Maximum Lifetime Routing in Wireless Ad-hoc Networks

Arvind Sankar and Zhen Liu

Abstract— **Routing problems in mobile ad-hoc networks (MANET) have been receiving increasing attention in the last few years. Most of the proposed routing protocols concentrate on finding and maintaining routes in the face of changing topology caused by mobility or other environmental changes. More recently, power-aware routing protocols and topology control algorithms have been developed to address the issue of limited energy reserve of the nodes in ad-hoc networks. In this paper we consider the routing problem in MANET with the goal of maximizing the life time of the network. We propose a distributed routing algorithm that reaches the optimal (centralized) solution to within an asymptotically small relative error. Our approach is based on the formulation of multicommodity flow, and it allows to consider different power consumption models and bandwidth constraints. It works for both static and slowly changing dynamic networks.**

I. Introduction

The network model we consider is the *wireless ad-hoc network*, consisting of a set of nodes connected by wireless links. The topology of the network is not under our control, but is determined purely by the current geographic location of the nodes and other environmental conditions, and the characteristics of the radio transcievers that the nodes possess. The nodes wish to communicate among each other, and we assume that they are willing to relay packets in order to facilitate this communication. The problem is to design effective routing protocols to meet a variety of performance objectives.

Typical examples of ad-hoc networks are wireless sensor networks, where the nodes are sensors that gather environmental data and send the information to computational nodes for further processing, or to base stations for relay to a wired network. Such networks could be deployed in hazardous locations, for example in disaster areas to aid rescue efforts, for mineral or oil prospecting, in defense applications in the battlefield etc.

Most of the previous routing protocols ([1]–[5]) for wireless ad-hoc networks concentrate on finding and maintaining routes in the face of changing topology caused by mobility or other environmental changes. Typical protocols use shortest path algorithms based on hop count, geographic distance, or transmission power. The first two are important in minimizing delay and maximizing throughput. The third objective is peculiar to wireless ad-hoc networks, and is important because typically the nodes involved have a limited power supply, and radio communication consumes a large fraction of this supply.

To address this issue, several power-aware routing protocols and topology control algorithms have been developed ([6]–[10]). In most of these, the aim is to minimize the energy consumed per packet in order to deliver it to the destination. The typical approach is to use a distributed shortest path algorithm in which the edge costs are related to the power required to transmit a packet between the two nodes involved. The problem with this technique is that nodes on the minimum-energy path are quickly drained of power, affecting the network connectivity when they fail. Some of the more sophisticated routing algorithms associate a cost with routing through a node with low power reserves ([6], [8]). But this remains at best a heuristic solution.

In [11] and [12], a rigorous formulation using linear programming is presented which attempts to capture the issue of power consumption more precisely. The idea is to make the goal of routing the maximization of the *network lifetime*, which is the time to network partition because of node failures. They give a heuristic algorithm to solve the linear program approximately, but which can perform arbitrarily badly in the worst case. In the later paper [13], a centralized algorithm to determine the maximum lifetime is presented, based on the Garg-Koenemann [14] algorithm for multicommodity flow.

In this paper, we explicitly formulate the problem as a maximum concurrent flow problem. All routing problems have analogous flow realizations, where the flow represents the routes that packets take, and the demands represent the rate at which packets are generated by various nodes. This formulation allows one to apply the extensive literature of maxflow algorithms to the problem of routing for maximum network lifetime. Many of these algorithms are not well-suited to distributed implementation as is necessary for a routing protocol, or do not adapt well to changing network topology. Moreover, they are typically better-suited to the problem of wired network routing where the constraints are link bandwidth (the number of packets that can be routed over an edge in unit time) and node capacity (the number of packets that can be forwarded by a node in unit time).

We adapt a distributed flow algorithm due to Awerbuch and Leighton ([15], [16]) to our situation. The algorithm finds an approximation to a feasible flow if one exists. The difference between the current paper and earlier heuristic approaches is that the approximation factor is guaranteed, and we theoretically analyze the algorithm, giving lower bounds on its performance. The algorithm is applicable to various power consumption models in static networks as well as dynamic networks with slowly changing edge costs.

The advantage of this approach over that of [13] is that it is a distributed, local-control approach and does not require a central node with global knowledge of the network. This is

0-7803-8356-7/04/$20.00 (C) 2004 IEEE

also the reason why it works well with dynamic networks as well.

The paper is organized as follows: in section II, a rigorous problem formulation is presented. In section III, the flow algorithm is presented and analyzed. In section IV, the feasibility algorithm is extended to actually determine the optimal routing that maximizes the network lifetime. In section V, some implementation issues are addressed, and simulation results are presented. Finally, in section VI, we discuss possible extensions and make some concluding remarks.

II. Problem formulation

In this section, we will formulate the problem more precisely and give both a linear programming and a flow interpretation.

The network is represented as a graph, with N nodes and M edges, with the nodes representing wireless devices and the edges the wireless links between them. Associated with each node i is a quantity E_i, representing the initial energy reserve of the device. Each edge ij has a cost e_{ij}, which is the energy required to transmit one packet of data across the corresponding link.

