Guesswork: Robust Routing in an Uncertain World

Tom Parker*   Koen Langendoen
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology, The Netherlands

*Supported by the Dutch Organisation for Applied Scientific Research (TNO)
E-mail: {T.E.V.Parker, K.G.Langendoen}@ewi.tudelft.nl

Abstract

Guesswork is an adaptive, probabilistic routing algorithm for wireless ad-hoc sensor networks, using local knowledge of best guess next-hop nodes to efficiently implement source-to-sink routing. Guesswork is partially based on the existing ETX and ExOR routing methodologies, but extends both in new ways to create a unified robust method for sink information distribution and source data transmission. It is designed to work efficiently in a wide variety of application scenarios, being able to cope with low quality links as well as both static and mobile networks.

1 Introduction

Most routing protocols have a number of significant problems when applied to Wireless Sensor Networks. The major trade-off is between adaptability to changing network conditions and maintenance overhead, but this trade-off illustrates clearly the issues with WSNs vs. conventional wired networks (limited power, limited memory storage, limited processing, etc).

Of these, the most critical focus is on power usage. Everything a WSN needs to do uses power - sending messages, listening for/receiving messages, doing processing, reading from sensors - and the power resources (i.e. batteries) are generally scarce. WSN protocols must therefore be designed to use the minimum amount of power possible, in order to increase network lifetime as much as possible. Therefore, any power usage must be carefully considered, and non-essential or redundant effort should be removed wherever possible.

Given this restriction, adapting to changing network conditions is very difficult, as this often requires the re-dissemination of routing information for a sink or other critical node across a significant proportion of a network. The overhead required to keep standard routing information up-to-date when the actual amount of data being transferred across said network is taken into consideration is often unacceptable.

This paper introduces Guesswork - an adaptive routing algorithm, using a series of extended variants of the ExOR protocol [1], in combination with the ETX routing metric [4], in order to reliably distribute sink information, keep source-to-sink routes up-to-date and provide message passing along these routes, with a minimum amount of overhead even in networks with low or variable quality links.

We firstly introduce the main framework of the Guesswork protocol, then our extensions to ExOR, and show how these can be integrated easily with a variety of existing MAC protocols, then present simulation results for Guesswork and other routing protocols on top of a variety of MAC protocols.

2 Existing work

In this section we will look at ETX and ExOR - two technologies that will be used later on for our routing protocol.

2.1 ETX

Most routing metrics have cut off levels - a link is considered to be arbitrarily good or bad. For most realistic scenarios, this is often not the case [5]. Sometimes we will have a lot of good links, and then we can discard more, sometimes we will have a very bad connection to the sink node, but we still need to be able to communicate.

Expected Transmission (ETX) count [4] provides an improved metric for routing decisions, based on the expected number of transmissions via a particular next-hop node to reach a particular destination node. This allows for adapting to the complete variety of node link conditions [20] - everything from perfect links to dealing with broken and partial links.

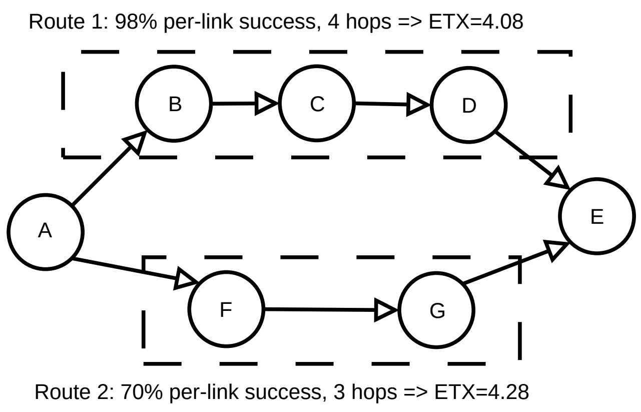
Route 1: 98% per-link success, 4 hops => ETX=4.08
A
B
C
D
E
F
G
Route 2: 70% per-link success, 3 hops => ETX=4.28

Figure 1. Not all links are equal

A partial link for example will increase the ETX value for a path because it is not always totally reliable, and broken links can be handled by significantly increasing the ETX value of a node that cannot find its next-hop neighbour, as a node with no next-hop neighbour can thought of as a node with a with a **very** high ETX value to the destination node.

