2019 IEEE International Conference on Software Architecture (ICSA)

An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics

Faheem Ullah[a,b] and Muhammad Ali Babar[a,b]
[a] CREST – the Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia
[b] Cyber Security Adelaide, The University of Adelaide, Australia
Email: {faheem.ullah, ali.babar}@adelaide.edu.au

***Abstract*—Big Data Cyber Security Analytics (BDCA) systems leverage big data technologies (e.g., Hadoop and Spark) for collecting, storing, and analyzing large volume of security event data to detect cyber-attacks. *Accuracy* and *response time* are the two most important quality concerns for BDCA systems. However, the frequent changes in the operating environment of a BDCA system (such as quality and quantity of security event data) significantly impact these qualities. In this paper, we first study the impact of such environmental changes. We then present *ADABTics*, an architecture-driven adaptation approach that (re)composes the system at runtime with a set of components to ensure optimal accuracy and response time. We finally evaluate our approach both in a single node and multi-node settings using a Hadoop-based BDCA system and different adaptation scenarios. Our evaluation shows that on average *ADABTics* improves BDCA's accuracy and response time by 6.06% and 23.7% respectively.**

***Keywords*—big data, cyber security, adaptivity, software architecture, accuracy, response time.**

I. INTRODUCTION

Big data technologies (e.g., Hadoop and Spark) have gained tremendous popularity over the last few years, with applications in several domains such as healthcare [1], business analytics [2], and smart environments [3]. The traditional cyber security solutions (e.g., Intrusion Detection System (IDS)) are unable to cope up with the high Volume, Velocity, and Variety (i.e., 3V) of security event data (e.g., NetFlows) [4, 5]. For example, an enterprise as large as HP generates around one trillion security events per day [4, 5]. Hence, it is inevitable to incorporate big data technologies in the cyber security mix [4]. A cyber security analytics system that combines big data technologies and cyber security is called **B**ig **D**ata **C**yber security **A**nalytics (**BDCA**) system, which is defined as "*A system that leverages big data tools and technologies for collecting, storing, and analyzing large volume of security event data to protect organizational networks, computers, and data from unauthorized access, damage, or attack*" [6]. BDCA systems are of two types – (i) *Generic BDCA system* (e.g., an IDS supported with big data tools) that are designed to detect the variety of attacks such as DoS, brute force, and SQL injection (ii) *Specific BDCA system* (e.g., a phishing detector supported with big data tools) that are designed to detect a specific type of attack such as phishing attack.

Accuracy and *response time* are the most importance quality attributes of a BDCA system [5-7]. *Accuracy* measures the ability of a BDCA system to detect malicious access and not become a shield in the face of any legitimate access [7]. *Response time* measures how quickly a BDCA system collects and analyzes security event data to generate alerts, which are then used to prevent an attack [7]. In addition to commonly known factors (e.g., algorithm efficiency, hardware failures, and parameter configuration) that impact *accuracy* and *response time*, the software architecture of a BDCA system plays a significant role in accurately and quickly detecting attacks [6, 8, 9]. Different techniques are employed in BDCA systems to achieve different quality attributes (e.g., accuracy, scalability, and reliability) [6]. For instance, an anomaly-based detector (which detects attacks based on deviation from normal behaviour) is integrated with signature-based detector (which detects attacks based on known attack patterns) to boost *accuracy* [10]. A technique for boosting the *response time* is the selection of features from the security data to reduce the size of data [11].

The effectiveness of these techniques varies with changes in an operating environment. In our context, operating environment means the enterprise network where a BDCA system is deployed for detecting attacks. Among others, the most important changes are the changes in the quality and quantity (e.g., change in number of features in data) and speed of security event data ingested into a BDCA system [7]. Any such change has a direct impact on the effectiveness of the employed techniques and ultimately the *accuracy* and *response time* of the system. For example, a feature selection technique can help reduce response time in an environment, which generates security data with large number of features. Otherwise, the technique only negatively impacts the response time (§II.B). Accuracy and response time are not impacted equally with changes. For example, a BDCA system (having fixed processing power (e.g., number of CPUs)) subjected to an increase in data influx rate will make an entire compromise on *response time* while keeping the *accuracy* largely intact (§II.C). This leads to one quality (i.e., *accuracy*) being optimized and another quality (i.e., *response time*) being totally compromised. Furthermore, some techniques expose a trade-off situation between accuracy and response time. To illustrate the trade-off, consider the case of integrating an anomaly-based detector with a signature-based detector in a BDCA system. The integration enables two-phase processing – security data is first processed by the signature-based detector followed by the anomaly-based detector, which makes sure that most of the malicious activities are detected [10]. However, such increase in accuracy comes at the cost of increase in response time due to two-phase processing (§II.A). This raises the concern whether such techniques should be employed. Given the dynamic nature of a BDCA system's operating environment [12] and the aforementioned sensitivities to the changes in operating environment pose the problem - "***How to enable a BDCA system to ensure optimal accuracy and response time in the face of changes in the operating environment?***".

We present *ADABTics*, an **A**rchitecture-**D**riven **A**daptation approach for **B**ig data Cyber security Analy**Tics**, which combines *component-based architecture* [13, 14] with *architecture-driven adaptation* [15, 16] to ensure optimal accuracy and response time. By architecture-driven adaptation, we mean the ability of a BDCA system to modify its structure in terms of adding and removing components at runtime in response to changes in the operating environment. Given the two well-known types of adaptation (i.e., parameter- and architecture-driven adaptation) [16-18], we opted for architecture-driven adaptation. This is because parameter-driven adaptation is not suitable for the changes that occur over time such as gradual increase in the response time [19]. Accordingly, *ADABTics* views and

978-1-7281-0528-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSA.2019.00013

subsequently composes a BDCA system as a set of independent components (§III.A). Each component embodies a technique (e.g., feature selection) devised for boosting accuracy or response time. In addition to other advantages (e.g., code reuse, maintainability, and independent development), organizing a BDCA system as a set of components renders the flexibility to add and remove components from the available pool of components at runtime [13]. Such flexibility is exploited by the *ADABTics*' adaptation mechanism (§III.B), which monitors the accuracy, response time, and properties of the operating environment (i.e., data quality and influx speed) of a BDCA system. Upon detection of deviation from the expected values set by a user (e.g., accuracy falls below 80%), adaptation is triggered, which re-composes the architecture of the system constituting a different set of components. The adaptation ensures that the system (now composed of a new set of components) delivers optimal accuracy and response time. We have implemented *ADABTics* on Hadoop [20] for evaluation in three *execution modes* using four well-known security datasets. Our evaluation reveals that on average *ADABTics* improves a BDCA system's accuracy and response time by 6.06% and 23.7% respectively.

Contributions: Our work makes the following contributions:

1. Demonstrates the impact of changes in the operating environment on the accuracy and response time of a BDCA system (§II)
2. Proposes a component-based architecture for BDCA (first part of *ADABTics*) (§III.A)
3. Introduces an architecture-driven adaptation approach for BDCA (second part of *ADABTics*) (§III.B)
4. Implements and evaluates the effectiveness of *ADABTics* through comprehensive experiments (§V)

Paper Organization: §II motivates the undertaken work. §III describes our approach, which is followed by implementation in §IV. §V presents the results. §VI discusses the results. §VII outlines the limitation. Related work is presented in §VIII. §IX concludes the paper.

II. MOTIVATION

In this section, we illustrate the accuracy-response time trade-off and expose the impact of environmental changes.

