2. Design of Real-Time WSN Applications

This section discusses real-time application design based on data reduction algorithms, in our case, a data stream based algorithm. In Figure 1, we present the desired behavior of the data stream considering real-time requirements. Once a sensor node receives a data stream from the wireless medium the data stream is classified by the *Stream organizer*, and the *Stream processing* chooses the sample size or particular algorithm for generating the sample. The blocks are detailed as follows:

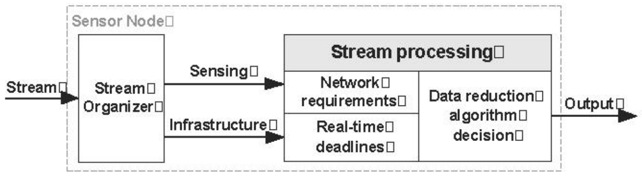


Figure 1. Real-time stream application.

The first design phase is the organization of the streams generated in the sensor network, *Stream organizer*. This can be done according to the following classification: *Sensing* is the most natural stream. It represents the sensed data in the network and its transmission model depends on the network communication type. In this case, we generally maintain the data values. *Infrastructure* is used to support network functions such as routing, data fusion, and data compression algorithms. For example, the data being forwarded in the routing process can be considered as a data stream and be processed by a data stream algorithm. In this case, the use of original data depends on the application. For each stream type we have a different treatment and deadlines. In a real-time application design, the *Stream organizer* is responsible for identifying the data stream using the information of received data. This fact allows the stream to be properly processed to attend real-time requirements.

In the second design phase, *Stream processing*, a given stream executes three stream processing functions: (i) *Network requirements*: there are different sensor network requirements (scalability, energy, packet loss, delay, and quality of data). These requirements are used to decide the best reduction solution to be used in the network. This occurs because the reduction may lead to different outputs with different "data qualities"; (ii) *Real-time deadlines*: to use deadlines in a sensor network, we must consider hard and soft real-time applications: Hard real-time applications are typically found when interacting at a low level with a physical hardware, in embedded systems. Soft real-time applications are typically found when there is a need to perform some concurrent access to a data storage from different processes. In sensor networks, it is common the use of soft real-time because the environment is not controlled. The applications usually use probabilistic methods to treat the data and have no communication acknowledgment. These aspects hide the use of hard real-time. In the design of a soft real-time system, we must know the delay behavior of the data for each solution being used. In our case, the deadline requirements of the applications can be met by the stream algorithms or the adopted sample size; and (iii) *Data reduction decision*: in the last step of the design, the decision of the best solution to treat the stream input depends on the network requirements, deadlines, and data quality. This decision can be on-line or in design level. The data quality is important because the reduction may degrades the data.

3. Data Reduction Algorithm

The data reduction problem to attend the design of Section 2, can be stated as follows:

> **Problem Statement**: Given a sensor stream, we want to meet WSN requirements by reducing data traffic (by using techniques based on data streams) and assuring a minimum data quality that allows to reduce energy consumption and delay.

This problem can be further assessed by answering the following questions: (i) *Time limits*: What are the time limits for real-time data stream applications in sensor networks? In real-time design, we must determine the time limits of our solutions. In general, a lower bound can be determined by the shortest path between source and sink nodes considering only one data stream of a given size. One stream can be used to prevent the influence of other network characteristics such as routing stack saturation; (ii) *Data reduction and data quality*: How can we evaluate the quality of the processed data? Due to network limitations and data characteristics only samples of the data stream can be sent. Thus, we must evaluate if the reduced data is representative. To perform this evaluation we can use statistic tests to know whether the original sensor stream and the sampled one are equivalent, and also compare the distance between the average of their data values; and (iii) *Project requirements*: Can the desired behavior of data stream applications, considering real-time requirements presented in Section 2, be achieved? To make it possible we need to know the behavior of the proposed solution regarding all requirements addressed in *Stream processing*. For the *Stream organizer*, we only identify the packet type. In this work, we attend the *Stream processing* requirements by using our algorithm.