For example, we may also have cases like Figure 1, where a shortest-hop algorithm would pick Route 2. But, this has a higher ETX and therefore a higher average transmission count than Route 1, despite the fact that it requires less hops.

ETX also allows for dealing with heterogeneous networks with gateway nodes (nodes with a faster link over another network to the sink, and usually an external source of power) - these can simply declare their ETX cost to be very small, as their faster link and external power means they represent a much better route to the sink.

2.2 ExOR

ExOR [1] uses a “one send, many replies” idea to do localised greedy routing, providing a better utilisation of the basic broadcast medium available to WSNs. This is based on the idea of a set of neighbours receiving a message from a sender node, all of the specified neighbours sending an Ack for the message, and the best next-hop node for a particular destination (of the set that receive the message) gets chosen, without having to do additional communication beyond the Data/Ack sequence already performed.

Figure 2 shows an example timeline for an ExOR packet. The DATA segment is sent by the sender node, which includes a list of neighbours in the order of how many hops it would take to get from that neighbour to the destination node. The neighbours (nodes 3,2,4 and 1 in this case) reply in the order that was specified in the DATA message. If a neighbour node does not hear any reply messages for nodes

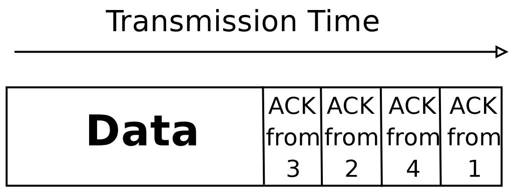

Figure 2. Example ExOR packet timeline

earlier in the order than itself, then it forwards on the message. If all of the neighbours can hear each other, then this will result in only one sending on the message. To help this process, if a node has already heard a reply message when it sends it’s reply message, then it replies with not it’s own id, but the id of the best (i.e. earliest in the neighbour list) node that it has heard a reply message from.

The use of shortest-number of hops by the originally proposed version of ExOR (which isn’t necessarily a good choice for routing decisions, as we discussed on the preceding page) relied on the knowledge of a local node about its neighbours, because it required that the neighbour list of an ExOR packet should always be in order of preference, based on information available at the start of the ExOR sequence. This creates a situation which is prone to allowing out-of-date information to be maintained for longer than it should be and also stops a number of possible useful extensions to the protocol, as this relies on locally-cached data about neighbours vs. querying the neighbours for their current state.

3 Guesswork

In the original ETX specification [4], De Couto et al. used probe packets to determine what the ETX cost for a link would be. This is expensive, with an unacceptably high overhead cost for low transmission rate scenarios i.e. most WSN applications. Additionally, because the probe packets are generally a lot smaller than actual data packets, probe packets do not necessarily provide an accurate picture of how good a link is for data packets.

Guesswork is a technique for creating and adapting ETX values for destination sinks over time using information from the data sent over a network, and so be able to adapt to changing network scenarios. These ETX values are then used to for routing decisions. Guesswork also utilises the extended ExOR variants that we will present later in this paper.

3.1 Initialisation

In order to formulate a good solution to the source→sink routing problem, first we need to look at our discovery process for the sink nodes themselves. One way is that all nodes are automatically told about all of the sink nodes at startup i.e. by hard-coding the sink node information into our nodes. This is however inflexible, assumes that we already have the sink information at the time of system deployment, and is generally unsuited for ad-hoc WSN scenarios.

A better solution is to do initial flooding of the sink information to the network. If we have a reliable broadcast mechanism then we can reduce the flooding to a single instance per sink. When a new node starts up, it can request the sink information known to its neighbours. A possible extension to this is the inclusion of query information within the sink broadcast (i.e. what kind of data a particular sink is interested in), but for now we are only considering the situation where all sinks want all the information. With this initial sink→sources flood, we can discover the initialisation ETX values (one per source node) for a sink.

This information may not be perfectly accurate (asymmetric links will cause problems for example), but it represents a reasonable first approximation to the correct current ETX for a sink.

3.2 Message Transmission

Guesswork relies on the existence of a reliable protocol to implement source-to-sink transmission once an ETX value for a particular sink is known by a source node. Packets being transmitted with Guesswork also contain a TX count so far for the packet.

