The RatNest Routing Protocol for Ad-Hoc Circuits Over Fixed Radio Networks

Scott A. Mitchell

***Abstract*— We propose the RatNest protocol for low-overhead ad-hoc routing over a small wireless radio network in support of certain novel communication patterns and nodes. Nodes have fixed position in an urban environment and have up to five dedicated wireless radio channels to nearby nodes. Links vary in quality and reliability. The protocol provides for the quick establishment and repair of routes to support circuit-like communication of both streaming video and sensor-chirps. As in link-state routing, our protocol stores a graph representation of the network at each node, annotated with transient information about the network state. Transient information is gathered using both pro-active and reactive mechanisms. To establish a circuit, a shortest-latency or highest-throughput path is computed locally on this graph, then the route is locally adapted and verified on the actual network. In this paper we focus on the main algorithms of the protocol and some analysis. We do not report simulation results in this paper, but we have implemented a prototype in C++ and integrated it into OPNET Modeler.**

***Index Terms*— Communication system routing**

I. Introduction

We propose the RatNest protocol for ad-hoc communication over a small radio network. Our scenario of interest is non-traditional for ad-hoc networks, but solves a particular scenario of military surveillance in foreign urban environments. The application network is relatively small, comprising a few hundred nodes. The dynamics of the network are limited. The nodes are non-mobile, but fail and recover occasionally. Nodes have essentially unlimited power, but limited compute capabilities. Nodes are sparsely arranged in an urban environment, say at street corners; see figure 1. Links communicate through line of sight; mainly along streets or urban canyons. Each node has a dedicated radio link to each of its neighbors, but it only has a few (up to 5) neighbors. Link quality varies slightly, so some links are preferred over others. We weight links inversely proportional to bandwidth, and weights are bounded between 1 and 4.

The communication pattern is that of long-lived circuits. Circuits are requested by a central (human) controller, are time-sensitive, and have no predictable pattern. Each circuit transmits either high-bandwidth streaming video/audio, or low-bandwidth (but latency-sensitive) sensor chirps. We consider two metrics for paths: for sensor chirps we use path length, the sum of link weights along the path; for streaming data we use bottleneck, the maximum weight over all links along the path.

The structured society of naked mole rat colonies provides some metaphors and inspiration for RatNest, but we are not "bio-inspired"[12] in the sense of accurately implementing in software specific natural activity. Nodes of the network are called "nests," after the hub-like nests of naked mole rats. Unless otherwise stated, features described in this paper have been implemented in a C++ prototype integrated with OPNET.

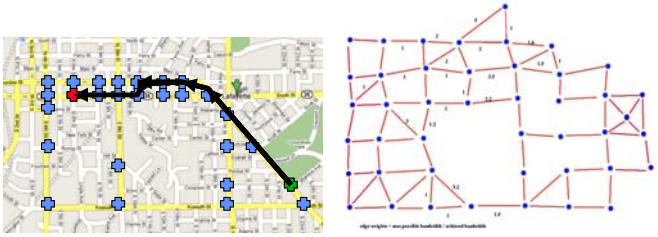

Fig. 1. Right, circuit in a sample of an urban-canyon network. Left, an induced graph G with a few edge weights labeled.

A. Protocol Goals

In our urban surveillance scenario it is important to establish a circuit quickly and reliably maintain it in order to monitor and record an activity of interest. The following describes the prescribed goals and derived requirements, in priority order:

1. Low routing overhead, especially when data are in hand.
   - Only a few bits for establishing routes, a few bits of message overhead for maintenance.
2. Low memory and computation for each nest.
   - *Messages are not queued.* If a circuit can not be established (or was broken and cannot be repaired) then packets are dropped.
   - Little memory for storing routing tables, graphs, etc.
   - A simpler and faster routing engine is preferred.
3. High transmission data-rates (good circuits).
   - High bandwidth for streams, low latency for sensors.
4. Route repair or "self-healing."
   - In the event of a nest failure, find an alternate route. Since packets are dropped while the route is repaired, repairing the route quickly and locally is more important than finding the best-metric route.

In addition, a very simple protocol is desired for

Manuscript received March 31, 2009.