A. Accuracy-Response Time Trade-off

Accuracy and response time pose a trade-off situation (i.e., improving one compromises another) at several design decisions in a BDCA system. Such decisions are characterized by the incorporation of architectural components. For example, integrating a **S**ignature-based **D**etection (SD) component with an **A**nomaly-based **D**etection (AD) component improves accuracy but at the cost of increased response time. To exemplify such trade-offs, we investigated the impact of two important components (i.e., SD and **A**lert **R**anking (AR)) on accuracy and response time in a Hadoop-based BDCA system. The metrices used for measuring accuracy and response time are presented in Table I and the implementation details of the experiment are available in §IV. The boxplots in Fig. 1 shows the difference between the accuracy and response time of the system with and without the components (i.e., SD and AR). Fig. 1a and Fig. 1b illustrates that on average the use of SD component increases accuracy by 2.31% but at the cost of 46.06% increase in training time. Our qualitative analysis of the ranked alerts indicates that the alerts ranked on top are more severe, hence, the use of AR component improves usability and subsequently accuracy. However, Fig. 1c shows that the component increases the prediction time by 46.2% on average. Given that a BDCA system is architected using a variety of components [6], these results highlight the underlying trade-offs and point towards the challenging decision about the selection of components. For example, whether AR component should be included in the BDCA architecture.

TABLE I. EVALUATION METRICES AND THEIR DESCRIPTIONS

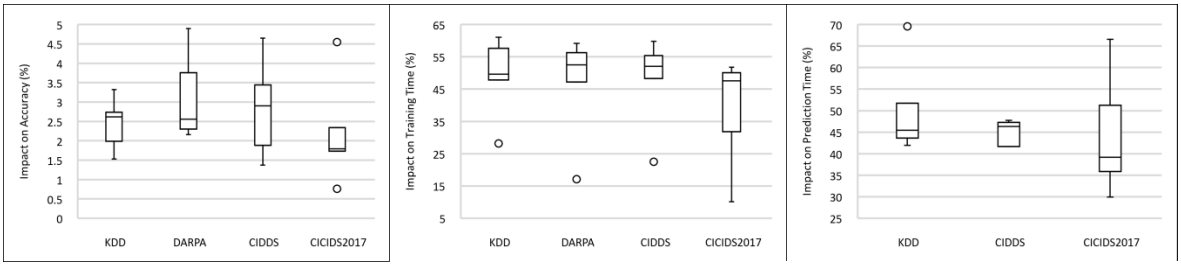
Metrix	Description	Equation
Accuracy	Percentage of correctly detected attack and normal instances	$\text{Acc} = \left(\frac{TP+TN}{TP+FP+FN+TN}\right) * 100$
F1 Score	The weighted harmonic mean of the precision and recall.	$\text{F1} = 2/\left(\frac{1}{Precision} + \frac{1}{Recall}\right)$
Detection Rate	Percentage of correctly detected attack instances	$DR = \left(\frac{TP}{TP+FP}\right) * 100$
False Positive Rate (FPR)	Percentage of normal instances detected as attack instances	$FPR = \left(\frac{FP}{FP+TN}\right) * 100$
Training Time	Time taken by the system in seconds to train the model using training data	
Prediction Time	Time taken by the system in seconds to detect attacks in the testing data	
True Positive (TP) = No. of correctly detected attack instances False Positive (FP) = No. of normal instances detected as attack instances False Negative (FN) = No. of attack instances detected as normal instances True Negative (TN) = No. of correctly detected normal instances Precision $= \left(\frac{TP}{TP+FP}\right) * 100$ and Recall $= \left(\frac{TP}{TP+FN}\right) * 100$		

B. Sensitivity of Architectural Components to Data Variety

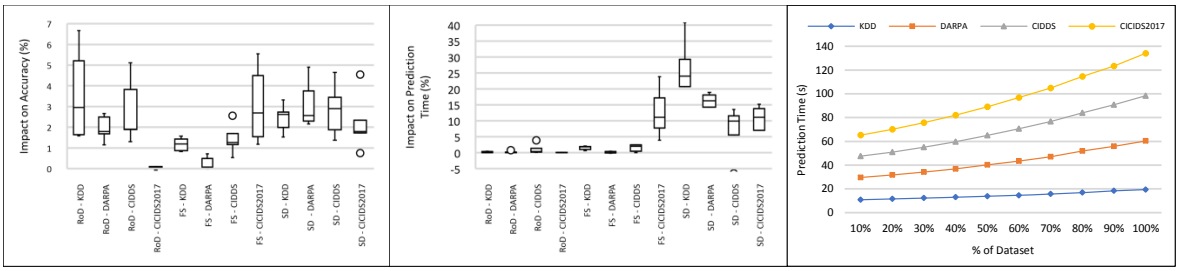
The characteristics of the security data (e.g., number of features in each network connection or amount of incorrect values in the collected security data) often changes in the underlying network [7, 12]. Such changes impact the performance of various components in the system. To illustrate the impact, we investigated the impact (in terms of accuracy and response time) of change in data quality on various components. Fig. 1d shows the impact on accuracy observed for three components (i.e., Removal of Duplicates (RoD), Feature Selection (FS), and SD) for different security datasets. The impact is measured by executing the experiment on the system with and without the component. It is evident from Fig. 1d that the impact (or contribution) of the components varies with the change in data quality. For instance, the accuracy for KDD [26] improves by up to 6.7% with the RoD component while the same component has negligible impact for CICIDS2017 [29]. The same variation is observed for prediction time (Fig. 1e). The use of FS component reduces the average prediction time by 12.7% for CICIDS2017 but achieves a reduction of only 1.51% for KDD. Similarly, SD component reduces the prediction time by 24% for KDD but only achieves a reduction of 10% for CICIDS2017. From these findings, it can be deducted that the various components in a BDCA system are highly sensitive to quality of security data. Hence, the components should be operationalized only when the operating environment (e.g., data quality) is as such that the component can play a significant role. For instance, operationalizing RoD component for CICIDS2017 will only be counterproductive as evident from Fig. 1d.

C. Sensitivity of Architectural Components to Data Velocity

The velocity of security data in the underlying network infrastructure is quite dynamic [7, 12]. For example, the network of a bank is very active in working hours as compared to non-working hours. Thereby, data is generated at a higher speed in working hours. The response time of a BDCA system (having fixed computational capacity) is likely to decrease with the increase in the data speed. To better highlight this relation, we conducted an experiment where a BDCA system is subjected to varying size of security data to be processed at a given time. In each execution, the system is processing a subset (e.g., 10%, 20%, and so on) of the total dataset. Our results show that the amount of data (to be processed at a given time) does not have any significant impact on the accuracy. On the other hand, Fig. 1f illustrates a constant increase in the prediction time as the size of the data increases. For instance, the prediction time for KDD



(a) Increase in accuracy with inclusion of SD component (b) Increase in training time with inclusion of SD (c) Increase in prediction time with inclusion of AR component



(d) Increase in accuracy with inclusion of various components for various datasets (e) Impact on prediction time with inclusion of various components for various datasets (f) Impact of increase in data speed on the prediction time

Figure 1. Impact of changes in operating environment on the accuracy and response time of a BDCA system.

increases from 8 sec to 20 sec as the size of data increases from 10% to 100%. The findings of this experiment enable us to conclude that as the data speed (and subsequently data size) increases, the BDCA system is keeping the accuracy largely intact and making the entire compromise on the prediction time, which is not an effective approach. A delay in response to an attack can have detrimental consequences [6].

