5	15.4	15.1

Table 4.3: Distribution of traffic across different path lengths.

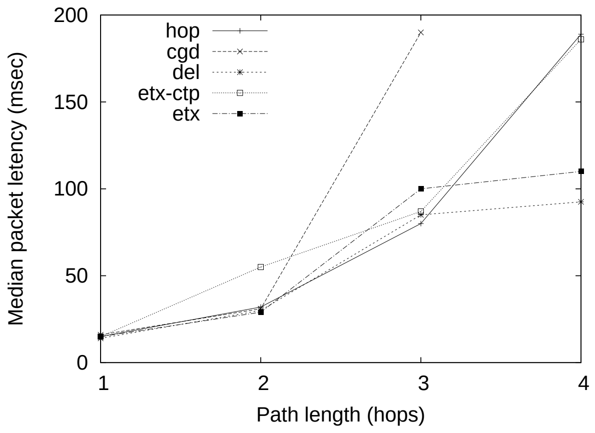


Figure 4.11: Delay per hop.

increased the length of the path (e.g. the delay metric returns paths of average length 1.5), yet they are 100% more efficient than the hop-count metric. The best efficiency is achieved by ETX, a metric that ensures that the number of transmissions is low; interestingly, it is the one metric with the highest hop stretch.

This implies that one cannot decouple routing metrics from stretch (performance); the two must be considered in unison. One cannot decouple, for example, the vagaries of wireless links, as measured by RSSI, from congestion since they are entangled in unknown ways.

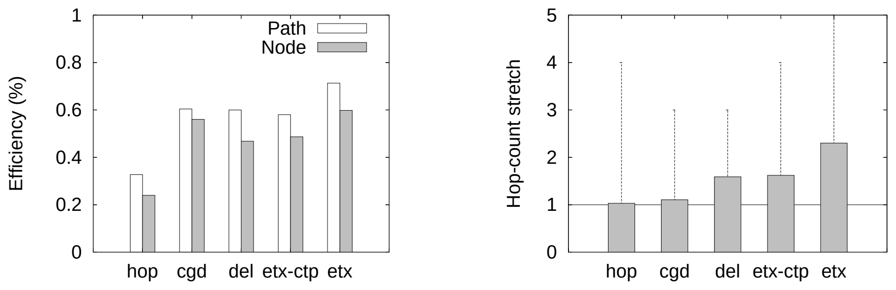


Figure 4.12: Hop-count stretch and the efficiency of paths at Level-5 for different metrics.

This also enforces a more intuitive argument, that the wireless metric space (and subsequently, the optimal metric space for wireless networks of sensors) is not yet fully understood. Take, for example, geographic routing: is the wireless metric space isomorphic to the Cartesian space?

4.6 Summary

This chapter redefined the notion of distance $d(x, y)$ between a pair x-y of nodes. It has done so by introducing a cost function Φ. The key is to normalize link costs to hops: if a link has cost c, an alternate path of $c - 1$ hops should be preferred.

Table 4.4 summarizes the link and node metrics discussed in this chapter and their properties. A "*" in the cell indicates that a metric satisfies a property implicitly. In particular:

ETX A routing should direct traffic over paths that require the least number of transmissions.

RSSI A routing should avoid bad links, but not to the extent that they are not used by any active route.

Remaining energy A routing should shed traffic to alternate, possibly longer, paths to evenly distribute the available energy budget.

Property	ETX	Delay	RSSI	Energy	Cong. Degree
Eventual delivery	Yes	–	No	No	No*
Stability	No	–	Yes	No*	No
Energy awareness	No*	–	No	Yes	No
Load awareness	No	–	No	No	Yes

Table 4.4: Summary of the properties of link and node metrics.

Congestion degree A routing should divert some flows to alternate, possibly longer, paths to mitigate congestion.

Because each metric is associated with a property, it opens the design space to explore combinations of link and node metrics.

A cost function is a map $\Phi : c(x, y) \rightarrow d(x, y)$. It changes the distance semantics, but it does not invalidate the notion of k-neighborhoods. This allowed us to explore the impact of k-neighborhoods on the route estimation and selection process.

This chapter has also evaluated routing metrics in the context of many-to-one traffic. First, as a proof of concept, it has shown that $F(1)$ can address many-to-one traffic patterns (this is similar to protocols that exist in the literature, such as CTP [24]). For $k = 1$, different link and node metrics have been evaluated. For the case of link metrics, the ETX metric appeared to be the best. This result is not surprising. However, it has also been shown that node metrics have a place in the metric space. In particular, this chapter has evaluated an energy-aware and a congestion-aware metric; it has been shown that node metrics can achieve load-balancing. This is true when all links are equal (i.e., when quality is perfect), and it is also true when quality varies with time.

When a different distance function is applied to graphs, the size of a k-neighborhood decreases. This shows that state can further be reduced, and as a result the proactive overhead as well, which is a limiting factor in the case of proactive routing schemes.

Chapter 5

Design apparatus

This chapter describes in more detail the design and implementation of the $\mathbb{F}$ routing scheme within the TinyOS operating system [31]. It provides a functional summary of the $\mathcal{F}(k)$ routing protocol in terms of its operations on basic protocol data structures.

It is important to stress that this dissertation never intended to introduce yet another routing protocol. Given the sheer number of routing protocols for wireless *ad hoc* networks available in the literature, there is no requirement for another point in the design space. On the contrary, the purpose of the dissertation is to navigate in this design space.

5.1 Protocol data structures

Every node (either a source x, a next hop w, or a destination y) is assigned a unique 16-bit address. The address with value `0xffff` is reserved for broadcast. A message consists of a header (that contains the destination address), a payload (which is populated by the protocol's data structures), and a footer (also followed by certain metadata, e.g. the received signal strength).

A destination y corresponds to exactly one entry in the routing table of x. Table 5.1 shows the structure of a routing table entry. The purpose of the option bits is mainly to distinguish between nearby and far-away destinations. The routing table is populated (or updated) upon receipt of an advertisement (from a nearby destination) or a reply (from a far-away destination).

Field	*Size* (Bytes)
Destination's address, y	2
Address of the next hop w toward y	2
Cost (e.g. number of hops) to y, $c(x, y)$	2
Option bits	1

Table 5.1: A routing table entry at a node x.

Field	*Size* (Bytes)
Destination's address, y	2
Originator's address, x	2
Cost (e.g. number of hops) to y, $c(x, y)$	2
Originator's sequence number	2

Table 5.2: A route request and reply packet.

In general, there are three types of messages: queries (or requests), replies, and advertisements. Requests and replies are similar in nature (see Table 5.2). A request is for an unknown destination y, thus initially the cost is set to *infinity* and it is sent to the broadcast address.

Advertisements contain the minimum of a single routing table entry (see Table 5.3).[1] This design choice was made to emphasize the following two assumptions that motivated this work:

a1) a routing table entry exists either in transit or in memory; and

a2) an advertisement is also a route request for a known destination.

[1]A message can carry more than one table entry. For example, the maximum payload size of a CC2420 data frame is 128 bytes; hence it can contain a vector (a list) of approximately 20 destinations.

Field	*Size* (Bytes)
Originator's address, x	2
Scope of message, k	1
Routing table entry	7

Table 5.3: An advertisement.

5.2 Functional overview

Every node in the network runs a copy of the routing function

$$\textsc{Update}(\text{x,y}) \text{ or } \textsc{Query}(x, y).$$

At start-up, each node initializes the aforementioned protocol data structures. Once the radio is functional, nodes start to self-advertise (an advertisement with initial cost 0). Each advertisement is retransmitted k-times, where k is the additive cost from the originator of the advertisement to the current recipient node. This way, nodes discover routes to their nearby destinations, namely their k-neighbors. For far-away destinations, nodes send requests.

5.2.1 k-neighborhoods

As mentioned in Chapter 2, the basic level of abstraction for state compactness is a k-neighborhood. Currently, the design choice is to allow a node to belong in more than one neighborhood. The effect of such multiple neighborhood membership is open to two opposing interpretations.

On the negative side, in a flat routing system there can be a significant number of unused entries in the routing table of each node (also referred to as *noise* in this dissertation). Whereas, in area or landmark hierarchies (see [63] and [84], respectively) there is a representative node from each k-neighborhood. The problem is then to find a minimum set of nodes that covers the entire network (a sparse cover). Finding such a set (a set such that all nodes are k-hops away from at least one representative node) is an

NP-complete problem [2] and nodes usually run heuristics to approximate a solution.

