Data Stream-based Intrusion Detection System for Advanced Metering Infrastructure in Smart Grid: A Feasibility Study

Mustafa Amir Faisal, Zeyar Aung, John R. Williams, and Abel Sanchez

***Abstract*—As Advanced Metering Infrastructure (AMI) is responsible for collecting, measuring, and analyzing energy usage data, as well as transmitting these information from a smart meter to a data concentrator and then to a headend system in the utility side, the security of AMI is of great concern in the deployment of Smart Grid (SG). In this paper, we analyze the possibility of using data stream mining for enhancing the security of AMI through an Intrusion Detection System (IDS), which is a second line of defense after the primary security methods of encryption, authentication, authorization, etc. We propose a realistic and reliable IDS architecture for the whole AMI system which consists of individual IDSs for three different levels of AMI's components: smart meter, data concentrator, and AMI headend. We also explore the performances of various existing state-of-the-art data stream mining algorithms on a publicly available IDS dataset, namely, the KDD Cup 1999 dataset. Then, we conduct a feasibility analysis of using these existing data stream mining algorithms, which exhibits varying levels of accuracies, memory requirements, and running times, for the distinct IDSs at AMI's three different components. Our analysis identifies different candidate algorithms for different AMI components' IDSs respectively.**

***Index Terms*—Smart grid, advanced metering infrastructure, intrusion detection system, data stream mining.**

I. Introduction

Smart Grid (SG) is a set of technologies that integrate modern information technologies with present power grid system. Along with many other benefits, two-way communication, updating users about their consuming behavior, controlling home appliances and other smart components remotely, monitoring power grid's stability, etc. are unique features of SG. To facilitate such kinds of novel features, SG needs to incorporate many new devices and services. For communicating, monitoring, and controlling of these devices/services, there may also need many new protocols and standards. However, the combination of all these new devices, services, protocols, and standards makes SG a very complex system that is vulnerable to increased security threats — like any other complex systems are. In particular, because of its bidirectional, inter-operable, and software-oriented natures, SG is very prone to cyber attacks. If proper security measures are not taken, a cyber attack on SG can potentially bring about a huge catastrophic impact on the whole grid and thus to the society. Thus, cybersecurity in SG is treated as one of the vital issues by NIST (National Institute of Standards and Technology) and FERC (Federal Energy Regulatory Commission) [1].

In this paper, we will focus on the security of Advanced Metering Infrastructure (AMI), which is one of the most crucial components of SG. AMI serves as a bridge for providing bidirectional information flow between user domain and utility domain [2]. AMI's main functionalities encompass power measurement facilities, assisting adaptive power pricing and demand side management, providing self-healing ability, and interfaces for other systems, etc. AMI usually constitutes of three major types of components, namely, 1) smart meter, 2) data concentrator, and 3) central system (a.k.a AMI headend), and bidirectional communication networks among those components. Being a complex system in itself, AMI is exposed to various security threats like privacy breach, energy theft, illegal monetary gain, and other malicious activities. As AMI is directly related to revenue earning, customer power consumption and privacy, the utmost important is to secure its infrastructure.

In order to protect AMI from malicious attacks, we look into the Intrusion Detection System (IDS) aspect of security solution. We can define IDS as a monitoring system for detecting any unwanted entity into a targeted system (like AMI in our context). We treat IDS as a second line security measure after the first line of primary AMI security techniques like encryption, authorization, and authentication such as [3]. However, Cleveland [4] stresses that these first line security solutions alone are not sufficient for securing AMI.

IDS can be signature-based, specification-based, and anomaly-based [5]. A signature-based IDS builds a black list of attacks. It is not suitable for AMI because new types of attacks may introduce in AMI as an emerging system. On the other hand, a specification-based IDS can be a potential solution for AMI according to [5] and [6]. Nonetheless, developing a specification for AMI networks is neither easy nor cost effective. As AMI becomes mature gradually, fresh specifications need to be included. Hence, changing specifications in all key IDS sensors would be expensive and cumbersome. In this research, we choose to employ anomaly-based IDS using data mining approaches. However, instead of considering conventional static mining techniques, we elect stream mining, precisely "evolving data stream mining", as this approach is more realistic approach in real-world monitoring and intrusion

Mustafa Amir Faisal is with Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA. Email: Mustafa.Faisal@utdallas.edu. A major portion of this research was done while he was with Masdar Institute of Science and Technology, Abu Dhabi, UAE.

Zeyar Aung is with the Institute Center for Smart and Sustainable Systems (iSmart), Masdar Institute of Science and Technology, Abu Dhabi 54224, UAE. Email: zaung@masdar.ac.ae.

John R. Williams and Abel Sanchez are with Engineering Systems Division, Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA. Emails: {jrw, doval}@mit.edu.

detection for AMI as various novel attacks can be introduced in AMI infrastructure. Rodrigues and Gama [7] mention that SG networks have various distributed sources of high-speed data streams. These stream data can be attributed as open-ended, high-speed, and are produced in non-stationary distributions. Thus, the dynamics of data are unpredictable. As the number of smart meters suppose to grows eventually as well as their roles in AMI evolves over time, the topology of SG networks may change. Thus, data distribution in AMI networks can be changed. Hence, the model should be coped with evolved data.

In this paper, we propose a new AMI IDS architecture based on the AMI architecture presented by OPENMeter [8], which is a project deployed by several European countries to reduce gap between the state-of-the-art technologies and AMI's requirements. We use the data stream mining algorithms available in the Massive Online Analysis (MOA) software [9], [10] in order to simulate the IDS systems of the proposed architecture.

The contributions of this research are that:

- We have proposed a reliable and pragmatic IDS architecture for AMI.
- We have conducted a set of experiments on a public IDS dataset using state-of-the-art data stream mining techniques and observed their performances.
- We have performed a feasibility study of applying these data stream mining algorithms for different components of the proposed IDS architecture.

Note that proposing a new data stream mining algorithm is out of the scope of this research, and is planned as a future work.

This paper is a significantly improved and extended version our previous work [11] in the following ways:

- More stream mining algorithms (7 instead of 4 as in [11]) are explored and analyzed in detail.
- Two different versions of datasets (as opposed to just one dataset in [11]) are used against seven algorithms to understand the feasibility of these algorithms for various components of AMI in changed data distribution.
- Comprehensive experiments of a variety of algorithms and datasets with different distributions will help future researchers to apprehend and extend existing stream mining algorithms for building robust IDSs, used in various components of AMI and other critical systems.

II. Advanced Metering Infrastructure

AMI is an updated version of AMR (Automatic or Automated Meter Reading) [2]. Present traditional AMR helps a utility company in reading meters through one way communication. However, as AMR cannot meet the current requirements for two-way communication and others, AMI is introduced.

AMI comprises of smart meters, data concentrators, central system (AMI headend), and the communication networks among them. These AMI components are usually located in various networks [5] and different realms like public and private ones [12]. Figure 1 gives a pictorial view of AMI integration in a broader context of power generation, distribution, etc. From this figure, we can see that smart meter, responsible for monitoring and recoding power usage of home appliances etc., is the key equipment for consumers. Home appliances and other integrated devices/systems like water and gas meters, IHD (In Home Display), PEV (Plug-in Electric Vehicle) / PHEV (Plug-in Hybrid Electric Vehicle), smart thermostat, rooftop PV (Photovoltaic) system, etc. constitute a HAN (Home Area Network), which is connected to the smart meter. For communicating among these constituents, Zegbee or PLC (Power Line Communication) can be used. A number of individual smart meter communicates to a data concentrator through NAN (Neighborhood Area Network). WiMAX, cellular technologies, etc. are possible means for this network. A number of data concentrator are connected to an AMI headend in the utility side using WAN (Wide Area Network). Various long-distance communication technologies like fiber optic, DSL (Digital Subscriber Line), etc. are used in WAN. The AMI headend located in the utility side consists of MDMS (Meter Data Management System), GIS (Geographic Information System), configuration system, etc. These subsystems may build a LAN (Local Area Network) for inter communication. We characterize the nature of each AMI component and the data it handles in Table I.

TABLE I
Characteristics of smart meter, data concentrator, and AMI headend.

Smart meter	Data Concentrator	AMI Headend
Similarity		
Data is continuous in each of them.		
Differences		
Amount of data in an individual smart meter is small as data sources are customer's HAN and its associated devices (like water and gas meters).	Amount of data is comparatively larger as it has to handle data from about a few hundred to tens of thousands of smart meters [5].	Data amount in cental system is in a huge volume as it has to tackle data from about several millions of smart meters [5].
Resources like main memory (in kilobyte range), processor capacity etc. are very restrictive.	Resources like main memory (in megabyte range [13]), processor capacity etc. are more powerful.	Resources are very powerful because they are usually high-end servers.
Data speed is comparatively low because of non-frequent requests at the smart meter.	Data speed is high as it aggregates a good number of smart meters' data.	Data speed is very high as it has to handle a huge amount of meter data, event data, commands, etc.

