Foxtrot: phase space data representation for correlation-aware aggregation

Tom Parker* Koen Langendoen
{T.E.V.Parker, K.G.Langendoen}@tudelft.nl
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology, The Netherlands
(*Supported by the Dutch Organisation for Applied Scientific Research (TNO))

Abstract— **Most existing work in aggregation and querying for sensor network data has focused on the use of standard statistical operations (average, median, etc) to reduce the quantity of transmitted data within a network. These operations have been carried out without considering the nature of the actual data in the network. In this paper, we show how aggregation without correlation awareness will result in variable (often high) levels of error in the end results, and reformulate the nature of the aggregation problem in terms of information loss v.s. packet rate reduction.**

We instead propose Foxtrot - using phase space data representation combined with novel aggregation methods to limit errors to application-specific ranges. Foxtrot reduces data rates without significant information loss. We demonstrate that Foxtrot is able to achieve these goals, both using simulation methods and a TinyOS implementation.

I. Introduction

Whatever the nature of the application, Wireless Sensor Networks (WSNs) generate data. With increasing numbers of nodes, the volume of data becomes ever greater. Therefore, one of the fundamental problems for these networks is managing the outputted data. Given the energy and space limitations of WSNs, moving increasing quantities of data to a sink node/end-user computer where the data can be stored and analysed will reduce the operational lifespan of such a network. It can be concluded that reducing the amount of data that is transmitted is a required goal in order to achieve usable network lifetimes. This goal, however, conflicts with the purpose of these networks: to gather information. One approach to resolving these conflicting issues is the use of aggregation techniques to remove redundant or unimportant data and only transmit useful information.

A simplistic approach would involve the use of standard statistical techniques to combine many data points into a smaller set. However, given that only incomplete information is available at any given node, the choice of techniques is somewhat limited. An exploration of existing work in this area is in Section II. Unfortunately, existing statistical aggregation work is not correlation aware: all data points are automatically considered as inputs for the aggregation mechanism, without consideration to the nature of the data. Additionally, rare events (e.g. a single sensor with different data) are not considered statistically significant, whereas for WSN applications a single sensor reading may well be important, e.g. only one sensor is attached to a tree that is on fire. Discarding the bulk of unimportant data (e.g. limiting data from the large areas of a forest that are not on fire) whilst keeping the useful information should be a focus for WSNs. Location data is also vital for WSNs (“Which tree is on fire?”), and when correlation awareness is considered, we must also check where multiple data points were measured in order to determine whether they can be aggregated.

Given all of these problems, new approaches are required. In this paper, we show the faults of existing aggregation techniques, propose the use of phase space representation to incorporate all data sources in a single view, demonstrate novel aggregation techniques that selectively merge phase space data such that information loss is limited, and show this can be all achieved in a fully-distributed manner on speed-, memory- and energy-limited node hardware.

II. Existing work

Much work has already been done in both aggregation, and in querying mechanisms for WSNs that indirectly use aggregation techniques (e.g. SQL-like techniques - like TinyDB [6], TAG [5] and STREAM [2]) for whole network queries). Given that one of the major purposes of statistical techniques is reducing large bodies of data down to a limited set of more usable values, it seems an obvious choice for aggregation techniques.

Averaging is a popular choice [5], [6] (partly because of the Central Limit Theorem [10]), as this can reduce any number of data points down to just an averaged value plus a count of the elements merged (the count is required to aid merging of multiple averages at later nodes). This has several major advantages - it is fully distributed; the end result is always very small (and the same size) regardless of how many data points were initially available; processing and space costs are minimal; and an average value for a region is a common example query to be asked of a sensor network. Overall, it is a very easy (and common) question to answer, and one that many aggregation protocols have optimised towards.

The one question that these systems fail to ask is: is the answer useful? Take the common example of "what's the average temperature on this floor?", and using it as part of a feedback loop to keep the temperature at 20 degrees celsius. Imagine that there are 5 sensors on a floor of a building and the answer is 22 degrees. The common assumption would be that most of the sensors have a temperature of about 22 degrees, and so the system would drop the temperature by 2 degrees. But then a report comes in of a room at 18 degrees. What actually happened is that 4 of the sensors were at 20 degrees, and the 5th sensor had been placed near the output vents of a computer which rose the temperature around that node to 30 degrees. The averaging algorithm, without any correlation awareness, merged all the values into a single value that says nothing about the actual nature of the true temperature values, and thus too much information was discarded.