Here, the approach is different. Rather than selecting (or electing) a subset of nodes as representatives (using the proactive method), each node could search for them (using the reactive method, instead). Or, a node could apply the method of successive approximations (as discussed in § 2.2.2), to coarsen the representation of its k-neighborhood. Thus, on the positive side there is selection: since nodes search for a sparse cover, they have a number of alternative routes to choose from.

The construction of a k-neighborhood is essential to provide routing stretch guarantees, not only for nearby destinations (those within scope k) but also, as it is later shown, for far-away destinations. There are two ways to maintain a k-neighborhood. Either using scoped distance vectors (where routing state is sent to the broadcast address) or using a database exchange method (where state is sent to a direct neighbor). This dissertation leaves open the question whether nodes should synchronize their routing tables using broadcast or unicast messages (and revisits it in Chapter 7). For now, the implementation uses scoped (by k) distance vectors.

The algorithm for scoped distance vectors is simple, yet robust: every node y sends advertisements periodically. Upon receipt of an advertisement, node x stores y in its routing table. If the cost to y is less than k, node x rebroadcasts the message; otherwise, it suppresses it. Both transmissions and retransmissions are randomized, at different time granularities (with re-transmissions being the most frequent). Figure 5.1 shows an example construction of a k-neighborhood for some node x. Node x receives advertisements from both y and z, but only z's advertisement is retransmitted to node v.

The next section discusses the route discovery and route establishment process of the protocol for far-away destinations.

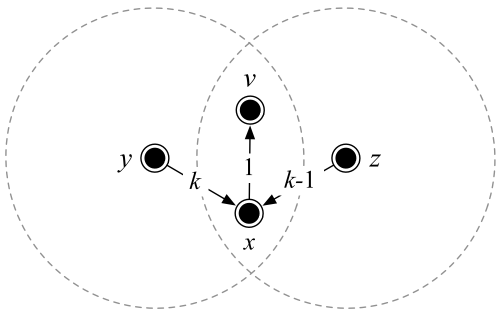


Figure 5.1: Soliciting the k-neighbors of node x.

5.2.2 Beyond k-neighbors

This section discusses the discovery and maintenance of routes to nodes beyond k-neighbors. This is achieved with route requests. Here is a reminder of the reactive method.

A route request is sent only if a valid route to the intended destination does not exist. There can be more that one outstanding request at a time, identified by a sequence number. Route requests are broadcast packets, constrained by a Time-To-Live (TTL) field that limits their scope by the maximum path length (the diameter) of the network.[1] All intermediate nodes, unless they are (or they know about) the intended destination, increment an additive metric (e.g. hop count) and rebroadcast the request; meanwhile they cache the sender as the next hop to the originator.

Whenever a request for a destination y arrives either at node y or at a k-neighbor of y, a reply message is sent to the originator of the request using the previous sender as the next hop towards the originator. Multiple routes can be reinforced by different k-neighbors of y. In turn, the originator can discard, use, or store these paths, based on the routing strategy it employs (e.g. single-path or multi-path routing).

[1]The TTL is a user-specified parameter, always available in off-the-shelf implementations of routing protocols [65, 71]. If the diameter is unknown, TTL is set to *infinity*.

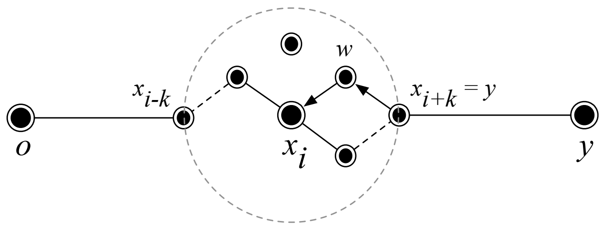


Figure 5.2: Maintaining a route within a k-neighborhood of node x.

The mechanism for route discovery is a best-effort flooding algorithm and does not necessarily return optimal routes. Any guarantees on the optimality (or uncertainty) of routes are provided by the maintenance mechanism. Once a route is established (via a reply) between an originator o and a destination y, nodes along that route send advertisements for node y. Figure 5.2 illustrates an example: node x_i will use w as the next hop towards a destination y.

5.3 Implementation overview

The $\mathcal{F}(k)$ networking stack has been implemented within the TinyOS operating system [31], version 2.

TinyOS programs are written in nesC (network embedded system C), a component-based C variant (see [55]). Each component declares the functions it provides (or implements) and the functions it uses (or calls), together with any memory space it requires (e.g. a routing table). A TinyOS program consists of a collection of components.

Each component has three computational abstractions, commands, events, and tasks. For example, consider the following three code snippets:

a) `command error_t f.findRoute(uint16_t y);`

b) `event void f.routeFound(uint16_t y, uint16_t w);`

c) `task void forwardTraffic();`

Suppose now that a sensor networking application wants to find the (best) next hop to some destination `y`. Then, it will call the `findRoute` function (which is provided by a component `f`) as follows:

a) `call f.findRoute(y);`

In turn, component `f` may use additional components (e.g. a radio component to send a request message, in the reactive case) to find a route to destination `y`. Once a next hop `w` is found, component `f` can signal the application of its success:

b) `signal f.routeFound(y, w);`

The application component specifies any tasks to perform upon finding a route:

c) `event void x.routeFound(...)` { `post forwardTraffic();` }

Tasks are executed by a non-preemptive FIFO scheduler. Apart from the scheduler, other essential components (those necessary to instantiate a networking application in TinyOS) include timers and platform abstractions.

Timers are mainly used to schedule events in the future, e.g. time-outs for stale routing state (cf. § 2.2.1):

d) `call timer.next(hello_interval * allowed_loss);`

In TinyOS, hardware is also represented as a collection of components. Certain components are drivers for specific chips. For example, the TMoteSky platform has a MSP430 microcontroller (MCU) component and a CC2420 radio component. Other components abstract basic hardware functionality. For example, a radio component can send and receive packets.

The basic networking abstraction in TinyOS is an active message [54, 56]. An application can use one or more active message senders and receivers, depending on the types of packets it requires (e.g. data messages, query or reply messages, and route advertisements). Each sender component will receive a fair share of transmission opportunities. Active messages provide the basis upon which networking components are built.

By default, TinyOS ships with the following four networking components:

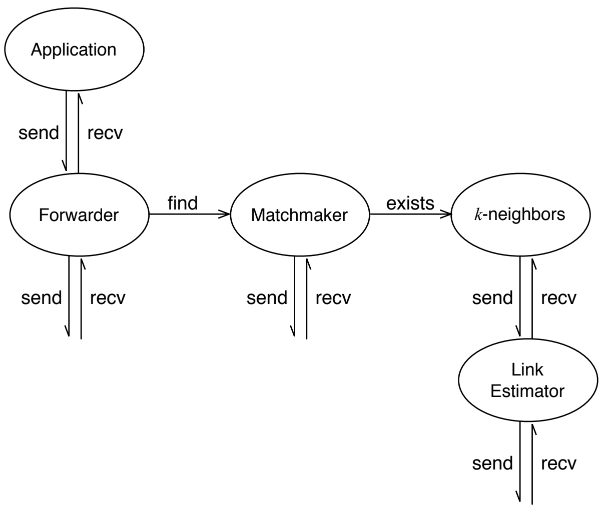


Figure 5.3: The $\mathcal{F}(k)$ blueprint.

a) a forwarder, that sends (receives) unicast packets, e.g. data messages, to (from) the next (previous) hop to a sink;

b) the Collection Tree Protocol (CTP) [24], a proactive distance-vector routing protocol that computes routes to a single destination, the sink.

c) a link estimator, that implements a variant of the ETX mechanism (cf. § 4.2.1.1) [18]; and finally

d) a Carrier Sense Multiple Access (CSMA) Medium Access Control (MAC) protocol.

5.4 The $\mathcal{F}(k)$ networking stack

Figure 5.3 shows a high-level overview of the $\mathcal{F}(k)$ networking stack.

The *application* component calls one or more sensor components periodically. Sensors signal their measured data (e.g. temperature) back to the application, where data are organized into packets. The application sends each packet to a specific destination via the forwarder.

Upon receipt of a packet, the *forwarder* component calls the routing protocol for the next hop to the given destination. Packets are queued for transmission until a route to that destination is found. When a route is known (a next hop), the forwarder attempts to transmit the packet to the next hop, until either it receives an acknowledgment or the maximum number of retransmissions is reached.