III. Related Works

Securing information networks using stream data mining is relatively young. Chu *et al.* [14] use both anomaly and signature-based mechanisms for network IDS with single-pass approach. In their proposed architecture, IDS is divided into couple of modules: passive module for monitoring data stream using signature-based detection and active module for using anomaly-based detection approach. For deploying the architecture, they propose a single pass algorithm named FP-Stream. Oh *et al.* [15] provide a clustering solution for anomaly detection taking account of the number of clusters is unknown where single cluster can be partitioned into a pair of clusters and two of clusters are united into a cluster in agreement with object distribution in data stream. For higher

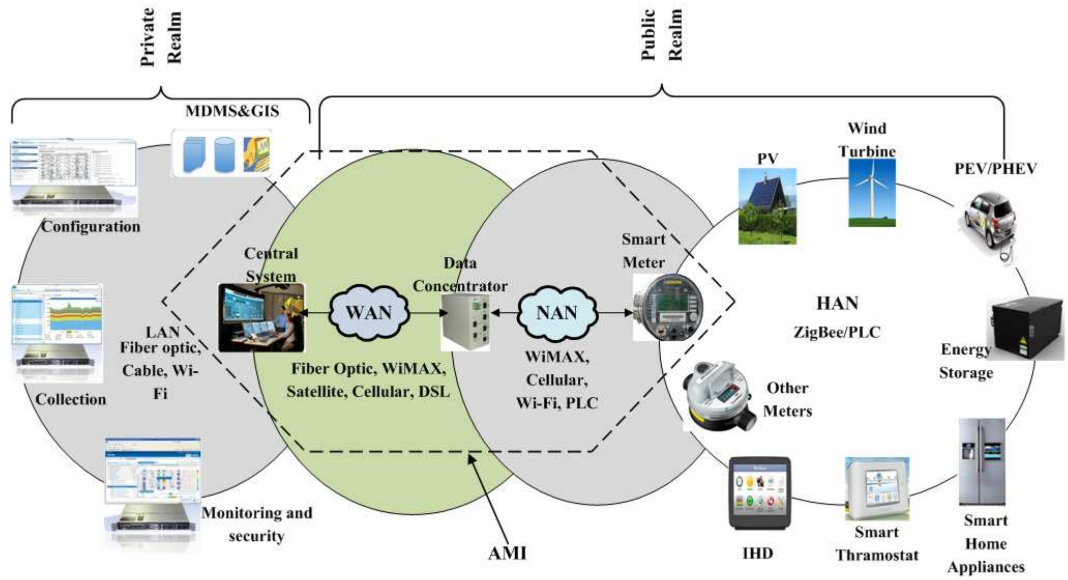

Fig. 1. AMI overview

accuracy and better efficiency, Liu *et al.* [16] show a model based approach on sequence mining for network IDS. In this model, authors use multidimensional item set for outlining network events. In addition, sequence mining algorithms as well as sliding window are considered for gathering network data stream and detecting intrusion respectively. Fuzzy logic is used by Khan [17] for introducing an automated annotation for obtaining results from data stream clustering. The principle feature of this approach is facilitating understanding the characteristics of clusters, anomaly or normal, without human interaction. Recently, Qun and Wen-Jie [18] propose TFSE (Time-table-joined Frequent Serial Episodes) for extracting necessary patterns and rules for intrusion detection from time series stream.

Few researches have been done so far for IDS in AMI. In [19], using a multi-layer network architecture scheme for whole SG including SCADA and AMI, Zhang *et al.* propose a distributed IDS architecture. Three IDSs would be placed in HAN, NAN, and WAN. Two clonal selection algorithms named CLONALG and AIRS2Parallel, which are derived from artificial immune system (AIS) and Support Vector Machine (SVM) are used for detecting anomaly assuming that data can be investigated anytime.

In [5], Berthier *et al.* discuss various types of IDSs and its possible components. An AMI monitoring architecture, proposed by the authors for AMI, uses a distributed system where sensors located in meter network to process anomaly related data. Upper level alerts collection and coordination of sensors' tasks are accomplished by a central system. Resource requirements for this IDS architecture are: network configuration, protocol specifications, system and network security policies, and statistical profiles. The belief of specification-based IDS would be the best approach for AMI by the authors for following reasons: specification-based IDS has better accuracy level over signature-based IDS; lack of empirical data to build a blacklist of signatures; and limited number of protocols and applications will be monitored in AMI and for this specification would be cost-effective as AMI is a controlled environment. To extend their work, Berthier and Sanders in [6] use sensors, characterized with specification-based intrusion detection mechanism, placed in key access points to monitor network, transport, and application layers in the OSI model. Through studying constraints, state machine, and requirements, authors build four rules to detect traffic modification, injection, replay, compromised meters, preventing large set of meters from a malicious node, and DOS (Denial of Service) attacks. Moreover, they emulate AMI environment where these four rules test the performance. At the application-layer, a formal verification of the specifications and monitoring operations are conducted.

IV. Proposed IDS Architecture

Let us look at the first component of AMI, namely the smart meter. Along with houses of ordinary people, smart meters are also installed in crucial places like companies, banks, hospitals, educational institutes, government agencies, parliaments, and presidential residences. Thus, the security of smart meters is a vital issue.

To our best knowledge, current smart meters do not possess IDS facility yet. If we are to furnish smart meters with IDS, one possible approach is to develop an embedded software for IDS, such as the one proposed in [20], and update the firmware of the smart meter to include this embedded IDS. Although this can be done with relatively ease, the main problem is the limitation of computing resources in the current smart meters. They are mostly equipped with low-end processors and limited amounts of main memory (in kilobyte to a few megabyte range) [12], [21]. Although this may change in a near future, since a good number of smart meters has already been deployed in many developed countries, it is not very easy to replace them or upgrade those existing ones with more powerful resources. Since a smart meter is supposed to consume most of its processor and main memory resources for its core businesses (such as recording electricity usage, interaction with other smart home appliances, and two-way communication with its associated data concentrator and ultimately the headend), only a small fraction of its already limited resources is available for IDS' data processing purpose.

We try to solve this problem of resource scarceness by proposing to use a separate IDS entity, either installed outside the smart meter (for existing ones) or integrated within the smart meter (for new ones). We name such an entity a "security box". A possible design of smart meter with this security box is provided in Figure 2(a) based on the one presented in [22]. Here, we show security box as a simple meter IDS. However, it is open for this component to cover other security functions like firewall, encryption, authorization, authentication, etc. Care should be taken that the security box for smart meter should not be too expensive and hence should be equipped with resources just enough to perform computations for IDS (and other security-related calculations, if applicable) at the meter level. The proposed configuration of a meter IDS is provided in Figure 2(b).

It should be noted that our proposed IDS architecture follows a sequential process. Communication data from various sources are inserted to Acceptor Module. Pre-processing Unit is responsible for producing data according to predetermined attributes by monitoring the communication data. This generated data would treat as input for Stream Mining Module. Stream Mining Module runs a data stream mining algorithm over the data generated by Pre-processing Unit. Decision Maker Unit decides whether it should trigger an alarm or not. This module also keeps records of the information associated with attacks. These records will be used for further analysis and improving the attack database.

The proposed IDS architecture for the other two types of AMI's components, data concentrator and AMI headend, is more or less similar to that of smart meter IDS. Again, the security boxes for those components can be either inside (in the form of software or add-on hardware card) or outside (in the form of dedicated box or server). In order to simultaneously monitor a large number of data flows received from a large number of smart meters and detect security threats, such a security box hardware (or the host equipment in the case of software) must be rich in computing resources.

The complete IDS architecture for the whole AMI is shown in Figure 3(a). For inter-communication, IDSs can use separate network as advocated in [2]. Though this dedicated network is expensive, it increases reliability. If IDSs use the same network as other AMI data, in the event of a node being compromised by an attacker, the purpose of the whole IDS system will be defeated.

The flow chart for IDS from smart meter to AMI headend is depicted in Figure 3(b). Likewise, IDS can also work reversely from headend to smart meter. It should be noted that some devices like O&M modules can be connected to IDS locally.

This architecture does not depend on any specific protocols. For this reason, legacy as well as newly coined protocols can be used with it. The IDS of a smart meter will monitor the flow of both incoming and outgoing data. Whenever it encounters any anomaly data, it will trigger an alarm to the IDS at the next level to further investigate the data. This is because the current IDS may face such kind of anomaly for the first time. This IDS should have enough storage to keep information about the anomaly and can send this information to the next IDS.

For all three types of IDSs for three respective AMI components, we believe that it is more practical to regard the data it as a "stream" in the AMI's network. That means, the data received by each component of AMI is sequentially continuous, much larger in size than the device's main memory (however large it is), and most importantly a mining algorithm can trace a particular piece of data for a very limited number of times (in most cases only once). Thus, we propose to explore a number of data stream mining algorithms to detect the anomalous events or attacks for all three types of IDSs.

