A Multi-Level Methodology for Explaining Data Streams

LUKA STOPAR, Jozef Stefan Institute
PRIMOZ SKRABA, Jozef Stefan Institute
DUNJA MLADENIC, Jozef Stefan Institute
MARKO GROBELNIK, Jozef Stefan Institute

This paper presents a novel multi-scale methodology for modeling a collection of continuously time-varying data streams. The data streams are aggregated using unsupervised data mining methods. Typical system states are then computed on the aggregated data. This is used as input to construct a Markovian transition model capturing the dynamics of the monitored system. The hierarchical organization of the states enables a visual representation of the dynamics on multiple aggregation levels.

CCS Concepts: **•Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; **•Networks** → Network reliability;

Additional Key Words and Phrases: [TODO]

ACM Reference Format:
Luka Stopar, Primoz Skraba, Dunja Mladenic, Marko Grobelnik, 2015. A Multi-Level Methodology for Explaining Data Streams. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 10 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Sensory systems typically operate in cycles with a continuously time-varying. These systems include, for example, the solar system, manufacturing systems or weather systems. Such systems can be characterized by a set of states, along with associated state transitions. States, on a high level, may include a ”day” state and a ”night” state or maybe states with high and low productivity. For example, when a pilot wishes to change an aircrafts heading, they will put the aircraft into state ”banking turn” by lowering one aileron and raising the other, causing the aircraft to perform a circular arc. After some time, the wings of the aircraft will be brought level by an opposing motion of the ailerons and the aircraft will go back into state ”level”.

Such high-level states can be decomposed into lower-level states, giving us a multi-level view of the system and allowing us to observe the system on multiple aggregation levels. For example, a ”banking turn” state can be decomposed by the aircrafts roll and angular velocity, resulting in perhaps three states: ”initiate turn”, ”full turn” and ”end turn”.

We present a methodology for modeling such systems and demonstrate its implementation called StreamStory. StreamStory models such systems as a hierarchical Markovian process by automatically learning the typical low-level states and transi-

[TODO] This work is supported by the National Science Foundation, under grant CNS-0435060, grant CCR-0325197 and grant EN-CS-0329609.
Author's addresses: G. Zhou, Computer Science Department, College of William and Mary; Y. Wu and J. A. Stankovic, Computer Science Department, University of Virginia; T. Yan, Eaton Innovation Center; T. He, Computer Science Department, University of Minnesota; C. Huang, Google; T. F. Abdelzaher, (Current address) NASA Ames Research Center, Moffett Field, California 94035.

DOI: 0000001.0000001

tions, and aggregating them into a hierarchy of Markovian processes. As such it gives users a unique view into the monitored system.

Furthermore, we divide the inputs streams into two sets: observation and control set. Attributes in the observation set are the attributes that tell us the state of the system and, we assume, cannot directly influence its dynamics. These are parameters that users cannot directly manipulate, like aircraft tilt from the previous example, which must be indirectly manipulated through the angles of ailerons. We use observation attributes to identify, and aggregate, low-level states, detect outliers (anomalies) and determine the current state of the system.

In contrast, users can directly manipulate attributes in the control set. These are attributes like angles of ailerons that may directly influence the behavior (observation attributes) and performance of the system. For example, when an operator in a steel factory sets the cooling temperature to a high value, the product will take longer to go from state ”hot” to state ”cool”. As such, we assume, control attributes may influence the occurrence, and expected time, of undesired states, associated with undesired events.

Our system uses control attributes to model state transitions, allowing us to observe the dynamics with respect to the current configuration and gives us insight into the expected dynamics with respect to some alternate attribute configuration.

We implement our methodology in a four step process, which includes:

(1) aggregating and resampling the input streams, interpolating wherever needed, producing feature vectors,
(2) clustering the feature vectors to obtain lowest-level states and computing statistics on each state,
(3) aggregating lowest-level states into a state hierarchy and computing the level on which each aggregated state lives and
(4) modeling state transitions with respect to control attributes and computing statistics on transitions.

[TODO The remainder of this paper is structured as follows]

2. PROPOSED MULTI-LEVEL METHODOLOGY

To implement our multi-level approach, we propose a three step methodology shown in figure 1.