The routing problem consists of a set of K source and destination pairs, which are pairs of nodes in the network. Each pair has a throughput requirement Q_c, the number of packets per second that must be routed between source and destination nodes. The routes are given by variables f_{ij}^c, which represent the rate at which packets are transmitted across link ij for connection c.

A. Linear programming formulation

In [12], the problem is formulated as a linear program. We give a slightly simplified description (their formulation includes the possibility that a connection has multiple sources and/or destinations).

The *lifetime* of the network is defined as the time at which the first node failure occurs, that is, the time at which some node's energy reserve is reduced to zero. Our goal is to route packets in such a manner that the lifetime is maximized while the throughput requirements are satisfied. We denote the lifetime by T. Define new variables $\hat{f}_{ij}^c$ denoting the total number of packets for connection c transmitted from node i to node j over the lifetime of the network.

The total energy consumed at node i is given by

$$\sum_{j,c} e_{ij} \hat{f}_{ij}^c$$

where the sum is over all nodes adjacent to i and all connections c. At every node except the source and destination for a particular connection c, the number of packets received must equal the number of packets transmitted. At the source, the number of packets transmitted equals $Q_c T$ over the lifetime T of the network.

Hence the linear program is

$$\begin{array}{lrcll} \max & T & & & \\ \text{s.t.} & \hat{f}_{ij}^c & \geq & 0 & \forall i,j,c \\ & \sum_{j,c} e_{ij} \hat{f}_{ij}^c & \leq & E_i & \forall i \\ & \sum_j \hat{f}_{ij}^c - \sum_k \hat{f}_{ki}^c & = & \begin{cases} Q_c T \\ 0 \end{cases} & \forall i,c \quad \begin{array}{l} i \text{ a source for } c \\ \text{otherwise} \end{array} \end{array}$$

The second constraint is due to the finite power supplies at the nodes, and the third represents the throughput requirements at the sources and the conservation constraints at other nodes.

This formulation is not very useful directly, but serves to show that the problem can theoretically be solved exactly in polynomial time.

B. Multicommodity flow

Routing problems can generally be interpreted as flow problems, and ours is no exception. The variables f_{ij}^c can be thought of as flow values of a commodity c. The requirement that incoming packets arrive at the same rate as outgoing packets leave is equivalent to the flow conservation constraint that at any intermediate node, the amount of incoming flow equals the amount of outgoing flow. The sources and destinations of the routing problem become sources and sinks in the flow problem. The throughput requirements Q_c represent the rate at which flow is produced at the sources and consumed at the sinks. So at each instant of time, the routing is represented as a multicommodity flow satisfying demands Q_c.

We will consider the time evolution of the network as proceeding in a sequence of synchronous rounds, each lasting for unit time. In each round, Q_c packets are generated at each source node. Packets may be transmitted between nodes, and the energy reserve of each node is reduced by the total cost of the packets it transmits. So, if f_{ij} is the number of packets transmitted by node i to node j, the energy reserve of i is reduced by $\sum_j e_{ij} f_{ij}$ in this round. Since each node will actually be forwarding packets for a number of connections, the reduction in the energy is given by

$$\sum_{j,c} e_{ij} f_{ij}^c$$

From the problem formulation, it might appear that a more general version is to ask for how many rounds a feasible multicommodity flow can be maintained, where the flow is allowed to vary with time, and some intermediate nodes might fail before the demands become infeasible. This does not lead to any improvement in the lifetime achievable over a static flow that lasts only till the first node failure, however, as can be seen by the following argument: suppose the flow achieving the maximum lifetime T is $f_{ij}^c(t)$ at time t, and satisfies the demands at every instant. Replace this variable flow by its time average

$$\tilde{f}_{ij}^c = \frac{1}{T} \int_0^T f_{ij}^c(t)\, dt$$

This static flow will satisfy the same demands at every instant, and now no node will fail before the lifetime T. Since the flow is static, the energy consumed in any round at a node i cannot exceed E_i/T.

Thus it is enough to look for a multiflow satisfying all the demands, and constrained by

$$\sum_{j,c} e_{ij} f_{ij}^c \le \frac{E_i}{T} \qquad \text{for all } i$$

to achieve a lifetime of T. We denote by L an upper bound on the length of the longest flow path in the feasible flow f_{ij}^c. An approximation algorithm for this problem will be described in the next section.

III. Local-control flow algorithm

The algorithm is expressed in terms of the flow interpretation. The goal is to determine whether a given value of the lifetime T is feasible. That is, is it possible to route flow in the network so that all the demands are satisfied, and no node runs out of energy before T rounds have gone by.

We will assume that there exists a multiflow that satisfies not just the demands Q_c, but slightly more, namely the demands $(1+\epsilon)Q_c$, for some parameter $\epsilon > 0$.

Theorem 1 *If the demands $(1+\epsilon)Q_c$ are feasible, then there exists a routing algorithm that will satisfy the demands $(1-\delta)Q_c$, for any $\delta > 0$, after running for*

$$\frac{8ML\left(K + \ln(6K/\epsilon)\right)}{\epsilon\delta}$$

rounds.

We will prove the theorem by actually describing such an algorithm and giving a complete analysis of its performance. The original algorithm in [15], [16] is for the case where there are edge capacity constraints, such as for example bandwidth limits. In our case, the constraints are on the nodes, since each node must respect its energy budget E_i/T.