V. Data Stream Mining Algorithms

We use Massive Online Analysis (MOA) [9], [10] in our experiment. It is an open source data stream mining framework Though there are various static and evolving stream mining classification algorithms available in this software environment, we are only interested in the evolving ones. Evolving classification algorithms care about the concept change or distribution change in the data stream.

All experiments are done in a single PC with Intel Core i7-2600 3.4 GHz processor, 8.00 GB DDR3 main memory, 1.0 TB SATA hard disk drive, and Microsoft Windows 7 Professional 64-bit operating system.

A. Algorithms Explored

There are 16 evolving data stream classifiers in MOA. After an initial trail on those 16 classifiers, the 7 ensemble classifiers listed in Table II are selected. (From now on, we will write classifier names in MOA in *Italic*. In many cases, MOA's classifiers names are self-explanatory.) These seven classifiers are chosen because they offered the highest accuracies (evaluated with *EvaluatePrequential* method in MOA) on the training set.

Different variants of Hoeffding Tree [31] are used as the base learner in MOA. The algorithm establishes the Hoeffding bound, which quantifies the number of observations required to estimate necessary statistics within a prescribed precision. Mathematically, Hoeffding bound can be expressed using Eq. 1, which says that the true mean of a random variable of range R will not differ from estimated mean after n independent examples or observations by more than ϵ with probability $1 - \delta$.

$$\epsilon = \sqrt{(R^2 \ln(1/\delta))/2n} \tag{1}$$

The descriptions of these seven selected ensemble learners are as follows. (For more detailed descriptions, refer to in the original papers [23], [24], [25], [26], [27], [28], [29], [30].)

(1) *AccuracyUpdatedEnsemble* (Accuracy Updated Ensemble - AUE): It is a block-based ensemble classifier, an improved version of AWE (Accuracy Weighted Ensemble), for evolving data stream. AUE makes this enhancement by using online component classifiers which updates the base classifiers rather than only adjust their weights as well as updating them according to present distribution. In addition, this method also

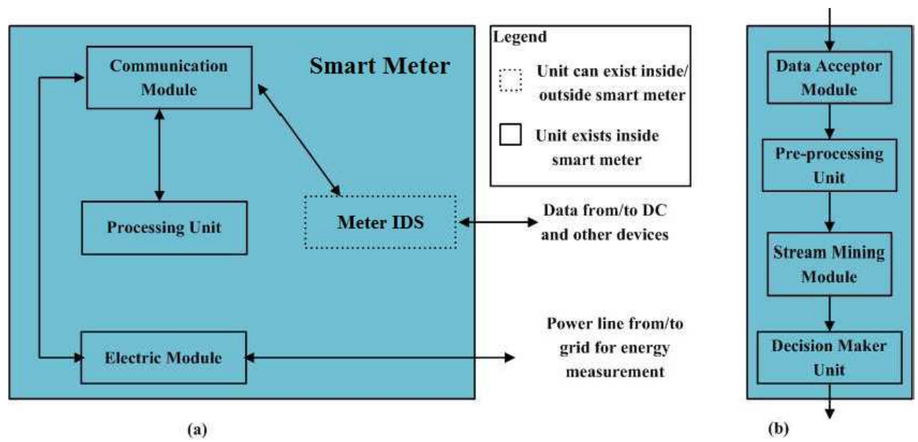

Fig. 2. (a) Smart meter with a security box (meter IDS). (b) Configuration of a meter IDS.

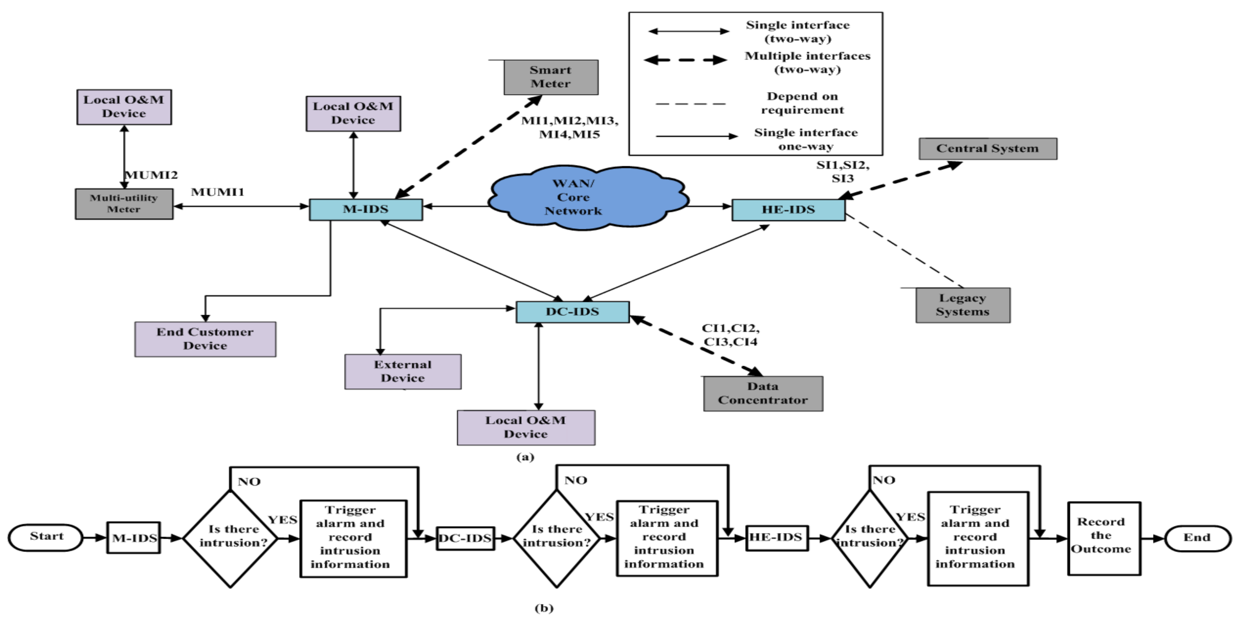

Fig. 3. (a) Architecture of whole IDS in AMI (b) Intrusion detection procedure from smart meter to AMI headend. (CI = Concentrator Interface; DC-IDS = Data Concentrator IDS; HE-IDS = Headend IDS; M-IDS = Meter IDS; MI = Meter Interface; MUMI = Multi-utility Meter Interface; O&M = Operations and Maintenance; SI = (Cental System) Interface.)

TABLE II
SEVEN DATA STREAM CLASSIFIER ALGORITHMS EXPLORED IN OUR EXPERIMENT.

Sr.	Method name in MOA	Description	Base Learner	Reference
1	*AccuracyUpdatedEnsemble*	Accuracy Updated Ensemble (AUE)	*HoeffdingTreeNBAdaptive*[1]	[23]
2	*ActiveClassifier*	Active Classifier	*HoeffdingTreeNB*[2]	[24]
3	*LeveragingBag*	Leveraging Bagging	*HoeffdingTreeNB*[2]	[25]
4	*LimAttClassifier*	Limited Attribute Classifier	*LimAttHoeffdingTreeNBAdaptive*[3]	[26]
5	*OzaBagAdwin*	Bagging using ADWIN	*HoeffdingTreeNB*[2]	[27], [28]
6	*OzaBagASHT*	Bagging using Adaptive Size Hoeffding Tree	*ASHoeffdingTree*[4]	[28]
7	*SingleClassifierDrift*	Single Classifier Drift	*HoeffdingTreeNBAdaptive*[1]	[29], [30]

[1]Hoeffding Tree with majority class and naive Bayes learners at the leaves. [2]Hoeffding Tree with naive Bayes learners at the leaves. [3]Hoeffding decision trees with a limited number of attributes. [4]Adaptive Size Hoeffding Tree used in Bagging using trees of different size.

leverages the drawback of AWE by redefining the weighting function. AUE determines the weight for the corresponding component classifier C_i by measuring the error rate on the most recent data block D_i which is provided in the following equations.

$$MSE_i = \frac{1}{D_i} \sum_{(x,c)\in D_i} (1 - f_c^i(x))^2 \tag{2}$$

$$w_i = 1/(MSE_i + \epsilon) \tag{3}$$

where $f_c^i(x)$ is the probability given by C_i that c is class for x in Eq. 2. In Eq. 3, here ϵ is very tiny constant value to tackle situations when $MSE_i = 0$, which are rare.

(2) *ActiveClassifier* (Active Classifier): The motivation of active learning algorithm is to build a model from as few amount of labeled data as possible [24]. The principal reason behind this is labeling of data is expensive so we may not except that all training data are labeled. This is an important assumption in an error prone network like various wireless networks. Various active learning strategies are used for maximizing prediction correctness under assumption that there exists concept drift in data stream.