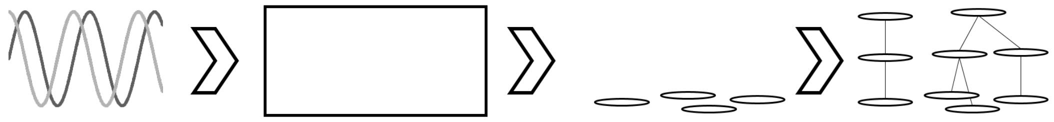

Fig. 1. The proposed multi-level methodology.

As shown in the figure, we split our methodology into three main steps: merging and processing data streams, state aggregation and transition modeling. These will be explained in detail in the following subsections.

2.1. Merging and Processing Data Streams

In recent years much attention of the research community has been focused on a class of applications where the data are modeled best not as persistent relations, but rather as transient data streams [Babcock et al. 2002]. Data streams differ from conventional batch data in several ways:

(1) the data elements arrive online, sequentially and the system has no control over their order,
(2) data streams are potentially unbounded.

As such a data stream can be defined as a sequence of pairs $(t_i, x_i)_{i \geq 0}$ where t_i is the timestamp of the i-th example and x_i is its value. In the first step our methodology consumes batches of multiple such streams and merges them to produce feature vectors, interpolating the values wherever appropriate resulting in a [TODO].

2.2. State Aggregation

To be able to represent the data streams in a qualitative manner

Once the data streams are merged we construct a state hierarchy. We do this by constructing a dendrogram of the previously constructed feature vectors. A dendrogram is a nested sequence of partitions with associated numerical level and is thus ideal for our multi-level representation. The literature proposes many algorithms for constructing dendrograms. These generally fall into two groups [Maimon and Rokach 2005]:

(1) **Agglomerative** - each object initially represents a cluster of its own. Clusters are then iteratively merged, until the desired structure is obtained.
(2) **Divisive** - when the procedure starts, all object belong to the same cluster. Then clusters are recursively divided into sub-clusters until the desired structure is obtained.

In general however, the time complexity of agglomerative clustering algorithms is $\Omega(n^2)$ [Sibson 1973], which is impractical for large datasets. Thus we compute an initial set of partitions using k-means and use them as singletons of the dendrogram as well as the lowest level state space in our methodology.

2.3. Modeling Transitions

We model transitions using a Markovian model. The main characteristics of Markovian models is that they retain no memory of where they have been in the past. This means that only the current state can influence where the process will go next. In this work we are interested only in processes that can assume a finite set of states, called Markov chains [Norris 1998]. More formally, we are interested in Markov chains $(X_t)_{t \geq 0}$ which can assume values in a finite state space S. We define $p_{ij}(t)$ to be the probability of the process being in state j at time t when starting from state i at time 0. We define a stochastic matrix $P(t)$, where $(P(t))_ij = p_{ij}(t)$. Thus each row of $P(t)$ is a probability distribution over the state space. Such a chain can be represented by a transition rate matrix Q, where Q satisfies the following system of equations:

$$\frac{d}{dt}P(t) = P(t)Q, \qquad P(0) = I. \tag{1}$$

Each non-diagonal entry of Q thus represents the rate of going from one state to another. Q has the following properties:

The ij-th entry of Q then represents the rate of going from state i to state j.

We assume that $P(t)$ is recurrent for all t and define $\pi_j = \lim_{t \to \infty} p_{ij}(t)$. It can be shown that for recurrent chains π_j is independent of the starting state. The vector $\pi = (\pi_1, \pi_2, ..., \pi_k)$ is called the stationary distribution of $(X_t)_{t \geq 0}$ and represents the proportion of time the process spends in each state after funning for an infinite amount of time.

(1) $-\infty < q_{ii} \leq 0$
(2) $q_{ij} \geq 0$ for $j \neq i$
(3) $\sum_{j \in S} q_{ij} = 0$

Using this representation, the stationary distribution π can be computed as the left eigenvector of Q corresponding to eigenvalue 0.