The sampling based data stream algorithm is motivated by the problem stated above. The *data reduction* will be achieve by sampling the original data. This solution aims to keep the data quality and the sequence of sensor stream.

Algorithm 1: Pseudo-code of the sampling algorithm.

```
   **Data**: $\_stream[n]$ – window of original data stream
   **Result**: $\_sample[m]$ – sample set resulted
1  **begin**
2      $Sort(\_stream)$
3      $widthClass \leftarrow$ "Histogram class width"
4      $first \leftarrow \_stream[0]$
5      $numElemColStream \leftarrow 0$
6      $index \leftarrow 0$
7      $j \leftarrow 0$
8      **for** $i \leftarrow 0$ **to** $n - 1$ **do**
9          **if** $\_stream[i] > first + widthClass$ **or** $i = n - 1$ **then**
10             $numElemColSample \leftarrow \lceil m \times \frac{numElemColStream}{n} \rceil$
11             **while** $numElemColSample > 0$ **do**
12                 $index \leftarrow$ "Random element in the histogram class"
13                 $numElemColSample \leftarrow numElemColSample - 1$
14                 $\_sample[j] \leftarrow \_stream[index]$
15                 $j \leftarrow j + 1$
16             **end**
17             $numElemColStream \leftarrow 0$
18             $first \leftarrow \_stream[i]$
19         **end**
20         $numElemColStream \leftarrow numElemColStream + 1$
21     **end**
22     $Re - sort(\_sample)$ {according to the original order};
23 **end**
```

Our sampling-based algorithm provides a solution that allows the balance between best data quality and network requirements. The sample size can vary, but it must be representative to attend the data similarity requirement. According to network requirements, we can set the amount of samples between $\log n$ and n. Thus, it can attend the quality requirements in relation to the network requirements.

The sampling algorithm can be divided into the following steps: (i) Build a histogram of the sensor stream; (ii) Create a sample based on the histogram obtained before. To create such a sample, we randomly choose the elements of each histogram class, respecting the sample size and the class frequencies of the histogram. Thus, the resulting sample will be represented by the same histogram; and (iii) Sort the data sample according to its order in the original data. The pseudo-code of the sampling algorithm is given in Algorithm 1. We also consider n as the number of elements in the original data stream, and m as the adopted sample size.

Analyzing the Algorithm 1 we have: Line 2 executes in $O(n \log n)$. Lines 11–16 define the inner loop that determines the number of elements at each histogram class of the resulting sample, which takes $O(m)$ steps. Lines 8–21 define the outer loop in which the input data is read and the sample elements are chosen. Because the inner loop is executed only when condition in line 9 is satisfied, the overall complexity of the outer loop is $O(n) + O(m) = O(n + m)$ since we have an interleaved execution. Consider $numClass$ the number of histogram classes, $numElemColStream_i$ and $numElemColSample_i$, respectively, the columns in original and sampled histograms, where $0 < i \leq numClass$. Basically, before evaluating the condition of line 6, $numElemColStream_i$ is counted and $\frac{n}{numClass}$ interactions are executed. Whenever this condition is satisfied, $numElemColSample_i$ is built and $\frac{m}{numClass}$ interactions are executed (loop 11–16). In order to build the complete histogram, we must cover all classes $(numClass)$, then we have $numClass \times (\frac{n+m}{numClass}) = n + m$. Line 22 re-sorts the sample in $O(m \log m)$.

Thus, the overall complexity is $O(n \log n) + O(n+m) + O(m \log m) = O(n \log n)$, since $m \leq n$. The space complexity is $O(n + m) = O(n)$ because we store the original data stream and the resulting sample. Since every source node sends its sample stream towards the sink, the communication complexity is $O(mD)$, where D is the largest route (in hops) in the network.