To describe the setup for this classifier, let consider X_t is an instance at time t with label y_t. Thus, data stream is $X_1, X_2, X_3, ..., X_{t-1}, X_t, X_{t+1}, ...$ The budget for the stream is B which indicates that the portion of incoming data are labeled. For example, if $B = 0.4$, then we expect 40% of the data stream has been labeled. An *ActiveLearningStrategy*$(X_t, B, ...)$ method is called to decide for current instance whether the framework should call for true class label or not. This method can use additional parameters depending on the underline implemented strategy like random strategy, fixed uncertainty strategy, variable uncertainty strategy, uncertainty strategy with randomization etc. For change detection, Drift Detection Method (DDM) [29] is used.

(3) *LeveragingBag* (Leveraging Bagging): A set of variants for leveraging bagging has been proposed in [25]. Mainly two randomization improvement techniques have been done to enhance bagging [32] performance. They are 1) increasing re-sampling, and 2) using output detection codes.

Though re-sampling is done in online Bagging using Poisson Process 1, Bifet *et al.* [25] propose to use higher value of λ to compute the Poission distribution's value which would increase this re-sampling weights. In second enhancement, randomization is added at the output of the ensemble using error-correcting output codes. A binary string of length n is assigned to each class and thus, an ensemble of n binary classifiers is built. Each classifier learns one bit for each position in this binary string. When a new instance x arrives, it is assigned to the class with nearest binary code. An error-correcting code can be viewed as a form of voting in which a number of incorrect votes can be corrected. The main motivation for using random code instead of deterministic code is each classifier in ensemble will predict a different function which may reduce the correlation effects among the classifiers. Subsequently, the diversity of the ensemble will be increased. Each classifier m and class c are assigned binary value $\mu_m(c)$ in a uniform, independent, and random way. Exact half of the classes are mapped to 0. The output of the classifier for an example is the class which has more votes of its binary mapping classes. To deal with concept drift in data stream, ADWIN [27] has been used.

(4) *LimAttClassifier* (Limited Attributes Classifier): This ensemble classifier combines restricted Hoeffding Trees using stacking. A classification model based on an ensemble of restricted decision trees are generated. Each decision tree is built from a unique subset of attributes. The whole model is formed by mixing the log-odds of the predicted class probabilities of these trees using sigmoid perceptrons, with single perceptron for individual class. ADWIN is used for setting perceptrons' learning rate as well as for resetting Hoeffding trees when they no longer perform well. Instead of forming an ensemble classifier in a greedy fashion like in standard boosting approach, Limited Attribute Classifier builds each Hoeffding tree in sequence and assigns related weights as a by-product. Thus, each tree generated in parallel and then these trees are combined using perceptron classifiers by adopting the stacking approach. Adaptive naive Bayes Hoeffding Trees with limited attributes show better performance instead of using individually naive Bayes or majority class for prediction.

For combining Hoeffding Trees' prediction, the log-odds of raw class probabilities estimates using following equation:

$$b_{ij} = (\log p(c_{ij}|\vec{x})) \,/\, (1 - p(c_{ij}|\vec{x})) \tag{4}$$

where $p(c_{ij}|\vec{x})$ is the probability estimate for class i and instance $\vec{x}$, received from j Hoeffding Tree in the ensemble. The output of the sigmoid perceptron for class i with $\vec{a_i}$, log-odds vector, as well as $\vec{w_i}$ and b_i coefficients.

$$f(\vec{a_i}) = 1/(1 + e^{\vec{w_i}\vec{a_i} + b_i}) \tag{5}$$

The weight vector of each perceptron is updated while a new training instance is obtained from the data stream using stochastic gradient descent. The learning rate of this stochastic gradient descent is determined by the equation:

$$\alpha = 2/(2 + m + n) \tag{6}$$

where m is the number of attributes and n is total number of instances. However, whenever ADWIN detects a change in data stream, n is set to 0. Thus, α is re-initialized.

The main computational complexity depends on generating tress for all possible attributes subsets of size k. For p number of attributes, there exists $(p!)/(k!(p-k)!) = \binom{p}{k}$ number of subsets. If number of attributes is big, then it is better to select small value for k, usually $k = 1$. Large value for k is feasible only for a small number of attributes.

(5) *OzaBagAdwin* (Oza Bagging with ADWIN): The main idea of this algorithm is to use a sliding window, not fixed a priori, whose size is recomputed in online according to the change rate observed from the data in window itself. The window will grow or shrink keeping pace with change in data stream. For this reason, OzaBagAdwin uses ADWIN2, an enhanced version of ADWIN in term of time and memory efficiency. This change detection technique holds a window of length W with $O(\log W)$ memory and update time. The classifier provides a performance guarantee by bounding the rates of false positives and false negatives.

(6) *OzaBagASHT* (Oza Bagging with Adaptive Size Hoffding Tree): It uses adaptive size (variable size) Hoeffding Tree (ASHT) with bagging. ASHT is derived from Hoeffding Tree with couple of differences: 1) ASHT has a maximum number of size or split nodes, and 2) the tree has a size reduction mechanism for exceeding maximum value and which is deletion of several nodes.

The principal intuition behind this is: smaller trees adapt more quickly to changes and on the other hand, larger trees do better during periods with no or little change as they have been built on more data. Thus this method increases diverse trees. Trees restricted to size s will be reset about twice as often as trees with a size of limit of $2s$. Due to the overflow of tree size, the algorithm takes two remedies: first one is to delete the oldest node, the root, and all of its children except where the split has been made and second one is to restart from a new root. The maximum legitimated size for the n-th ASHT tree is twice the maximum allowed size for the $(n-1)$-th tree. And each tree has a weight proportional to the inverse of the square of its error, and it monitors its error with the exponential weighted moving average (EWMA) with α = 0.01. The initial tree size is 1, though the size can be set by user.

(7) *SingleClassifierDrift* (Single Classifier Drift): It is an evolving classifier with a wrapper on it for handling concept drift in data stream. For this, drift detection method (DDM) [29] or early drift detection method ($EDDM$) [30].

(7.1) *DDM* (Drift Detection Method): In [29], Gama *et al.* propose this approach where the number of errors produced by the learning model during prediction is controlled. Two windows are compared. Fist window contains all the data and second window contains only the data from the beginning until the number of errors increases. Instead of storing these windows in memory, this method only holds required statistics and a window of recent errors. In addition, this method considers that the number of errors in a sample of examples is modeled by a binomial distribution. A considerable increase in the error of the algorithm, suggests that the class distribution is changing and thus, the original decision model is supposed to be inappropriate. They check for a warning level and a drift level. Beyond these levels, change of context is considered.

The number of errors in a sample of n examples is modeled by a binomial distribution. For each point i for being sampled in the sequence, the error rate is the probability of misclassifying (p_i), with the standard deviation of $s_i = \sqrt{(p_i(1-p_i))/i}$. A significant increase in the error of the algorithm suggests that there is a change in class distribution, and hence the actual model is supposed to be inappropriate. Thus, they store the values of p_i and s_i when $p_i + s_i$ reaches its minimum value during the process (obtaining p_{min} and s_{min}). For this, DDM checks when the following conditions trigger:

1) $p_i + s_i \geq p_{min} + 2.s_{min}$ for the warning level. Beyond this level, the examples are stored in application of a possible change of context.
2) $p_i + s_i \geq p_{min} + 3.s_{min}$ for the drift level. Beyond this level, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered.

(7.2) *EDDM* (Early Drift Detection Method): Manuel *et al.* [30] have proposed this method to enhance the detection in presence of gradual concept drift maintaining good performance with abrupt concept drift. The fundamental idea is to consider the distance between two errors classification instead of considering only the number of errors. Increasing the average distance between two errors and improving the prediction are the improvements of this method. The average distance between two errors ($p_i^{'}$) and its standard deviation ($s_i^{'}$). The values of $p_i^{'}$ and $s_i^{'}$ are stored when $p_i^{'} + 2.s_i^{'}$ reaches its maximum value ($p_{max}^{'}$ and $s_{max}^{'}$).

EDDM also checks when the following conditions trigger:

1) $(p_i^{'} + 2.s_i^{'})/(p_{max}^{'} + 2.s_{max}^{'}) < \alpha$ for the warning level. Beyond this level, the examples are stored in application of a possible change of context.
2) $(p_i^{'} + 2.s_i^{'})/(p_{max}^{'} + 2.s_{max}^{'}) < \beta$ for the drift level. Beyond this level, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. In addition, $p_{max}^{'}$ and $s_{max}^{'}$ are also reset.

After some experiment, the default value of α and β are set to 0.95 and 0.90 respectively. If the similarity between the original value of $p_i^{'} + 2.s_i^{'}$ and the maximum value of $p_{max}^{'} + 2.s_{max}^{'}$ increase over the warning threshold, the stored examples are removed and the method returns to normality.

B. Tuning Parameter Values

There are a number of parameters to be tuned each of the above seven algorithms. In order to tune the parameter values, we perform the following procedure for each algorithm.