III. OUR APPROACH

We present our approach, *ADABTics,* that enables a BDCA system to achieve optimal accuracy and response time. *ADABTics* consists of two parts – component architecture and adaptation approach.

A. Component Architecture

In this section, we first present the meta-model followed by the component-based BDCA architecture.

1) BDCA Architecture Meta-model: The BDCA meta-model characterizes the elements of a BDCA architecture and the relationship among the elements. The meta-model consists of four architectural elements shown in Fig. 2. **Component:** A software component is a unit of encapsulated behavior that is used to organize operations and data [13]. **Interface:** An interface is a collection of operations that are used to access a component. The interfaces in our model are defined in terms of ports, which are points of connection between a component and its environment. For instance, Fig. 2 shows that *Component 1* has two types of ports (i.e., P1A [1..*] and P1B [1..*]) with associated interfaces (i.e., I1A and I1B). **Connector:** A connector is a runtime communication link between two components [21]. Each connecter is specified by its name and the interfaces it connects. **Composition:** In our model, the entire BDCA system is modeled as a composite component, which is composed of other components.

2) BDCA Component-based Architecture: The component-based architecture of a BDCA system is presented in Fig. 3 as managed system. We first present the architecture overview followed by the description of various features. *Overview:* The architecture is organized into layers, modules, phases, and components with layers being at the highest

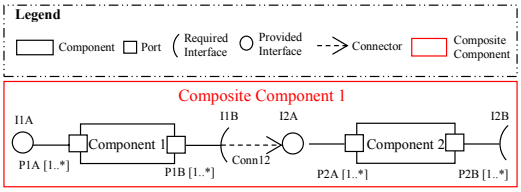


Figure 2. Meta-model for BDCA Component Architecture.

abstraction level and components at the lowest. There are two layers – *Security Analytics Layer* that collects, prepares, and analyzes the security data and *Big Data Support Layer* that supports the storage and processing of the large volumes of heterogenous natured security data. Each layer is composed of several modules with *Security Analytics Layer* being composed of three modules (i.e., Data Collection, Data Analytics, and Visualization) and *Big Data Support Layer* being composed of two modules (i.e., Storage Support and Processing Support). Each module embodies several components except for Data Analytics module, whose complexity first warrants a categorization into different phases pertaining to the data lifecycle.

a) Security Analytics Layer: This layer implements the key features responsible for detecting attacks. It is composed of the following three modules.

Data Collection: This module collects security data from data sources (e.g., Packet data, NetFlow data, IDS log data, firewall logs, email servers, operating system logs, and application servers). This way external data sources are connected to the BDCA system. Data Collection module uses different tools (e.g., Wireshark and nmap) for collecting the data.

Data Analytics: The data analytics module is responsible for analyzing the collected security data to extract the desired insight (i.e., information about malicious cyber activities). The module consists of five phases. Data Engineering: This phase engineers or preprocess the security data to enable it to be used efficiently in the subsequent phases. The phase incorporates the following components. The *Removal of duplicates* component removes duplicate records from the security data. This is important because the existence of duplicate records makes the Machine Learning (ML) model (in AD component) biased towards more frequent records and

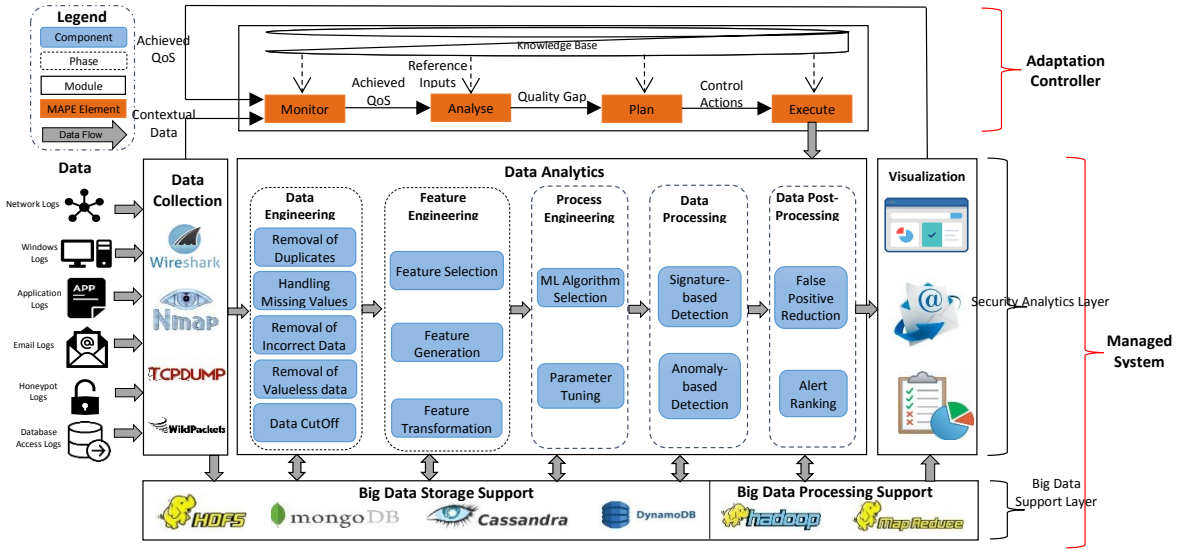


Figure 3. BDCA Component Architecture.

prevents it from learning rare behaviors that pertains to more dangerous attacks [21]. The *Handling missing values* component asses the number of records in the collected security data that has missing values. Based on the statistical assessment, this component either deletes the records with the missing values or opt for data imputation to fill the missing values. The *Removal of incorrect data* component filters the collected data to remove records with incorrect values. For example, a negative value (e.g., -20) for number of bytes in a network connection is considered as an incorrect value. The *Removal of valueless data* component removes the data that does not contribute to attack detection. For example, a network sniffer collects zero-byte data, which is used for handshaking in TCP/IP connection. Such data does not have any relevance for security analytics, hence, it should be discarded. The *Data CutOff* component imposes a customizable limit on the data to be included for a connection or process. For instance, the component can impose a cutoff limit of 15 KB for each network connection, which ensures to only retain the first 15 KB data for each connection. *Feature Engineering:* This phase is responsible for selecting, generating, and/or transforming features from the engineered data. Given that each record in security data is composed of several features (e.g., 41 features in KDD and 82 features in CICIDS2017) and not all of them contribute to attack detection, the selection of specific features for subsequent processing is crucial for achieving accurate and speedy results. This phase consists of four components. The *feature selection* component leverages intelligent algorithms (e.g., InfoGain [11]) to select the most relevant features. The *Feature generation* component creates new features from the available features with the aim of accuracy improvement. To help the ML model discriminate more clearly among various classes (e.g., benign and malicious record), the *Feature transformation* component transform the features into a different space using techniques such as normalization of features into a range of 0 to 1. *Process Engineering:* This phase consists of components responsible for taking various measures to ensure effective data processing in the subsequent step as shown in Fig. 3. In our architecture, the phase involves components – (1) *ML algorithm selection* component selects the most suitable ML algorithm based on applying and validating available algorithms (2) *Parameter Tuning* component handles the tuning of various parameters (e.g., value of K for Kmeans) required in data processing phase. *Data Processing:* This phase processes the prepared security data in parallel fashion to extract insight that signals towards a potential attack. Based on the processing technique, BDCA can be categorized into (i) Signature-based detection and (ii) Anomaly-based detection [7]. Accordingly, this phase consists of two components. The *Signature-based detection* component detects attacks based on already known attack patterns while *Anomaly-based detection* component uses ML algorithms to detect attacks based on deviation from already learned normal behavior. *Data Post-processing:* This phase processes the analyzed results to remove (any) false alarms and facilitate the user in responding to the detected attacks. The phase consists of two components – *False Positive Reduction* component leverages vulnerability signatures to reduce the false positive rate and *Alert Ranking* component ranks the generated alerts based on their dangerousness and severity [22] to facilitate a user in responding to more dangerous attacks on priority basis.