The $\mathcal{F}(k)$ routing protocol is realized by the *matchmaker* and *k-neighbors* components that implement the reactive and proactive method, respectively, and a *link estimator* that implements a cost function based on a set of predetermined metrics (cf. Chapter 4).

Upon request (a call to the `find` function by the forwarder), the matchmaker first searches for the intended destination locally by calling the `exists` function of the k-neighbors component. If it does not exist, then it broadcasts a query message. The k-neighbors component maintains a table of nodes that are at cost k or less from it. The cost to each neighbor is determined by the link estimator component.

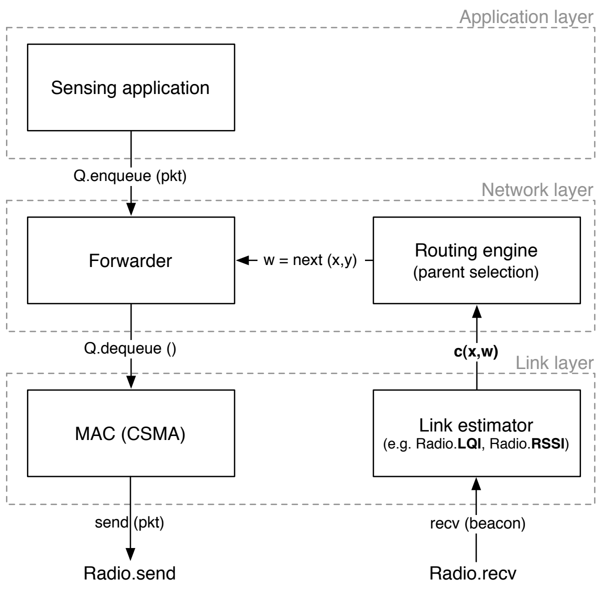
Figures 5.4(a) and 5.4(b) show the behavior of the networking stack when a control message is received and when a data message is received, respectively. The routing engine represents both the proactive (k-neighbors) and reactive (matchmaker) component of the protocol.

The routing engine returns a k-neighbor, say w, that minimizes the estimated cost to route to a destination y. The cost to route from node x to node w is computed by the link estimator, based on control messages (beacons) it received from node w (Figure 5.4(a)).

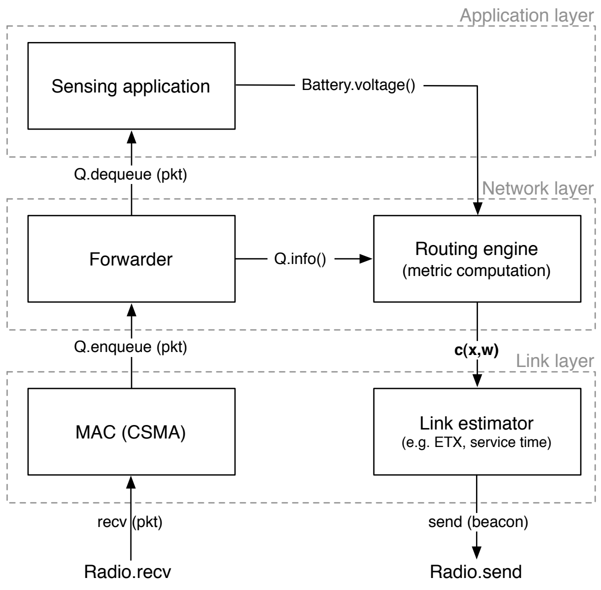
While data packets are in transit, the cost to traverse a node may change. For example, the service time of packets may increase, or the battery level may decrease. This information is fed back to the routing engine, which in turn, when the change is sufficiently large, may broadcast an update message (Figure 5.4(b)).

5.5 Summary

This chapter has described the design of the $\mathcal{F}(k)$ scheme, and discussed some of the design issues and choices that have been made for the $\mathcal{F}(k)$ routing algorithm. It has given a description of the protocol messages and formats. Finally, it has described the implementation of the $\mathcal{F}(k)$ networking stack within the TinyOS environment. The implementation can be loaded onto sensor devices, such as the TMoteSky, but may also be used in the TinyOS simulation environment, TOSSIM.



(a)



(b)

Figure 5.4: The $\mathcal{F}(k)$ networking stack.

Chapter 6

Efficacy of $\mathbb{F}(r)$

At the begining of the dissertation (cf. § 1.1), proactive and reactive routing schemes have been associated together by a hybrid scheme $\mathbb{F}(r)$. The value of r denotes the r-first neighbors of each node x. In order to argue about the optimality of the scheme $\mathbb{F}$, the value of r has been associated with k-neighborhoods. In particular, it has been shown that $\mathcal{F}(k)$ can converge to shortest paths in networks that contain cycles of length less than $2k + 1$. This chapter determines the value of r, the number of neighbors within a k-neighborhood.

The value of r depends upon the *growth* of wireless networks.

6.1 The nature of wireless links

Wireless links (those formed by a collection of radio transceivers) do not lead to arbitrary undirected graphs (cf. § 3.1.1). This section introduces some of their prominent characteristics in order to justify the assumptions that are inherent in the simulation method of wireless networks, in general, and in this dissertation, in particular.

The most basic model for wireless transceivers is a Unit Disc Graph (UDG) [66]. It assumes that all nodes are identical and places them in an ideal environment, where each transmission covers a circular coverage area of the same radius, R. Then, by scaling R to be of unit length, one can derive a unit disc graph by adding a (bidirectional) link $\langle x, y \rangle$ if and only if

$\|x - y\| \leq 1$. Practice, however, indicates otherwise.

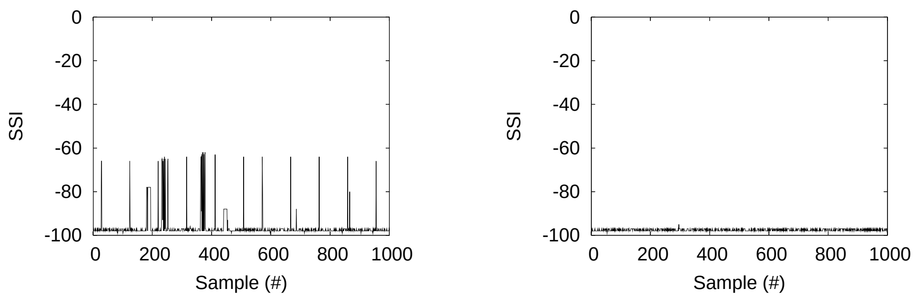
A message is transmitted as a Radio Frequency (RF) signal. Signals travel from a transmitter to a receiver over a channel. In the case of wireless networks, the channel is free space. A signal does not reach the receiver in its original form; its amplitude decreases with distance as it spreads out. This free space loss is typically in the order of $\frac{1}{d^2}$, where d is the distance from the transmitter, depending on the environment. Exponents higher than 2 also account for multi-path effects (reflection).[1] Solid obstacles (e.g. trees or walls) further attenuate the signal.

A receiver also overhears signals from sources other than the intended transmitter. These sources can be part of the same network or belong to a different network altogether. For example, in the case of sensor networks, the IEEE 802.15.4 PHY-MAC standard defines 16 channels, numbered from 11 to 26, in the 2.4GHz band ($2405 - 2480$MHz). The channels are 5MHz apart, overlapping with 802.11b (Wi-Fi) and 802.15.1 (Bluetooth) [80]. Figure 6.1 shows an example of such foreign noise, as measured by a single node on different channels.

A wireless link that consists of two identical transceivers should be symmetric. However, this is not always the case. For example, consider the snippet of a sensor network deployment depicted in Figure 6.2. The snippet is part of the Level-5 testbed (cf. § 4.4; see Figures 4.4 and 4.7) and shows the percentage of broadcast packets that have been successfully delivered along each link, in either direction. Notice, for example, that the link from node A to B is better than the link from A to i, although the Euclidean distance between nodes A and B is twice that of nodes A and i. Such results are in agreement with previous observations of wireless link dynamics (e.g. [80]). For further discussion on the causes of loss in wireless (sensor) networks, see [79].

This dissertation considers metric spaces throughout; this chapter considers the hop-count metric space. Recall from § 2.2 that the hop-count metric

[1]When obstacles reflect a signal, copies of it travel over multiple paths simultaneously. Since each path has different path loss characteristics, the receiver will receive multiple copies of the signal, each with a different magnitude and delay.