There are in fact only two interesting scenarios for averaging: a) all data values are approximately similar (or at least vary around a common centre), in which case a sampling of a subset of the nodes would get as good an answer as averaging, but with less network traffic; or b) data values vary widely (as in the example), resulting in a result that bears no resemblance to the actual data.

Alternately, we could use the median of a set of values instead. However, for an accurate median we need all the original data points at a single point. Q-Digest [8] attempted to reduce this problem by only transmitting a subset of the candidate data values, and providing a method to merge candidate sets together. This gave a reasonable approximation to the median (as well as other statistical values) while reducing packet rates, but ran into the same problem as averaging: is the answer useful? In the temperature example above, it would have given us the 20 degree value (and probably also the 30 degree value depending on the level of merging). This is an improvement over averaging, but in situations with a larger number of nodes, rare events will still be discarded in favour of lots of information about common events. Also, Q-Digest gives us no location awareness; in the temperature example, we would be unable to locate the problematic sensor without using other techniques.

Other statistical techniques are mostly limited by their requirements for needing most of the data to be merged in one location, combined with the issue of merging multiple subsets that have been previously merged. Further work in this area could possibly reduce these two problems, but we would still need to find a technique that gives useful answers, and that is a much harder problem.

We conclude that a newer approach, focused on the usefulness of the end result to the users of the application, is required for an improved aggregation technique. We also conclude that the major goal of aggregation is a trade-off between information loss and packet data rate reduction, and that any aggregation technique that attempts to discard packets without considering what level of information loss this will cause is fundamentally flawed.

III. Phase space data representation

In order to consider how to create a technique that would combine correlation awareness with location knowledge, we turned to phase space representations of the data. Phase space [11] uses an abstract n-dimensional space to represent all of the possible states of a system. Each degree of freedom in the space represents a different data value. Our initial approach to this focused on single sensor values, plus their associated location data. We later realised that the same techniques could not only be used to combine multiple sensor values, but to also incorporate other data sources. In total, we identified three possible sources of data:

- Raw sensor values (e.g. humidity or temperature)
- Internal node data sources (e.g. location data)
- Functions of other data sources (e.g. rate of change)

The state of a node at a particular instant in time can be represented by a point in phase space defined by the values of all of the sources of data being used. Most applications would normally only be interested in a small number of sensors, plus 2 dimensions for location data, but the capability for extra data sources is automatically available. Irregardless of what data is being used, the data can still be represented only as an abstract concept of a series of values without any knowledge of which of the three categories the original data source was. Individual axes may however specify certain source specific limits on what can be done with data in that particular axis.

The basic data unit is that of a point in phase space, but we also want to be able to merge data points into larger regions also within the same phase space. A region in phase space represents a range of values. A region is defined by a set of numbers $\{min_1,...min_n\}$ and $\{max_1,...max_n\}$ for an n-dimensional space, and covers all points of the form $\{v_1,...v_n\}$ such that $\forall x, x \in \mathbb{N}, 1 \leq x \leq n, min_x \leq v_x \leq max_x$. A point is defined as a zero-sized region i.e. $\forall x, x \in \mathbb{N}, 1 \leq x \leq n, min_x = v_x = max_x$

IV. Region merging

Aggregation can now be specified as merging of multiple phase space regions into a different (generally smaller) quantity of phase space regions. We also need to note that some regions may not be mergeable, and that any processing time spent attempting to merge unmergeable regions is effectively wasted. Therefore, one design aim is that if the merging fails, it should fail as early as possible to reduce wasted effort.