Visualization: This module consists of tools (e.g., dashboard, email notifier, and reports) that are used to communicate the results of security analytics to the user.

b) ***Big Data Support Layer:*** This layer leverages big data technologies to support the *Security Analytics layer* by managing all the issues related to data storage and data processing distribution. The layer consists of two modules. The *Big Data Storage Support* module manages the back and forth storage of the security data after being processed in each module and phase of the *Security Analytics layer*. The module uses different data storage tools (e.g., HDFS, MongoDB, or Cassandra) to support the required data storage. The *Big Data Processing Support* Layer provides the processing framework for distributed processing of the data. To support such processing, the module uses big data framework (i.e., Hadoop) and its programming framework (i.e., MapReduce).

B. Adaptation Approach

We now present our adaptation approach that leverages the component-based BDCA architecture.

Adaptation Goal: Our adaptation goal specifies the primary reason as to why a BDCA system should be self-adaptive. As

revealed in §II, the Quality-of-Service (QoS) delivered by a BDCA system is sensitive to the changes in the environment. This sensitivity drives our adaptation goal, which is to "*enable a BDCA system to ensure optimal QoS in the face of changes in the operating environment*". Given that QoS encompasses a multitude of qualities (e.g., response time, scalability, and reliability), our focus is only on accuracy (measured in terms of attack detection rate, FPR, accuracy, and F1 score) and response time (measured in terms of prediction time)(Table I).

Reference Inputs: The reference inputs specify the concrete target/state to be achieved in the BDCA system by our adaptation approach. Like most research on self-adaptation, the reference input in our approach is the *ideal QoS* that a BDCA system aims to achieve through the adaptation approach. The *ideal QoS* is calculated using the utility function (Eq 1), which normalizes and sums the values for desired qualities. In Eq (1), *F(k)* is the obtained QoS at k^{th} iteration (workflow) and ω denotes the weight (specifying the importance) for a quality (e.g., detection rate) assigned by a user. The value functions (e.g., *FPR(k)*) are scaled using Eq (2) and Eq (3) [23] before calculating *F(k)*. In Eq (2) and Eq (3), $V_N(k)$ and $V_P(k)$ denotes the value function. Some quality attributes have a negative impact i.e., the higher the value, the lower the quality (e.g., FPR) while some quality attributes have a positive impact i.e., the higher the value, the higher the quality (e.g., detection rate). Eq (2) is used to scale qualities having negative impact and Eq (3) is used to scale qualities having positive impact.

$$F(k) = \frac{\omega_1 * FPR\ (k) + \omega_2 * PrT\ (k)}{\omega_3 * Acc\ (k) + \omega_4 * DR\ (k) + \omega_5 * F1\ (k)} \quad (1)$$

$$V_N(k) = \begin{cases} \frac{V_{max} - V_k}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (2)$$

$$V_P(k) = \begin{cases} \frac{V_k - V_{min}}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (3)$$

Measured Outputs: These are the set of values that are measured and compared with the reference inputs to evaluate whether the managed system has achieved the desired state. In our approach, we use *achieved QoS* as the measured output. The *achieved QoS* specifies the QoS achieved after the system has adopted to the changes. The *achieved QoS* is measured using Eq (1). The difference between *ideal QoS* and *achieved QoS* is termed as *quality gap*. We define a *threshold* for the *quality gap*. If the *quality gap* is within the threshold, the system is delivering satisfied QoS and vice versa.

Control Actions: These are the actions undertaken to make the managed system adapt to the changes in the operating environment. With the introduction of any such change, the *quality gap* gets out of the already defined *threshold* range. Thereby, the control actions aim to bring the *quality gap* within the *threshold* range. In our approach, the control action is the re-composition of the architecture of a BDCA system. Each time a QoS violation is detected (i.e., *quality gap* exceeds *threshold* range), the control action is triggered, which computes the QoS (i.e., *achieved QoS*) delivered by different composition of the system's architecture. The compositions are specified by **W**ork**F**lows (WF) shown in Appendix A. The workflows differ in terms of the number of components instantiated in each workflow. For example, WF-2 consists of two components (i.e., RoD and **A**nomaly-based **D**etection (AD) and WF-8 consists of three components (i.e., RoD, FS, and AD). The algorithm for the selection and ultimately recomposing the architecture of the system is shown in Algorithm 1. Upon activation of adaptation trigger, the algorithm starts calculating the *achieved QoS* for different workflows (line 6 and 7) and discards workflows with *achieved QoS* greater than the *threshold* (line 8-10). There are three conditions (specified by S_{cond}) that stop and recompose the architecture – (i) *QoS satisfaction:* as soon as the QoS for a workflow is found within the *threshold* range, the process is stopped and the particular workflow is selected for re-composition of the architecture (ii) *User-defined limit:* the user defines a maximum number for workflows to be executed upon adaptation. When the maximum number of workflow executions is reached, the workflow with the best QoS is selected and (iii) *Most optimal QoS:* the QoS for all the possible workflows is calculated and the workflow with the best QoS is selected for re-composition of the architecture. The algorithm also enables the user to apply constraints on the workflow selection (line 12 and 13). The constraints filter out workflows with acceptable overall QoS but violating a specific user requirement (e.g., FPR should always be less than 0.05%).

Algorithm. 1. Workflow Selection Algorithm

Input: QoS_{idl} = ideal QoS
$QoS_{thrshold}$ = QoS threshold
WF = {WF_1, WF_2, …., WF_N} // *Set of possible workflows*
C = {C_1, C_2, …., C_Z} // *Set of user-defined constraints*
I_{max} = maximum number of iterations
S_{cond} = {QoS Satisfaction, User-defined Limit, Most Optimal QoS} // *user chooses stopping condition*
Output: WF_{opt} = Optimal Workflow
Steps:

```
1.  WF_opt = WF_1
2.  QoS_gap = |Q_idl – F(WF_1)|
3.  If S_cond = most optimal workflow
4.          I_max = N
5.  For i = 1 to I_max
6.          Calculate Utility Function F(WF_i)
7.          QoS_gap (WF_i) = |QoS_idl – F(WF_i)|
8.          If QoS_gap (WF_i) > QoS_thrshold
9.                  Jump to Next Iteration of ith For loop
10.         Else
11.         For j=1 to Z
12.             If (WF_i violates any constraint)
13.                 Terminate the jth For loop
14.             Else
15.                 {
16.                  If (S_cond = threshold entry)
17.                 {   Wopt = WF_i
18.                     Terminate the process
19.                 }
20.                 Else
21.                     { QoS_gap = QoS_gap (WF_i)
22.                      WF_opt = WF_i
23.                     }
```

// The final value of WF_{opt} is the optimal workflow selected for re-composition

System Architecture: The system consists of two subsystems –*managed system* and *adaptation controller*. The different aspects of the managed system (i.e., BDCA system) have already been reported in §III.a. The adaptation controller follows the structure of the MAPE-K loop, which consists of five main elements. *Monitor* collects the information related to the QoS delivered by the managed system. As shown in Fig. 3, the QoS delivered by the managed system is directly monitored through the *monitor* element. The information collected by *monitor* is analyzed through *analyzer*, which compares it with the reference inputs and determines if the *quality gap* has crossed the *threshold*. In case of quality gap crossing the threshold (indicating that the system requires adaptation), the *analyzer* triggers the *planner* element. The *planner* computes the controls actions that suggest a new workflow for the composition of the managed system as mentioned previously. The *executor* recomposes the structure of the managed system according to the new workflow. The *knowledge base* maintains the data of the managed system (e.g., QoS constraints), operating environment (e.g., data influx rate), and data shared among the four MAPE elements (e.g., reference inputs).