3.3 Adaption

Nodes have an initial value for the ETX value for the sinks in the network (from the Initialisation phase), but this will change over time as the network alters (dead nodes, broken links, new links, etc), so we need to be able to adapt the ETX value for a sink over time. One way this can be done is by propagating route-update packets back to source nodes when a successful transmission to a sink node has differing values for the number of transmissions used vs. the original ETX value. The route-update contains the actual TX count that was used for the particular source→sink.

Route discovery for the sink→source route can be done by the nodes on the path (P) that a particular source node (A) uses to get to a sink node (S), because all of the P nodes will have a TX count for the incoming packet from A, and so they can record this as an estimated ETX to return to A. The new ETX to S for each of the nodes along the path can also be updated, by subtracting the ETX value that the packet had on the way in at a particular node from the total TX value that is being reported in the route-update packet.

If we have a route-update, then we have a new ETX value. This can now be used to update our current recorded ETX value, using a learning function based on these two values. This is done because a single changed route cost doesn't necessarily mean all transmissions to that sink will be equally low/high. The simplest form of this is:

$$\begin{aligned}&ETX_{updated} = \\ &ETX_{new} * LearningConstant \\ &+ETX_{old} * (1 - LearningConstant)\end{aligned}$$

where the value for $LearningConstant$ is some value smaller than 1. Earlier work in similar areas [14] suggests values in the 0.2 to 0.4 region, but more experimental testing would be needed to test for suitable values for typical applications.

An additional optimisation is the aggregation of route-update packets, as these are only used to propagate the ETX value back along the sending path. Instead of sending one route-update for every message with a changed ETX, an aggregate update consisting of an averaged ETX (with a packet count) for a set of packets can be sent back to the sender node. A suitable mechanism for this would be waiting until no packets have been received from a sender for x seconds (10 for example) before sending the aggregate route-update. Also note that if the aggregate ETX is very close (e.g. <5% change) to the original source node's ETX for our sink, then we can simply discard the route-update completely as no update is necessary. These measures reduce the number of route-update packets so that they are only generally sent when the network is changing and during periods of network stability they do not need to be sent at all.

3.4 Failure Resilience

New route discovery in the event of a failure (no responding neighbouring node has a lower ETX than ours) consists of bouncing the packet back up to the previous node along the chain and repeating the send sequence. In this case, we can also update the failed node's ETX value for the destination sink by using a value for ETX_{new} that is higher than any other ETX value that we have already seen. Possible values include $ETX_{Highest} + 1$ or $ETX_{Highest} * 2$, but experimental testing will also be needed to resolve better values for this. We should now have an increased ETX (due to the updating from the failure) and so another possible neighbour will probably be chosen instead.

For example, if a node A has a message for sink S and it chooses node B as it's next-hop neighbour, but B is unable to forward the packet, then B updates it's ETX for S to a much higher value, and bounces the message back to A. At this point, A will go through the message-forwarding mechanism again, and B could potentially be chosen again, but given that B has just had it's ETX for S significantly increased, it is likely that another node will be chosen.

4 Generalised ExOR

Guesswork relies on the existence of certain lower level facilities - specifically, a reliable broadcast mechanism, and a reliable way to do source-to-sink data transmission. ExOR has some limited abilities towards these goals, but it has limitations. In this chapter, we demonstrate a series of alterations to basic ExOR, broadening it's scope and allowing for a variety of "choice functions", including the use of ExOR as a mechanism for reliable broadcast.

4.1 Choice Functions

Generalised ExOR specifies the neighbour list in an arbitrary order, and the neighbours respond with a particular value (~1 byte of data for most choice functions). Which value, and the resulting actions depending on that value, depend on the particular effect that is required. A choice function defines how the protocol responds to ExOR messages, including how a particular given node receiving an ExOR message will then decide whether the message should be forwarded onto other nodes.

Some useful possibilities include:

1. Sending the lowest hop count that this node has heard (or its own if it has heard none so far), and forwarding if we have the lowest hop-count heard. This is the original ExOR choice function.

2. Sending the lowest ETX value that this node has heard (or its own if it has heard none so far), and forwarding if we have the lowest ETX value heard. This is referred to as ExOR-ETX.