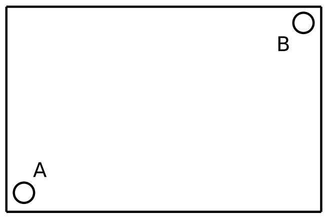

Fig. 1. Greedy merging of two points

An initial greedy approach to merging would be simply to merge any and all packets into a large region that contains all of them. This approach has a number of problems, as demonstrated in Figure 1. Specifically, the approach is too greedy, and ends up describing regions that not only contain the original points, but also large areas that are not in the original data set, and so can provide results that differ significantly from the original data. Additionally, greedy merging of sensor data will result in large ranges in the results e.g. for the temperature example in Section II, we would get the range “20-30 degrees”, losing significant amounts of information. We do however still want to be as greedy as we can in the merging algorithm, as a greedier algorithm will result in being able to merge greater numbers of regions into a single region. Therefore, in order to find an algorithm that is greedy enough, but not too greedy, we need to constrain how data points are merged, and also decide if some points can in fact not be merged at all. A particular set of data points are only mergeable if all sources are mergeable for the particular points.

The constraints required for sensor data and for location data differ in their requirements. For location data, we want to be as greedy as we can, provided that the end region does not cover areas that were not implied by the original data. For sensor data, we have more fixed constraints. For example, an application may specify that an acceptable level of data loss from a temperature sensor is 1 degree. In this case, if two points are further apart than 1 degree in the temperature dimension, then they cannot be merged. Conversely, if they are no more than 1 degree apart, they can always be merged. This contrasts with location data, where “close” values may create over-sized regions whereas “distant” points may not. See Figure 2 for examples of these two cases. In the “distant” case we have nodes in all of the corners of the created region, making the assumption that the central region contains similar values a reasonable inference. In the “close” example, we have no points in the top-left and bottom-right areas, so merging these points would infer much more without evidence to back up the assumption. If, on the other hand, we had data from nodes in the top-left and bottom-right areas, then creating the “close” region would be much less likely to cause problems. Another factor that makes this form of estimate more reliable is the use of proper heuristics for dealing with overlapping regions, which we will look at in Section V-B.

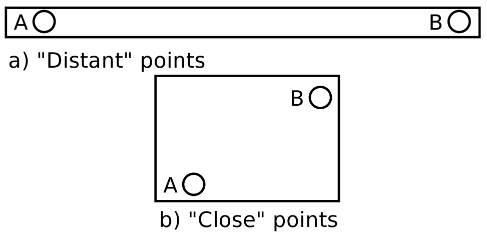

Fig. 2. Location merging examples

A. Constraints

Given the two differing forms of constraint, we define two classes of data source: statically and dynamically limited. In general, these will correspond to sensor data and location data respectively, but this may vary on a per-application basis, and for the purposes of merging we only need to know the class of a data source.

Statically limited data sources have the criteria that a data point with a particular value from this source can be merged with any other data point that is not further away (difference between two values) than a specific value e.g. a temperature source may say that the limit is 1 degree. This means that two temperatures that are more than 1 degree apart will never be merged, thus giving a guaranteed limit on the amount of information that will be discarded.

Dynamically limited sources are more complicated, and are merged as a set (i.e. all dynamic sources are tested at the same time). They have the advantage that they have no fixed limits as to which can be merged, but instead have a series of criteria to guarantee that the created region does not expand into regions that are not suitably defined by the original regions used. Complete details of the algorithm are in Appendix I, but the method resolves working with the corner points of all the regions involved, and not expanding a region in a particular direction unless there are suitable points in that direction to indicate that it is safe to expand in that direction, in order to avoid overly greedy merging.

V. Foxtrot

Foxtrot takes the data representation and merging concepts introduced in Sections III and IV, and builds an aggregation protocol. In common with any other aggregation protocol that wants to do in-network aggregation, Foxtrot requires a source-to-sink routing protocol that allows packets passing through a particular node to be altered and/or dropped depending on the choices of the aggregation protocol. In fact, the easiest way for aggregation protocols to do this is is for the routing protocol to not automatically forward incoming packets, but to hand them to the aggregation protocol, which then may later give (some) packets back to the routing protocol for further forwarding.

A. In-network nodes

For all in-network nodes (i.e. all nodes aside from the sink) Foxtrot takes data from the data sources provided by the application, converts them into the phase space data representation and then hands them off to the underlying routing protocol for forwarding towards the sink. If packets arrive at a node, then Foxtrot will attempt to merge them together along with any other packets currently stored at this node, and then hand over the results to the routing protocol.