Adaptation Properties: These properties measure the effectiveness of the adaptation approach [24]. Our approach is measured in terms of following properties – (i) *Adaptation*

Accuracy measures how close the achieved QoS approximates to the ideal QoS upon adaptation. (ii) *Adaptation Stability* measures the ability of the adaptation approach to ensure that the managed system adapts under different scenarios instead of indefinitely searching for an optimal workflow (iii) *Adaptation Time* measures the speed of the adaptation approach i.e., how quickly the managed system adapts.

IV. IMPLEMENTATION

After presenting our approach, we now describe the details of our implementation approach.

Instrumentation: We implemented *ADABTics* on Apache Hadoop [20] – a widely adopted big data processing framework. We used five ML algorithms in our BDCA system, which are implemented in a distributed fashion using Hadoop's machine learning library – Apache Mahout [25].

Execution modes: *ADABTics* has been executed in three different modes – ***S**tand**A**lone (SA), **P**seudo-**D**istributed (PD), and **F**ully-**D**istributed (FD)*. This is because execution mode closely relates to response time, which is one of the key qualities considered in this study. *SA mode* mimics sequential execution, where Hadoop uses local file system and all jobs run in a single Java process. *PD mode* mimics distributed Hadoop cluster and stores data in HDFS but runs on a single machine. The machine used for SA and PD executions is an HP EliteBook 850 G5 configured with 2x Intel Xeon processors (2.60 GHz and 2.70 GHz), 8 GB RAM, 128 GB hard drive, and installed with CentOS 6.4 and JVM 1.7.0. *FD mode* distributes data processing among multiple machines. We configured a Hadoop cluster on OpenStack cloud for our experiments. The cluster consists of 17 nodes (16 slaves and 1 master) each running CentOS 6.4. Each slave node is equipped with 2GB RAM and 20GB disk space while the master node is equipped with 8GB RAM and 80GB disk space.

Managed System and Adaptation Controller: The various components of the managed system and adaptation controller have been implemented using Java MapReduce paradigm. Among several components presented in §III.A, we implemented only six components (i.e., Removal of Duplicates (C1), Handling Missing Values (C2), Feature Selection (C3), Signature-based Detection (C4), Anomaly-based Detection (C5), and Alert Ranking (C6)) in the workflows (Appendix A) for evaluation of *ADABTics*. This is because a BDCA system can be designed using a multitude of components as presented in §III.A, hence, implementing all components, and generating and executing subsequent workflows is a challenging and time-consuming undertaking. Such an undertaking will only increase the number of workflows, which will have little (if none) impact on our evaluation results. We have implemented *Removal of Duplicates* using *NullWritable* class available in *org.apache.hadoop.io* library that detect and discard exactly similar records. For implementing *Handling Missing Values*, we used the mapper function to monitor each feature of each record and discard records with missing values. We implemented the *Feature Selection* component using InfoGain algorithm [11]. We followed the approach proposed in [10] for implementing *signature-based detection* component – leveraging C4.5 to generate signatures that can accurately detect known attacks. *Anomaly-based detection* is the core component and is instantiated in each workflow (Appendix A). The component leverages ML algorithms, which are implemented using Mahout [25]. We separately used five state-of-the-art algorithms (i.e., Kmeans (KM), XGBoost (XGB), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF)) in the evaluation. For *Alert Ranking*, we employed the ranking model proposed in [22] that ranks the alerts based on the number of bytes embodied in each alert. The implementation details and source code of these components are available at https://goo.gl/oJ5e2U.

Datasets: We used four well-known security datasets (i.e., KDD [26], DARPA [27], CIDDS [28], and CICIDS2017 [29]) for the evaluation of our approach. These datasets vary in terms of size, redundancy, quality and quantity of features, and nature and percentage of attacks. Hence, we assert that our evaluation is rigorous and realistic. The various types of attacks contained in each dataset are shown in Fig. 4. The size of training and testing data and percentage of normal and attack instances in each dataset are shown in Table II.

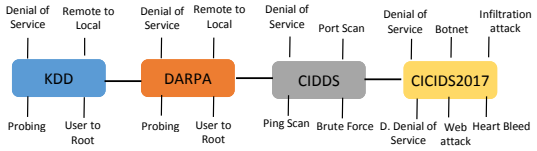


Figure. 4. Attack types in each dataset.

TABLE II. STATISTICS OF THE SECURITY DATASETS

Dataset	Connection	Training Data		Testing Data	
		No. of Records	Percentage of Records	No. of Records	Percentage of Records
KDD	Normal	972781	19.8%	60593	19.4%
	Attack	3925650	80.14%	250436	80.5%
DARPA	Normal	1145946	42.07%	658797	43.3%
	Attack	1577550	57.92%	863513	56.7%
CIDDS	Normal	4706999	83.54%	2303898	81.78%
	Attack	927349	16.46%	513275	18.22%
CICIDS2017	Normal	1109933	84.6%	555005	84.6%
	Attack	201889	15.4%	100905	15.4%

V. EVALUATION

In this section, we first outline the adaptation scenarios and configurations used for evaluation. We then formulate the evaluation objectives and finally present the results.

A. Adaptation Scenarios:

We have evaluated our approach under three different scenarios in which a BDCA system is likely to find itself. ***B**ase**L**ine (**BL**)* – this is the (initial) state of the system in an operating environment where it is ensured that the system's achieved QoS is within the tolerance/acceptable level. Hence, there is no need of any adaptation. *Varying **D**ata **Q**uality (**DQ**)* – the system is subjected to a change in the quality of security data (e.g., from KDD to DARPA). With the change in data quality, the achieved QoS is either forced out of the acceptable zone or remains in the acceptable zone. In the former case, adaptation is triggered to bring back the QoS within the acceptable level. *Varying **D**ata **S**peed (**DS**)* – the system is subjected to a change in the quantity of security data (e.g., from 500 MB to 1000 MB per unit time) to be processed at a given time. Similar to DQ, the achieved QoS either gets out of the acceptable zone or remains within acceptable zone with the change in data speed. The adaptation is triggered only in the case when achieved QoS gets out of the acceptable zone.