Scott A. Mitchell is with Sandia National Laboratories, MS1316, PO Box 5800, Albuquerque, NM 87185 USA (phone: 505-845-7594; fax: 505-845-7442; e-mail: samitch@sandia.gov).

verification of reliability. These goals compete, so the protocol design seeks reasonable tradeoffs among them. The RatNest protocol is tailored to the strengths and weaknesses of the nests' hardware.

B. Our Scenario Compared to Other Settings

Our setting has some similarities to traditional sensor networks, but some key differences. Node failure patterns and the goal of low routing overhead are similar. Network layout, and the lack of GPS position and time information is similar to a deliberately placed (non-scattered) sensor network. However, our nodes are much more compute-capable and have essentially unlimited power. Our setting includes high bandwidth data as in other recent multimedia surveillance sensor networks[1], but also low-latency few-bits messages, and time-sensitive human-selected circuit establishment. Circuits are of interest in telecommunications, but our network is different: our network is more dynamic, smaller, and simpler; nodes are less capable; links have less (routing) bandwidth. Also the fraction of the network being used for active circuits is much less so it is undesirable to pre-compute a global schema of collision-avoiding routes because individual routes would be inherently sub-optimal.

Our setting also has some similarities to traditional wireless radio networks, but some key differences. The main difference is that nodes are fixed in our setting, which motivates graphs rather than routing tables. There has been some recent interest in streaming video over MANETs[5] (Mobile Ad-Hoc Radio Networks), so our interest in circuits is novel but not unique. Centralized control, and the high-consequences for data collection, allows the RatNest protocol to ignore some issues of route admission. In contrast to both traditional sensor and wireless settings, our nodes have multiple dedicated directed channels.

II. Concepts and Definitions

A. Definitions

P = a directed circuit path from source S to destination D over intermediate nests T_i, i.e. P = {S=T_0, T_1, … T_k=D}

L = the number of nests in P, i.e. k when P = {T_0, T_1, … T_k}

N = number of nests in the network, between 100 and 1000.

G = persistent graph of the network: nodes are all the nests, directed edges are radio links between pairs of nests. This graph is fixed regardless of whether any nest is actually up.

H = transient graph of the network: G annotated with (possibly out-of-date) information X on which nests are currently up, and which links are in use by established circuits.

X = set of transient network features in a packet used to update H. Either a nest is up or it is down. Either a link is dedicated to an established circuit or it is available.

Blinking = when a nest fails it "blinks off" and when it recovers it "blinks on." (Naked mole rats are nearly blind.)

Rat = a packet containing X, that traverses the network. Nests receive, interpret, modify, and forward rats.

B. Local Graphs G and H

Since nests are non-mobile, we chose for each nest to maintain a graph model of the network in its local memory, as in link-state routing, e.g. OSPF[2]. This model includes the graph *G* of all nests of the network, with edge weights related to the bandwidth of each link. This "persistent" graph *G* is annotated with transient information about which nests are currently blinked on or off, and which links are currently dedicated to any circuit, to produce the transient graph *H*. (The nest keeps detailed information about the entire circuits through its own links, but to reduce memory requirements it does not store which circuit is using which edge for non-local links.) Graphs *G* and *H* require about the same memory as distance-vector routing tables. Competing paradigms that exploit geographic information[4], and also hierarchical routing protocols with implicit address capabilities, are able to use less memory; but our scenario does not have that structure. Constructing virtual geographic coordinates[9] or overlay networks[4] is possible but would overly complicate our algorithms and does not seem well suited to our traffic patterns. The graph model is size O(*N*), the same order as the minimal amount of information needed to store just the existence (address) of nests in the graph. The geographic persistence of our network, and the nature of the transient information makes the graph much more appealing than in most highly mobile ad-hoc scenarios in the literature.[10]

1) Time Stamping

Transient information on *H* may be out of date. The transient information comes pro-actively from rats and re-actively from route validation and other feedback. While the nests have a local clock for time-outs, there is no global clock, no GPS. We consider two methods of dealing with the asynchrony of graph information.

The first method is "overwrite," and simply assumes that the last-received information is most accurate, and overwrites *H* whenever new information is received. This is the simplest approach and performs well enough because the network is relatively static on the time-scale of route validation, repair, and pro-active and event-driven rats; and these all have very local information based on probing the status of a nest's neighbors.