2.4. Transition Rate Estimation

We estimate the transition rates of $(X_t)_{t \geq 0}$ by first discretizing the the continuous parameter space $t \in [0, \infty)$ into a discrete sequence $(0, h, 2h, ...)$ and estimating the transition probabilities $\tilde{p}_{ij} = p_{ij}(h)$. Once $\tilde{p}_{ij}$ are estimated, the transition rates can be calculated as

$$q_{ij} = \frac{h}{\tilde{p}_{ij}}. \tag{2}$$

As proposed in section 1 we allow the users to simulate the dynamics based on an alternate configuration of the attributes in the control set A_S. Lets say the process is in state i at time 0 and define a random variable $J_i = j \Leftrightarrow X_h = j$. Thus J_i has a multinomial distribution with parameters $(p_{i1}, p_{i2}, ..., p_{in})$ which can be modeled using a generalized linear model (GLM) [Dobson and Barnett 2008]. Because there is no natural ordering in the response category, we use nominal logistic regression to estimate p_{ij} based on values $x_k \in A_S$. We select reference category as p_{ii} and model the relationship between x_k and p_{ij} as follows:

$$\text{logit}(p_{ij}) = \log\left(\frac{p_{ij}}{p_{ii}}\right) = \beta_{ij} x_k \tag{3}$$

thus, for each state we get a set of $n - 1$ equations which are used simultaneously to estimate the parameters β_{ij}. Once these have been obtained, the linear predictors can be calculated as:

$$\tilde{p}_{ij}(x_k) = \frac{e^{\beta_{ij} x_k}}{1 + \sum_{l \neq i} e^{\beta_{il} x_k}}. \tag{4}$$

2.5. State Aggregation

Once the low level process is being modeled we need to be able to model it on higher levels. Recall from section 2.2 that our methodology partitions the data space using a dendrogram. A dendrogram is a nested sequence of partitions with associated numerical levels. Thus it provides, on each level l, a unique partition of the data space $P^l = (P_1, P_2, ...P_{k_l})$ where each partition P_i is the union of one or more partitions on level 0. Since each state of the Markov chain corresponds to a single partition on level 0, we aggregate then to be able model the higher level process on level l. Suppose the data space is partitioned into partitions $(P_1, P_2, ..., P_k)$ on level 0.

We note that simply lumping lower level states is not sufficient since it may result in a non-Markovian process [TODO ref]. Therefore we must construct a new Markov chain $(Y_t)_{t \geq 0}$ which should retain some the behavior of the original chain $(X_t)_{t \geq 0}$. Let $(X_t)_{t \geq 0}$ be the Markov chain on level 0 with state space $S = 1, 2, ..., n$ and transition rate matrix Q. Now suppose we would like to aggregate two sets of states $A = \{i_1, i_2, ..., i_k\} \subset S$ and $B = \{j_1, j_2, ..., j_h\} \subset S$ and would like to preserve the stationary distribution of the original process. Let $\tilde{\pi}$ be the stationary distribution of $(Y_t)_{t \geq 0}$. Then $\tilde{\pi}$ should have the following properties:

(1) $\tilde{\pi}_A = \sum_{i \in A} \pi_i$, $\tilde{\pi}_B = \sum_{j \in B} \pi_j$
(2) $\tilde{\pi}_k = \pi_k$ for all $k \notin A \cup B$.

We can compute the transition rate matrix of such a process using the following formula:

$$q_{AB} = \frac{\sum_{i \in A} \pi_i \sum_{j \in B} q_{ij}}{\sum_{i \in A} \pi_i}. \tag{5}$$

2.6. Visualization

[TODO]

3. THEORY

This section presents the theory behind the StreamStory systems. We begin with a general discussion about data streams, then we give some insight into the theory behind clustering algorithms and stochastic processes.

Data streams can be defined by a set of tuples (t_i, x_i), where t_i represents the timestamp of the i-th observation x_i. They can be characterized by the following:

(1) the elements of the data stream arrive sequentially, and the developer has no control over their timing,
(2) each stream can provide elements at its own schedule: they do not need to have the same data rates or types
(3) a data stream is potentially unbounded in size.

A data stream may contain sensor data, image data or maybe internet and web traffic. High dimensional data streams arise in many applications. In many cases these are stochastic in nature or can be approximated by a stochastic process [Crosskey and Maggioni 2014].