B. Sink node

Sink nodes, similarly to in-network nodes, receive packets consisting of regions in the phase space for the application. This data should then be handed over to the application (which may well then give the data to the sink-connected PC, store the data for future reference, or any other action that the sink node wishes to do). However, the format in which this data is provided to the application brings up a number of issues. Applications will probably have to be adapted to Foxtrot, but this applies to most other aggregation protocols as well (e.g. an application designed for raw data would have to be changed to use averaged values correctly). The current implementation provides the data in the standard Foxtrot format i.e. a series of phase space regions. These can be combined to provide a complete picture of the network without much effort.

One issue that can come up is that it is possible for the regions gathered by the sink node to overlap. How an application wishes to deal with this problem may well vary. The simplest options is to provide all of the possibilities for overlapping nodes e.g. a node may be marked as either being between 20 and 21 degrees celsius, or being 30-31 degrees celsius. Each measurement comes from a separate region, but because of the merging, it is unknown which answer is correct. Foxtrot does guarantee that the correct value does not lie outside the reported regions, but it is difficult to eliminate the wrong option. A number of heuristics ("pick the smallest box", "lower values are more likely", etc) have been tested against various application scenarios, but the best option will be application-specific. Alternately, the ranges can be simply averaged. This will provide less accurate values at the uncertain points, but the values will represent a reasonable compromise between the various choices. Notably, vs. conventional averaging, Foxtrot does provide information about which nodes have uncertain values, and which are certain, which may also be of use in some applications.

C. Timing issues

The simplified protocol model detailed above does not deal with the timing issues common to all in-network aggregation protocols. The first major problem is that periodic data measurement does not in general result in synchronised data. For example, if a particular application measures data every 10 minutes, and Node A's data can be aggregated with Node B's data, we have no guarantee that the two nodes will measure data at the same time, and therefore there could be up to a 10 minute delay between the measurements from the different nodes. This means that to allow aggregation, a node will have to delay the forwarding onwards of a packet for a much longer time than if the nodes are synchronised.

In order to solve this problem, we implemented a global time synchronisation algorithm related to the Global Schedule Algorithm [4], giving us synchronised cross-network timers and allowing all nodes to measure values no more than 2-3ms apart. This also gives better results for many scientific applications, as being able to know that the sensor data represents a snapshot of the monitored area over a short period of time is generally more useful than a measurement spread over a larger period, especially for cases where the source of the data values is moving.

The second problem is how much a protocol should delay before sending a packet onwards, and this has been dealt with in some detail in earlier work ([1], [3], [9]). At the moment, we are using values based on knowledge of the routing for the whole network, with a delay value based on the number of hops from the current node to the highest hop-count child node in this part of the tree.

Notably, Foxtrot only requires a solution to the problem of how long to delay a packet. Our current solution to the delay problem also requires a solution to the synchronisation problem, but other solutions can also be used with Foxtrot. Inaccurate answers will result in less optimal results (because of later and/or less aggregation) than correct ones, but Foxtrot will still work.

VI. Results

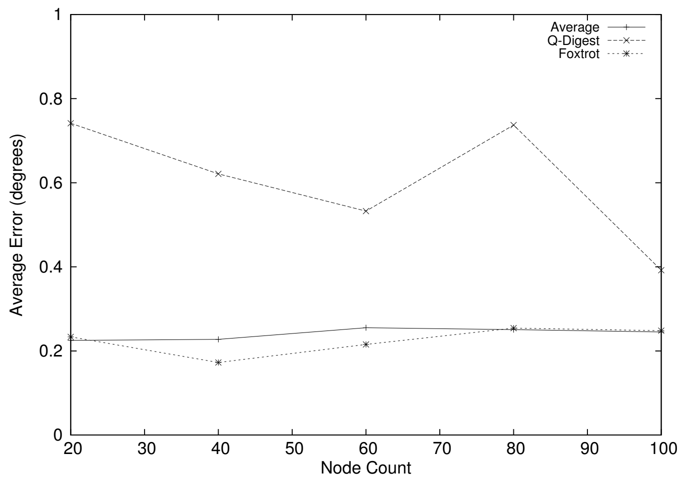

Fig. 3. Scenario 1 (Spread)