1) We first identify relevant parameters for which algorithm performance can be influenced. The identification process has been done with the help of corresponding literatures.
2) For each parameter a range of value has been selected. A batch has been created with this range of values for that individual parameter. The procedure helps us to measure the influence of corresponding parameter of the algorithm for this dataset.
3) Then we perform trial runs with the ranges of values for all required parameter for every interested evolving algorithms. This procedure helps us to understand the combined effect of these parameters. For a given dataset, we conduct 10 trial runs before we record the results in order to reduce stochastic effects of tuned parameter values.
4) After all these trial runs, we select parameter values for which the highest accuracies are received.

VI. Experimental Results

A. Datasets

Since AMI can be treated as a combination of computer network and the power system with some additional new characteristics, some communication scenarios of a computer network can resemble those of an AMI network. For example, some attacks like blocking data and theft of information are common in both networks. Since no real AMI transaction dataset is publicly available to our best knowledge, the widely-used public KDD Cup 1999 dataset [33] for intrusion detection in computer networks is selected to be used in our experiment. Although this dataset is based on the communication scenarios in computer networks, the majority of features of the record samples such as the service duration time, data size to/from source/destination, various error rates, etc. are also applied to the SG communication scenarios. The same data has also been used in the studies for the distributed IDS for SG by Zhang *et al.* [19].

TABLE III
STATISTICS FOR KDD CUP 1999 "FULL" AND "IMPROVED" DATASETS.

Type of transaction	Number of network transaction instances			
	Full		Improved	
	Training	Testing	Training	Testing
Normal	972,781	60,593	67,343	9,711
R2L	1,073	11,980	942	1,656
U2R	105	4,437	105	1,298
DOS	3,883,370	229,853	45,927	7,458
Probing	41,102	4,166	11,656	2,421
Total	4,898,431	311,029	125,973	22,544

The KDD Cup 1999 dataset can be regarded as an evolving stream (a.k.a concept drift) dataset in which the sequence of events of normal network transactions and attacks are continuously happening and also changing over time [34].

This dataset is a version of the generated dataset through the 1998 Intrusion Detection Evaluation Program initiated by DARPA (Defense Advanced Research Projects Agency, USA) and managed by MIT Lincoln Labs [35]. From seven weeks of network traffic, raw training data, about four gigabytes of compressed binary TCP dump data, was processed into about five million connection records. At the same way, for two weeks, test data was extracted from two million connection records. The dataset has 41 features as well as training and testing sets have 24 and 38 distinct type of attacks respectively. However, these attacks can be categorized into 5 broad types which are: 1) normal 2) denial-of-service (DOS) 3) unauthorized access from a remote machine (R2L) 4) unauthorized access to local super user (root) privileges (U2R), and finally, 5) surveillance and other probing.

The original data set released for the KDD Cup 1999 contains 5,209,460 network transactions. We name this as the **Full Dataset**. However, this full version of dataset contains many redundant records as well as repeating items of DOS. Also, it has biased distribution of various attack types which makes accurate classifications of U2R and R2L difficult. Thus, Tavallaee *et al.* [36] proposed an improved version of the KDD Cup 1999 dataset to serve as a better benchmark dataset. It contains a total of 148,517 network transactions. We call this data the **Improved Dataset**. The number of transactions in both Full and Improved Datasets are given in Table III.

B. Performance Metrics

The performance of each of the 7 algorithms are measured in terms of the following 7 metrics.

1) **Accuracy (%):** the percentage of correctly classified network transaction instances divided by total number of instances.
2) **Kappa Statistics (%):** the measurement of inter-rater agreement for nominal or categorical items (attack categories in our case) [37].
3) **Model size (KB):** the size of classifier model in kilobytes.
4) **Running time (seconds):** the wall-clock time taken by the model.
5) **Model cost (RAM-Hours):** the amount of RAM (in gigabytes) allocated to the classifier model multiplied by time (in number of hours) the model has been deployed. This results in a combined metric of the model size and the running time. This pricing method is used by GoGrid cloud hosting service [38].
6) **False positive rate (FPR) (%):** the percentage of instances which are incorrectly classified as any kind of attacks (while they are actually normal).
7) **False negative rate (FNR) (%):** the percentage of instances which are incorrectly classified as normal (while they are actually attacks).

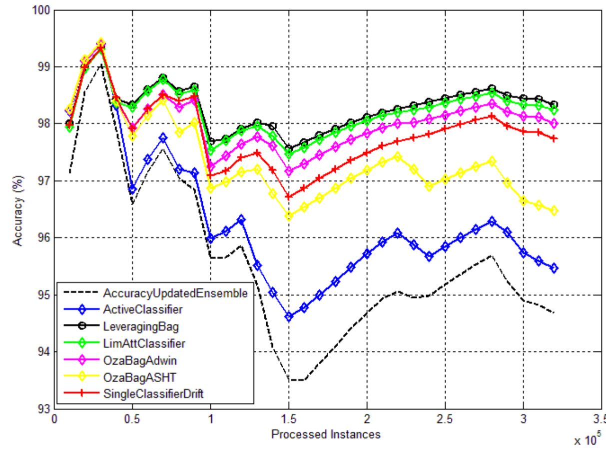

Fig. 4. Accuracies by the seven classifiers (on the Full Dataset).

C. Results for Full Dataset

The performances of the seven algorithms on the test set of the Full Dataset are given in Table IV. From this Table, we can see in the test set, the best performer in term of accuracy is *LeveragingBag* whereas *AccuracyUpdatedEnsemble* is a poorest performer. Lowest RAM-Hours and time requirements are shown by *ActiveClassifier*. However, this classifier shows poor outcomes for accuracy, kappa statistic, FPR, as well as for FNR. *LeveragingBag* also receives better result for FPR, and FNR. The highest model cost is incurred by *OzaBagASHT*. The performance of *LimAttClassifier* is impressive except that it is most costly in term of time requirement among the seven classifiers. *SingleClassifierDrift* also shows good results for time, memory consumption, FPR, etc. in spite of its bad performance for FNR. Both *OzaBagASHT* and *AccuracyUpdatedEnsemble* do not show attractive performance, though both of them receive better FPR.

The accuracy comparison among the classifiers during the evaluation is depicted in Figure 4.

Until around first $30,000$ network transaction instances, all algorithms except *AccuracyUpdatedEnsemble* show about the same accuracy. After around $30,000$ instances each classifier shows decreasing accuracy and reach its individual lowest accuracy at around 1.5×10^5 instances. After 1.5×10^5 instances, all classifiers show increasing accuracy. However, for the ending instances each classifier shows decreasing accuracy. This observed trend is because of the evolving (concept drift) phenomenon that the dataset exhibits.

To get a better understanding about the detection ability of concept drift of these selected classifiers, we take a snapshot

TABLE IV
PERFORMANCES OF THE SEVEN CLASSIFIERS FOR THE TEST SET OF THE FULL DATASET.

Classifier	Accuracy (%)	Kappa Statistics (%)	Model Size (KB)	Running Time (secs.)	Model Cost (RAM-Hours)	FPR (%)	FNR (%)
AccuracyUpdatedEnsemble	92.72	83.11	705.66	34.27	6.48E-6	1.85	22.77
ActiveClassifier	94.67	87.28	134.55	3.46	1.23E-7	3.31	9.13
LeveragingBag	98.33	95.97	401.01	20.92	2.22E-6	0.78	5.15
LimAttClassifier	98.24	95.74	380.07	48.06	4.84E-6	0.78	5.26
OzaBagAdwin	98.00	95.18	295.88	37.77	2.96E-6	1.14	5.31
OzaBagASHT	96.48	91.57	5269.16	41.25	5.78E-5	1.86	7.86
SingleClassifierDrift	97.74	94.55	187.30	6.74	3.34E-7	1.07	6.79

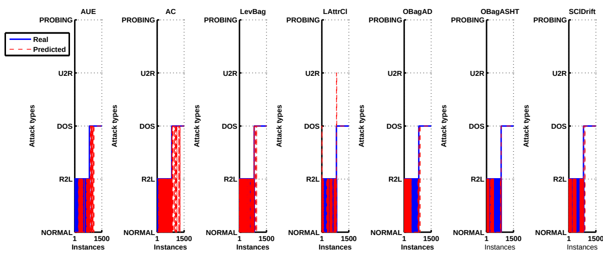

Fig. 5. Detecting of concept drift by the seven classifiers (on the Full Dataset).

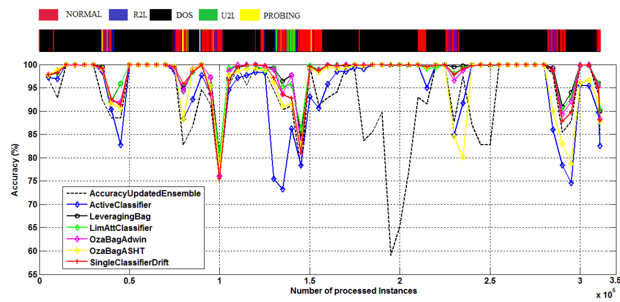

Fig. 6. Performances of the seven classifiers in dealing with both concept drift and static data (on the Full Dataset).