B. Configurations

As mentioned in §III.B, *ADABTics* requires the user to configure the values for ideal QoS, QoS threshold, constraint, and weights. The values configured for the sake of the evaluation are shown in Table III. *Ideal QoS* is calculated by feeding the best possible values (i.e., target values) for each metric into Eq 1 generating value 0 for ideal QoS. For computing *QoS threshold*, we collected the values for each metric for the 20 cases (4 datasets × 5 algorithms) considered in our experiment. We then computed the mean for each

metric shown in Table III. The mean values are then fed to Eq 1, which generated a value of 1.07 for QoS threshold. For setting constraints, we first plotted boxplots for each metric value obtained for the 20 cases to get the distribution of points. The upper bounds of the distribution are set as constraints for the positive QoS metrices (e.g., prediction time < 17sec) while the lower bounds of the distribution are set as constraints for the negative QoS metrices (e.g., Accuracy > 78.16). We assigned same weight (i.e., 1) to all metrices.

TABLE III. QOS METRICES CONFIGURATION VALUES

QoS Metric	Ideal	Mean	Constraint
Accuracy (%)	100	86.2	> 81.16
Detection Rate (%)	100	87.4	> 82.3
F1 Score (%)	100	86.9	> 81.4
False Positive Rate (%)	0	0.52	< 1.34
Prediction Time (sec)	0	17.3	< 28.4

C. Evaluation Objectives:

The evaluation presented in this section aims to answer the following Research Questions (RQ).

RQ1: ***How close to the ideal QoS does the QoS of BDCA system reaches using ADABTics (i.e., adaptation accuracy)?***

RQ2: ***How often does ADABTics is successful in making BDCA system adapt to the changes (i.e., adaptation stability)?***

RQ3: ***How long does it take for ADABTics to make a BDCA system adapt to the changes (i.e., adaptation time)?***

D. Results:

1) RQ1: Adaptation Accuracy

Adaptation accuracy is measured by how close is the best QoS (among several QoS values for a specific case) to the ideal QoS. Fig. 5 shows the QoS achieved by the BDCA system for each of the dataset and for each of the ML algorithm in various workflows. For the 20 experimental cases (i.e., 4 datasets × 5 algorithms), the average adaptation accuracy achieved by *ADABTics* is 0.42. The best accuracy (i.e., 0.07) is achieved with CICIDS2017 + XGB in workflow 14 (Fig. 5d). The metrices values for the best adaptation accuracy are detection rate 95, F1 score 94.8, accuracy 94.61, FPR 0.11, and prediction time 25.94. The components operationalized in workflow 14 are C3, C4, and C5. Among datasets, the most accurate (average) results are obtained for CICIDS2017 (0.13) followed by CIDDS (0.37), KDD (0.44), and DARPA (0.72). This trend is in line with the number of features in each dataset. The system achieves high detection rate, F1 score, and accuracy for CICIDS2017, which has the largest number of features (i.e., 82). Hence, these metrices dominate Eq (1) to generate lower QoS values for CICIDS2017. Similar to datasets, the adaptation accuracy varies among the ML algorithms employed in the system with SVM taking the lead to achieve an adaptation accuracy of 0.37 on average. The remaining algorithms (i.e., XGB, KM, NB, and RF) achieves an adaptation accuracy of 0.39, 0.42, 0.43, and 0.47 respectively. The workflows violating the applied constraints are specified by the red cross in Fig. 5. As can be seen, the workflows generating the most accurate QoS do not violate any constraint. Therefore, adaptation accuracy is not affected by the applied constraints. Fig. 6a illustrates the QoS achieved for each dataset under different execution modes. FD mode with a mean adaptation accuracy of 0.45 outclasses SA and PD mode, which achieves mean adaptation accuracy of 0.48 and 0.49 respectively. However, this difference is not in line with the computational power (16 nodes to FD and 1 node to SA and PD) allocated to each mode. This is because the fully distributed implementation of Hadoop is more efficient for larger datasets [20, 30, 31]. In our implementation, this is evident from the difference observed for the large size dataset as compared to the small size dataset. For example, the average adaptation accuracy for the three modes in case of CIDDS (which is a large size dataset) is 0.32 (FD), 0.36 (SA), and 0.37 (PD) as compared to 0.5 (FD), 0.52 (SA), and 0.53 (PD) for KDD (a small size dataset).

2) RQ2: Adaptation Stability

Adaptation stability is measured by the number of times achieved QoS falls within the threshold region. The greater the adaptation stability, the greater are the chances that the system will adapt upon activation of adaptation trigger. It is important to note that adaptation stability depends upon the choice of threshold range [24]. The results presented here are in accordance with our threshold range discussed in §V.B. For the 600 QoS points illustrated in Fig. 5, 284 falls within the threshold region. This way *ADABTics* achieves an adaptation stability of around 47.3% (i.e., 284/600). The result is important from two perspectives. First, it reveals that 52.7% of BDCA's composition (workflows) are not able to deliver the desired QoS, which motivates the need for adaptation. Second, the result illustrates that there are 47.3% chances of making the system adapt to the operating environment. The highest stability is achieved with CICIDS2017, where 135 out of 150 (30 workflows × 5 algorithms) QoS points lies within the threshold region. This is followed by KDD, CIDDS, and DAPRA for which 56, 54, and 39 QoS points lies with the threshold region respectively. Among classifiers, KM achieves the best adaptation stability (i.e., 58.3%) on average, which is followed by NB, SVM, XGB, and RF with a mean stability of 49.1, 46.6, 41.6, and 41.2 percent respectively. The QoS values for workflows that violate any of the applied constraints lies primarily out of the threshold region. Fig. 5 shows that only 9 out of 284 QoS values lying within the threshold region violates a constraint. This brings down the adaptation stability of *ADABTics* to 42% for the scenario where constraints are considered. The adaptation stability is highest (47.3%) in FD mode and lowest (41.8%) in PD mode.

3) RQ3: Adaptation Time

Adaptation time underlines the speed with which our adaptation approach makes the BDCA system adapt to the changes. As the reader may recall, the stopping conditions in our algorithm (described in §III.B) put a barrier on the execution time of the search process for selecting the optimal workflow. Therefore, we report our results with regards to the three stopping conditions – (i) *QoS satisfaction* (ii) *User-defined limit* (iii) *Most optimal QoS*. The time taken by *ADABTics* to adapt the system under the three operating conditions is shown in Fig. 7. The average adaptation time with *QoS satisfaction* condition is 4.6 times lower than *User-defined limit* condition and 16.2 times lower than *Most optimal QoS* condition. This trend is in line with the number of workflows to be executed under each stopping condition. It is important to note that unlike *User-defined limit* (set as 10 in this experiment) and *Most optimal QoS*, there is no control over the number of workflows execution under *QoS satisfaction* condition. This is because under *QoS satisfaction* condition, as soon as the algorithm finds a workflow with a QoS within the threshold, the algorithm selects the workflow and stops the search process. It is found that under *QoS satisfaction* condition, the adaptation time is lowest with DARPA (26.9 sec), which is followed by KDD (46.2 sec), CIDDS (46.7 sec), and CICIDS2017 (64.5 sec). With *User-defined limit* and *Most optimal QoS* conditions, the system adapts fastest with KDD followed by DARPA, CIDDS, and