3. Sending a bit-field representing the set of nodes that this node has heard an ACK for (including its own), expressed as a series of bits in the same order as the original transmitted set of neighbours. The original sender can then OR together the received bit-fields and infer the list of nodes that have received the message, thus allowing for a reliable broadcast mechanism with reduced cost compared to unicasting to every neighbour. The OR'ing together of the received bit-fields allows the sender to get a good picture of what nodes have received the message, even in the case of asymmetric links. This is referred to as ExOR-Bcast.

4.1.1 Multi-hop Reliable Broadcast

ExOR-Bcast can further be improved as a method for reliable broadcast over multiple-hops i.e. a message flooding scenario, by overhearing of broadcast messages from neighbours. This can be used to reduce the set of neighbours that we need to send a message to by eliminating those that we have heard an ACK for (or have seen another ACK that contains the relevant bit set for that neighbour).

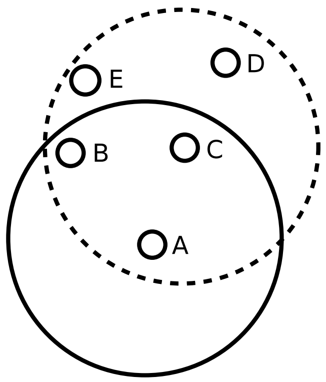

Figure 3. Multi-hop broadcast example

For example, see Figure 3. A is a node that has sent an initial ExOR-Bcast to B and C. C then proceeds to do ExOR-Bcast to D and E. If B overhears any of the replies from D or E, it can determine if they have heard the broadcast, and so therefore it may not be necessary for B to do ExOR-Bcast at all, or it can at least reduce the set of neighbours that it needs to send the message to.

4.1.2 ExOR-ETX

With ExOR-ETX, if two nodes have the same ETX value (which is a quite likely scenario, especially in cases with reliable links) and both receive a ExOR-ETX message, they will both by default decide to forward on the message, resulting in duplication. This may occur many times, resulting in an significant increase in the number of messages sent in the attempt to successfully deliver one message. In a number of likely scenarios this is not so much of a problem, as in many cases with equally good next-hop nodes they will both be travelling a similar path to the sink, and so the two routes will likely overlap later on and the duplicate message can be filtered out at that point.

However, a better solution to the duplication problem is to change the metric field for ExOR-ETX, reserving one bit (usually the high bit) for use as a "sender" bit. A node sets this bit to indicate that it has taken responsibility for making sure that the message is sent, not that it will necessarily send the message. If a node has decided it is to be a sender, and it sees no other replies with a sender bit, then it sends out the message. If however a node sees another message with a sender bit and an equal ETX, then it must make a decision whether it is a sender or not. This decision is based on an arbitrary, but fixed (for a particular network) function for any given network, that will always be able to decide a single sender node. The current implementation uses node ids, and the node with the highest id is the sender.

4.2 Inverted ExOR

One problem with the ExOR methodology is what we call the "neighbour bootstrap" problem. When a network has very little traffic, then it may be the case that a node does not receive any messages from its neighbours, or only from a subset of them, and that the node needs to be able to send messages to the neighbours that are being quiet and not announcing their presence. Communicating with these nodes is difficult due to the fact that ExOR requires neighbour knowledge for the scheduling of the ACK messages.

To combat this problem, we provide another variant on ExOR using an inverted neighbour list i.e. the neighbours that are in the list in the ExOR message should not respond. For a bootstrap scenario with no currently known neighbours, the list may well be empty. The message specifies instead a time period (specified as the number of control packet intervals) after the message has been sent, in which nodes not in the list should respond. Each node that responds picks a random interval within the specified time period to respond (using slotted aloha [16] with the transmission time of a control packet as the interval).

With this random choice, the likelihood of collisions increases significantly, but the focus of this method is acquiring information from some neighbours, in order to reduce the problem of having sparse neighbour information. In the case that we have not received information from enough neighbours, then the message can be repeated, adding the successful neighbours from earlier stages to the "do not send" list each time.

An additional constraint on the design of the control function that can reduce the collision rate is only having neighbours that have something useful to say responding (as opposed to the "everyone should respond" models used for standard ExOR) e.g. in a message routing scenario, only nodes that have a good route to the sink should respond.

5 Implementation

So far, we have specified what we want to do, and some high-level details of how we will implement these choices, but another important factor to consider is that in most WSN systems there are existing other protocols that we must interact with.