One of the major challenges when modeling such processes is the ability to reach experimentally relevant timescales [Pande et al. 2010].

Markov state models build models with N states and parametrize the model with the rates between states. The challenges when building Markov state models include [Pande et al. 2010]:

(1) defining the states of the model and
(2) using the state decomposition to build a transition matrix in an efficient manner.

To define the models' states, we chose to partition the data streams.

3.1. Gaussian Mixture Models and K-means

A mixture model is a probabilistic model that represents the presents of subpopulations in an observed population. A mixture model corresponds to a mixture distribution that represents the distribution of observations in the overall population. In a Gaussian mixture model, the data is assumed to arise from the following distribution:

$$p(x) = \sum_{i=1}^{k} \pi_i N(\mu_i, \Sigma_i) \tag{6}$$

where k is the number of components, π is the mixing coefficient and μ_i and Σ_i are parameters of a Gaussian distribution. [TODO] A related model is provided by the K-means objective function, which constructs a hard partitioning of the data set. Given a set of points $x_1, x_2, ..., x_n$, the K-means objective function attempts to find partitions $c_1, c_2, ..., c_k$ that minimize the following objective function:

$$\min_{bounds} \sum_{j=1}^{k} \sum_{x_i \in r(j)} \|x_i - \mu_j\|_2^2 \, where \mu_j = \frac{1}{|r(j)|} \sum_{x_i \in r(j)} x_i \qquad (7)$$

3.2. What is left

Once the model is build it can be used for quantitative simulations as well as qualitative interpretation.

To define the models' states, we use

4. IMPLEMENTATION

This section presents the detailed implementation of the StreamStory system, by presenting in details each step of our four step process. We begin the section by first discussing the preprocessing step proposed in step 1 of our four step process defined in section 1.

4.1. Data Stream Aggregation and Resampling

In recent years much attention in the research community has been focused on a new class of applications: applications in which the data is modeled best not as persistent relations but rather as transient data stream [Babcock et al. 2002].

Data streams differ from conventional batch data in several ways:

- the data elements arrive online, sequentially and the system has no control over their order and
- data streams are potentially unbounded.

As such a data stream can be defined as a sequence of pairs $(t_i, x_i)_{i \geq 0}$, where t_i represents the timestamp if the i-th measurement and x_i represents its value. In our work we assume the streams follow the following model: $x_{i+1} = x_i + X$, where X is a normally distributed random variable $X \sim N(\mu, \sigma)$. [TODO really???]

To build a model in offline mode, StreamStory consumes batches of multiple such data streams and merges them, to that all the stream are sampled at the same timestamps. This results in a single joined data stream in the following form:

$$\left\{ \left(t_1, x_1^{(1)}, x_1^{(2)}, ..., x_1^{(d)}\right), \left(t_1 + \Delta t, x_2^{(1)}, x_2^{(2)}, ..., x_2^{(d)}\right), ..., \left(t_0 + (k-1)\Delta t, x_k^{(1)}, x_k^{(2)}, ..., x_k^{(d)}\right) \right\}$$

Since in general not all the input streams are equally sampled, missing values need to be interpolated. For this purpose StreamStory supports two interpolation methods: linear and previous point interpolation. The choice of the interpolation method influences the online behavior of the system. Although in theory linear interpolation produces better results [TODO ref], linear interpolation cannot interpolate a value until it observes at least one value in the [TODO future]. Which results in a certain lag (non-real-time behavior) of the system.

4.2. State Identification

Once the data streams are preprocessed, StreamStory identifies their typical lowest-level states. This is achieved by clustering the joined data stream.

Clustering is an unsupervised machine learning procedure, which organizes a collection of patterns, usually represented as a vectors of measurements or points in a multidimensional space, into clusters based on a chosen measure of similarity or distance [Jain et al. 1999]. There is no universally agreed upon definition of a cluster.

Most researchers describe it by considering the internal homogeneity and external separation [Xu and Wunsch 2005].

Intuitively patterns in the same cluster should be more similar to each other than to patterns in other clusters. There are many clustering algorithms that solve different problems. These include partitioning, [TODO].