The second method is "sequence-numbering:"[6] nests assign a sequence number to any packet they send out, and a receiving nest can use it to tell whether received or stored information is more current. Links are unidirectional, so an edge is owned by a unique nest and its sequence numbers are consistent. When a nest fails and recovers it must gather its old sequence number from its neighbors' memories. This more complicated approach is required when the network is changing rapidly compared to the amount of protocol traffic. ("Sequence-numbering" is not implemented in the prototype.)

III. RatNest Protocol

One main algorithm of the protocol is circuit establishment, which includes path computation, adaptation, and updating H. We also describe route repair, which is similar to path adaptation. Network initialization is fairly standard and relatively unimportant in our application. Packet acknowledgement and route dismissal are standard, modulo some time-out values which depend on whether streaming video or sensor chirps are expected, so these are omitted for brevity. We describe a couple of pro-active strategies for keeping H up to date.

A. Network and Persistent Graph Initialization

When the network is deployed, nests discover their neighbors up to two hops away and dedicate channels to avoid radio interference. After a delay, nests flood the network with a packet describing its ID (address) and links. Using the radios in promiscuous mode, rather than as dedicated channels, speeds this process. Each nest builds G from these messages.

B. Circuit Establishment:

A distinguishing feature of RatNest is that each source nest first computes the desired (shortest) path locally on its transient graph *H*, rather than going directly to the network. That is, the initial desired path is fully source routed as in DSR.[4] The source nest then seeks to validate this route in the actual network; there may be competing circuits or blinked-off nests that the source does not know about. The protocol attempts to deal with these problems locally. Some data bandwidth is reserved for routing protocol traffic, so that a nest fully participates in all rat and route traffic, even if its links are reserved or participating in a circuit, unless the nest itself is off. (Circuit establishment with the detour variation is fully implemented. The fork variation is not implemented.)

1) Path Computation

A source nest S is told by the central (human) controller to establish a circuit to a given destination nest D. It computes a shortest weighted path to D, $P = \{S=T_0, T_1, \ldots T_k=D\}$, over its local graph H,. Here "shortest" means either highest-bandwidth or lowest-latency, and is also specified by the controller. The computed path may be sub-optimal, because of stale information in H. (E.g. if a link is thought to be in another circuit, then it is not part of H.) S then attempts to establish this path in the "real" network. Beginning with S, a nest T_i sends a route validation packet with the desired path to the next nest T_{i+1} on the path. Nests reserve their requested link. If D is reached the circuit is established and two return route acknowledgement messages are sent backward along the route from D to S. The first message is lightweight, containing only the loop-free circuit that was established. This is done in order to establish the circuit as quickly as possible. The second message is heavier weight, containing transient graph information used to update the H of each nest.

2) Path Adaptation

It may be that a nest T_i can not establish the prescribed path to nest T_{i+1} because either T_{i+1} has blinked off or the directed link from T_i to T_{i+1} is already part of an established circuit for a different (S, D) pair; in either case we say that T_i has encountered a problem X and must attempt to find an alternative route. The alternate route is always the concatenation of the sub-route of P from S to T_i, plus a new shortest route from T_i to D. This route is likely to be longer than the shortest route that the source could compute if it had known about X ahead of time. Hence this protocol represents a choice of low overhead and local adaptation (robustness) over getting the best-metric routes, which is consistent with our protocol goals. The protocol does not burden forward communication with the overhead of transient information until a problem is encountered, and then only with the particular problems X encountered. This reduces overhead and speeds up circuit completion. See figure 2.

a) Path Adaptation Variants Detour and Fork

We consider two path adaptation variants, called detour and fork. Both variants accumulate and forward certain problems X. If the problem is a competing circuit reservation for a link, all links of the competing circuit are accumulated in X. (Recall this information is available because a nest keeps detailed information about the circuits it is a part of, namely the entire route.) This transient information may be useful downstream right away, as these nests might need to re-route and avoid those X's. Non-problem transient information is updated only on route acknowledgement.

Detour. T_i computes a new shortest path from itself to D using its H updated with any X's in the route validation packet. It updates the route validation packet with the new path P′, appends X, and forwards the packet to T'_{i+1}. The protocol continues as before.