Most existing WSN routing protocols [2, 9, 10] come in two forms: weak and strong binding to the MAC layer - those that just treat the MAC as a black box that will send packets, and those that rely heavily on one particular MAC. The former methodology is unsuitable for ExOR use as the uncertain delay between one packet transmission and the next (due to MACs doing things like sleeping [3, 18] and letting other possible parts of an application use the radio), and the latter would reduce the flexibility of ExOR to interact with a variety of protocol stacks (which is a problem given the heterogeneity of current WSN systems).

5.1 Partial Binding

As a compromise, we propose instead a partial binding to whichever MAC protocol is used. At some levels this is similar to some of the ideas proposed by B-MAC [13], but we assume the existence of a minimal MAC layer below our protocol to implement things like scheduling of message times and possibly sleeping during quiet times. B-MAC could be used to implement this minimal MAC layer. Our bindings specify certain additional facilities above and beyond what a MAC protocol would normally provide, but these should be fairly simple to implement (certainly vs. creating a new transmission method in an existing MAC protocol).

These facilities are then used to implement ExOR above the MAC layer, as opposed to implementing it in every separate MAC. There is one requirement of the lower-layer MAC - that it is possible to allocate contiguous blocks of time that can be used for both sending and receiving by a node. This is possible for all contention-based MACs, and for some TDMA-based MACs (but not all of them).

Our extensions are split into two groups: commands and events, in a similar way to the nesC [6] language that is used by TinyOS. The full details are specified in Table 1 on the next page.

5.2 Building ExOR

Using our MAC extensions, we can now build generalised ExOR as follows. In all cases, we assume that at startup we call PacketTime() with the payload length of a reply packet (1 byte for most choice functions) to get the value $ReplyTime$.

For all types of node, on a BroadcastReceive() event, where the incoming packet is from a Receiving node, then apply the control function as appropriate for the ExOR variant in question, and return 0.

ExOR Sending node:

1. Call PacketTime() with the length of the message to get $BroadcastTime$
2. Call ClaimMAC() with Time equal to $BroadcastTime + ReplyTime * Neighbours$.
3. Call BroadcastNow() with the message that we want to send.
4. On EndTimer(), perform whatever cleanup operations are associated with the used choice function (e.g. for ExOR-Bcast, record which neighbours we now have additionally managed to send the message to).

Command	Description
ClaimMAC(Time)	ClaimMAC attempts to reserve Time milliseconds for the guaranteed use by this part of the program. This time is guaranteed to be not used by any other sub-program on this node (including MAC-level control packets), and the MAC will be listening for the entire period. Carrier-sense or equivalent operations will be performed beforehand to help ensure the reservation of this time
BroadcastNow(Packet)	Sends the Packet right now, with no waiting of any sort. Only works if the MAC is currently reserved, and the specified amount of time has not run out.
EndClaim()	Finish a block of time allocated to this program module before the previously specified ending time i.e. because we've done everything we want to do.
PacketTime(Length)	PacketTime returns an estimate of how a long a Packet of payload Length bytes would take to send, with all of the MAC and lower-layer headers added. Carrier-sense and other similar delays are ignored, and only the time needed to send the bytes is returned.
GetNeighbours()	GetNeighbours returns a list of the known neighbours of this node. Note that it is acceptable for a MAC to always return a zero-length list, as the list should always be assumed to be not necessarily complete, but is merely an aid to help/initialise higher-level neighbour-discovery protocols.

Event	Description
BroadcastReceive(Packet)	BroadcastReceive is fired whenever a Packet is received. Implementations of the handler for this event should return the number of additional milliseconds that the MAC layer should be reserved for this application (in a similar way as the guaranteed time for ClaimMAC, but this only succeeds if no other sub-section has current control of the MAC). Some protocols (ExOR for example) may want to set an additional timer during this routine for when they are to send a reply.
EndTimer()	EndTimer is fired to notify a node that its previously reserved time has ended.

Table 1. MAC Extensions

ExOR Receiving node:

1. On a BroadcastReceive() event, where the incoming packet is a message from a Sending node, then record the sender node id
   a) If invert is switched off for the message, and we are in the neighbour list, then set z to be our index in the neighbour list.
   b) If invert is switched on for the message, and we are not in the neighbour list pick a random value between 0 and n as z, where n is the maximum number of slots for the reply packet.
   c) If neither a) nor b) apply, return 0.