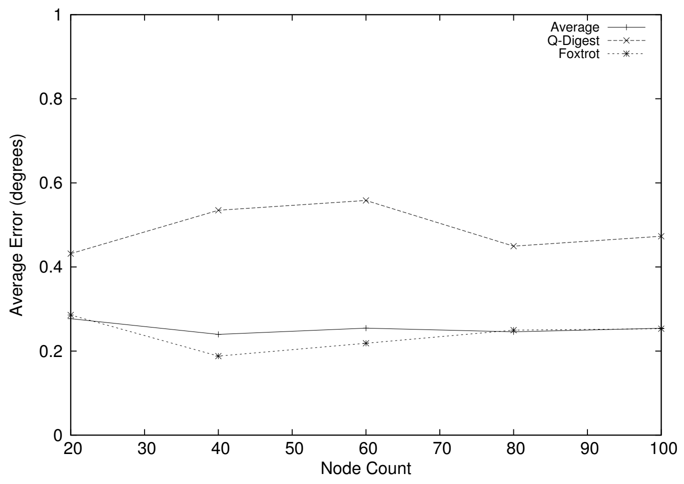

Fig. 4. Scenario 2 (Sparse)

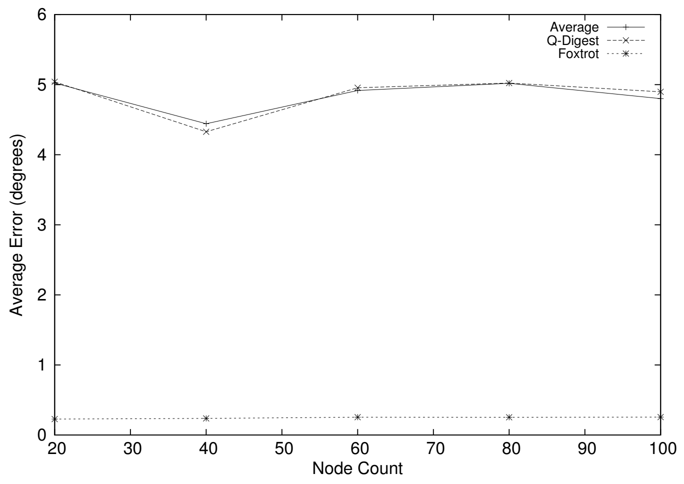

Fig. 5. Scenario 3 (Division)

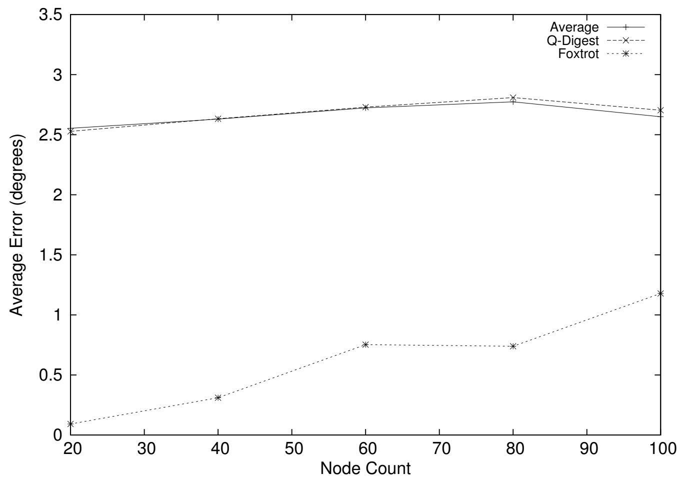

Fig. 6. Scenario 4 (Random)

We tested Foxtrot in two ways; firstly in simulation against averaging and Q-Digest, using a generic "smart" routing protocol; and secondly as a TinyOS implementation both in TOSSIM and on our node hardware testbed.

Our two metrics of interest were information loss and the number of transmitted packets. Information loss was calculated as the average per-node difference between the estimate provided to the sink node and the true data value of a sensor at each node. In all cases we ran the tests 20 times, and the data here is an average of all of those results.

The simulations modelled a network of nodes with temperature sensors, with a variety of different floating point values for the temperature readings. All tests were done in an area of 30m by 30m, with a 14m radio range. Radio links were assumed to be bi-directional and perfect. Q-Digest was run with the temperature values placed into 0.1 degree "bins" (in order to generate the integer values required by Q-Digest from our floating point temperature data), and Foxtrot set the maximum allowed temperature merging range to 1 degree. Our "smart" routing protocol used flooding from the sink to generate shortest-hop routes. Routing overhead packets were ignored for the purposes of packet counts.