4. Algorithm Evaluation in Real-Time Design

From the design phases showed in Section 2 in order to analyze the data reduction behavior in real-time environment, we consider four aspects: (i) *Stream organizer*: We use only one type of data stream arriving sensing; (ii) *Sensor network requirements*: We analyze the energy behavior using our solution because energy is the most import aspect of WSNs; (iii) *Real-time deadlines*: We analyze the delay limits for our data reduction solution and observe which deadlines can be supported by our solution; and (iv) *Data reduction decision*: We analyze the data quality behavior for our solution and discuss about the project decision from the results combination.

The evaluation of the algorithms is based on the following assumptions: (i) *Simulation*: We perform our evaluation through simulations and use the NS-2 (Network Simulator 2) version 2.30. Each simulated scenario was executed with 33 random topologies. At the end, for each scenario we plot the average value with 95% of confidence interval; (ii) *Network topology*: We use a tree-based routing algorithm called EF-Tree [11] as the routing protocol, the density is kept constant, and all nodes have the same hardware configuration. To analyze only the application, the tree is built just once, before the traffic starts; (iii) *Stream generation*: The streams used by the nodes are always the same, following a normal distribution, where the values are between [0.0; 1.0], and the generation periodicity is 60s. The size of the data packet is 20 bytes. For larger samples, these packets are fragmented by the sources and re-assembled at the reception; and (iv) *Evaluated parameters and stream size*: We varied the number of nodes, stream size, and number of nodes generating data. For each evaluated parameter we analyzed the application and network behavior by using sample sizes of $\frac{n}{2}$ and $\log n$. All parameters used in the simulations are presented in Table 1.

In order to evaluate the data quality by distribution approximation between the original and sampled streams, we

Table 1. Simulation parameters.

Parameter	Values	Parameter	Values
Network size	Varied with density	Simulation time	5000s
Queue size	Varied with stream	Traffic start	1000s
Source location	Random	Traffic end	4000s
Number of sources	$1, 5, 10, 20$	Stream periodicity	60s
Number of nodes	$128, 256, 512, 1024$	Sink location	$0, 0$
Stream size (n)	$256, 512, 1024, 2048$	Radio range (m)	50
Sample size	$\log n, n/2, n$	Bandwidth (kbps)	250

use the Kolmogorov-Smirnov test (K-S test) [12]. This test evaluates if two samples have similar distributions, and it is not restricted to samples following a normal distribution. Moreover, as the K-S test only identifies if the sample distributions are similar, it is also important to evaluate the discrepancy of the values in the sampled streams, i.e., if they still represent the original stream. To quantify this discrepancy (*Data Error*) we compute the absolute value of the largest distance between the average of the original data and the lower or higher confidence interval values (95%) of the sampled data average, $Data\,Error = Max\{|lower_{value} - Generate_{avg}|, |higher_{value} - Generate_{avg}|\}$, where the pair $(lower_{value}; higher_{value})$ is the confidence interval of data sample and $Generate_{avg}$ is the average of original data.

In the following, we show the simulation results of the data reduction algorithm to address the design requirements for real-time applications.

DEADLINE BEHAVIOR. An important issue to be considered when evaluating real-time requirements is the possible time limits of each sample size of data streams. These time limits can be very difficult to determine due to the possible dynamic conditions during the network operation, such as different number of sources, data sizes, and topologies. Thus, we performed this study by considering the amount of data sent in the network and by changing the sample sizes.

Table 2. Time limits

Stream size	n	$\frac{n}{2}$	$\log n$
256	0.62	0.36	0.11
512	1.12	0.67	0.11
1024	1.98	1.19	0.11
2048	3.70	1.94	0.11

In order to do this, we consider an network with a fixed size of 256 nodes, running the tree-based routing infrastructure, and only one source of data generating streams with different sizes. We use the lowest average delay obtained through simulations by considering random topologies and a tree with the minimum number of hops between the nodes. The delay is determined by measuring the time between the first data packet sent by the source and the last packet received by the sink, i.e., it is the time for a stream to be entirely received by the sink. Table 2 summarizes the time limits, in seconds, for our data reduction algorithm using different sample strategies and the considered data stream sizes. These limits are used in the *Real-Time Deadline* module present in Section 2 to help in network decision when our solution is applied in a real-time environment. So, if the deadline requirement met this time limits our solution can be used in the real-time application.