   ```
   If $z > 0$ then
     set a 'reply packet' timer for $z * ReplyTime$ ms
   else
     send a reply to the sender with BroadcastNow()

   Return $(z + 1) * ReplyTime$
   ```

2. If our 'reply packet' timer goes off, then send a reply using BroadcastNow() to the recorded sender.

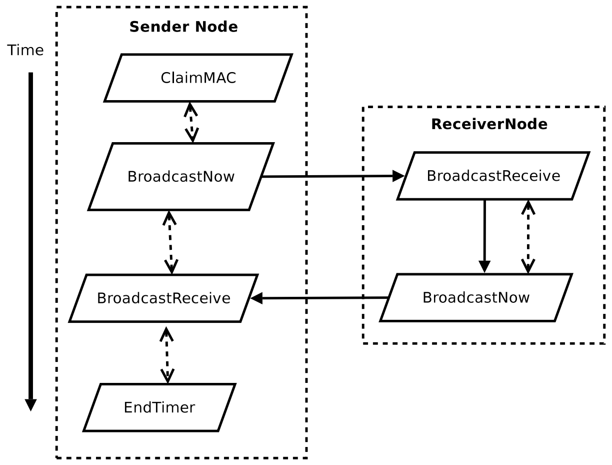

Figure 4. ExOR time line example

5.3 Building Guesswork

Guesswork (as we discussed in section 3 on page 2) requires a reliable broadcast mechanism, and a method for reliably transmitting messages using ETX. We have now provided these, with ExOR-Bcast and ExOR-ETX, but there are a few remaining details of the Guesswork use of these algorithms to be mentioned.

Namely, the choice of how many neighbours to have in a neighbour list. For ExOR-Bcast, the answer is generally fixed - in our implementation, we stick to a maximum of 5 neighbours - more would make information spread faster, but as longer messages tend to have a greater probability of failure (due to interference from other nodes, and effects like the "hidden terminal" problem) this appears to be a good value. Also, because we actually want to talk to everyone in ExOR-Bcast, talking to too many neighbours is less of an issue. Additionally, as ExOR-Bcast is generally only used by Guesswork during application setup, optimisations to this will have a minimal effect.

However, the choice of neighbours used by ExOR-ETX is more important - partly because this variant is more often used, and partly because ExOR-ETX only actually wants to talk to one neighbour and the others are only for redundancy. As excessive redundancy is overhead, a good value for this is application specific, but as we would like the algorithm to work with a variety of applications, a method for automatically deciding on this value is useful. One option is using the following algorithm:

1. Each node starts with a neighbour count value (e.g. 5), which is used to determine how many neighbours are used in ExOR-ETX
2. A node keeps track of the last known ETX value for each of it's neighbours, and every time an ExOR-ETX sequence is executed, it checks which node its cached values for the neighbour ETXs would have chosen vs. the actual winning node.
3. Every time the node guesses correctly, it decrements the neighbour count value (down to a minimum of 2) and every time it gets it incorrect, it increments the neighbour count (possibly up to a maximum value e.g. 10)

The idea behind this is that if a node can correctly guess the correct node to send to next, then the network is probably moving towards a stable configuration with stable links. If a node guesses incorrectly, then it is probably worth expanding the neighbour list to check against other nodes. The neighbour list in ExOR-ETX effectively represents a "candidate node" and a list of backup options. Therefore, giving more backup options in a unstable situation, and less in a stable scenario is a good idea. The limit of 2 neighbours as a minimum ensures that there is always a backup neighbour, and avoids the node collapsing towards the fixed route scenarios that Guesswork intends to avoid.

For the purposes of the simulation testing, we created a fixed-length neighbour list and did not implement this extension.

6 Results

We proceeded to test Guesswork against other routing algorithms, in combiantion with a series of different MAC protocols. The simulation framework is based upon Positif [11], but extended and altered to work with routing protocols rather than localization protocols. The MAC protocol implementation is taken from our existing work on MAC protocols [3], which has been extended to interface with Positif to create a unified simulation framework.