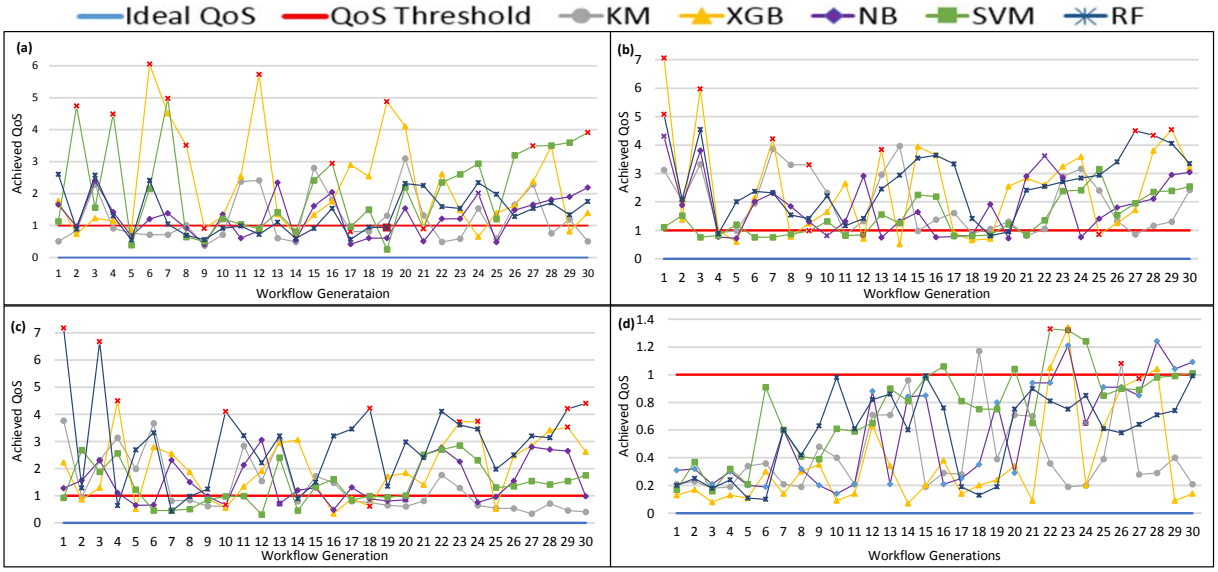


Figure 5. Achieved QoS for various datasets (a) KDD (b) DARPA (c) CIDDS (d) CIDIDS2017.

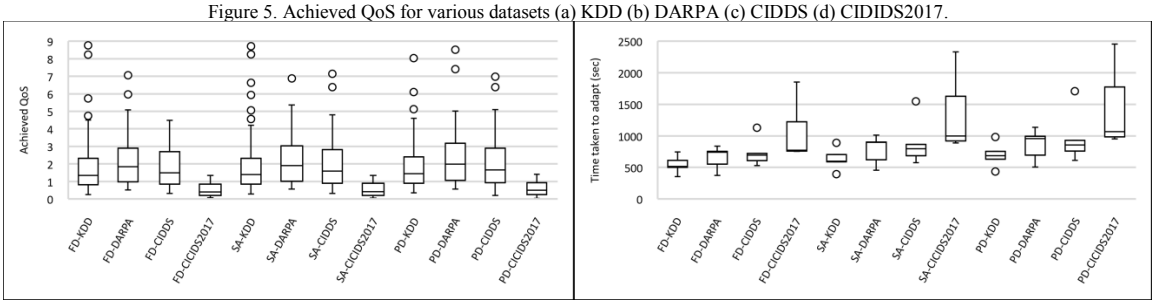


Figure 6. a) QoS achieved in various execution modes b) Adaptation time in various execution modes.

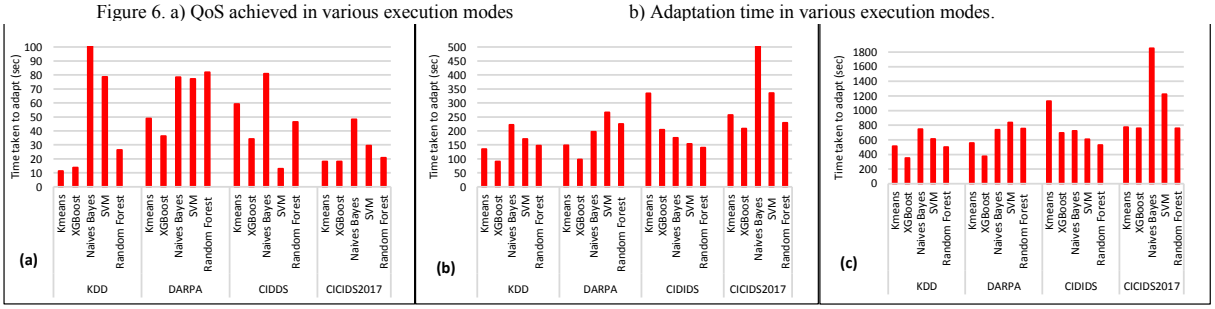


Figure 7. Adaptation Time under various conditions a) QoS satisfaction b) User-defined limit c) Most optimal QoS

CICIDS2017. This trend shows that under these two conditions, the adaptation time increases proportionally with the increase in the number of records in the dataset (mentioned in Table II). The average adaptation time taken by various ML techniques under *QoS satisfaction* condition is XGB (25.5 sec), KM (34.3 sec), RF (43.9 sec), SVM (49.5 sec), and NB (77.17 sec). The enforcement of constraints does not have any significant impact on adaptation time under second and third stopping condition as the constraints are checked after the QoS is calculated for the workflow (Algorithm 1). Although the constraints do not impact the adaptation time even under *QoS satisfaction* condition in our experiments, this finding cannot be generalized. This is because it is quite likely that a different set of constraints might be violated by the first workflow with satisfied QoS, in which case the adaptation time will be negatively impacted. Fig. 6b shows the impact of execution mode on the adaptation time for *Most optimal QoS* condition. Unlike the marginal impact of execution mode on adaptation accuracy (where accuracy related metrices dominate Eq (1)), the impact of execution mode is quite significant for adaptation time. The average adaptation time with FD mode (751.7 sec) is 17.8% faster than SD mode (914.5 sec) and 24.3% faster than PD mode (993.4 sec).

VI. DISCUSSION

Quantifying the achieved Optimization: We now turn to the question (related to our goal) that how much *ADABTics* optimizes the accuracy and response time. We try to answer this question by comparing the accuracy and response time of the BDCA system exactly before and after the adaptation. Table IV shows the mean values for accuracy, prediction time, and adaptation time for the 20 experimental cases under the three different stopping conditions before and after the adaptation. Under *QoS satisfaction* condition, adaptation improves accuracy and prediction time by 5.43% and 22.1% respectively. The average improvement recorded under *User-define limit* and *Most Optimal QoS* conditions are 23% and 26.11% in accuracy and 6.49% and 6.26% in prediction time respectively. To investigate whether the achieved

improvement is statistically significant, we applied Wilcoxon signed ranked test [32] (with significance level $\alpha = 0.05$) on the two set of values (i.e., before and after adaptation). The test generated p values of 0.00222, 0.0021, 0.0019 for accuracy and 0.0096, 0.0097, and 0.00998 for prediction time under *QoS satisfaction*, *User-defined*, and *Most Optimal QoS* condition respectively. All six values are lower than α (i.e., $p<0.05$), which concludes that the improvement for both accuracy and prediction time is statistically significant under all three stopping conditions.