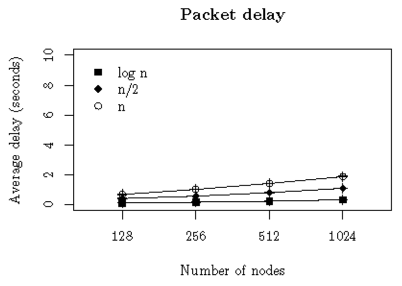
A more detailed evaluation of the delay performance is presented in Figure 2. This evaluation considers the delay of the entire network to delivery a data packet to the sink. In this evaluation, we use different sample sizes ($\log n$ and $\frac{n}{2}$) and the complete sensor stream (n). These cases are analyzed with different network scenarios by varying the network size, the amount of generated data at the source, and the number of sources.

We observe in all cases that when the sample size is diminished the delay diminishes accordingly. The $\log n$ sample is the best result because the number of elements in packet has a little increased. Analyzing the figures separately, when the number of nodes varies (Figure 2(a)), the delay varies a little. This occurs because only one source is used, and both the size of the sensor stream and the network density did not change. In this scenario, the $\log n$ sample has less impact on the delay.

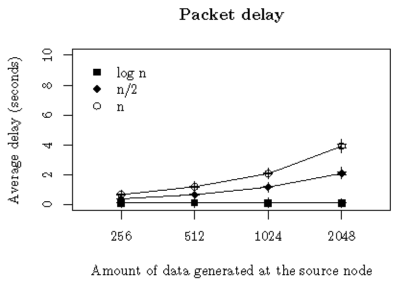
When the size of the sensor stream varies (Figure 2(b)), we can observe the impact of our solutions in the delay. The $\log n$ sample has the best performance in all cases, and the delay does not vary when the sample size increases. In the $\log n$ sample, this occurs because the number of elements in packet is increased only when we increase the sensor stream size (256, 512, 1024, 2048). The other results (samples of $\frac{n}{2}$ and n) have worse performances because the number of packets is increased proportionally when the sensor streaming size is increased.

When the number of nodes generating data varies (Figure 2(c)), the sample of $\log n$ have the best performance for all cases. This occurs because, in this scenario, more packets are passing through the network when we increase the number of nodes generating data. Each source using the sample of $\log n$ uses only one packet (the packet size is no more than 20 bytes) to send its data to the sink. For the other results (samples of $\frac{n}{2}$ and n) each source node generates more than one application packet, overloading the network, and causing delay. These results are close to time limits showed in Table 2.

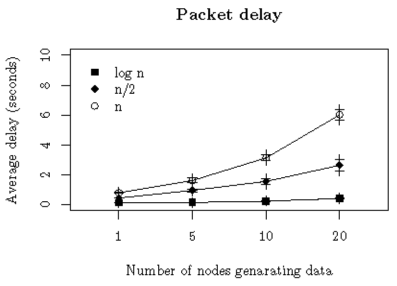
ENERGY BEHAVIOR. This evaluation considers the energy consumption of the entire network to delivery a data packet to the sink. Again, we use different sample sizes ($\log n$ and $\frac{n}{2}$) and the complete sensor stream (n). These cases are analyzed with different network scenarios by varying the network size, the amount of generated data at the source, and the number of sources. According to the delay behavior, as a result when the sample size decreases, the consumed energy decreases for the same reason that delay behavior. Again, the same effect of the number of node variation is observed. When the sensor stream size and number



(a) Different network sizes.



(b) Different stream sizes.