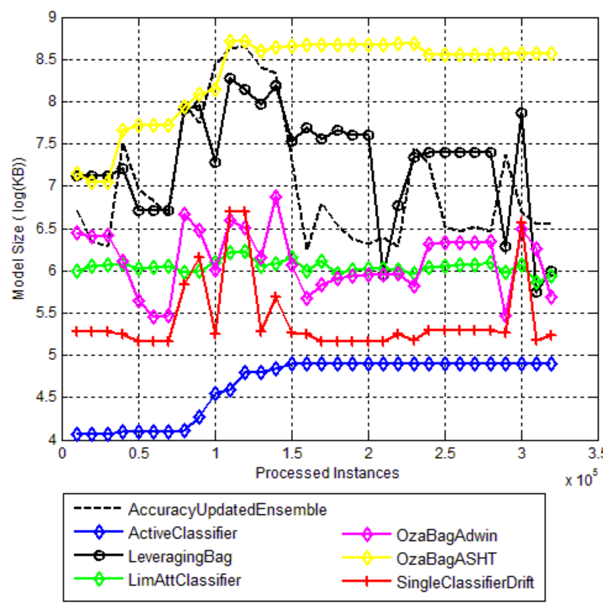

Fig. 7. Model sizes by the seven classifiers (on the Full Dataset).

for first $1,500$ instances with a single concept drift (see Figure 5). From this figure, we see only *OzaBagASHT* can detect the concept drift perfectly, although it shows poor performance for detecting static instances. Very bad performance is shown for detecting concept drift by *ActiveClassifier* and *AccuracyUpdatedEnsemble*. Though *ActiveClassifier* can detect concept drift, it performs a considerable large number misclassifications after the concept drift. Abrupt transition from R2L to DOS is almost smoothly identified by *LimAttClassifier*, *OzaBagAdwin*, and *SingleClassifierDrift*. *LeveragingBag* shows better performance for detecting both usual instances and concept drift except small number of misclassifications after concept drift.

To predict the performance of these seven evolving classifiers both in static and evolving data, we use *WindowClassificationPerformanceEvaluator* evaluator with window size $5,000$ for *EvaluateInterleavedTestThenTrain* task on full version test data. In Figure 6, we see the result. Except AUE, ActiveClassifier, and OzaBagASHT the rest four classifiers show nearly same accuracy level. However, AUE is the worst performer. Noticeable thing, except AUE, all classifiers do better (about 100% accuracy level) while data is static. On the other hand, for evolving data the accuracy deviates from 100% for all classifiers. For example, near 1×10^5 instances, all classifiers show bad accuracy (between 75% and 80%).

Model sizes for the seven classifiers are pictorially shown in Figure 7. For getting a better visualization of memory consumption nature among the classifiers, instead of using KB, we take $\log(KB)$ for the model sizes. *ActiveClassifier* presents very low memory usage and so does *SingleClassifierDrift* except some fluctuations. Both *LimAttClassifier* and *OzaBagAdwin* show their moderate memory requirement. On the other hand, *OzaBagASHT* shows its highest memory requirement followed by *AccuracyUpdatedEnsemble* and *LeveragingBag*.

Almost the same nature as for memory requirement is shown by the seven classifiers for their running times (see Figure 8). Smaller time requirement is the characteristic for both *ActiveClassifier* and *SingleClassifierDrift* classifiers. Moderate time requirement can be found for four classifiers: *LeveragingBag*, *OzaBagAdwin*, *OzaBagASHT*, and *LimAttClassifier* respectively. The model cost (in RAM-hours) for each algorithm shown in Figure 9 shows the combined effect of the two factors: the model size and the running time.

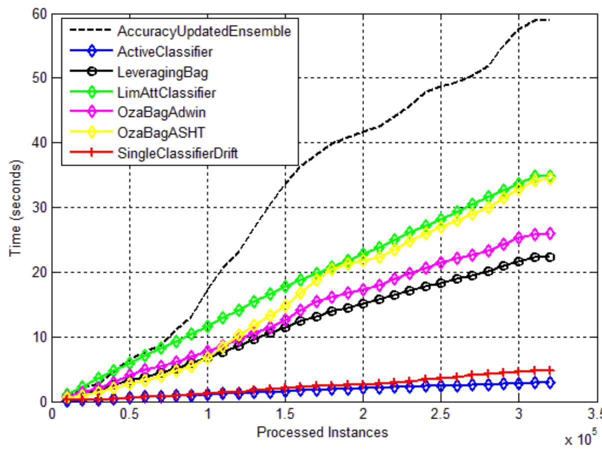

Fig. 8. Running times by the seven classifiers (on the Full Dataset).

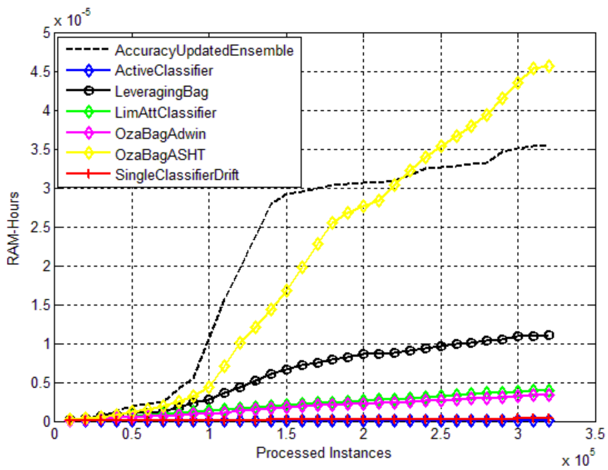

Fig. 9. Model costs by the seven classifiers (on the Full Dataset).

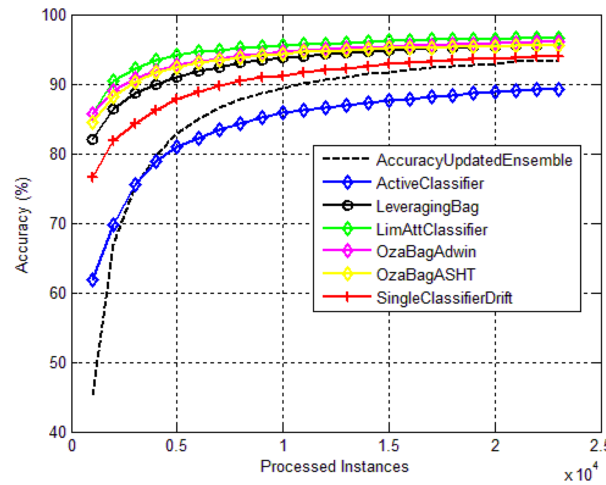

Fig. 10. Accuracies by the seven classifiers (on the Improved Dataset).

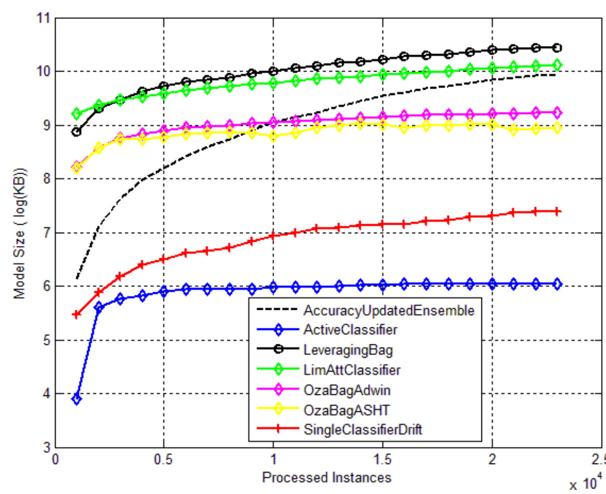

Fig. 11. Model sizes by the seven classifiers (on the Improved Dataset).

D. Results for Improved Dataset

In case of the Improved Dataset, which is very small in size, the performances of the seven algorithms on the test set are given in Table V.

A graphical demonstration of accuracy comparison of the classifiers for Improved Dataset is presented in Figure 10. *AccuracyUpdatedEnsemble* represents a quite bad performance at the beginning even though it shows a considerably better performance gradually. Same trend is shown by *ActiveClassifier*. *SingleClassifierDrift* shows a moderate performance where the rest four classifiers represent relatively better performances. However, *LimAttClassifier* shows the best performance among these four. A mentionable observation is all classifiers show increasing trend with increasing number of instances.

However, for the memory and the time requirement, *ActiveClassifier* shows an impressive result. Though it receives the worst performances for accuracy, kappa statistic, and FNR, it shows the best performance for memory and time usages. On the other hand, *LimAttClassifier* requires the highest amount of memory and time in spite of its better performances for accuracy, kappa statistic, FPR, as well as FNR. *LeveragingBag* shows a moderate performance with lowest FPR and considerable model cost. Reasonable performance is shown by both *OzaBagAdwin* and *OzaBagASHT* classifiers. One noticeable concern is that for all classifiers the accuracy and the kappa statistic deceases when compared to those of the Full Dataset due to reduction in the number of instances in this Improved Dataset.