We support two partitioning methods: K-Means and DPMeans. The first is a well studied method which, given k - the desired number of clusters, partitions the dataset into k clusters by minimizing the distance between data points and the nearest centroid [Coates and Ng 2012]. It works by, initially selecting k random centroids, then in each iteration it first assigns all the feature vectors to their nearest centroid constructing a Voronoi diagram regions are associated with centroids $C = \{c_1, c_2, ..., c_k\}$. More formally a region of the diagram is defined as $R(c_i) = \left\{x \in \mathbb{R}^d | d(c_i, x) \leq d(c_j, x) \, \forall j\right\}$. The method then recomputes each centroid c_i as the mean value of all the data points that lie in region $R(c_i)$:

$$c_i = \frac{1}{\|R_i\|} \sum_{p_j \in R(c_i)} p_j.$$

The procedure terminates when the partitions are the same in two consecutive iterations.

The second method, DP-means, proposed by [Kulis and Jordan 2011] behaves similarly to K-Means with the exception that a new cluster is formed whenever a feature vector is further than λ away from every existing centroid. Thus, the number of clusters on the output is not known in advance, but is controlled by λ.

When identifying states, we first remove timestamps from the joined data stream, resulting in feature vectors of the form

$$x_i = \left(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(d)}\right).$$

These are then clustered using one of the above methods, resulting in a set of k partitions which are used as lowest level states. When StreamStory [TODO sees] a new data point it will assign it to the state (partition) with the nearest centroid.

$$S(p_i) = \underset{p_j \in R(c_j)}{\operatorname{argmin}} \, d(p_i, c_j)$$

Where d represents the Euclidean distance.

4.3. State Aggregation

Once the lowest level states are computed, StreamStory aggregates them into a state hierarchy. Before grouping states, we need to measure their relative distance to each other. Since StreamStory was developed to work with dense dataset, we use the Euclidean distance. Many hierarchical clustering techniques have been proposed in the literature. They can be characterized as greedy in the algorithmic sense [Murtagh and Contreras 2011]. Assuming that a pair of states is merged or agglomerated at each step, the techniques construct a binary tree commonly known as a dendrogram. This produces a set of states at each level - or each threshold value which produces a new partition. Hierarchical clustering methods generally fall into two groups [Maimon and Rokach 2005]:

- **Agglomerative** - each object initially represents a cluster of its own. Clusters are then iteratively merged, until the desired structure is obtained.

- **Divisive** - when the procedure starts, all object belong to the same cluster. Then clusters are recursively divided into sub-clusters until the desired structure is obtained.

These groups can be further subdivided according to the manner that the distance measure is calculated:

- **Single-link** - methods that consider the distance between two clusters equal to the minimum distance of any member of one cluster to any member of the other.
- **Complete-link** - methods that consider the distance between two clusters equal to the maximum distance of any member of one cluster to any member of the other.
- **Complete-line** - methods that consider the distance between two clusters equal to the average distance of any member of one cluster to any member of the other.

StreamStory uses an agglomerative clustering technique and supports all three of the above mentioned linkage strategies.

4.4. Modeling Transitions

We model transitions using a Markovian model. The main characteristic of Markovian models is that they retain no memory of where they have been in the past. This means that only the current state can influence where the process will go next. In this work, we are interested only in processes that can assume a finite set of states, called Markov chains [Norris 1998].

More formally, we are interested in Markov chains $(X_k)_{k \geq 0}$, which can assume values in a finite state space S. We define $p_{ij}(k) = P(X_k = j | X_0 = i)$ to be the probability of the process being in state j at time k when starting from state i at time 0. Because of the memoryless property $P(X_k = j | X_0 = i) = P(X_{k+m} = j | X_m = i) = p_{ij}(k)$.

We define a stochastic matrix $P(t)$, where $(P(t))_{ij} = p_{ij}(t)$. Thus the i-th row of $P(t)$ is a probability distribution over the state space at time t when the process starts from state i. We assume that $P(t)$ is recurrent and define $\pi_j = \lim_{t \to \infty} p_{ij}(t)$. Then $\pi = (\pi_1, \pi_2, ..., \pi_m)$ is the stationary distribution of $(X_t)_{t \geq 0}$ and represents the proportion of time the process spends in each state after running for an infinite amount of time.