(a) Channel 18 (b) Channel 26

Figure 6.1: Foreign noise on two different channels. Signal Strength Indicator (SSI), at 1KHz by reading the RSSI register of the CC2420 radio for 1 minute.

defines the direct neighbors of a node, those within the scope of one broadcast. A transmission from a node x may or may not reach all direct neighbors $y \in C_1(x)$. Hence, according to [66], $c(x, y) \leq 1$; this is the necessary and sufficient condition to add link $\langle x, y\rangle$ to the set of all edges E. That is,

$$\langle x, y\rangle \in E \Leftrightarrow c(x, y) \leq 1.$$

This communication model suffices to measure the routing complexity of an optimal routing scheme. It is equivalent to a unit disc graph, if the cost function $c(x, y)$ is the Euclidean distance between nodes x and y. It is equivalent to a network that consists of links with perfect reception probability, if the cost function $c(x, y) = 1$. Further equivalent models can be derived by selecting an appropriate cost function. The results herein hold for any derived hop count space from a mapping of a user metric space (a user-specified cost function) into hops.

Although metric spaces may assume that reception is perfect, packets can be lost due to collisions, e.g. due to hidden terminals. This means that an algorithm, say with $O(n)$ message complexity, might require more than n messages due to those losses. Such overhead is considered to be the **cost of broadcast**.

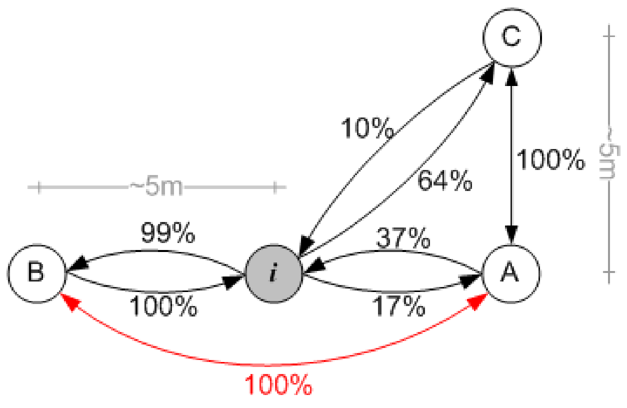


Figure 6.2: A snippet of the Level-5 network. Wireless links are asymmetric.

6.2 Experiments

The experiments consider both random and grid networks. For random networks, a generator attempts to construct n-vertex, ρ-regular graphs (where each of the n vertices has average degree ρ) by filling a unit disc in a way that every node has approximately $\frac{\rho}{n}$ direct neighbors.[1] The experiments consider sparse ($\rho \simeq 8$), dense ($\rho \simeq 16$), and very dense ($\rho \simeq 32$) topologies; the network size n ranges from 64 to 256 nodes.

The complexity measures to evaluate the $\mathcal{F}(k)$ family of protocols are

a) routing state, r;

b) setup and update overhead, as the number of advertisements, queries, and replies.

The first set of experiments evaluates the route discovery process. Given a network of n nodes with sufficient state, there are two distinct cases for *any* source-destination pair $x - y$. First, if $y \in C_x(k)$ then

a) $r =$ the cardinality of $C(k)$;

b) $\Delta_m = 1$ (or $\Delta_\alpha = 0$);

[1]The network generator is a variant of the one used in [19, 61].

c) the number of queries is 0, the number of replies is 0, the number of advertisements is $O(r)$.

This first case is true by construction of a k-neighborhood. That is, every node computes and maintains routes to its k-neighbors proactively. Node y sends an advertisement to all its k-neighbors, hence the upper bound of $O(r)$ messages on the setup overhead. By the proactive method, the routes to all k-neighbors are optimal ($\Delta_m = 1$). Since routes are optimal, the update overhead is 0.

Thus, the experiments focus on the second case where

$$y \notin C_x(k)$$

That is, for a given value of k, the experiments consider routes of length $k + 1 \leq c(x, y) \leq \mathrm{Diam}(G)$. Thus, for the second case,

a) $r =$ the cardinality of $C(k)$, plus 1 for the entry resulting from the query.

Chapter 3 has established that

b) $\Delta_m = 1$ (or $\Delta_\alpha = 0$);

c) the number of queries is $O(n-r)$, the number of replies is $O(Diam(G))$, and the number of advertisements is $O(r)$.

All that remains is to determine the value of r for a given value of k.

6.3 Experimental setup

Certain experiments are run in the `tinyos-2.x` environment, using the TOSSIM discrete event simulator [56]. The default TOSSIM Medium Access Control (MAC) is a CSMA/CA protocol. These types of protocols avoid collisions as follows.

Before a data transmission, a sender transmits a short Ready-To-Send (RTS) packet to the receiver. The latter replies with a Clear-To-Send (CTS) packet. This way, the sender occupies the channel and transmits a DATA packet. The process repeats (for a finite number of times) until an ACK packet from the receiver acknowledges a successful data transmission. Unfortunately, this message handshake does not apply to broadcast packets: there is no unique receiver to acknowledge the data transmission.

The probability of a successful transmission is based on a Signal-to-Noise ratio (SNR) curve, derived from empirical measurements (cf. Figures 4.4 and 6.1).

6.4 The growth of wireless networks

Consider a series of n nodes $X = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$ placed on an infinite line. With no loss of generality, assume that node $x \in X$ begins to construct a network. Initially, $r = 1$. After k iterations, the value of r is

$$r(k) = 1 + \underbrace{2 + 2 + 2 + 2 + \ldots}_{k \text{ times}} = 1 + 2k,$$

since every iteration adds two nodes in the k-neighborhood of x (one on the right and one of the left). The growth rate is arithmetic, with the difference between two successive iterations being 2. It is possible to construct networks with exponential growth, viz. by building a tree. For example, if every node (parent) has ρ neighbors (offspring), then the size of a k-neighborhood at iteration k is

$$r(k) = 1 + \rho + \rho^2 + \cdots + \rho^k = \frac{\rho^k - 1}{\rho - 1}.$$

and the $(k+1)^{th}$ term is

$$r_{k+1} = \rho r_k.$$

Trees have exponential growth. Such exponential growth is the lower bound on the value of r. In particular, similar (but stronger) bounds on r have been shown in [6], where the order r of ρ-regular networks of girth g is

$$r \geq 1 + \rho \frac{(\rho - 1)^{\frac{g-1}{2}} - 1}{\rho - 2}$$

when the girth g is odd (i.e. $g = 2k + 1$, thus $k = \frac{g-1}{2}$), or

$$r \geq \rho \frac{(\rho - 1)^{\frac{g}{2}} - 1}{\rho - 2}$$

when g is even (i.e. $g = 2k$, thus $k = \frac{g}{2}$). It remains to determine an upper bound on the value of r with respect to the value of k. It is very difficult to find an upper bound on the value of r for networks of a given (minimal) density and girth.[1]

Networks, and in particular wireless networks, are of bounded growth [66]. The growth of wireless networks is bounded by physical limitations (e.g. a deployment occupies an area). As the k-neighborhood of each node grows, cycles begin to occur whenever a link connects two already known nodes. Such links appear whenever the network reaches its maximum capacity, n. The (normalized) limiting factor is $\frac{r}{n}$. (This also follows by construction of random networks in this dissertation.) Given a maximum network size n,

$$\frac{dr}{dk} = \rho r \left(1 - \frac{r}{n}\right)$$

and the number of nodes at iteration k (thus, the order r of a k-neighborhood) is

$$r(k) = \frac{n}{1 + (n-1)e^{-\rho k}}.$$

This bounded growth[2] of the network is shown in Figure 6.3 for networks of size $n = 256$ and variable density.

6.4.1 The impact of size and density

Consider the randomly-generated networks underlying the curves of Figure 6.3. There is trade-off between size and density. The sparser the network, the larger the diameter, and therefore the larger the value of k required to converge to the complete network view (that is $r = n - 1 = 255$). For example, when $d \simeq 8$, $k = 27$. Dense networks require smaller values for k to

[1]As stated in [6], it is "the most important unsolved problem."

[2]Also known as the logistic growth curve.

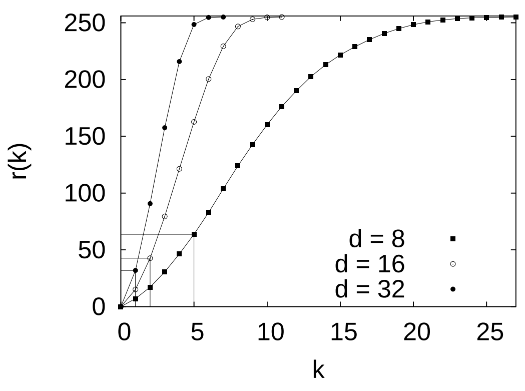


Figure 6.3: The growth of a wireless sensor network for different densities.

converge (e.g. $k = 11$ for $d \simeq 16$ and $k = 7$ for $d \simeq 32$), but the size of routing tables increases faster (high growth rate).