Fork. T_i forwards a copy of the route validation packet (along with X) to *each* of its blinked-on neighbors; T_{i-1} is skipped unless it is the only blinked-on neighbor. Each neighbor computes a new shortest path to D and the protocol continues as in the detour variant. If another problem X is hit, the packet detours but does not fork again. If a packet hits a nest that another fork already visited, the packet attempts to detour around that nest and so establish a nest-independent alternate path to D. The first route validation packet to reach D "wins" and generates a route acknowledgement. ("Fork" is not implemented; given the extra overhead it would only be worth doing in niche contexts.)

b) New Path Features

It is possible that the shortest path (perhaps the only path) from an intermediate nest to the destination is to revisit prior nests and re-trace prior hops, e.g. $T'_{i+1} = T_{i-1}$. This is allowed temporarily in order to keep the circuit request progressing quickly. But care is taken so that it does not degrade the final path: backtracks and other types of loops are pruned from the circuit when the destination is reached. Note that retracing is

not stymied by competing circuit reservations along the links in the backwards direction, because a nest can determine that a retraced link will eventually be pruned and allow the route validation traffic to temporarily use it. See figure 2.

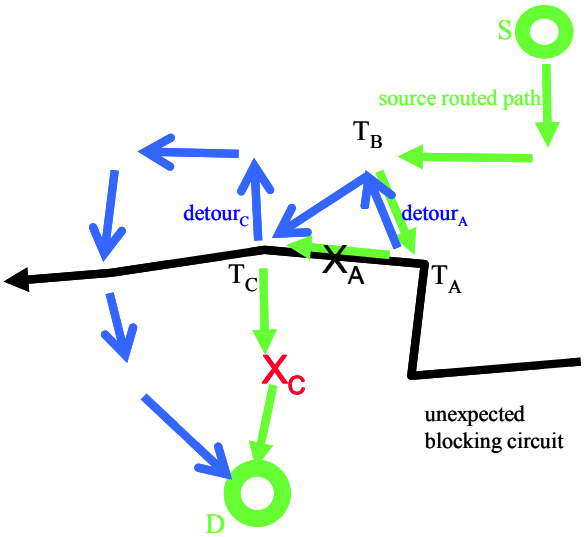

Fig. 2. Detour example. Source S does not know about the unexpected blocking circuit, so source-routes using one of its links. At T_A the blocking link X_A is discovered and $detour_A$ is selected, which goes through T_B to T_C and then continues on the original path. At T_C it is discovered that nest X_C is unexpectedly blinked off. So $detour_C$ is selected, which avoids both X_C and all the links of the unexpected blocking circuit. The route validation packet reaches D. The first link of $detour_A$ is pruned from the route and a route acknowledgement is sent back along the other links from D to S.

c) Recourse

It may be that there is no route on H from a nest to D. In that case, the nest attempts a "recourse" strategy. Each nest may attempt a recourse strategy only once for a given circuit request. The nest attempts to find a path on a modified graph H′ which has more available edges and nests than H. Here H′ is G annotated with just the accumulated problems X and any other information ensured to be current. (E.g. It is ensured that all the nests upstream in the traverse are blinked-on, and the active circuits through this nest make some edges unavailable. But it is uncertain whether circuits elsewhere in the network have been dismissed, and whether remote nests have blinked off, so these features are not included in H′.) If there is a recourse route, it is validated on the real network as before. Otherwise, the nest sends circuit acknowledgement messages back to the source saying there is no route. To keep overhead low, only a few, O(log(L), nests attempt a recourse.

d) Termination

This route validation strategy works well in practice for our scenario. In addition, the protocol is provably guaranteed to terminate, although the theoretical bounds are not optimal. Each computed path is finite, O(N). A new path is only computed when the route validation packet goes through a nest that has a neighbor that is unexpectedly down, or a link that is unexpectedly reserved for another circuit. Each nest has only a few neighbors, 5, so the number of reroutes is also O(N). Hence the number of messages sent throughout the network is $O(N^2)$, and each message is O(N) in size. (The size of a message is the length of the path plus the size of accumulated X's.) In practice the number and size of messages are usually much smaller. Initial paths are usually shorter than the diameter of the graph. Even when competing circuits completely block the destination from the source, usually only a handful of reroutes are attempted, and they are usually only a few hops long in an attempt to get around a single nest. In our scenario only 10% of the nests are likely to be down at any given time. Hence nest-blinked-off blockages X are rare and sparse, and it is likely that such a problem can be re-routed around locally. Note also that nests do not "flicker", meaning that if a nest is down or up, it stays in that state longer than the time needed to establish a route. Hence it will switch states at most once and even that with low probability. In addition, for large networks, we add a maximum hop count to the packet.