A. Description

Each node maintains K queues for each of its links. In each round, the flow in a queue may be moved to the other end of the link. Flow may also be redistributed among the queues at a single node.

Denote by q_{ij}^c the height of the queue for commodity c on link ij at node i. The link has another queue at its other end, denoted by q_{ji}^c.

We define a potential function $\phi(q)$ associated with every queue, where q is the queue height. The potential function will be completely specified when we analyze the performance of the algorithm, for now note that it will be a twice-differentiable convex function. The algorithm will operate so as to minimize the sum of the potential functions of all queues.

In each round of the algorithm the following phases are performed:

- *Inject flow*. For each commodity c, add Q_c flow at the corresponding sources.
- *Balance nodes*. Equalize the queues at each node.
- *Push flow*. This step is the most complex and contains the crux of the algorithm. Associate each pair of queues q_{ij}^c and q_{ji}^c for an edge ij with the node where the queue height is greater. To reduce the potential, we must move packets from the higher queue to the lower (this follows from the convexity of ϕ). So we minimize the function

  $$\sum_{j,c} \phi(q_{ij}^c - f_{ij}^c) + \phi(q_{ji}^c + f_{ij}^c)$$

  where the sum is over edges ij associated with node i, and the variables are the f_{ij}^c subject to the constraint

  $$\sum_{j,c} e_{ij} f_{ij}^c \le \frac{E_i}{T}$$

  The solution f_{ij}^c is then used to move flow from node i to node j.
- *Drain flow*. Absorb the flow for commodity c at its sink. That is, set the height of all queues for c that are at the sink to zero.

B. Analysis

We will show that the potential in the system remains bounded as time goes on, which implies that most of the flow injected into the network reaches the appropriate sink.

The idea will be to lower bound the decrease in potential in a round by comparing what the algorithm does to what would happen if we picked the flow values according to a feasible multiflow instead. This will in turn imply an upper bound on how large the potential can become as the algorithm is repeatedly executed.

For simplicity in writing the expressions, we will assume that the source nodes and the destination nodes both have unit degree, and that all flows and queues are scaled so that the demand for each commodity is unity.

The increase in potential after the first two phases will be at most

$$\sum_c \phi(s_c) - \phi(s_c - 1) \le \sum_c \phi'(s_c)$$

where s_c is the total amount of commodity c at the source after flow injection.

The potential decreases when we push flow, and to estimate the decrease in potential, we consider what happens if we use a feasible flow f_e^c satisfying the demands $(1+\epsilon)$, instead of the minimizing flow values we compute. The expression is

$$\sum_{e,c} \phi(q_{eh}^c) + \phi(q_{et}^c) - \phi(q_{eh}^c + f_e^c) - \phi(q_{et}^c - f_e^c)$$

where the summation extends over all edges in the network, and q_{eh}^c and q_{et}^c denote the queue heights for commodity c at the head and tail of the edge respectively. We apply the extended mean value theorem to ϕ and simplify the expression to

$$\sum_{e,c} f_e^c(\phi'(q_{et}^c) - \phi'(q_{eh}^c)) - \frac{1}{2}(f_e^c)^2(\phi''(q_{eh}^c + \zeta_{eh}^c f_e^c) + \phi''(q_{et}^c - \zeta_{et}^c f_e^c))$$

where the ζ_{et} and ζ_{eh} are suitable constants between 0 and 1. Note now that the first derivative terms will cancel out except

at the source and sink, because in the node balancing phase, all the queue heights for a given commodity at a given node were made equal, and f_e^c satisfies the flow conservation property. We will pick our potential function so that the second derivative terms make a very small contribution to the sum. In particular, assume that their contribution is at most $\frac{\epsilon}{2}\phi'(s_c)$. Hence the potential drop is at least

$$\sum_c (1+\epsilon)\left(\left(1-\frac{\epsilon}{2}\right)\phi'(s_c) - \phi'(0)\right)$$

So overall during the round, the potential drops by at least

$$\sum_c \left(\frac{\epsilon}{2} - \frac{\epsilon^2}{2}\right)\phi'(s_c) - (1+\epsilon)K\phi'(0)$$

Note that if we ignore the flow values f_e^c that move flow from a lower height queue to a higher queue, we will only improve the potential drop. Since our algorithm considers all such feasible updates while minimizing, the potential drop it finds will be better than the one we have just derived.

Let S be the value that satisfies the equation

$$\left(\frac{\epsilon}{2} - \frac{\epsilon^2}{2}\right)\phi'(S) = (1+\epsilon)K\phi'(0)$$

So if $s_c \geq S$ for any commodity, then the potential drop during the round will be non-negative. We wish to bound the amount of commodity c in the network by $2MS$, by showing that no queue can exceed S in height.

To do this we borrow from [16] the trick of using 'overflow buffers' at each source node. The idea is that if the queue at the source has reached S, then we put additional injected flow in a special overflow buffer, which has potential $b\phi'(S)$ if it has b amount of commodity. With this potential function, the increase in potential when flow is injected remains upper bounded by $\phi'(s_c)$. The advantage of bounding the source queue is that no other queue can exceed this bound. For it is easy to see that given a particular value of queue height, S in this case, the first queue to exceed this height must be at the source.