Dense networks have better locality (more nodes to choose from within a k-neighborhood), but there are many (unnecessary) entries in the routing tables. Therefore, growth is independent of the network size. It is rather related to k, and subsequently, the length of cycles.

Chapter 3 has argued that a family of protocols $\mathcal{F}(k)$ can converge to shortest paths whenever the length of cycles is less than or equal to $2k + 1$. Figure 6.3 also shows the values of k that initially satisfy these networks. For example, in a 256-node network of average density $d \simeq 8$ and circumference $c(G) = 11$, $k = 5$ and $r(k) \simeq 63$. Similarly, for $d \simeq 16$ and $c(G) = 5$, $k = 2$ and $r(k) \simeq 42$. As the density increases it is more likely for cycles to occur in a bounded space. For example, $k = 1$ suffices when $d \simeq 32$.

Since $\mathcal{F}(k)$ k-satisfies these networks, then it becomes evident that there is also a value of $r < n - 1$ that satisfies them; therefore, there is no need to apply the most expensive solution (one that requires $n - 1$ routing table entries) to converge to optimal routes; one only requires r(k) routing table entries.

6.4.2 Limitations of broadcast

An inherent limitation of broadcast algorithms (in particular, flooding) is that broadcast packets are not acknowledged. Hence an algorithm may require additional iterations to stabilize. These additional iterations quantify the **cost of broadcast.**

As an example, consider an n-node Hamiltonian network[1] and assume that all links are equiprobable, say of probability p, where p is the probability of a successful transmission along a link. A unicast message m requires

$$\underbrace{\frac{1}{p}+\frac{1}{p}+\cdots+\frac{1}{p}}_{n-1\text{ times}}=\frac{n-1}{p}$$

transmissions to traverse the network, since every node (re-)transmits the message $\frac{1}{p}$ times and all transmissions are independent. For a broadcast message, however, the cost of broadcast is multiplicative; this is because a message flood requires

$$\underbrace{\frac{1}{p}\cdot\frac{1}{p}\cdot\cdots\cdot\frac{1}{p}}_{n-1\text{ times}}=\left(\frac{1}{p}\right)^{n-1}$$

transmissions, since broadcast packets are not acknowledged and the success of the $(n-1)^{th}$ transmission is dependent on the success of the $(n-2)^{th}$ transmission, which is dependent on the $(n-3)^{th}$ one, and so on.[2]

Scoped distance vector algorithms do not retransmit a packet if the new state is already known. However, a node cannot know whether all direct neighbors have received the previous transmission, unless it is aware of its density. In the latter case, where density is known, a node could count the number of retransmissions (e.g., by eavesdropping on the channel). Note

[1]A Hamiltonian network contains a path that traverses all nodes in the network exactly once, hence it is the longest path in the network: a message requires $n-1$ transmissions to traverse it.

[2]The independence (or dependence) of transmissions refer to the method of retransmissions, not interference; for example, even if all other nodes remain silent, there is foreign noise.

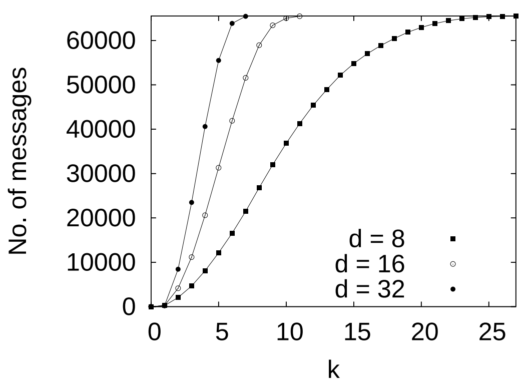


Figure 6.4: The proactive overhead to construct a k-neighborhood.

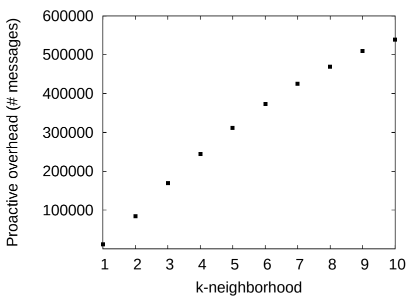
however that only after the network has converged, could one apply density-aware (counter-based) methods to reduce the cost of flooding. This dissertation does not wish to make this assumption. In § 2.7, the cost of broadcast has been associated with m (the number of links in the network), or better μ, the cost to traverse each link in the network rather than n (or r) the number of nodes in the network.

Figure 6.4 shows the proactive overhead that would be required to construct the k-neighborhoods of 256-node networks, under the assumptions that

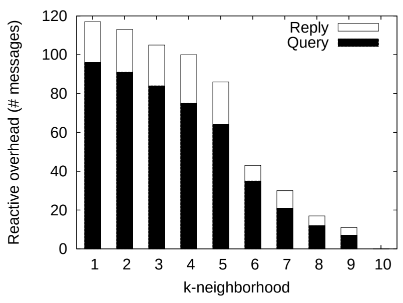
a1) advertisements carry state for a single routing table entry; and

a2) advertisements are successfully transmitted.

When advertisements can carry more than one table entry, this proactive overhead can only improve. If the second assumption is relaxed however, the proactive message overhead can only increase. This is further elaborated in the following section.



(a) Proactive overhead.



(b) Reactive overhead

Figure 6.5: Discovery overhead.

6.4.3 Route discovery

Figure 6.5(a) show the proactive overhead to construct a k-neighborhood in an 100-node grid network, of average density 8. Figure 6.5(b) show the reactive overhead to discover the longest path in the network, after nodes have found their k-neighbors; the shaded portion of the bars show the number of requests sent, while the unshaded portion the number of replies. The proactive overhead is broadcast and might require more that one rounds to converge to k-neighborhoods. It is also a continuous process over time (hence the 2-order of magnitude difference from $O(n^2)$, and the 3-order of magnitude difference from $O(n)$). The reactive overhead is partly broadcast, since a query is flooded to the network, but reply messages are unicast packets: they are retransmitted on a hop-by-hop basis. As the value of k increases, the scope of path queries decreases. This reduces the reactive overhead, but it causes an increase in the proactive overhead. Figure 6.6 shows the relative decrease (resp. increase) of the proactive overhead (resp. the reactive overhead) as the scope k of advertisements (resp. queries) decreases (resp. increases). Figure 6.7 shows the multiplicative stretch of the longest path ($Diam(G)$) in a 100-sensor grid network, as discovered by the reactive method, but without an update process that converges to shortest paths. This (traditional) reactive method cannot guarantee shortest paths. As k

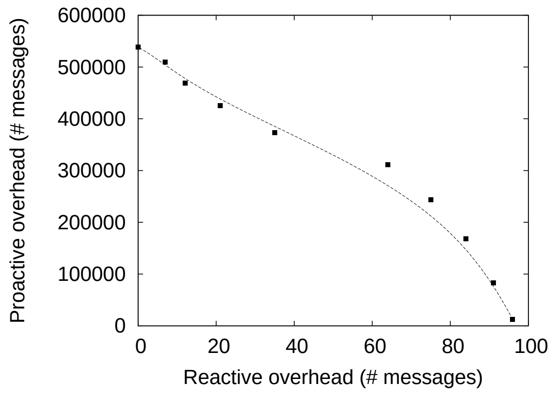


Figure 6.6: Proactive vs. reactive overhead.

increases, it is more likely to converge to some approximation Δ close to an optimal path, but it is still unreliable, until $k = Diam(G)$, in which case the proactive method guarantees optimality.

6.5 Summary

This chapter has evaluated the efficacy of $\mathbb{F}(r)$.

First, it has shown that state (r) is upper-bounded by the size n of the network. Because wireless networks are bounded in growth, the growth of k-neighborhoods is independent from the network size. A k-neighborhood is a network itself – it is a subgraph of the original network – and its size increases with k. The growth rate of a k-neighborhood depends upon density. In dense networks, a k-neighborhood covers the entire network for smaller values of k, but routing tables grow faster.