e) Failure

It may be that there is no path (in either H or H′) from a nest T_i to D. In this failure condition, the nest returns two route validation packets with a failure message, using the same mechanisms as described below. It sends these to its "spawning nest." In the detour variant, the spawning nest is S. In the fork variant, the spawning nest is the one that generated the multiple forked messages. Since some other fork may have found a valid route to D, the spawning nest only sends a failure message to S if it receives a failure messages from all of its forks.

f) Pruning and Route Acknowledgement

The hoped-for case is that a route validation packet actually reaches D. In this case, D generates two route acknowledgement packets. The first is to quickly establish the circuit, the second to update H and clean up unneeded link reservations. The destination computes a pruned, loop-free circuit in the following simple way. The destination checks each nest of the loop, starting from the source, in sequence. If the nest appears more than once in the path, the portion of the path between its first and last occurrence is removed (pruned). The brute-force solution takes $O(L^2)$ time and O(L) space, where L is the length of the traverse taken from S to D. (Theoretical bounds are O(L log L). In the worst case L is $O(N^2)$ but in practice it is usually smaller than the diameter of the graph. The length of the circuit is O(N), since each nest appears at most once. The first acknowledgement packet is sent along the pruned circuit only, and contains just the pruned route and a message of "success." Each nest of the circuit stores the complete circuit, and forwards the acknowledgement to the next nest back towards the source. The source S is now ready to send data to D.

The simple pruning algorithm avoids some complicated choices in the case of a complicated set of loops as in figure 3.

After a delay, the destination sends the second acknowledgement packet, which contains the full traverse the

packet took from S to D, along with the pruned circuit, and the X problems accumulated during the forward traverse. Each nest updates its link reservations, removing any reservations it made during the traverse that are not in the pruned circuit. The first (or only) time a nest is visited on the backwards traverse, the nest gathers its knowledge of all circuits through it, and the blinked on/off status of all of its immediate neighbors; although these graph updates are not "problems," it appends them to X to be forwarded to the prior nest of the traverse. The second (or only) time a nest is traversed, these updates X are applied to H. The updates implied by the successful traverse and established (S, D) circuit are also applied to H: all the nests traversed were up, and, if the circuit was successful, the links of the circuit are now unavailable.

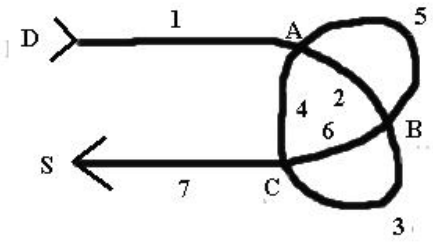

Fig. 3. A trefoil presents some interesting choices for loop pruning. Since nest C is the first nest visited twice from S to D, RatNest would prune at nest C by removing sub-paths 6, 5, 4. The resulting circuit is sub-paths 7, 3, 2, 1.

g) Reservation

Care is taken so that two incompatible routes (between different S/D pairs) are not established nearly simultaneously, and to avoid other race conditions. Intermediate nests T_i reserve requested links immediately upon receiving the first route validation packet, not waiting for the route acknowledgement (except during fork). This policy keeps negotiations and route acknowledgement simple, but requires that if the route is not established, or the route that is established detours around that link, the nest should be informed. Reservations time out if a route acknowledgement is not received, which also covers the rare case of a nest failure occurring between reservation and acknowledgement which cuts off the acknowledgement packet to upstream nests. In the fork variant, nests downstream of the fork set reservations only on route acknowledgement. This prevents too much of the network from being reserved for alternative routes that are never used, at the risk of non-reserved links being taken by other circuits. (Details of the mechanisms that handle this case are omitted for brevity.)