Now we argue by induction on the number of rounds that the potential must always be less than $2MK\phi(S)$. For as long as the overflow buffers are empty for all commodities, this remains true. If, on the other hand, the flow injection phase overflows some source queues, then we know that the potential actually drops during this round.

Hence the maximum size of an overflow buffer is $2MK\phi(S)/\phi'(S)$. This result will be needed in the next section, so we restate it as a lemma.

Lemma 2 *If S satisfies the equation*

$$\left(\frac{\epsilon}{2} - \frac{\epsilon^2}{2}\right)\phi'(S) = (1+\epsilon)K\phi'(0)$$

then the maximum size of an overflow buffer is

$$2MK\phi(S)/\phi'(S)$$

The maximum amount of commodity c within the edge queues is $2MS$, hence combining this with the upper bound on the overflow buffers, we get an upper bound of $2M(S + K\phi(S)/\phi'(S))$ on the total amount of commodity in the system.

Now we set the potential function $\phi(q) = e^{\alpha q}$, for $\alpha = \epsilon/4L$. This gives, for ϵ small enough

$$\begin{aligned}
\sum_e \frac{1}{2}(f_e^c)^2(\phi''(q_{eh}^c + \zeta_{eh}^c f_e^c) + \phi''(q_{et}^c - \zeta_{et}^c f_e^c)) & \\
\leq & \sum_e (f_e^c)^2\alpha^2 e^{\alpha(S+f_e^c)} \\
\leq & (1+\epsilon)\alpha e^{\alpha(1+\epsilon)}\sum_e f_e^c\phi'(S) \\
\leq & (1+\epsilon)\frac{\epsilon}{4L}e^{\epsilon(1+\epsilon)/4L}L(1+\epsilon)\phi'(S) \\
\leq & \frac{\epsilon}{2}\phi'(S)
\end{aligned}$$

The equation for S gives $S \leq \left(\frac{4L\ln(6K/\epsilon)}{\epsilon}\right)$. So the total amount of a given commodity in the system is at most $R = 8ML(K + \ln(6K/\epsilon))/\epsilon$.

Suppose that we run the algorithm continuously for R/δ rounds. The amount of commodity c pumped into the network is R/δ, and the amount remaining in the network is at most R. Hence the average amount of commodity transported through the network per round is $1-\delta$. This proves theorem 1.

Another parameter of interest is the average delay a packet experiences while being routed by our algorithm. Since the amount of commodity in the system is bounded by R in the long run, the average delay is also upper bounded by R, by Little's Law.

Lemma 3 *The average delay of a packet is at most*

$$\frac{8ML(K+\ln(6K/\epsilon))}{\epsilon}$$

IV. Maximizing lifetime

Now we describe how to use the algorithm to converge to the routing appropriate to the maximum lifetime. We will imagine for the moment that we have centralized control, and can update information at all the nodes at the same time.

The node constraints will be set to E_i/T, and we will vary T till it converges to the optimal value T^*. The idea is that if T is too large, then the node constraints are too tight to allow for a feasible flow, hence the source queues will grow without bound. Hence we start out T at some value that is smaller than the true lifetime and increase it till we find that some overflow buffer exceeds the bound $2MK\phi(S)/\phi'(S)$. Now we know the optimal lifetime T^* to within a factor of two, and can perform a bisection search to determine it exactly.

Let us flesh out the outline above. Suppose we want to determine the optimum lifetime T^* to within a relative error of ϵ. Initialize T to some suitable value T_0, which we assume to be less than the optimum T^*. Let the algorithm run for $2R$ rounds. If no overflow buffer exceeds

the bound $2MK\phi(S)/\phi'(S)$, then we know that the average fraction of commodity transported through the network is at least one-half by theorem 1.

Notice that if demands λQ_c can be satisfied with constraints E_i/T, then scaling all the flow values by $1/\lambda$ shows that demands Q_c can be satisfied with constraints $E_i/\lambda T$. Hence if demands $Q_c/2$ are being satisfied on average with constraints E_i/T, then we know that demands Q_c can be satisfied with constraints $2E_i/T$, and so $T^* \geq T/2$. Double T and run for another $2R$ rounds. We repeat this process until some overflow buffer bumps up against the $2MK\phi(S)/\phi'(S)$ upper bound, say when $T = T_1$. At this point we know that

$$(1+\epsilon)T_1 \geq T^* \geq \frac{T_1}{4}$$

The upper bound is because if demands $(1+\epsilon)Q_c$ had been feasible with constraints E_i/T_1, then by lemma 2, the overflow buffers cannot exceed $2MK\phi(S)/\phi'(S)$.

We stop the algorithm as soon as an overflow buffer hits its bound, even if the full $2R$ rounds have not been executed. Because of this, the maximum overshoot that can happen in an individual buffer is one unit, corresponding to a potential overshoot of $\phi'(S)$. To get the potential back under the bound $2MK\phi(S)$, we run a round of the algorithm without injecting any new packets at the sources that overshot. Suppose that K' of the source overflow buffers exceed their bounds. Then the decrease in potential that this round produces will be at least $K'\phi'(S)$, hence will be sufficient to reduce the potential below $2MK\phi(S)$.