Second, the algorithm requires $O(n^2)$ messages, in the worst case, to construct all k-neighborhoods. When $O(n^2)$ messages are exchanged, nodes establish a complete network view. In the process of constructing the k-neighborhoods, the results have also shown the impact of the wireless losses. Due to self-interference and hidden terminals, nodes can further increase

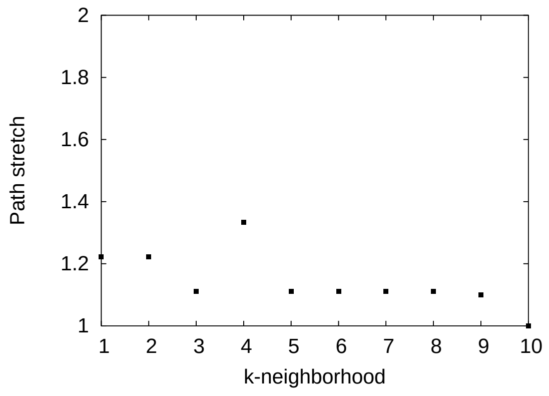


Figure 6.7: Routing stretch.

the proactive overhead by orders of magnitude. This means that the upper bound now becomes the lower bound and everything above it is sub-optimal.[1] The reactive overhead is complementary to the proactive overhead. As k increases, the number of messages required to establish a path decreases. Since the discovery process is based on flooding, it is highly likely that the resulting path is not the shortest.

Most importantly, this chapter has shown that there is a value of r that satisfies every network G of circumference $c(G) \leq 2k + 1$. This value can be less than $n - 1$. Thus, there are cases where $\mathbb{F}(r)$ converges to optimal routes.

Since $r < n - 1$, is it possible to converge to optimal routes when $r = 0$? It is now appropriate to discuss $\mathbb{F}(0)$, which is essentially a stateless routing scheme. As discussed, a protocol $\mathcal{F}(k)$ bounds the scope of advertisements (or queries to nearby destinations) of a node x by k so as to discover an r-neighbor (a neighbor whose cost to reach is less than or equal to k) that can provide node x with a better route to a destination, say y. Once this r-neighbor is found however, one could argue that node x may simply ignore the rest of the r-neighbors, until the network changes again. However, this

[1]This is a good measure for the efficiency of broadcast protocols.

implies either that the metric space is static, i.e. an unweighted network, or that the metric space can be discovered upon request, i.e. there is a passive metric that accurately reflects upon the metric space (the received signal strength, for example). This question remains unanswered by this dissertation. A positive answer would imply that sensors could self-organize into an optimal routing system, using the reactive method alone, and in the absence of any control (imposed by the proactive method).

Chapter 7

Summary and conclusions

In summary, the key contributions of this dissertation are the following.

1. The dynamic routing problem, finding an optimal route between arbitrary pairs of nodes in spite of network dynamics, poses a unique challenge – complexity.

There is a trade-off between the memory cost of a routing solution (a routing function) and the extent of its uncertainty (stretch). Proactive routing schemes – and, in particular, compact routing schemes – have been successful in finding Δ-optimal solutions that scale sublinearly with the size of the network while, at the same time, providing guarantees on the value of uncertainty, Δ. However, such solutions are static. In other words, proactive systems require a management hierarchy in order to bound Δ. As a network evolves, and whenever the cost between any two nodes changes by a value greater than Δ, new configurations are required to reassure an upper bound on stretch.

Given the fundamental trade-off between the size of routing tables, the number of messages needed to construct it, and optimality, how many routes – denoted by a number r – should be pre-determined before a routing system provides any stretch guarantees? This work argues that the value of r is the cardinality of the initial segment of an ordering $\prec_x$ such that

$$I_x(r) = \{y | c(x, y_r) < k\}$$

and, for

$$k \geq \frac{c(G) - 1}{2},$$

this value of r guarantees optimal routes. In particular, it has been shown that there is a family of hybrid routing protocols, namely $\mathcal{F}(k)$, that can converge to minimum-cost routes in networks of circumference $c(G) \leq 2k+1$. More importantly, this work has shown that the $\mathcal{F}(k)$ family of protocols can construct a dynamic, flat routing system that matches the complexity of the best static hierarchical scheme.

2. The alternative solution is to search for an optimal solution. In the context of this dissertation, reactive routing schemes represent exactly this property of nodes – their ability to search. Until now, it was unclear as to whether reactive protocols can converge to an optimal solution or not (and, most importantly, why).

It has become evident that reactive routing schemes lack a *sense of direction* (as opposed, for example, to geographic routing schemes) and, therefore, they must resort to flooding. However, once a sense of direction is established – an initial, but not necessarily optimal, route from a node x to a destination y – nodes can collaborate to converge to a minimum-cost route from x to y. In this work, the reactive method has been revisited and completed by using a hybrid protocol to study the theoretical limits of reactive protocols.

3. Finally, and to return to the ordering of nodes, it is necessary to revisit the elementary proposition stated at the beginning of the dissertation (Chapter 1) – that an optimal route is computable by either the proactive method of y or the reactive method of x.

p) The proactive method $\mathcal{P}$ of y is

p1) At node y, advertise y.

p2) At node x, if $c(x, y) > c(x, w) + c(w, y)$, then the *best next hop* from x to y is w.

p3) At an intermediate node w, do as p1) and/or p2).

r) The reactive method $\mathcal{R}$ of x is

  r1) At node x, query y.

  r2) At node y, if $c(x, y) > c(x, w) + c(w, y)$, then the *best next hop* from x to y is w.

  r3) At an intermediate node w, do as r1) and/or r2).

These two methods are essentially identical (interchange x with y and x's queries for y's advertisements) with their difference being who decides on the *best next hop* w from x towards y. (Or, who preserves the order $\prec_x$ of y).

The system of functions $\mathcal{F}(k)$, namely $\mathbb{F}(r)$, preserves the order between proactive schemes $\mathbb{P}$ and reactive schemes $\mathbb{R}$, in which case

$$\mathbb{P} = \mathbb{R}.$$

Hence proactive and reactive schemes are equivalent, up to an isomorphism. This work has shown that the two equivalence classes of routing protocols are proactive and reactive schemes; and the equivalence relation between them is an *order-isomorphism.*

This dissertation has provided a basis for reasoning about the trade-offs of routing schemes in dynamic networks, such as sensor networks, using well-ordered sets. This is the first work that has shown the relationship between well-order and the optimality of routing protocols.

4. There is a large gap between algorithms and systems – i.e., theory and practice. The work throughout this dissertation has focused on the lower three layers of the networking stack; the physical, the data-link, and the network layer of sensors:

1) The physical layer (viz. the radio) determines the edges of a graph (hence, its connectivity). But for a given transport task, topology awareness alone – often represented as an unweighted graph – cannot guarantee routing efficiency (minimum-cost routes) because wireless links exhibit dynamics that are difficult to capture with static metrics alone.

2) The data link layer determines edge and node weights. It uses active or passive measurements to derive a set of link and node metrics that can satisfy a certain set of user requirements (thus, map the network metric space to a user policy space).

3) Finally, the network layer determines the stretch of a path, based on the choice of algorithms it uses, so as to ensure the fidelity of the routing system to user policy, while minimizing the overhead of route computation.

This work argues that the sub-optimality of routing protocols (routing stretch) should not be decoupled from measured metric spaces (routing metrics).

5. In conclusion, there is no need to always apply the most expensive solution, $\mathbb{F}(n-1)$, to find optimal routes. There is a system $\mathcal{F}(k)$ of routing protocols (functions) that trade off size of routing tables and communication overhead to compute routes that k-satisfy different application requirements. At the very least, this dissertation urges the networking community to *search* for paths on-line.

Chapter 8

Future work

This section explores some of the possible avenues for future work. It lists a number of questions and problems left open by this dissertation.

8.1 The cost of broadcast

The framework that has been developed and is available for experimentation uses distance vectors to disseminate routing state. It is debatable whether wireless routing algorithms should rely on distance vectors or should use a more reliable exchange method based on unicast (cf. 6.4.2). Distance vectors rely on broadcast; unfortunately, broadcast packets can be lost due to collisions.

The rule for distance vector protocols is simple. Upon receipt of an advertisement at some node w, if the state (at a minimum, a routing table entry) carried by the message is different than the one currently stored in memory (indicating either a cost increase or decrease), then node w must rebroadcast it. Distance vector protocols work well in wired networks, where losses are minimal, whereas in wireless networks one should account for link dynamics. To see this, consider a direct neighbor of node w, say z. Node z cannot safely infer whether the absence of any distance vectors is due to stability (the system has converged to an optimal solution) or due to lost packets. Any inference at node z should rely upon link-quality estimators (at the data-link layer) or signal-quality estimators (at the physical layer).