3) Route Repair

Once established, a circuit can not be interrupted by any other circuit request. However, a nest T_i (=X) on the circuit could blink off. In this case, a route repair is initiated. The main goal is to quickly re-establish the circuit; getting a short path is of secondary consideration. The nest prior to the problem, T_{i-1}, will be the first to notice the problem. T_{i-1} computes a new shortest path from it to any nest in the sub-path T_{i+1} to D; this shortest path is called a "patch," T_{i-1} is called the "patch head," the sub-path T_{i+1} to D is the "tail", and the nest of the tail that is the patch's destination is the "patch tail." The patch head then attempts to establish that path on the real network using the same basic mechanisms as for route validation. That is, nests may locally reroute. If loops in the whole circuit are created, which may occur even if the patch is loop-free, they are pruned on acknowledgement as before. If there is no path from T_{i-1} to the tail, then the protocol backs up two hops to T_{i-3} and attempts to use it as a patch head. Every time there is no route, the protocol backs up the patch head twice as many hops as before (i.e. 2, 4, 8, 16 ...) and attempts to reach the same tail. (In order to keep overhead low, recourse rerouting is only allowed if the patch head is S.) If it is determined that there is no route to the tail, or a route repair is not successful within a certain number of seconds (implemented, but the choice of the length of time has not been vetted), then the route is dismissed. If a route repair is successful, the patch tail prunes the new circuit and sends an updated description to all nests upstream and downstream. The segments of the old circuit that are circumvented are dismissed.

C. Shortest Path Computations

Shortest paths are computed using dynamic programming. When the metric is the sum of link weights, then this is Dijkstra's algorithm, as in OSPF[2][3]. The same basic dynamic program also works when the metric is the bottleneck, so implementing both options was easy. The only difference is what value to use when updating distances. Let $D_j(S,T)$ denote the intermediate stage j computation of the distance from S to T. When updating distances using link L of nest U, then for the bottleneck metric $D_{j+1}(S,T) = \max(D_j(S,U), \text{weight}(L))$, in contrast to the usual $D_{j+1}(S,T) = D_j(S,U) + \text{weight}(L)$ in Dijkstra. Recall that dynamic programming can compute the shortest path from S to *all* nests of the graph, so that the theoretical time-bound for shortest path computations for route repair is no more expensive than those for circuit initialization.

D. Pro-active transient information mechanisms

Pro-active mechanisms are not included in the C++ prototype.

1) Soldier Rats

A fixed pool of soldier rats (say N/10) pro-actively roams the network and update nests with transient information. Having a fixed pool of agents is a common bio-inspired strategy[12] with the following two desirable features: it is a relatively simple protocol and it transitions well between static and dynamic networks. By "transitions well", we mean: it requires no information about the rate of dynamics; it provides a lot of useful information if the network is relatively static; the information is less useful if the network is highly dynamic, but the overhead does not grow if the network is dynamic.

A soldier rat saves transient information about the last 20 nests visited and their links. For a given nest, it saves its blink state, the reservations of all the edges of all circuits through it,

and the blink state of its neighbors. At each hop, H is updated, then the soldier rat is updated by deleting the oldest information and adding the newest. We take care not to add redundant information, but update its age, which reduces protocol overhead at the expense of some computation.

On initialization a soldier rat picks a random destination D in G. The nest it is on computes a shortest path on G to D and the soldier rat deterministically follows that path until it reaches D or encounters a nest that has blinked off. Then it picks a new random destination. In this way, we suspect that soldier rats will preferentially spend time in the important parts of the graph, the areas that are on a lot of shortest paths, as well as on problem (dynamic) areas of the graph. (Alternatives include gossip-based strategies and ways to bias the search towards regions that have not been explored recently.[2])

A soldier rat may die (i.e. the nest it is on blinks off) or get isolated (i.e. the network becomes disconnected and part of the network has no soldier rats). These are rare but possible. To keep the population stable, if a nest has not been visited by a soldier rat in a given time, it generates a new soldier rat. This is another common swarm strategy.

2) Tunneling Rats

Tunneling Rats are event-driven rats. The events that generate a tunneling rat are route establishment, route dismissal, nest blinking on, and nest blinking off.