The power consumed at node i during this process of finding an upper bound on T^* is at most

$$2E_iR\left(\frac{1}{T_0} + \frac{1}{2T_0} + \cdots\right) \leq \frac{4E_iR}{T_0}$$

Now that we have upper and lower bounds, we will perform a search to determine the value of T^* more accurately. The naïve approach is to use a bisection search, running the algorithm for R/ϵ rounds to cut the search interval approximately in half. But a more intelligent technique cuts the search time significantly. If T_h and T_l are the current upper and lower bounds (initially $T_h = T_1$ and $T_l = T_1/4$), pick a test value $T = (2T_h + T_l)/3$, a third of the way from T_h to T_l, and run the algorithm for R/δ rounds, where $\delta = (T_h - T_l)/3T$. The idea is that if no source overflow buffer exceeds its bound within these many rounds, then by theorem 1 we have

$$(1+\epsilon)T_h \geq T^* \geq T(1-\delta) = \frac{T_h + 2T_l}{3}$$

and if some buffer does exceed its bound, then

$$(1+\epsilon)T \geq T^* \geq T_l$$

In either case, the search interval is reduced by a factor of $2/3$. The excess power consumed at node i (over the ideal E_i/T^*) during these R/δ rounds is at most

$$\frac{R}{\delta}\left(\frac{E_i}{T} - \frac{E_i}{T^*}\right) \leq \frac{3E_iR}{T^*}$$

The number of rounds spent during this search process is at most

$$R\sum_{i=1}^{\lceil \log_{2/3}\epsilon/3\rceil}\left(4\cdot\left(\frac{3}{2}\right)^{i-1}+1\right) \leq \frac{25R}{\epsilon}$$

and the total excess power consumption at node i is at most

$$\frac{3E_iR\log_{3/2}(3/\epsilon)}{T^*}$$

The total number of rounds spent to converge to an accurate value of T^* is hence

$$R\left(2\log\frac{4T^*}{T_0} + \frac{25}{\epsilon}\right)$$

During the remaining rounds, the power consumed per round is at most $E_i/(1-\epsilon)T^*$. Hence if

$$\frac{4E_iR}{T_0} + \frac{3E_iR\log_{3/2}(3/\epsilon)}{T^*} \leq \epsilon E_i$$

then the node will last for at least $(1-2\epsilon)T^*$ rounds. Hence if we pick $T_0 = 8R/\epsilon$, and if

$$T^* \geq \frac{6R\log_{3/2}(3/\epsilon)}{\epsilon}$$

then the remaining life of the node is at least $(1-2\epsilon)T^*$.

Theorem 4 *There exists a routing protocol that achieves a lifetime of* $(1-2\epsilon)T^*$*, if*

$$T^* \geq \frac{48ML\log_{3/2}(3/\epsilon)(K+\ln(6K/\epsilon))}{\epsilon^2}$$

For a given value of T^*, we should pick

$$\epsilon = \Theta\left(\frac{ML}{T^*}\left(K + \log\frac{T^*}{ML}\right)\right)^{\frac{1}{2}}$$

to get the best possible bound. (Notation: $f = \Theta(g)$ means that there exist constants c_1 and c_2 such that $c_1 g \leq f \leq c_2 g$.)

V. Implementation issues

There are three issues with implementing the algorithm as we have presented it so far: one is that thus far we have been treating the flow as a continuous quantity. However, in routing, we don't want to split up packets. So we must discretize the queues so that they contain an integral number of packets, and we must route an entire packet at a time. The second is computing the flow values that minimize the potential function at a node. The third is how to converge to the maximum lifetime in a distributed manner rather than in the centralized fashion we have analyzed.

Another problem that occurs in practice is that traffic does not arrive in the smooth, deterministic manner we have assumed so far, but may be intermittent or bursty in nature. To deal with this, we add a *leaky bucket* at every source to smooth out the traffic.

A. Discretizing flow

We modify the node rebalancing phase of the algorithm to rebalance the queues while maintaining an integer number of packets in each queue. This implies that the actual height of a queue might differ by up to one packet from its ideal height.

This introduces an error term into the analysis of the algorithm. While deriving the drop in potential by routing a flow f_e^c across an edge e, we obtained the expression

$$\sum_{e,c} f_e^c(\phi'(q_{et}^c) - \phi'(q_{eh}^c)) + \text{second derivative terms}$$

The first term might differ from its ideal value (when the q's are allowed to vary continuously) by

$$\sum_{e,c} \frac{f_e^c}{Q_c}(\phi''(q_{et}^c + \eta_{et}^c/Q_c) + \phi''(q_{eh}^c + \eta_{eh}^c/Q_c))$$

where η_{et}^c and η_{eh}^c are between -1 and 1. This sum is at most

$$\sum_{e,c} \frac{f_e^c}{Q_c}(2\alpha^2 e^{\alpha S}) \leq 2(1+\epsilon)L\frac{\epsilon}{4L}\phi'(S) \leq \epsilon\phi'(S)$$

So this does not significantly affect the analysis of the algorithm.