(c) Different number of sources.

Figure 2. Average delay.

of nodes generating data (Figure 3) vary we can observe the impact on the energy when our solution is used. In the all cases, the sample $\log n$ has the best performance. These results are used in the *Sensor Network Requirement* module present in Section 2 to help deciding when the application needs to save energy.

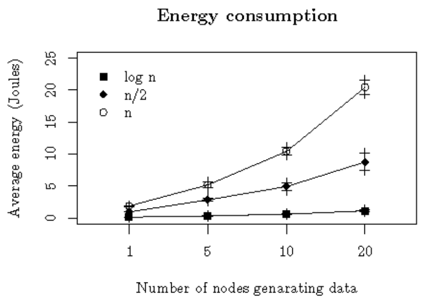


Figure 3. Total energy consumption.

DATA QUALITY BEHAVIOR. Here, we present the impact of our solution by evaluating the data quality. The sampling solution looses information in its process, therefore it is important to evaluate its impact on the data quality. Again, the impact of the sampling solution is made through the K-S test and the average error. Like the network evaluation, we use different sample sizes ($\log n$ and $\frac{n}{2}$) and the complete sensor stream (n) in different network scenarios. We vary the network size, the amount of data generated at the source, and the number of sources. Here, we show only the number of sources result because in all scenarios we have the same behavior.

Figure 4 shows the similarity between the original and sampled stream distributions. The difference between them is called *ks-diff*. The results show that when the sample size is decreased the *ks-diff* increases. Because the data streams are generated between $[0.0; 1.0]$, *ks-diff* is 20% for $\log n$ sample sizes, and *ks-diff* is 10% for $\frac{n}{2}$ sample sizes. The error is constant, since the data lost is small. The greatest error occurs when we use a smaller sample size but the data similarity is kept.

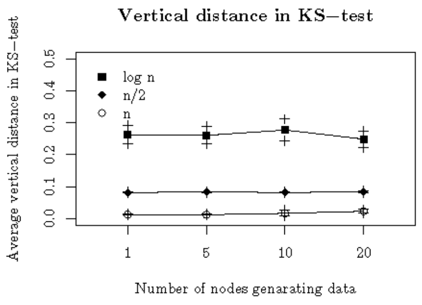


Figure 4. K-S distance.

We also evaluate the data quality through the discrepancy between the original and sampled stream average values (Figure 5). This error we call *data-error*. Like the *ks-diff*, when the sample size decreases, the *data-error* increases. However, *data-error* is 10% for samples of $\log n$, and *data-error* is almost zero for samples of $\frac{n}{2}$. Again, the error is constant for the same reason of the *ks-diff*. However an important observation is that the *data-error* is the same for samples of $\frac{n}{2}$ and n. Therefore, if we want to keep the maximum data quality, considering the *data-error*, we can send only samples of $\frac{n}{2}$. These results are used in the *Data Reduction Decision* module presented in Section 2 for help in decision about the data quality when the application require quality in data processing.

In summary, when we analyze the combination of data quality, network behavior and deadlines in the *Data Reduction Decision* module, presented in Section 2, we conclude that: (i) the sample of $\log n$ reduces the energy consumption and delay by reducing the amount of data being transmitted. However, the data quality is affected in the distribution similarity (20%) and average discrepancy (10%). But

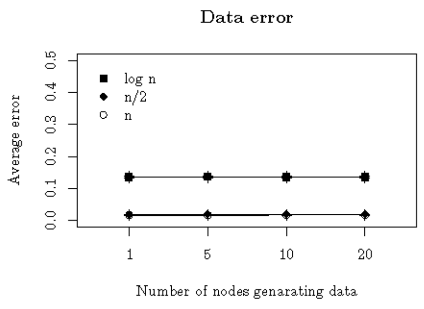


Figure 5. Average error.

this quality may be acceptable in WSNs applications when the network restrictions are strong; (ii) the sample of $\frac{n}{2}$ is interesting when the application priority is the average discrepancy (near zero); and (iii) it is interesting not to use our algorithm (sample of n) when we have to keep the same data quality similarity and we do not have to worry about the WSN restrictions. The results answer the questions *Data reduction and data quality* and can be applied to the problem addressed in Section 3.