We tested four scenarios for the dispersal of the temperature values:

1) Spread : Near-identical values, all nodes at values between 30 and 31 degrees
2) Sparse: 4% (1 in 25 nodes) of the nodes at between 20 and 21 degrees, all others at 30 to 31 degrees.
3) Division: Nodes on the left hand side are between 20 and 21 degrees, and nodes on the right hand side are between 30 and 31, with 50% of the total number allocated to each group.
4) Random: Random spread, all nodes between 20 and 31 degrees

The Spread and Sparse scenarios (Figures 3 and 4), are not particularly interesting, but do show that in the situations where conventional techniques are able to achieve low error values, Foxtrot is able to achieve identical performance. Q-Digest does quite badly because it provides complete information for a series of values between 30-31 that we end up having to average to get an estimate for any given node because of the lack of location data.

The Division scenarios (Figure 5) provides us with more useful results - Foxtrot's error values remain low, but the error rates for averaging and Q-Digest have both risen sharply. Similarly to Spread and Sparse, Q-Digest provides two different series of values for the overall network in this scenario, one at approximately 20 degrees, and a second at 30 degrees, reflecting an accurate picture of the network. However, due to the complete lack of location information, we are again forced to use a weighted average of the two series to derive estimates of the sensor data values, and so Q-Digest's performance is similar to averaging. The difference here is that both averaging and Q-Digest give inaccurate results, but with Q-Digest an end user would at least be aware of the situation. With Foxtrot, we get two sets of values plus location information, allowing accurate estimates of the data values, and maintaining the low error rates shown in the first two scenarios.

Foxtrot does not perform quite as well in the Random scenario (Figure 6), but this is the least likely of the scenarios to actually occur, given the correlation that tends to exist within multiple nearby readings for most physical values used by sensor nodes. Despite the low likelihood of this scenario, Foxtrot is still able to get much lower error values than other methods. In fact, one of the major sources of Foxtrot error is due to the ambiguity issues described in Section V-B. We use averaging here to resolve ambiguities, and this gives us higher errors than we might be able to achieve with more highly tuned methods.

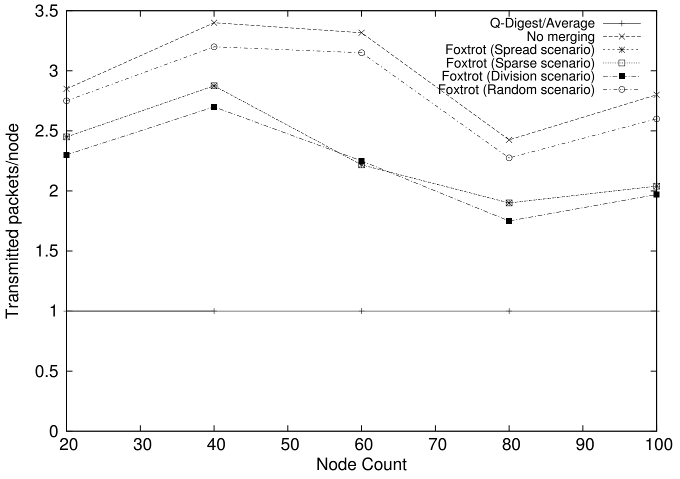

Fig. 7. Packet counts

The trade off is that Foxtrot requires more packets to be sent, as it is not necessarily able to always merge all data, which is shown in Figure 7. This graph shows average packets sent per node, in order so we can more easily see trends in the data. Firstly, Q-Digest and average both have exactly the same packet rate - 1 packet per node, as they merge everything. Foxtrot's packet rate varies substantially depending on the scenario, because the amount of unmergeable data in the network varies. For the Random scenario, the packet rate is fairly similar to the values for no merging, with only a reduction of ~7%. For scenarios 1-3 (Spread, Sparse, Division), the packet rate reduction is more substantial, with an average of a 23% reduction in overall packet transmissions.