B. Minimizing potential

The problem we have to solve at each node is

$$\text{Minimize } \sum_{j,c} \phi(q_{ij}^c - f_{ij}^c) + \phi(q_{ji}^c + f_{ij}^c)$$

subject to

$$\sum_{j,c} e_{ij} f_{ij}^c \leq \frac{E_i}{T}$$

Using the technique of Lagrange multipliers, the f_{ij}^c satisfy the system

$$\phi'(q_{ij}^c - f_{ij}^c) - \phi'(q_{ji}^c + f_{ij}^c) = \lambda e_{ij}$$

So we compute the maximum of the quantities

$$\frac{\phi'(q_{ij}^c) - \phi'(q_{ji}^c)}{e_{ij}}$$

(which must be equal when we reach the minimum potential solution) over the edges ij and connections c and route one packet for that edge and connection. We update the queue heights and repeat the process until we exhaust the energy budget E_i/T.

C. Distributed implementation

The core of the algorithm (described in section III) is easy to implement in a distributed manner. The difficult part is defining a distributed protocol for doing the work of maximizing the lifetime (section IV).

The idea will be that each source sends out a broadcast (forwarded to each neighbour along with data packets) whenever its overflow buffer hits the bound $2MK\phi(S)/\phi'(S)$. Each node will receive the broadcast in at most N rounds.

In the initial phase, every $2R$ rounds, every node doubles its local value of T. If during the next N rounds, it receives a broadcast message from a source, it switches into the next phase, and forwards the broadcast.

In the second phase, each node maintains the upper and lower bounds T_h and T_l, and computes the test value $T = (2T_h + T_l)/3$ to determine its local energy budget. If after $R/\delta + N$ rounds, no source broadcast is received, the value of T_l is updated to $(T_h + 2T_l)/3$. If a broadcast is received, the value of T_h is updated to T. This continues until the value of T^* is determined (to within relative error ϵ).

The protocol executed at each node can be summarized as:

- *Initialize* Set $T = T_0$.
- *Determine* T_h Run core algorithm until $2R + N$ rounds pass, a broadcast is received, or (if the node is a source) the overflow buffer hits $2MK\phi(S)/\phi'(S)$. If $2R + N$ rounds have gone by, double T and repeat this step, otherwise move to the next one.
- *Broadcast* If a broadcast message was received, forward it to our neighbours, else generate a new one.
- *Search initialization* Set $T_h = T$ and $T_l = T/4$.
- *Search* Compute $T = (2T_h + T_l)/3$, $\delta = (T_h - T_l)/3$. Run the core algorithm until $R/\delta + N$ rounds pass, a broadcast is received, or (if the node is a source) the overflow buffer hits $2MK\phi(S)/\phi'(S)$.
- *Update* If $R/\delta + N$ rounds have passed, set $T_l = (T_h + 2T_l)/3$. Otherwise, set $T_h = (2T_h + T_l)/3$ and either forward the broadcast or generate a new one. If $(T_h - T_l)/T_l \geq \epsilon$, go back to the *Search* step.

The analysis of this distributed protocol is similar to the centralized one and will lead to the same bounds except for a small constant factor, since N is small compared to R.

D. Leaky bucket

A leaky bucket is a controller used in network applications to smooth out traffic flow. The idea is that ρ tokens are generated per second, and tokens are dropped if there are more than σ of them. Each packet transmission consumes one token. If tokens are not available, the packets are buffered till they become available. Such a controller limits the traffic transmitted: the number of packets transmitted in a time interval T can never exceed $\rho T + \sigma$.

In our application, we use the special case where σ is zero, so that the traffic is smoothed out completely.

E. Acceleration of the algorithm

There are some issues with the algorithm as described above, principally that the delay can grow unexpectedly fast. On a route of length L, the delay a packet faces will grow as L^2 because the queues decrease linearly from source to sink (after convergence to the appropriate flow). To both reduce this delay and to accelerate convergence, the algorithm can be modified using ideas from [17] and [18]. The flow to be sent on an edge is modified by using some history, that is, the actual flow is computed as a linear combination of the flow that the original algorithm (or the *first-order* method) chose,

and the flow that was sent on the edge in the previous step. For a suitable choice of the coefficients of the two components, the convergence can be accelerated in this new *second-order* method. If the coefficient of the flow in the previous step is close to 1, then the final queue heights will be smaller than in the first-order method, thus reducing delay.

The actual flow in the second-order method is thus calculated as

$$f = \alpha f_0 + \beta f'$$

where f_0 is the flow given by the first-order method, f' is the flow sent in the previous step, and α and β are suitable parameters. The convergence can be significantly accelerated when $\alpha + \beta > 1$. The closer β is to 1 and the larger the value of α, the smaller the final queue heights and the faster the convergence. However, if α is too large, the algorithm becomes unstable.

An exact analysis can be done for the simple case of a path graph, which shows that the second-order method is stable as long as $\alpha < 1 + \beta$.

This method may call for flow values that are greater than the actual queue heights. There are two methods to deal with this, one consisting of simply sending the maximum possible when this happens, and ignoring the rest. Alternatively, the excess flow called for can be remembered and sent during subsequent time steps if possible.