When a route is dismissed, each nest on the route generates one tunneling rat. If the nest received an explicit route dismissal packet, then the tunneling rat contains the information that all of the links of the circuit are now available. Otherwise it is dismissing the route due to a time-out, and the tunneling rat only knows that the links of that particular nest (and perhaps its neighbors) are now available. Each nest sends its tunneling rat to a (different) random nest in G, using the same mechanism as for soldier rats. A tunneling rat dies when it reaches its destination or encounters a down nest, but recall rats may still traverse across links in an active circuit using the protocol-reserved bandwidth.

A nest that has not received normal traffic from one of its neighbor for a long time performs a simple handshake to discover if the neighbor has changed state. When a nest blinks off, each of its neighbors will detect it and generate a tunneling rat with the fact that the nest is blinked-off. When a nest blinks on, it recovers G and its last sequence number from its neighbors, then sends an event-driven tunneling rat with the fact that it is now blinked on. These rats are sent out in a sparse broadcast as in the case of route dismissal.

IV. Conclusion

The RatNest protocol meets the objectives of a particular surveillance scenario. The protocol uses a mix of features borrowed from other protocols and adapted to this scenario's capabilities and needs. We have described the main algorithms and some analysis, focusing on the aspects that appear to have the most general interest and applicability. We believe that our annotated graph approach has certain advantages, and has no critical shortcomings in this particular setting where the dynamics of the network are limited, and we are not constrained by other layers of the protocol. Follow-on work may involve tuning protocol parameters and simulation studies. We speculate that naked mole rats, with their hierarchical societies, specialized roles, complex brains, and network-like habitat, may provide some bio-inspired algorithms (say for network discovery and maintenance) that are fundamentally different than the more well studied insect-inspired ones.[12]

Acknowledgment

Scott A. Mitchell thanks Brian Van Leeuwen, Tom Tarman, Cindy Phillips, and Hamilton Link for discussions regarding protocols and applications; Scott especially thanks Brian for his OPNET integration of RatNest which clarified many requirements for the C++ prototype. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000. SAND2009-1895C.

References

[1] I. F. Akyildiz, T. Melodia, K. Chowdhury, "A Survey on Wireless Multimedia Sensor Networks", Computer Networks Journal, (Elsevier), 2007.

[2] R. Beraldi, The polarized gossip protocol for path discovery in MANETs, Ad Hoc Networks, Volume 6 , Issue 1, 2008, pp. 79–91

[3] G. Berkovski, M. Marom, "OSPF Tutorial, Creating the Routing Table," http://www2.rad.com/networks/2002/ospf/dijkstra.htm

[4] P. Bose, P. Morin, I. Stojmenovic, J. Urrutia, "Routing with Guaranteed Delivery in Ad-Hoc Wireless Networks," ACM Wireless Networks, Nov. 2001.

[5] Y.-C. Hu and D. B. Johnson, "Design and Demonstration of Live Audio and Video over Multihop Wireless Ad Hoc Networks," in proceedings of the MILCOM 2002 IEEE Military Communications Conference, IEEE, Anaheim, CA, October 2002.

[6] The Handbook of Ad Hoc Wireless Networks, eds. M. Ilyas, R. C. Dorf, CRC Press, Inc., 2003.

[7] D. Johnson, D. Maltz, Y.-C. Hu, The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks for IPv4, RFC 4728: Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing, eds. T. Imielinski, H. Korth, Vol. 353, Chapter 5, pp. 153–181, Kluwer Academic Publishers, 1996.

[8] B. McBride and C. Scoglio, "Characterizing Traffic Demand Aware Overlay Routing Network Topologies." Proceedings of IEEE Workshop on High Performance Switching and Routing 2007, New York, USA, 2007.

[9] J. Moy. "OSPF Version 2". Internet Engineering Task Force. 1998.

[10] C. E. Perkins, E. M. Royer, "Ad-hoc on-demand distance vector routing," in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[11] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, I. Stoica, "Geographic routing without location information," in proceedings of the 9th annual international conference on Mobile computing and networking, 2003, San Diego, CA, USA pp. 96–108

[12] H. F. Wedde, M. Farooq, "A comprehensive review of nature inspired routing algorithms for fixed telecommunication networks," in Journal of Systems Architectures, Vol. 52(8-9), 2006, pp. 461–484.