For the model cost (see in Figure 13), the model size (Figure 11), and the running time (Figure 12) all classifiers keep the same order as we see in Table V. One important observation is that all classifiers consumed more memory than those for the Full Dataset. That means, for the Full Dataset, the model cost is in 10^{-5} range where in this case it is in 10^{-4} range which is 10 times higher. Almost the same trend is observed in case of model size.

VII. Discussions

From the experiments, we can observe that *ActiveClassifier* provides admirable results in term of memory and time in both versions of datasets. However, it suffers from poor accuracy and high FPR. One mentionable outcome for *LeveragingBag*, *LimAttClassifier*, and *OzaBagAdwin* is that provide consider-

TABLE V
PERFORMANCES OF THE SEVEN CLASSIFIERS FOR THE TEST SET OF THE IMPROVED DATASET.

Classifier	Accuracy (%)	Kappa Statistics (%)	Model Size (KB)	Running Time (secs.)	Model Cost (RAM-Hours)	FPR (%)	FNR (%)
AccuracyUpdatedEnsemble	93.39	90.23	20632.44	3.96	2.16E-5	3.26	8.70
ActiveClassifier	89.26	84.19	420.43	0.28	3.20E-8	4.04	15.29
LeveragingBag	95.65	93.60	34375.00	1.62	1.85E-5	2.03	6.85
LimAttClassifier	96.59	95.02	24596.75	92.47	6.03E-4	2.49	3.39
OzaBagAdwin	96.05	94.21	10212.59	3.35	9.07E-6	2.15	5.55
OzaBagASHT	95.60	93.58	7724.45	3.16	6.48E-6	3.17	3.92
SingleClassifierDrift	93.97	91.09	1611.43	0.30	1.26E-7	2.49	8.23

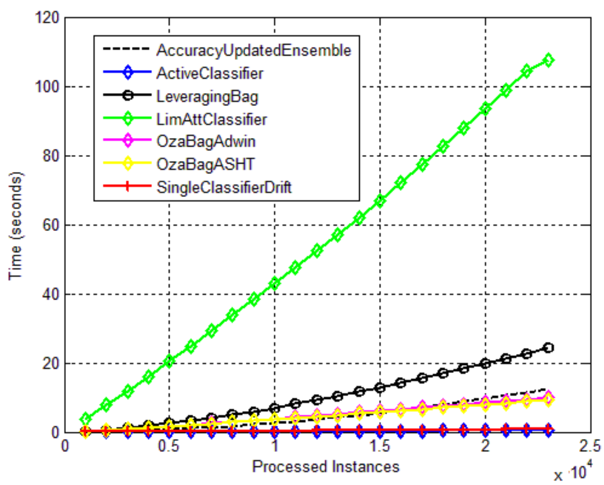

Fig. 12. Running times by the seven classifiers (on the Improved Dataset).

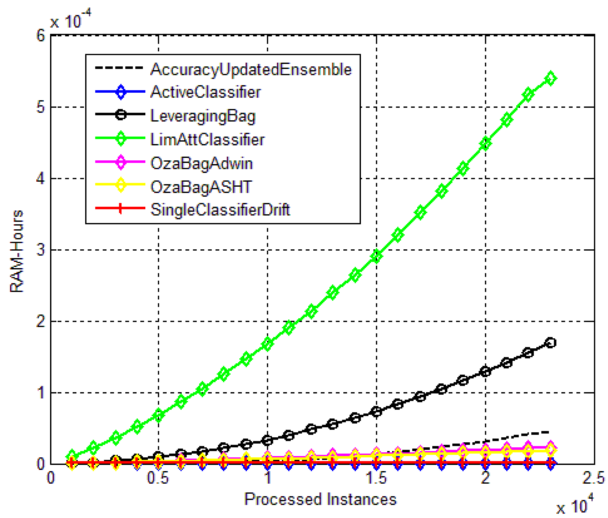

Fig. 13. Model costs by the seven classifiers (on the Improved Dataset).

ably high accuracy as well as lower FPR and FNR. However, they require higher memory amounts for their model sizes for the Improved Dataset, where the distribution of data changes quickly. These three classifiers require more memory when the change of data distribution is high. In the Full Dataset, the data distribution change is infrequent because of large quantity of DOS attacks, and less memory is required by these three classifiers.

A. Feasibility of Using Data Stream Classifiers for AMI's IDSs

Now, let us look into the feasibility of applying the some of the aforementioned seven data stream classifiers for AMI's three different types of IDSs.

For the smart meter IDS, the data stream algorithm should cope with the main memory primarily because a smart meter will not be accessed as frequently as like the rest of two components in AMI. Processed number of instances should be in reasonable range which will be determined by the amount of available main memory. However, additionally, the smart meter has to deal with small amount of data generated from smart devices in HAN as well as some operation and maintenance services. Because of the relaxed time requirement, data can be visited for more than one pass. Nonetheless, for this purpose, the complete set of instances must be kept in the memory and that is not feasible for the smart meter. Considering these constraints, we can see that only *ActiveClassifier* and *SingleClassifierDrift* classifiers that exhibits limited memory usage are the possible candidates. However, though *ActiveClassifier* shows very attractive performance in memory and time, it shows quite poor performances for other metrics. On the other hand, *SingleClassifierDrift* perform very well in memory and time consumption as well as moderate performances for other metrics. Other algorithms are not restrictive memory friendly even if the number of processed instances is reduced.

For the data concentrator IDS, the data flow is more frequent than a smart meter and less than a headend. Regardless of whether it is implemented as a dedicated hardware or software installed on the data concentrator, a moderate amount of memory will be available for the data concentrator IDS. So, although the speed requirement for an IDS algorithm for the data concentrator is much higher than that of the smart meter's, the memory restrictions can be relaxed. *LeveragingBag* is a promising classifier for this IDS as it has both moderate time and memory requirements with reasonable accuracy and FPR. However, still it has higher FNR which needs to be further reduced. *SingleClassifierDrift* should be enhanced in terms accuracy, FPR, and FNR to make it qualified for a data concentrator IDS.

For the IDS in AMI Headend, the algorithm must exhibit a response time of a fraction of second to a few seconds. Data speed is very high, and so the IDS has to process a huge number of instances within this limited time. The memory is naturally abundant. So far, from our analysis, we are not able to find a perfect classifier which will make a tradeoff between time and memory with acceptable accuracy, FPR, and FNR. One possible, albeit not perfect, candidate is

ActiveClassifier. It can process the data very quickly, but is not able to provide luxurious levels of accuracy, FPR, and FNR. The same observation applies for *SingleClassifierDrift*. Other algorithms, especially *LimAttClassifier*, cannot be deployed in this system due to their higher time requirement.

The algorithms studied here are ensemble classifiers depend on a collection of base classifiers. So, improvements can be done both in the base and the ensemble classifiers. For our experiment, we use default value 10 for ensemble size. Reduction in ensemble size may not bring better accuracy and in contrary, increasing the size will be costly (both in time and memory).

ActiveClassifier has very promising features like limited time and memory requirement and these can be the reasons for its relatively inferior performance in other metrics. In addition, this classifier assumes that not all data are labeled, which is a realistic assumption in an attack prone network where various kinds of new attacks can be introduced. Accuracy of *LimAttClassifier* can be further improve using higher number of attributes but this will be more expensive in terms of time and memory.

B. Comparison with Existing Works

Our work is related to Berthier *et al.* [5], [6] and Zhang *et al.* [19]. Among these two, our work is closely related with Zhang *et al.* as they also use anomaly based IDS. However, they propose IDS architecture for complete SG where we concentrate on AMI security with IDS. Moreover, we emphasize that it is more practical to apply evolving data stream mining techniques rather than static data mining techniques in AMI where memory requirements and time costs are the very concerned issues. Moreover, our proposed architecture is less expensive for deploying for existing components of AMI. However, Zhang *et al.* does not provide such critical solution. In addition, there is no details comparison of used classifiers in terms of accuracy, time, and memory which are crucial metrics in AMI. This work even does not consider different characteristics of various AMI components.

Berthier *et al.* emphasize on specification based IDS for AMI due to AMI's controlled network and lack of training data for anomaly based IDS. They propose to use sensor based architectural scheme. On the other hand, we have proposed anomaly based IDS using data stream mining in network layer in OSI model as we believe, building and updating specification for AMI will be expensive eventually. Sometime, updating the existing software with patch may crash the sensor node. For this we focus on security of individual meter and component in AMI. In stead of placing IDS sensors in key locations, we precisely have mentioned deployment places of IDSs in AMI. Optimizing sensor location will also become a significant issue in designing IDS infrastructure. In addition, if sensor node is compromised, how the system will response which is also most concerned issue in this architecture. This kind of centralize architecture bears various risks like system crash, software or hardware malfunction, cyber and/or physical attacks, etc.