F. Alternative implementation and simulation results

For the simulation, the graphs were generated by uniformly generating points in a square area, 5 units by 5 units. Nodes were given a random initial energy, uniformly distributed on the interval $[10000, 30000]$. Transmission cost was set equal to half the square of the distance, with a cut-off if the square was larger than 10. The number of connections was set to 1, with a demand of 1.0 for simplicity. Performance was measured by first computing the maximum lifetime using the algorithm from [13], running both the first- and the second-order methods for the lifetime number of rounds and noting how much flow actually reached the sink. In addition, we also tested the algorithm with known lifetime (without the lifetime-determining phase).

We actually implemented the algorithm in a simplified and more practical form. Firstly, the potential function was taken to be just the square of the queue heights rather than the exponential function. This makes determining the flows that minimize the potential function much simpler. With known lifetime, the algorithm performs very well, usually delivering over 97% of the flow.

Secondly, instead of using a distributed protocol to search for the true lifetime, the potential function was modified by adding a term proportional to the square of the cost of the flow, *i.e.*,

$$\left(\frac{\alpha}{E_i}\sum e_{ij}^c f_{ij}^c\right)^2$$

This works reasonably well as long as the constant α is chosen appropriately.

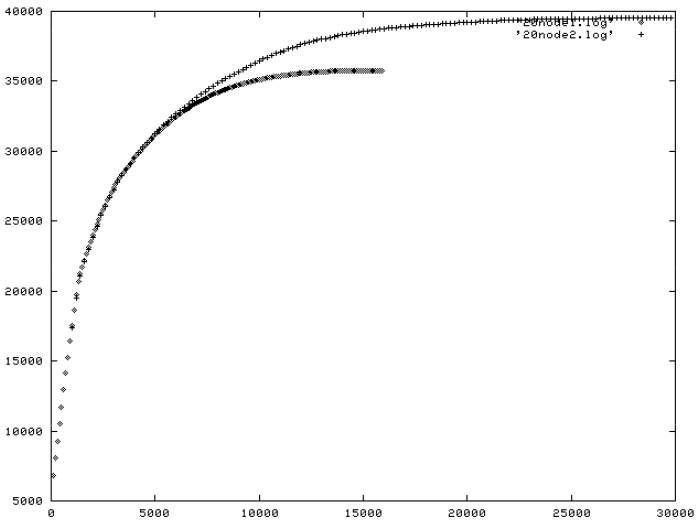

Fig. 1. Total flow reaching sink.
x-axis: α *y*-axis: Total flow

In figure 1, the total flow delivered in time equal to the optimum lifetime is plotted against the value of α used. The upper graph shows the second-order method with $\beta = 0.95$. The lower graph is the first-order method. The graph was generated on 20 nodes and the optimum lifetime of the network was determined to be approximately $43,000$.

The simulation shows that choosing α appropriately is largely a matter of making sure that it is not too small. Larger values reduce the total flow delivered, but only slightly. Further theoretical work is needed to determine how to choose both α and β appropriately, or perhaps even vary them with time.

VI. Extensions

The algorithm actually works for a wide variety of different constraints, either on the nodes or on the edges or even both.

From the proof of the core algorithm, it follows that it will work for any combination of local constraints (both nodal as well as edge capacity constraints) as long as there exists a flow satisfying demands $1 + \epsilon$ and meeting all the constraints. These constraints need not even be linear, as long as we are able to do the potential minimization.

A. Idle power

Typically, communication is not the only energy consumer in a wireless node. The node would be doing some computation or data collection which consumes power at a rate independent of the number of packets it transmits. Also, wireless interfaces consume power even when idle and not receiving or transmitting any packets. To take these factors into account, we can add a constant power consumption C_i to the node energy constraint, which becomes

$$\sum_{j,c} e_{ij} f_{ij}^c \le \frac{E_i}{T} - C_i$$

The core algorithm is not affected at all by this change. The only thing that we have to verify in order to show that the lifetime maximization protocol will still work is the scaling assumption that if demands λQ_c can be satisfied with lifetime T, then demands Q_c can be satisfied with lifetime λT. This follows because if flow values f_{ij}^c achieve demands λQ_c, then the flow f_{ij}^c/λ will satisfy Q_c, and

$$\sum_{j,c} e_{ij} \frac{f_{ij}^c}{\lambda} \leq \frac{E_i}{\lambda T} - \frac{C_i}{\lambda} \leq \frac{E_i}{\lambda T} - C_i$$

provided $\lambda \leq 1$, which is true whenever we apply this result ($\lambda = 1 - \delta$ in section IV).

B. Periodic recharge

An interesting case is when some nodes are recharged periodically, say every T_i seconds. In this case the right hand side of the constraint for this node can be replaced with E_i/T_i, since the node only has to survive till its next recharge. This node does not participate in the lifetime-determining phase of the protocol, except for forwarding control packets.

C. Edge constraints

One can also incorporate edge constraints, for example, bandwidth constraints. At each node, in addition to the energy constraint

$$\sum_{j,c} e_{ij} f_{ij}^c \leq \frac{E_i}{T}$$

we would have additional constraints

$$f_{ij}^c \leq B_{ij}$$

where B_{ij} is the maximum possible rate at which packets can be transmitted on link ij, perhaps due to signal-to-interference ratio (SIR) requirements.

It may also be necessary to modify the energy constraint to allow non-linear power dependences. For example, there may be an initial startup cost associated with using the transmitter or for setting up the wireless link with the node at the other end, which may be a significant fraction of the total power required.