The wireless metric space is not yet fully understood. Therefore, current systems could be enhanced with an algorithmic solution that relies on unicast, rather than broadcast. Initial experimentation with such a routing table synchronization process have shown that

1) all routing tables are eventually consistent, and
2) it is possible to eavesdrop on a unicast message exchange (thus, synchronize *silently*), as to later reduce the number of unicast messages sent.

8.2 Routing-enhanced duty cycles

Sensor networks usually remain idle, as regards data transmissions, in between successive measurements or events, wasting energy as they listen on an idle channel. Sensors may choose to turn off their radios to minimize energy consumption during these idle periods. They do so by means of an ON/OFF schedule (a duty cycle). Obviously, unsynchronized duty cycles among nodes in the network may disrupt any routing task.

As an example, consider the S-MAC protocol [90], one of the first MAC protocols for sensor networks that used duty cycles. In S-MAC, nodes within the k-neighborhood of a leader x establish a common duty cycle by following the ON/OFF schedule of x. Border nodes, nodes that can belong to two or more neighborhoods, conform to all duty cycles imposed on them by the different leaders; this way, they ensure that unsynchronized neighborhoods remain connected. Usually $k = 1$, thus a message traverses one hop per cycle. For $k = 2$, a packet traverses two hops per cycle, and so on; when k equals the diameter of the network, all nodes have a common duty cycle.

Thus far, the dissertation has ignored ON/OFF schedules for reasons of simplicity. This section asks whether one could enhance duty cycles with routing state so as to bound the increase in end-to-end delay (respectively, the decrease in throughput) due to this disruptive communication model.

Routing-enhanced duty cycles are key for the co-design of routing and Medium Access Control (MAC) protocols, especially if nodes are to turn off their radios (thus disrupt connectivity) to preserve resources, as in the case of sensor networks. Intuitively, the computed routes should consider duty cycles, otherwise they offer no guarantees. The proactive and reactive methods may simply fail to return a path (due to lack of synchronization) [39].

It is possible to use routing state (routing tables) to ensure that a computed route for a given source-destination pair remains synchronized for the duration of the transmission, as in the R-MAC protocol [15]. Protocols that require *a priori* knowledge of routes fall under the proactive method. For the reactive method, consider the example of the AIMRP protocol [51].

Given the k-neighborhood of a sink (the default destination y), AIMRP finds a route from a node x_k, a node that is k-hops away from y, to sink y as follows. Upon demand, x_k sends a request for a $(k-1)$-neighbor, say x_{k-1}, such that the ON periods of x_k and x_{k-1} overlap. Upon reply, data propagate to the next hop. In turn, node x_{k-1} asks for an $(k-2)$-neighbor and so on, until the data are eventually delivered to the sink.

Consider the following formulation of the routing problem (similar to [39, 91]). Given a routing task from node x to node y at time t_0 and an upper bound of t time units on the time it takes for a message from x to reach node y, find a neighbor w^*, such that

$$c(x, w^*) + c(w^*, y) = \min_i \{c(x, w_i) + c(w_i, y)\},$$

and

$$w^* \text{ is ON for the duration of } [t_0, t)\,.$$

A routing scheme that can positively answer this problem is able to compute a delay-bounded route between any pair of sensors $\{x, y\}$ within a given time window $[t_0, t)$. Because route interrupts along a path are usually predictable (sensors advertise their schedule to their neighbors), it is possible for a routing scheme to pre-compute such paths based on future network dynamics.

The routing protocol $\mathcal{F}(k)$ of node x, and given sufficient knowledge of the ON/OFF schedules of its k-neighbors, can compute a continuous path (if one exists) to a k-neighbor. For a more distant destination y, it remains to be seen whether $\mathcal{F}(k)$ can place an upper bound on the time it takes for a message from x to reach node y.

8.3 On sensor network management

The Open System Interconnection (OSI) management framework [36] divided Internet management and its standards into five functional areas – these are fault, configuration, accounting, performance, and security management – commonly abbreviated as FCAPS.

At the centre of FCAPS is configuration management. Specifically, a manager (a human operator or a machine) monitors (via query or update messages[1]) information from agents (end terminals) about statistics, diagnostics, or accounts on (mis)usage. The manager then configures the network accordingly, as to meet certain user requirements.

While FCAPS and SNMP may suffice to macro-manage a sensor network (e.g. specify a high-level policy), micro-management of sensors should be performed *in situ*. Sensor networks are typical examples of distributed systems that should operate unattended and thus, by necessity, autonomously. For truly autonomous networks, the centre of management becomes fault management.

Currently, there are multiple points of interaction between users and systems; thus, there is a need to move away from the traditional centralized paradigm where scientists design, implement, and manage a sensor network centrally. The less transparent a system is, the more complex it becomes. In order to scale, one could delegate management roles closer to end nodes, resulting in a hierarchy of roles (management by delegation[25]); or, one could experiment with the parameter k as to allow nodes to self-configure.

The study on k-neighborhoods has revealed a form of network that is 1-fault tolerant: a triangle that consists of a source x, a destination y and a k-neighbor that (i) ensures that the path from x to y is optimal, and also (ii) recovers the path in case the current next hop w fails. This also has implications on security. Are such networks secure? When all three nodes are non-malicious, then the answer is yes: it can also handle the addition of any node since, according to [52], more than $\frac{2}{3}$ of the nodes are trusted.

[1]Refers to the *pull* and *push* management model, the most common manifestation of which is the Simple Network Management Protocol (SNMP) [8].

This dissertation has shown ways of configuration, by static assignment of the value of k, when all nodes are non-malicious. There are two open problems of interest. Firstly, given a initial node x and $\mathbb{F}(0)$, construct an optimal (or near-optimal) routing system. This requires that node x be able to determine the value of k dynamically (i.e. close cycles of length less that $2k+1$). Secondly, given an initial system of three nodes $\{x, x_k, y\}$ and $\mathbb{F}(3)$, construct an optimal (or near-optimal) routing system.

8.4 Naming and addressing

Thus far, this dissertation employs a flat addressing scheme for sensors.

There is a spatial relationship between measurements or events and the physical world: sensors in close proximity report similar measurements (e.g. a temperature reading) or react to similar events (e.g. a fire alert). Communicating pairs are now identified by semantics, e.g. data attributes or geographic coordinates, rather than identifiers.

This dissertation has stated that in the absence of sense of direction, a node must resort to flooding in order to discover routes to unknown destinations. To establish an *a priori* sense of direction, the routing system $\mathbb{F}$ can be enhanced with a directory (or location) service. For example, the Beacon Location Service (BLS) [67] is a Distributed Hash Table (DHT) that maps node names to topologically meaningful addresses, based on their proximity to a set of landmarks (or beacons). Such a service can be easily incorporated in the $\mathcal{F}(k)$ framework.

As location-aware applications in networks become prevalent - e.g. find the nearest replica of a service - meaningful, yet compact, state is an issue of growing importance. Such a distributed directory service can be used to experiment with hybrid push-pull application models, such as [60], where data are pushed to predefined (or random) rendezvous points to be pulled from users later on.

8.5 Mobility

Thus far, the dissertation has considered non-mobile networks. Thus, nodes do not experience the thrashing of routing tables seen in mobile networks. This excludes a rather large class of ubiquitous applications where mobility is a requirement.

Mobility is a special case of a network change. Network changes occur at different time scales. There are changes in link or node quality, changes due to a failure, and changes due to mobility. Mobility, however, does not add an additional layer of routing complexity. While mobile nodes generate updates more frequently, issues of link churn have also been addressed in this dissertation: the scope of updates.

There are many application scenarios available for experimentation with the $\mathbb{F}$ routing system. For example, a mobile user could simply upload (download) some information to (from) the nearest k-neighborhood along its trajectory. The route discovery problem becomes more difficult if the destination in question is also mobile. The problem of finding a mobile destination can be studied in conjunction with the cost of broadcast (§ 8.1) and rendezvous services (§ 8.4).

Bibliography

[1] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *SPAA '04: Proceedings of the 16th annual ACM symposium on Parallelism in algorithms and architectures*, pages 20–24. ACM, 2004.

[2] A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM 2000: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 32–41 vol.1, 2000.

[3] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *STOC '89: Proceedings of the 21st annual ACM symposium on Theory of computing*, pages 479–489. ACM, 1989.

[4] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[5] M. Boguñá and D. V. Krioukov. Navigating ultra-small worlds in ultra-short time. *Physical Review Letters*, 102(058701), 2009.