Nonetheless, we are not able to directly compare our experimental results with those of the two methods mentioned above. This is because we have a different (and more realistic) assumption that the data is in form of stream rather than static, and thus the dataset in its entirety is never available to the mining algorithm at any given point of time.

In our previous work [11], we are more concerned about the architecture of IDS in AMI. As an extension of the earlier work, in this paper, we focus on experimental analysis in details. Here we use more classifiers to provide insight of existing ones to deploy for IDS. This gives us better understanding to improve existing ones or coin new ones for similar problems.

VIII. Conclusion and Future Works

In this research, we propose an architecture for the comprehensive IDS system in AMI which is designed to be reliable, dynamic, and considering the real-time nature of traffic for each component in AMI. Then, we conduct a performance analysis experiment of the seven existing state-of-the-art data stream mining algorithms on a public IDS dataset. Finally, we elucidate the strengths and weaknesses of those algorithms, and assess the suitability of each of them to serve as the IDSs for the three different components of AMI. We have observed that some algorithms that uses very minimal amount of computing resources and offers moderate level of accuracy can potentially be used for the smart meter IDS. On the other hand, the algorithms that requires more computing resources and also offers higher accuracy levels can be useful for the IDSs in data concentrators and AMI headends. As future works, we plan to develop our own lightweight yet accurate data stream mining algorithms to be used for the smart meter IDS, and also to set up a small-scale hardware platform for AMI to test our algorithms. In conclusion, we hope our research can make contributions towards more secure AMI deployments by the use of steam data mining-based IDSs.

References

[1] G. J. FitzPatrick and D. A. Wollman, "NIST interoperability framework and action plans," in *Proc. 2010 IEEE Power and Energy Society General Meeting*, 2010, pp. 1–4.

[2] X. M. Bai, J. X. Meng, and N. H. Zhu, "Functional analysis of advanced metering infrastructure in smart grid," in *Proc. 2010 International Conference on Power System Technology (POWERCON)*, 2010, pp. 1–4.

[3] D. Li, Z. Aung, J. R. Williams, and A. Sanchez, "Efficient authentication scheme for data aggregation in smart grid with fault tolerance and fault diagnosis," in *Proc. 2012 IEEE Power and Energy Society Conference on Innovative Smart Grid Technologies (ISGT)*, 2012, pp. 1–8.

[4] F. M. Cleveland, "Cyber security issues for advanced metering infrastructure (AMI)," in *Proc. 2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, 2008, pp. 1–5.

[5] R. Berthier, W. H. Sanders, and H. Khurana, "Intrusion detection for advanced metering infrastructures: requirements and architectural directions," in *Proc. 1st IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2010, pp. 350–355.

[6] R. Berthier and W. H. Sanders, "Specification-based intrusion detection for advanced metering infrastructures," in *Proc. 17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2011, pp. 184–193.

[7] P. P. Rodrigues and J. Gama, "Holistic distributed stream clustering for smart grids," in *Proc. 2012 Workshop on Ubiquitous Data Mining (UDM)*, 2012, pp. 18–22.

[8] (2009) Open Public Extended Network Metering. http://www.openmeter.com/.

[9] (2012) Massive Online Analysis, release 2012.03. http://moa.cs.waikato.ac.nz.

[10] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "MOA: Massive online analysis, a framework for stream classification and clustering," in *Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings, Workshop on Applications of Pattern Analysis*, vol. 11, 2010, pp. 44–50.

[11] M. A. Faisal, Z. Aung, J. Williams, and A. Sanchez, "Securing advanced metering infrastructure using intrusion detection system with data stream mining," in *Proc. 2012 Pacific Asia Workshop on Intelligence and Security Informatics (PAISI)*, ser. LNCS, vol. 7299, 2012, pp. 96–111.

[12] R. Shein, "Security measures for advanced metering infrastructure components," in *Proc. 2010 Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2010, pp. 1–3.

[13] (2011) Data Concentrator in AMI. http://www.meworks.net/userfile/44670/DataConcentratorforAdvancedMeteringInfrastructure(AMI)_1.pdf.

[14] N. C. N. Chu, A. Williams, R. Alhajj, and K. Barker, "Data stream mining architecture for network intrusion detection," in *Proc. 2004 IEEE International Conference on Information Reuse and Integration (IRI)*, 2004, pp. 363–368.

[15] S. H. Oh, J. S. Kang, Y. C. Byun, G. L. Park, and S. Y. Byun, "Intrusion detection based on clustering a data stream," in *Proc. 3rd ACIS International Conference on Software Engineering Research, Management and Applications (ICSEMA)*, 2005, pp. 220–227.

[16] Q. H. Liu, F. Zhaoi, and Y. B. Zhao, "A real-time architecture for NIDS based on sequence analysis," in *Proc. 2005 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 3, 2005, pp. 1893–1896.

[17] M. U. Khan, "Anomaly detection in data streams using fuzzy logic," in *Proc. 2009 International Conference on Information and Communication Technologies (ICICT)*, 2009, pp. 167–174.

[18] Z. Qun and H. Wen-Jie, "Research on data mining technologies appling intrusion detection," in *Proc. 2010 IEEE International Conference on Emergency Management and Management Sciences (ICEMMS)*, 2010, pp. 230–233.

[19] Y. Zhang, L. Wang, W. Sun, R. C. Green, and M. Alam, "Distributed intrusion detection system in a multi-layer network architecture of smart grids," *IEEE Transactions on Smart Grid*, vol. 2, pp. 796–808, 2011.

[20] S. Wu and Y. Chen, "An embedded intrusion detection system model for application program," in *Proc. 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application (PACIIA)*, vol. 2, 2008, pp. 910–912.

[21] D. Li, Z. Aung, J. R. Williams, and A. Sanchez, "P3: Privacy preservation protocol for appliance control application," in *Proc. 3rd IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2012, pp. 294–299.

[22] M. Costache, V. Tudor, M. Almgren, M. Papatriantafilou, and C. Saunders, "Remote control of smart meters: Friend or foe?" in *Proc. 7th European Conference on Computer Network Defense (EC2ND)*, 2011, pp. 1–8.

[23] D. Brzeziǹski and J. Stefanowski, "Accuracy updated ensemble for data streams with concept drift," in *Proc. 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS) Part II*, 2011, pp. 155–163.

[24] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes, "Active learning with evolving streaming data," in *Proc. 2011 European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD) Part III*, 2011, pp. 597–612.

[25] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Proc. 2010 European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD) Part I*, 2010, pp. 135–150.

[26] A. Bifet, E. Frank, G. Holmes, B. Pfahringer, M. Sugiyama, and Q. Yang, "Accurate ensembles for data streams: Combining restricted HoeffdingTrees using stacking," in *Proc. 2nd Asian Conference on Machine Learning (ACML)*, 2010, pp. 225–240.

[27] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. 2007 SIAM International Conference on Data Mining (SDM)*, 2007, pp. 443–448.

[28] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceesings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009, pp. 139–148.

[29] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. 2004 Brazilian Symposium on Artificial Intelligence (SBIA)*, 2004, pp. 286–295.

[30] M. Baena-Garcìa, J. d. Campo-Àvila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-bueno, "Early drift detection method," in *Proc. 4th International Workshop on Knowledge Discovery from Data Streams (IWKDDS)*, 2006, pp. 77–86.

[31] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2001, pp. 97–106.

[32] N. Oza and S. Russell, "Online bagging and boosting," in *Proc. 8th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2001, pp. 105–112.

[33] (1999) KDD Cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[34] P. Li, X. Wu, and X. Hu, "Learning from concept drifting data streams with unlabeled data," in *Proc. 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010, pp. 1945–1946.

[35] (1998) DARPA intrusion detection evaluation. http://www.ll.mit.edu/mission/communications/CST/darpa.html.

[36] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD Cup 99 data set," in *Proc. IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009, pp. 1–6.

[37] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, pp. 37–46, 1960.

[38] (2013) GoGird billing model. https://wiki.gogrid.com/wiki/index.php/Billing_Model.

Mr. Mustafa Amir Faisal (M.Sc. in Computing and Information Science, Masdar Institute of Science and Technology, UAE) is a Ph.D. candidate in the Department of Computer Science at University of Texas at Dallas, USA. His current research interests are control theory, smart grid security, data privacy, etc.

Dr. Zeyar Aung (Ph.D., National University of Singapore) is an Assistant Professor of Electrical Engineering and Computer Science, and is also affiliated to the Institute Center for Smart and Sustainable Systems (iSmart) at Masdar Institute of Science and Technology, Abu Dhabi, UAE.

Dr. John R. Williams (Ph.D., Swansea University, UK) is a Professor of Information Engineering, Civil and Environmental Engineering, and Engineering Systems, and is also the Director of Auto-ID Laboratory at Massachusetts Institute of Technology (MIT), USA.

Dr. Abel Sanchez (Ph.D., Massachusetts Institute of Technology, USA) is the Executive Director of Geospatial Data Center, and is also a Research Scientist and Software Architect of Auto-ID Laboratory at Massachusetts Institute of Technology (MIT), USA.