These modifications make the potential minimization problem at the nodes more complicated, and more sophisticated numerical algorithms may be required. Also, the lifetime maximization protocol needs to be re-analyzed for each particular case, because the scaling assumption is no longer valid.

D. Receive power

So far we have ignored power required to receive packets at a node. In real life, a transciever consumes power when operated in receive mode as well as in transmit mode. If the network is fairly static or slowly evolving, then this can be taken into account by charging it when the packet is actually forwarded instead of when it is received. Since all but a bounded number of packets at any node will be forwarded, we simply add the energy required to receive a packet to the energy required to transmit it in the node constraint, which takes the form

$$\sum_{j,c} (e_{ij}^t + e_{ij}^r) f_{ij}^c \leq \frac{E_i}{T}$$

where e_{ij}^t is the energy per packet for transmission, and e_{ij}^r is the energy per packet for reception and decoding.

E. Dynamic network

The core algorithm described in section III will work even if the network is dynamic and edge costs are varying with time, as long as it always true that there exists a multicommodity flow satisfying the demands Q_c.

However, because of the changing edge costs, it is possible that the lifetime may change. If any source overflow buffer hits its bound, then the current value of T is too high, and a new search can be begun using the distributed protocol with $T_h = T$ and $T_l = 0$. If the current value of T is too low, this will not be detected unless we re-initiate the entire lifetime maximization protocol. This can be done after at least R/ϵ rounds have gone by, using for the starting value T_0 the current value of T. Waiting for this many rounds ensures that we transmit $1 - \epsilon$ fraction of packets to the appropriate destinations. So if the rate of change in the network is slower than this, we will be able to track the lifetime as the network evolves.

VII. Conclusion

In power-limited wireless ad-hoc networks, battery power is an important consideration to take into account while picking routes. In this paper, we have proposed and analyzed a distributed routing algorithm for ad-hoc networks where the goal is to maximize the network lifetime. We have been able to establish a performance guarantee for this algorithm. This is a significant theoretical improvement over the heuristic algorithms previously proposed. We have shown that our algorithm works for both static networks as well as networks where the edge costs are varying slowly. The protocol can also be modified to take into account different constraints, of which some examples were discussed. It is also possible to handle slowly changing dynamic ad-hoc networks.

References

[1] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing* (T. Imielinski and H. Korth, eds.), ch. 5, pp. 153–181, Kluwer Academic Publishers, 1996.

[2] S. Murthy and J. J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *Mobile Networks and Applications*, vol. 1, no. 2, pp. 183–197, 1996.

[3] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. IEEE INFOCOM*, 1997.

[4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *Proc. of the* 3^{rd} *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 48–55, 1999.

[5] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proc. of the* 2^{nd} *IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, February 1999.

[6] S. Singh, M. Woo, and C. S. Raghavendra, "Power-aware routing in mobile ad hoc networks," in *Proc. ACM/IEEE Int. Conf. on Mobile computing and networking*, pp. 181–190, October 1998.
[7] V. Rodoplu and T. H. Meng, "Minimum energy mobile wireless networks," in *Proc. IEEE Int. Conf. on Communications*, pp. 1633–1639, June 1998.
[8] I. Stojmenovic and X. Lin, "Power-aware localized routing in wireless networks," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, May 2000.
[9] R. Ramanathan and R. Rosales-Hain, "Topology control of multihop wireless networks using transmit power adjustment," in *Proc. IEEE INFOCOM*, pp. 404–413, March 2000.
[10] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang, "Distributed topology control for power efficient operation in multihop wireless ad hoc networks," in *Proc. IEEE INFOCOM*, April 2001.
[11] J.-H. Chang and L. Tassiulas, "Routing for maximum system lifetime in wireless ad-hoc networks," in *Proc. of* 37^{th} *Annual Allerton Conference on Communication, Control, and Computing*, September 1999.
[12] J.-H. Chang and L. Tassiulas, "Energy conserving routing in wireless ad-hoc networks," in *Proc. IEEE INFOCOM*, pp. 22–31, March 2000.
[13] J.-H. Chang and L. Tassiulas, "Fast approximation algorithms for maximum lifetime routing in wireless ad-hoc networks," in *Lecture Notes in Computer Science: Networking 2000*, vol. 1815, pp. 702–713, May 2000.
[14] N. Garg and J. Koenemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *Proc.* 39^{th} *Annual Symposium on Foundations of Computer Science*, pp. 300–309, November 1998.
[15] B. Awerbuch and F. T. Leighton, "A simple local-control approximation algorithm for multicommodity flow," in *Proc. IEEE Symposium on Foundations of Computer Science*, pp. 459–468, 1993.
[16] B. Awerbuch and F. T. Leighton, "Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks," in *Proc. ACM Symposium on Theory of Computing*, pp. 487–496, 1994.
[17] S. Muthukrishnan and T. Suel, "Second-order methods for distributed approximate single- and multicommodity flow," in *Proc. RANDOM*, pp. 369–383, 1998.
[18] S. Muthukrishnan, B. Ghosh, and M. H. Schultz, "First- and second-order diffusive methods for rapid, coarse, distributed load balancing," *Theory of Computing Systems*, vol. 31, pp. 331–354, 1998.