[6] B. Bollobás. *Extremal graph theory*, chapter 3. Academic Press, 1978.

[7] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM, 1998.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP), May 1990. RFC 1157.

[9] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), October 2003. RFC 3626.

[10] D. S. J. D. Couto. *High-Throughput Routing for Multi-Hop Wireless Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2004.

[11] D. Cuff, M. Hansen, and J. Kang. Urban sensing: out of the woods. *Commun. ACM*, 51(3):24–33, 2008.

[12] S. M. Das, H. Pucha, K. Papagiannaki, and Y. C. Hu. Studying wireless routing link metric dynamics. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 327–332. ACM, 2007.

[13] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of multihop wireless networks: shortest path is not enough. *SIGCOMM Comput. Commun. Rev.*, 33(1):83–88, 2003.

[14] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[15] S. Du, A. K. Saha, and D. B. Johnson. RMAC: A routing-enhanced duty-cycle MAC protocol for wireless sensor networks. In *INFOCOM 2007: Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1478–1486. IEEE, 2007.

[16] C. T. Ee and R. Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 148–161. ACM, 2004.

[17] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: Definitions, properties, and classes. *Networks*, 32(3):165–180, 1998.

[18] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. In *HotNets VI: Proceedings of the 6th Workshop on Hot Topics in Networks*, pages 1–10, 2007.

[19] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: scalable point-to-point routing in wireless sensornets. In *NSDI '05: Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation*, pages 329–342. USENIX Association, 2005.

[20] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Comput. Optim. Appl.*, 29(1):13–48, 2004.

[21] C. L. Fullmer and J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. In *SIGCOMM '97: Proceedings of the 1997 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 39–49, 1997.

[22] S. Gal. *Search games*, chapter 2. Academic Press, 1980.

[23] C. Gavoille. Routing in distributed networks: overview and open problems. *SIGACT News*, 32(1):36–52, 2001.

[24] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2009.

[25] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*, page 333. IEEE Computer Society, 1995.

[26] Z. J. Haas. A new routing protocol for the reconfigurable wireless networks. In *ICUPC'97: Proceedings of 6th IEEE International Conference on Universal Personal Communications*, volume 2, pages 562–566. IEEE, October 1997.

[27] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006.

[28] Z. J. Haas and M. R. Pearlman. The performance of query control schemes for the zone routing protocol. *IEEE/ACM Trans. Netw.*, 9(4):427–438, 2001.

[29] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. F. Abdelzaher. Range-free localization and its impact on large scale sensor networks. *Transactions on Embedded Computing Systems*, 4(4):877–906, 2005.

[30] J. Heidemann, F. Silva, and D. Estrin. Matching data dissemination algorithms to application requirements. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 218–229. ACM, 2003.

[31] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.

[32] K. Hrbacek and T. Jech. *Introduction to set theory / 3rd ed. rev. and expanded*, chapter 6. CRC Press, 1999.

[33] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147. ACM, 2004.

[34] IEEE Computer Society. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), October 2003. 802.15.4 IEEE Standard.

[35] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[36] ISO. Information processing systems – open systems interconnection – basic reference model – part 4: Management framework. Technical Report ISO/IEC 7498-4: 1989 (E), ISO/IEC, November 1989.

[37] K. Iwanicki and M. van Steen. On hierarchical routing in wireless sensor networks. In *IPSN '09: Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 133–144, April 2009.

[38] J. Jaffe and F. Moss. A responsive distributed routing algorithm for computer networks. *Communications, IEEE Transactions on*, 30(7):1758–1762, July 1982.

[39] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158. ACM, 2004.

[40] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. *Kluwer International Series in Engineering and Computer Science*, pages 153–179, 1996.

[41] B. Karp. *Geographic Routing for Wireless Networks.* PhD thesis, Division of Engineering and Applied Sciences, Harvard University, October 2000.

[42] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM, 2000.

[43] S. Keshav. A control-theoretic approach to flow control. *SIGCOMM Comput. Commun. Rev.*, 21(4):3–15, 1991.

[44] A. Khanna and J. Zinky. The revised ARPANET routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[45] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 217–230. USENIX Association, 2005.

[46] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM, 2000.

[47] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks (1976)*, 1(3):155, 1977.

[48] D. E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, 1976.

[49] A. Korman and D. Peleg. Dynamic routing schemes for graphs with low local density. *ACM Trans. Algorithms*, 4(4):1–18, 2008.

[50] D. Krioukov, k. c. claffy, K. Fall, and A. Brady. On compact routing for the internet. *SIGCOMM Comput. Commun. Rev.*, 37(3):41–52, 2007.

[51] S. Kulkarni, A. Iyer, and C. Rosenberg. An address-light, integrated MAC and routing protocol for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 14(4):793–806, 2006.

[52] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[53] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo. The emergence of a networking primitive in wireless sensor networks. *Commun. ACM*, 51(7):99–106, 2008.

[54] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95. ACM, 2002.

[55] P. Levis and D. Gay. *TinyOS programming*. Cambridge University Press, 2009.

[56] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003.

[57] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2004.

[58] P. Levis, A. Tavakoli, and S. Dawson-Haggerty. Overview of existing routing protocols for low power and lossy networks. IETF, April 2009. Work in progress.

[59] J. Li, C. Blake, D. S. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 61–69. ACM, 2001.

[60] X. Liu, Q. Huang, and Y. Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 122–133. ACM, 2004.

[61] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *NSDI '07: Proceedings of the 4th USENIX Symposium on Networked System Design and Implementation*, 2007.

[62] K. Martinez, J. K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50–56, 2004.

[63] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol.* Addison Wesley Longman, Inc., 1998.

[64] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol*, chapter 6. Addison Wesley Longman, Inc., 1998.

[65] J. Moy. OSPF version 2, April 1998. RFC 2328.

[66] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4):1–17, 2008.

[67] J. Ortiz, C. R. Baker, D. Moon, R. Fonseca, and I. Stoica. Beacon location service: a location service for point-to-point routing in wireless sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 166–175. ACM, 2007.

[68] D. Peleg. *Distributed Computing: a locality-sensitive approach*, chapter 23. Society for Industrial and Applied Mathematics (SIAM), 2000.

[69] D. Peleg. *Distributed Computing: a locality-sensitive approach*, chapter 15. Society for Industrial and Applied Mathematics (SIAM), 2000.

[70] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[71] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing, July 2003. RFC 3561.

[72] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, Feb 1999.

[73] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994.

[74] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. Comput. Syst.*, 8(2):158–181, 1990.

[75] V. Ramasubramanian, Z. J. Haas, and E. G. Sirer. SHARP: a hybrid adaptive routing protocol for mobile ad hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 303–314. ACM, 2003.

[76] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108. ACM, 2003.

[77] C. A. Santivá nez, R. Ramanathan, and I. Stavrakakis. Making link-state routing scale for ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 22–32. ACM, 2001.

[78] C. Savarese, J. Rabaey, and J. Beutel. Locationing in distributed ad hoc wireless sensor networks. In *Proc. 2001 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 4, pages 2037–2040. IEEE, Piscataway, NJ, May 2001.

[79] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Understanding the causes of packet delivery success and failure in dense wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 419–420. ACM, 2006.

[80] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The β-factor: measuring wireless link burstiness. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 29–42. ACM, 2008.

[81] K. Srinivasan and P. Levis. Rssi is under appreciated. In *EmNets '06: Proceedings of the Third Workshop on Embedded Networked Sensors*, 2006.

[82] M. Thorup and U. Zwick. Compact routing schemes. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM, 2001.

[83] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[84] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *SIGCOMM '88: Proceedings of the 1988 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 35–42. ACM, 1988.

[85] J. Van Leeuwen and R. B. Tan. Interval routing. *The Computer Journal*, 30(4):298–307, 1987.

[86] C.-Y. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft. Overload traffic management for sensor networks. *ACM Trans. Sen. Netw.*, 3(4):18, 2007.

[87] C. Wang, B. Li, K. Sohraby, M. Daneshmand, and Y. Hu. Upstream congestion control in wireless sensor networks through cross-layer optimization. *Selected Areas in Communications, IEEE Journal on*, 25(4):786–795, May 2007.

[88] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, June 1998.

[89] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27. ACM, 2003.

[90] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, 2004.

[91] W. Zhao, M. Ammar, and E. Zegura. Multicasting in delay tolerant networks: semantic models and routing algorithms. In *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 268–275. ACM, 2005.