Let us now observe the elements if $P(t)$. Since each row of $P(t)$ is a probability distribution it is clear that $p_{ij}(t) \geq 0$ and $\sum_{j \in S} p_{ij}(t) = 1$. Furthermore $p_{ij}(t)$ must satisfy the memoryless property $P(X_{t+s} = j | X_s = i) = p_{ij}(t)$.

4.4.1. State Aggregation.

When observing the process on multiple resolutions we need a way to aggregate states in S. When aggregating states of the Markov chain, we choose a formula which preserves the stationary distribution. Suppose A and B are non-intersecting subsets of S. Then we can calculate the probability of the process going from set A to set B for any t as follows:

$$p_{AB}(t) = \frac{\sum \pi_i \sum_{j \in A} p_{ij}(t)}{\sum_{i \in B} \pi_i} \quad (8)$$

4.5. Visualization

[TODO]

5. VISUALIZATION AND USER INTERACTION

When interacting with the StreamStory system, the user is presented with a two panel user interface shown in figure 2.

The visualization panel on the left visualizes the hierarchical Markovian model. States are represented by circles, while transitions are represented by arrows. The

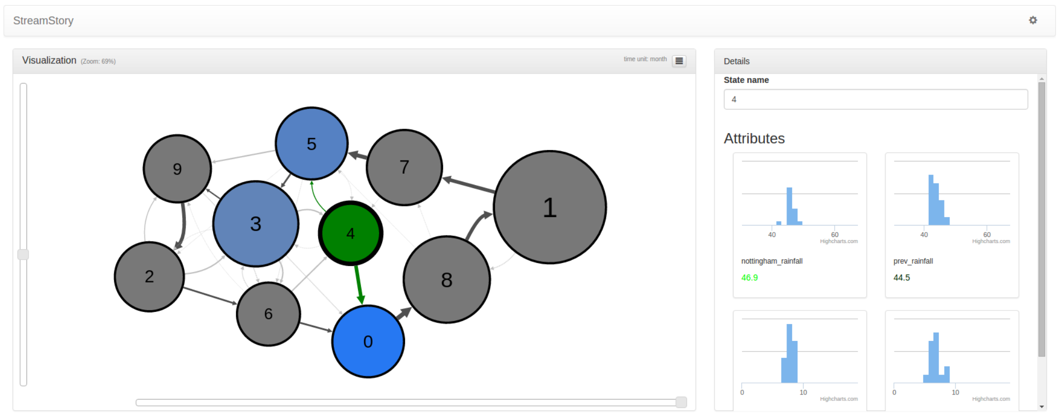

Fig. 2. User interface of the StreamStory system.

size of a state is proportional to the fraction of time the monitored process spends in the state. When StreamStory is used in online mode, the current state is colored green, the most likely future states blue and the previous state has a red border.

The thickness of an arrow is proportional to the probability of the corresponding transition, and the most likely transitions have a darker color.

When first opening user interface the user is presented with a top-level chain with only two states. They can then use the scroll function to zoom into the hierarchy and the states get automatically expanded.

By clicking on a state, the state becomes selected and its details are shown in the "Details" panel on the right side of figure 2. The details panel allows the user to name the state which, we believe, increases the models interpretability. It also shows the distribution of all the parameters inside the state as well as the mean value of each parameter in the state. The mean value is highlighted red or green depending on how specific the value is for that state. If the value is gray it means that the value of the attribute is normal compared to the value of the same attribute in other states. However, if the value is green or red, it indicates the value is the value is specific for this state compared to other states.

This is achieved by classifying the instances of the selected state against instances off all the other states on the same level using a logistic regression model [TODO ref] and extracting weights. Values highlighted green indicate a positive weight, while values highlighted red indicate a negative weight.

APPENDIX

In this appendix, we measure the channel switching time of Micaz [CROSSBOW] sensor devices. In our experiments, one mote alternatingly switches between Channels 11 and 12. Every time after the node switches to a channel, it sends out a packet immediately and then changes to a new channel as soon as the transmission is finished. We measure the number of packets the test mote can send in 10 seconds, denoted as N_1. In contrast, we also measure the same value of the test mote without switching channels, denoted as N_2. We calculate the channel-switching time s as

$$s = \frac{10}{N_1} - \frac{10}{N_2}.$$

By repeating the experiments 100 times, we get the average channel-switching time of Micaz motes: 24.3μs.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Maura Turolla of Telecom Italia for providing specifications about the application scenario.