	Parameter	
Protocol	Name	Value
Aodv	"Hello" messages	Disabled
Gossip	Fanout	2 for 1st 5 hops 1 afterwards
	TTL	20 hops
Guesswork	Neighbour list size	5 (fixed, no adaption)
SMac	Frame length	1000ms
	Timeout	100ms
TMac	Frame length	610ms
	Timeout	15ms

Table 2. Protocol parameters

In each case, we have a simple routing test, consisting of a source transmitting a packet every 10 seconds, until it has sent a total of 20 packets, and sending towards a single sink. We tested 3 different MAC protocols SMac [17], TMac [3], and a "simple, no carrier-sense" MAC (to provide a baseline comparison). The other routing algorithms being compared against are AODV [12] and Gossip (random walking with limited fanout). The AODV implementation was ported from the existing implementation for the GloMoSim [19] simulator. Given their lack of a sink-discovery mechanism, both AODV and Gossip were informed of the address of the sink at the start of each test.

56 nodes are present in each test, in a 50x50 area, with maximum radio range set at 14. The per-link reliability is set at 80% for all tests i.e. a random 20% of all packets sent by the nodes are randomly discarded, in order to simulate imperfect links. Note that 80% per-link reliability is a level that would be considered "good" by most algorithms that classify links as good/bad.

Each simulation is run for 300s before termination, and each result is the average of 20 runs of the particular combination of routing protocol and MAC protocol. The other parameters for the protocols are given in Table 2, but a couple are worth discussing further - Aodv has the "Hello" messages disabled, because they resulted in far too much overhead for our simple WSN example, and Gossip reduces it's fanout to 1 after 5 hops as otherwise it ends up reducing to simple flooding and having far too large overheads.

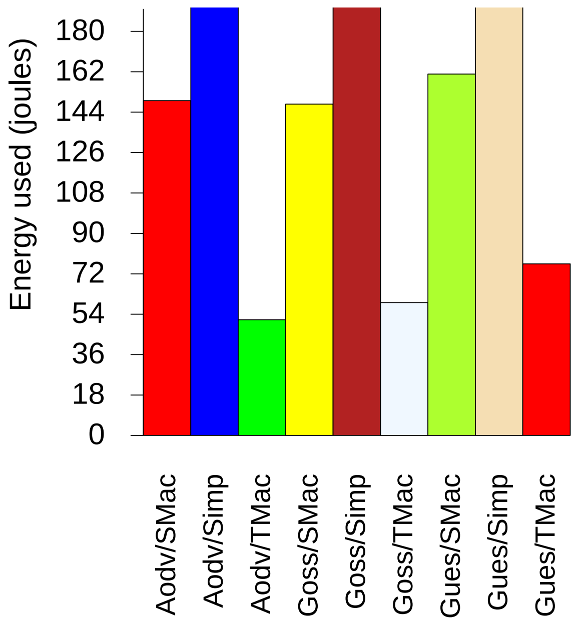

Figure 5. Energy cost

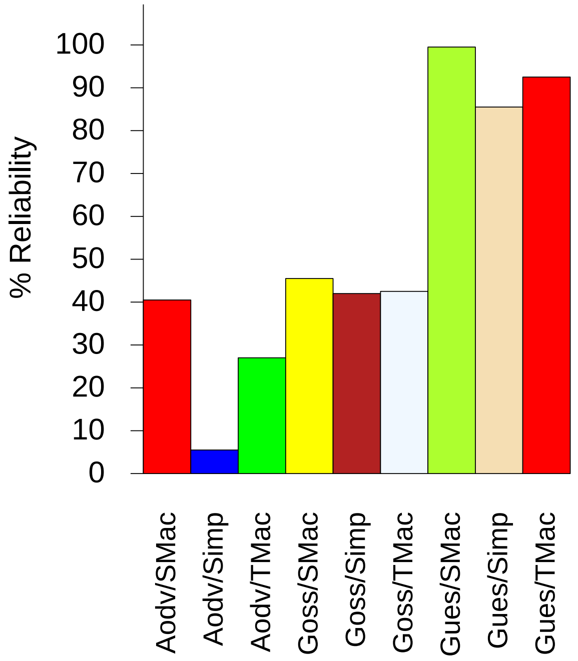

Figure 6. Reliability

We looked at two different evaluation metrics for the results: reliability (how many of the source messages get to the sink) and cost (power used for transmission and reception over all of the nodes in the experiment, using the listed costs for a typical node transceiver [15]). The produced graphs for these metrics have been altered in a few small ways for improved readability - some of the protocol names have been shortened ("Gossip" has become "Goss", "Guesswork" became "Guess" and the simple MAC is described as "Simp") and the "simple" power measurements have been clipped (due to its lack of power management, the simple MAC uses approximately an order of magnitude more power than the other protocols, and so displaying it fully would reduce the amount of usable information on the other protocols).