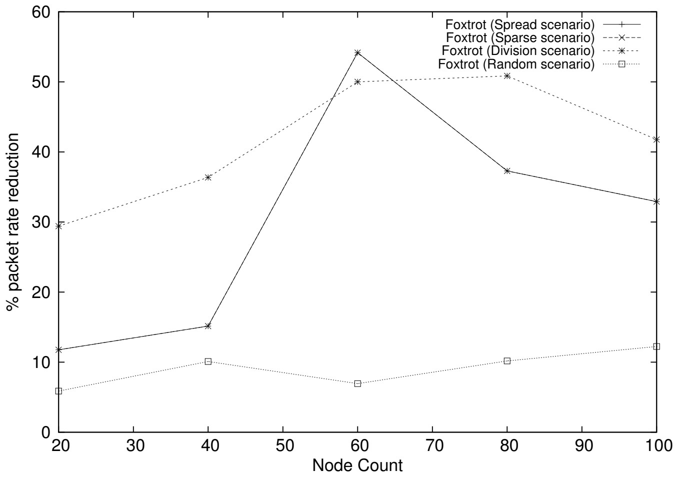

Fig. 8. Reduction in packets for sink neighbours

Different packet transmissions are not all the same in a real-world sensor network, and spreading packet load over the entire network as opposed to having most of the load near to the sink will also help to save power used by those transmissions due to to less contention and reduced idle listening time. Figure 8 shows the reduction in the number of transmitted packets vs. non-aggregated scenarios for nodes within one hop of the sink node. In Spread, Sparse and Division, we achieved an average reduction of 34% (with values up to 54% for some scenarios). Notably, Spread and Sparse get exactly the same results as each other for this test. The Random scenario had a 8.8% reduction.

To test that Foxtrot would work with actual node hardware, as opposed to just in simulation environments, we also implemented Foxtrot for TinyOS. We used Guesswork [7] to provide routing, but any other reliable sink-to-source routing algorithm would be a viable candidate. The resulting program for our mica2 derived nodes added up to a total of 44222 bytes of ROM. Getting exact values for the Foxtrot modules on their own is difficult, but the simple test program for the routing protocol takes up 38330 bytes in total, so a size for Foxtrot in the region of 6Kbytes is not unreasonable, and as that would be only 4.6% of the total program space of 128Kbytes, we can conclude that Foxtrot will not cause too many problems for application designers in terms of finding enough space on their nodes. Results from TOSSIM indicate that the TinyOS implementation behaves similarly to our earlier simulation data, and early testing with on node hardware indicate that this still holds true for when run on real hardware.

VII. Conclusions

We have shown here that existing aggregation techniques are much more lossy than earlier estimates may have thought, and that the error rates from these protocols may vary widely over the lifetime of a network. To combat these problems, we proposed Foxtrot, a limited information loss aggregation protocol. Foxtrot aggregates sensor data without significant information loss, and without losing location information. This increase in information comes at a cost in additional packet transmissions vs. more lossy aggregation protocols, but the resulting information is much more reliable due to continuously lower error rates.

Foxtrot points the way towards a new generation of sensor software, where application users will hopefully be willing to use aggregation techniques. Currently, many scientific application users have been cautious about the use of aggregation protocols, given the possibility of information loss. Techniques like Foxtrot, with its focus on information loss reduction within application-specific boundaries, may well help to persuade future projects to use aggregation without fearing the loss of experimental data.

A. Future Work

Foxtrot is a first generation attempt at limited information loss aggregation, and more research is required on the topic of creating aggregation protocols with similar aims to the ideas discussed in this paper.

Foxtrot could also be expanded in a number of ways. The dynamic sources merging algorithm is relatively conservative, and further exploration of the trade-off between accuracy and greediness for merging may find better candidates. Our use of phase space regions could also be expanded to cover other polytopes, which would allow the specifying of larger regions with less of the greediness issues.

The notion of correlation (whether multiple regions are mergeable) within Foxtrot could also be used with some routing protocols to provide additional optimisations, specifically when a locally held region is entirely enclosed by a region transmitted by another node. In this case, it is possible to discard the local region as transmitting it would not change the end results, thus further reducing required packet transmission rates. Correlation could also be used to "hint" to the routing protocol that sending a packet via a particular node would result in packet merging (and therefore reduced overall packets needed to be sent) and so this would be a good choice for the next hop node.