REFERENCES

Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02)*. ACM, New York, NY, USA, 1–16. DOI:http://dx.doi.org/10.1145/543613.543615

Adam Coates and AndrewY. Ng. 2012. Learning Feature Representations with K-Means. In *Neural Networks: Tricks of the Trade*, Grgoire Montavon, GeneviveB. Orr, and Klaus-Robert Mller (Eds.). Lecture Notes in Computer Science, Vol. 7700. Springer Berlin Heidelberg, 561–580. DOI:http://dx.doi.org/10.1007/978-3-642-35289-8_30

M. Crosskey and M. Maggioni. 2014. ATLAS: A geometric approach to learning high-dimensional stochastic systems near manifolds. *ArXiv e-prints* (April 2014).

Annette J. Dobson and Adrian G. Barnett. 2008. *An Introduction to Generalized Linear Models, Third Edition*. Chapman & Hall/CRC Press, Boca Raton, FL. http://eprints.qut.edu.au/15448/ For more information about this book please refer to the publisher's website (see link) or contact the author.

A. K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data Clustering: A Review. *ACM Comput. Surv.* 31, 3 (Sept. 1999), 264–323. DOI:http://dx.doi.org/10.1145/331499.331504

Brian Kulis and Michael I. Jordan. 2011. Revisiting k-means: New Algorithms via Bayesian Nonparametrics. *CoRR* abs/1111.0352 (2011). http://arxiv.org/abs/1111.0352

Oded Maimon and Lior Rokach. 2005. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Fionn Murtagh and Pedro Contreras. 2011. Methods of Hierarchical Clustering. *CoRR* abs/1105.0121 (2011). http://arxiv.org/abs/1105.0121

J.R. Norris. 1998. *Markov Chains*. Number št. 2008 in Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press. https://books.google.si/books?id=qM65VRmOJZAC

Vijay S. Pande, Kyle Beauchamp, and Gregory R. Bowman. 2010. Everything you wanted to know about Markov State Models but were afraid to ask. *Methods* 52 (2010), 99–105.

R. Sibson. 1973. SLINK: An optimally efficient algorithm for the single-link cluster method. *Comput. J.* 16, 1 (1973), 30–34. DOI:http://dx.doi.org/10.1093/comjnl/16.1.30

Rui Xu and II Wunsch, D. 2005. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on* 16, 3 (May 2005), 645–678. DOI:http://dx.doi.org/10.1109/TNN.2005.845141

Received February 2007; revised March 2009; accepted June 2009

Online Appendix to: A Multi-Level Methodology for Explaining Data Streams

LUKA STOPAR, Jozef Stefan Institute
PRIMOZ SKRABA, Jozef Stefan Institute
DUNJA MLADENIC, Jozef Stefan Institute
MARKO GROBELNIK, Jozef Stefan Institute

A. THIS IS AN EXAMPLE OF APPENDIX SECTION HEAD

Channel-switching time is measured as the time length it takes for motes to successfully switch from one channel to another. This parameter impacts the maximum network throughput, because motes cannot receive or send any packet during this period of time, and it also affects the efficiency of toggle snooping in MMSN, where motes need to sense through channels rapidly.

By repeating experiments 100 times, we get the average channel-switching time of Micaz motes: 24.3 μs. We then conduct the same experiments with different Micaz motes, as well as experiments with the transmitter switching from Channel 11 to other channels. In both scenarios, the channel-switching time does not have obvious changes. (In our experiments, all values are in the range of 23.6 μs to 24.9 μs.)

B. APPENDIX SECTION HEAD

The primary consumer of energy in WSNs is idle listening. The key to reduce idle listening is executing low duty-cycle on nodes. Two primary approaches are considered in controlling duty-cycles in the MAC layer.

© 2010 ACM. 1539-9087/2010/03-ART39 $15.00
DOI: 0000001.0000001