5. Conclusion

In real-time wireless sensor networks applications, a very important requirement is the time to deliver such data streams to the sink, and, as discussed in this work, the amount of data in transit through these constrained networks has a great impact on the delay. Thus, we proposed and evaluated a based data stream algorithm that uses a sampling over a histogram technique to reduce data traffic, and consequently the delay and energy consumption. This work represents a way to deal with energy and time constraints at the application level, as a complementary view to solutions that treat this problem in lower network levels.

The results show the efficiency of the proposed method by extending the network lifetime — since data transmission demands lots of energy — and reducing the delay without losing data representativeness. Such a technique can be very useful to achieve energy-efficient and time-constrained sensor networks if the application is not so dependent on the data precision or the network operates in exception situation (e.g., few resources remaining or urgent situation detection). Moreover, the proposed method meets the design requirements in real-time WSNs.

As future work, we intend to better evaluate the proposed technique by considering other network scenarios, and matching the proposed application-level solution with lower-level ones, for example, by considering some real-time-enabled routing protocol. We also plan to apply the proposed method to process data streams along the routing task. Thus, not only the data from a source is reduced, but similar data from different sources is also reduced, resulting in a more energy-efficient solution.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[2] L. S. Buriol, D. Donato, S. Leonardi, and T. Matzner. Using data stream algorithms for computing properties of large graphs. In *Workshop on Massive Geometric Data Sets (MASSIVE'05)*, Pisa, Italy, June 9 2005.

[3] T. H. Chan, C. K. Ki, and H. Ngan. Real-time support for wireless sensor networks. Unpublished, 2005.

[4] J. Ledlie, C. Ng, D. A. Holland, K.-K. Muniswamy-Reddy, U. Braun, and M. Seltzer. Provenance–aware sensor data storage. In *1st IEEE International Workshop on Networking Meets Databases (NetDB)*, April 2005.

[5] H. Li, P. J. Shenoy, and K. Ramamritham. Scheduling communication in real-time sensor applications. In *IEEE Real-Time and Embedded Technology and Applications Symposium – (RTAS'04)*, pages 10–18, Toronto, Canada, 25–28 May 2004. IEEE, IEEE Computer Society.

[6] A. Lins, E. F. Nakamura, A. A. Loureiro, and C. J. Coelho Jr. Beanwatcher: A tool to generate multimedia monitoring applications for wireless sensor networks. In A. Marshall and N. Agoulmine, editors, *Management of Multimedia Networks and Services*, volume 2839 of *Lecture Notes in Computer Science*, pages 128–141, Belfast, Northern Ireland, September 2003. Springer-Verlag Heidelberg.

[7] A. Lins, E. F. Nakamura, A. A. Loureiro, and C. J. Coelho Jr. Generating monitoring applications for wireless networks. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, Lisbon, Portugal, September 2003.

[8] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *IEEE Real Time Technology and Applications Symposium – (RTAS'02)*, pages 55–66, San Jose, CA, USA, 24–27 September 2002. IEEE, IEEE Computer Society.

[9] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, March 2005.

[10] S. Muthukrishnan. Data streams: Algorithms and applications. In *4th ACM-SIAM Symposium on Discrete algorithms*, Baltimore, Maryland, 2003.

[11] E. F. Nakamura, F. G. Nakamura, C. M. S. Figueredo, and A. A. F. Loureiro. Using information fusion to assist data dissemination in wireless sensor networks. *Telecommunication Systems*, 30(1/2/3):237–254, November 2005.

[12] S. Siegel and J. N. John Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill College, 2nd edition edition, January 1988.