A number of interesting results immediately appear from the graphs. Firstly, looking at Figure 5, we can see that the overall cost of the messages is dominated by the choice of MAC protocol - comparing differing routing protocols with the same MAC protocol shows a slightly higher cost for Guesswork and a slightly lower cost for AODV, but the difference is not significant. These results are consistent with current theory regarding such factors as idle listening (as noted in [7]), and show that despite the additional overhead of the multiple ACKs in Guesswork, it still represents a viable alternative.

The advantage of Guesswork becomes immediately apparent when we turn to Figure 6. Given that AODV depends on the reliability of a link at route discovery time, it performs badly when faced with links that are "reasonable", and may not always succeed. 80% is a high enough per-link reliability to provide links that can be used for routing, but low enough to cause enough failures to make otherwise viable connections be often discarded. Using the "Simple" MAC (with it's lack of carrier sense, and so hence much greater packet drop rate) reduces AODV to unusable levels. Gossip, with it's simpler methodology, is able to get some results (the redundancy from fanout is a significant factor there), but only with a reliability of 40-45%. Guesswork, on the other hand, is able to react to even low reliability levels and work around these problems, with end-to-end reliability at 85%+, climbing to 95%+ with the carrier-sense capable MACs. We have performed additional testing with Guesswork at lower reliability levels, and were able to maintain successful sink-source routing with link reliability levels down to ~30%, without alterating Guesswork in any way.

7 Conclusions

We have created a routing algorithm that performs reliable routing over significantly unreliable links, and without significant additional energy costs vs. existing routing protocols. The Guesswork algorithm, despite it's reliance on a packet sequence not normally supported by MAC protocols (ExOR), has also been shown to be able to integrate successfully with a range of MAC protocols.

To further the goal of integration with existing protocol stacks, and to facilitate additional testing, Guesswork is currently being implemented for TinyOS [8], and will be released to the wider community at a future date. Support for the partial binding extensions has already been released as part of the T-MAC implementation available at contrib/t-mac/ in the TinyOS CVS repository.

We would also like to explore ways of implementing Guesswork on top of TDMA protocols (which have very limited support at this point). More work also needs to be done with testing against both fixed/reliable networks and mobile networks, as the current testing is against what is a fairly realistic middle ground for WSN applications, but testing how well Guesswork performs vs. protocols designed for a particular niche in the search space of possible routing problems is of interest, as Guesswork is designed to work well (high reliability and low cost) in a variety of scenarios.

References

[1] Sanjit Biswas and Robert Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):69–74, 2004.

[2] David Braginsky and Deborah Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM Press.

[3] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pages 171–180, Los Angeles, CA, November 2003.

[4] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[5] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, October 2002. ACM SIGCOMM.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: a holistic approach to networked embedded systems, 2003.

[7] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[8] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGARCH Comput. Archit. News*, 28(5):93–104, 2000.

[9] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[10] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

[11] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. *Computer Networks, special issue on Wireless Sensor Networks*, (43):500–518, 2003.

[12] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA '99, February 25-26, 1999, New Orleans, Lousiana, USA*, pages 90–100. IEEE, IEEE, February 1999.

[13] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2004)*, pages 95–107, Baltimore, MD, USA, 2004.

[14] Lili Qiu, Yang Richard Yang, Yin Zhang, and Haiyong Xie. On self adaptive routing in dynamic environments - an evaluation and design using a simple, probabilistic scheme. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP '04)*, pages 12–23. IEEE Computer Society, 2004.

[15] RFM. *TR1001 868.35 MHz Hybrid Tranceiver*.

[16] Lawrence G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, 1975.

[17] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *21st Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1567–1576, June 2002.

[18] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *ACM/IEEE Transactions on Networking*, 12(3):493–506, June 2004. A preprint of this paper was available as ISI-TR-2003-567.

[19] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[20] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 125–138, New York, NY, USA, 2004. ACM Press.