References

[1] Abdelzaher, T., He, T., and Stankovic, J. Feedback control of data aggregation in sensor networks. In *Conference on Decision and Control* (2004).

[2] Babu, S., and Widom, J. Continuous queries over data streams. *SIGMOD Rec. 30*, 3 (2001), 109–120.

[3] Krishnamachari, B., Estrin, D., and Wicker, S. B. The impact of data aggregation in wireless sensor networks. In *ICDCSW '02: 22nd International Conference on Distributed Computing Systems* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 575–578.

[4] Li, Y., Ye, W., and Heidemann, J. Energy and latency control in low duty cycle MAC protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference* (New Orleans, LA, USA, March 2005).

[5] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 131–146.

[6] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30*, 1 (2005), 122–173.

[7] Parker, T., and Langendoen, K. Guesswork: Robust Routing in an Uncertain World. In *2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems* (Nov. 2005).

[8] Shrivastava, N., Buragohain, C., Agrawal, D., and Suri, S. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems* (New York, NY, USA, 2004), ACM Press, pp. 239–249.

[9] Solis, I., and Obraczka, K. The impact of timing in data aggregation for sensor networks. In *In Proc. of the IEEE International Conference on Communications (ICC), 2004* (2004).

[10] Wikipedia. Central limit theorem — Wikipedia, The Free Encyclopedia, 2006. [Online; accessed 6-December-2006].

[11] Wikipedia. Phase space — Wikipedia, The Free Encyclopedia, 2006. [Online; accessed 6-December-2006].

Appendix I
Dynamically limited sources merging

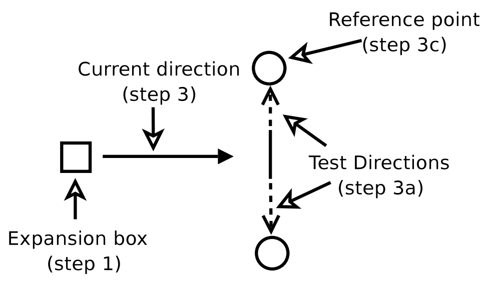

Fig. 9. Dynamically limited sources merging example in 2-D

To merge a set of regions defined by dynamically limited sources (see also Figure 9):

1) Define a initial zero-sized box in the centre of all the original regions, called Ψ.
2) Λ = set of all corners of the regions.
3) For each dynamically limited source α, perform steps a to e twice, firstly for the positive direction, and secondly for the negative direction. The current direction is specified as Υ.
   a) The set of test directions is defined as as the cartesian product $A_1 \times \ldots \times A_n$, such that $A_\beta = \{A_{pos}, A_{neg}\}$ (positive and negative) for all of the dynamic sources A_β and $\beta \neq \alpha$.
   b) Initialise a result variable Δ to the maximum possible value of α if Υ is positive, otherwise to the minimum possible value.
   c) For each test direction A_β, check if there exists a point in Λ that satisfies each direction in A_β for Ψ. For example, given a test direction $\{x_{pos}\}$, the point must have an x co-ordinate greater than or equal to the largest x co-ordinate of Ψ. Similarly, for $\{x_{neg}\}$, the point would need to have an x co-ordinate smaller than or equal to the smallest x co-ordinate of Ψ. If we have one or more points that satisfy this criterion; then if Υ is positive, set Δ to the minimum of all of their α values, else set Δ to the maximum of all of their α values.
   d) If we were unable to find one of the test values in step c), quit as these regions are not mergeable.
   e) If Υ is positive, set the maximum α value for Ψ to Δ, else set the minimum α value for Ψ to Δ.
4) If we have completed step 3 without quitting, then Ψ is a merged form of the original regions.
5) For each original region, check it against Ψ. If Ψ completely covers the original region, we can discard the original region. Alternately, Ψ may partially cover the original region. If Ψ completely covers the original region on every dynamically limited source aside from one, remove the part of the original region that is within Ψ. Otherwise, we cannot do anything with the original region.
6) If we were unable to completely cover any regions in step 5, then we have generated an extra region, and so the original regions were not mergeable. Otherwise, return the revised set of regions, including Ψ.