TABLE IV. ACCURACY AND PREDICTION TIME BEFORE AND AFTER ADAPTATION

Stopping Condition	Experimental Case	Before Adaptation		After Adaptation		Adaptation Time (sec)
		Accuracy (%)	Prediction Time (sec)	Accuracy (%)	Prediction Time (sec)	
QoS Satisfaction	KDD	86.2	21	93.4	13.28	46.2
	DARPA	78.06	14.66	85	14.68	26.9
	CIDDS	79.4	23.9	82.9	16.8	46.7
	CICIDS2017	90.45	32.5	94.5	26.91	64.5
User-defined Limit	KDD	86.2	21	95.5	13	152.8
	DARPA	78.06	14.66	86.8	14.24	186.8
	CIDDS	79.4	23.9	83.16	17.5	201.24
	CICIDS2017	90.45	32.5	94.6	26.2	307
Most Optimal QoS	KDD	86.2	21	94.52	13.3	545.3
	DARPA	78.06	14.66	86.8	14.18	651.8
	CIDDS	79.4	23.9	83.22	14.6	736
	CICIDS2017	90.45	32.5	94.6	25.94	1073

Adaptation Accuracy VS Adaptation Time Trade-off: Most of the prior work on architecture-driven adaptation (e.g., [18] ,[33]) takes a short sighted view to adapt for the the most optimal QoS. Such approaches though are successful in achieving optimal QoS, but at the cost of increased adaptation time. Unlike prior work, our approach takes a long-sighted view and employs the flexibility to do a trade-off between adaptation accuracy and adaptation time. As evident from Table IV, the accuracy and prediction time are the most optimal under *Most Optimal QoS* condition followed by *User-defined limit* and *QoS satisfaction* conditions. However, the adaptation time is lowest under *QoS satisfaction* condition followed by *User-defined limit* and *Most Optimal QoS* condition. Therefore, a long-sighted view is required to give the opportunity to the user to employ the adaptation intelligently based on his know-how of the operating environment. Accordingly, our approach enables the user to leverage his basic knowledge of the operating environment. If the environment is experiencing frequent changes, adaptation is triggered very often. Therefore, the user opts for *QoS satisfaction* condition to minimize adaptation time. For an average frequency of changes, the user chooses the *User-defined limit* to customize the adaptation time. Finally, if the environment rarely changes, the user chooses *Most Optimal QoS* condition to achieve highest level of optimization.

VII. LIMITATIONS

The initial configuration of *ADABTics* (e.g., setting up the thresholds and constraints) relies on the expertise of security experts. If setup unrealistically and not aligned with the underlying operating environment, *ADABTics* may degrade accuracy and response time instead of optimizing them. However, the assumption is common to most of the research on adaptation approaches (e.g., [18, 19, 34]). The workflow selection algorithm (Algorithm 1) has to explore all possible workflows before finding the workflow with the most optimal QoS, which makes it an NP-hard problem [34]. Whilst most architecture-based optimization algorithms face this limitation, some techniques (e.g., market-based control [34]) have been recently proposed to mitigate the problem. We leave the incorporation of such techniques into *ADABTics* as a future work. Furthermore, the workflows can also be defined automatically using a heuristic-based approach instead of pre-defining them as reported in this study. Given that a BDCA system can be architected using different set of components and different big data frameworks (e.g., Spark and Storm), we cannot guarantee the generalization of our findings. However, it is worth noting that the goal of this paper is not to generalize the results, but rather to demonstrate the ability of *ADABTics* to ensure optimal accuracy and response time.

VIII. RELATED WORK

Big Data Cyber security Analytics (BDCA): BDCA has been an active area of research and practice. A large number of Hadoop-based BDCA systems (e.g., [35-40]) have been proposed [6]. Of these, some (i.e., [35-37]) are single machine system while others (i.e., [38-40])) support distributed processing. The incorporation of ML algorithms is limited to two to three algorithms in these systems. Except [37], which incorporates three ML algorithms (i.e., NB, SVM, and RF), the rest of the systems incorporate either one ([36, 40]) or two ([35, 38, 39]) algorithms. Furthermore, only one system ([39]) is evaluated with two security datasets (i.e., KDD and CMDC2012) while the rest with only one dataset (i.e., KDD). In contrast, our implemented BDCA system incorporates several ML algorithms (i.e., KM, XGB, NB, SVM, and RF) and evaluated with four security datasets (i.e., KDD, DARPA, CIDDS, and CIDIDS2017) in three execution modes.

BDCA Architecture: Research efforts have been invested in proposing and evaluating BDCA architectures ([8, 30, 41][9, 31]). However, the existing architectures are of a higher abstraction level and do not follow the guidelines of component-based architectural style [13]. Ours is the first effort to propose a component-based architecture for BDCA, which improves system's structure and flexibility by allowing addition and removal of components at runtime.

QoS Optimization: The optimization of quality (e.g., accuracy and response time) is a well-trodden phenomenon explored in different domains such as embedded systems [42], cloud-based systems [43], radar systems [44], and object detection systems [45]. Whilst [45] investigates various architectural options (e.g., feature extractor and image size) where accuracy can be traded for response time, [42-44] present and evaluate approaches (i.e., controlling length of cryptographic keys, distributed simulations and symbiotic feedback loops, and service lever configurations based on tasks respectively) for optimizing the QoS. Unlike prior work, our work is in a different domain (i.e., BDCA) and explores a different approach (i.e., architecture-driven adaptation with stopping conditions) to ensure optimal accuracy and response time.

Architecture-driven Adaptation: Unlike control-based adaptation, there is little understanding and exploration of architecture-driven adaptation in different domains of software-intensive systems [16]. However, some recent efforts underline the increased interest in the application of architecture-driven adaptation in real-world software systems. This is evident from exploring its application in different domains (i.e., robotic systems [18], network monitoring [16], IoT [33], and business transactions [17]). Motivated by such efforts and the utmost need (§II), this work is the first of its kind that presents, applies, and evaluates architecture-driven adaptation in a different but an important domain (i.e., BDCA).

IX. CONCLUSION

The frequent changes in the operating environment of a BDCA system impacts the system's accuracy and response time. In this paper, we first expose the impact of such environmental changes. We then propose *ADABTics* – an architecture-driven adaptation approach that recomposes the architecture of the BDCA system at runtime in accordance with the environmental changes to ensure optimal accuracy and response time. Our evaluation demonstrates the effectiveness of our approach in terms of adaptation accuracy, adaptation stability, and adaptation time. To reduce the adaptation time, we plan to investigate in future if the QoS delivered by each workflow can be calculated at design time and leveraged at runtime. We also plan to evaluate *ADABTics* in an online mode to assess its potential in a real-world deployment. An interesting avenue for future research is to extend the approach to monitor and consider other important quality attributes (e.g., scalability and reliability) in the utility function for the optimization of overall QoS. The reported work can be further refined by assessing the dependencies among components while adding and removing them and analysing the impacts of such dependency checks.

REFERENCES

[1] W. Raghupathi, and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health information science and systems,* vol. 2, no. 1, pp. 3, 2014.

[2] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: from big data to big impact," *MIS quarterly*, 2012.

[3] N. Bessis, and C. Dobre, *Big data and internet of things: a roadmap for smart environments*: Springer, 2014.

[4] A. A. Cárdenas, P. K. Manadhata, and S. Rajan, "Big data analytics for security intelligence. Available at https://goo.gl/wxKqDV," pp. 1-22, 2013.

[5] T. Mahmood, and U. Afzal, "Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools," *Information assurance (ncia)*, 2013.

[6] F. Ullah, and M. A. Babar, "Architectural Tactics for Big Data Cybersecurity Analytic Systems: A Review. Available at https://goo.gl/PsjEPu.," 2018.

[7] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE communications surveys & tutorials*, pp. 303-336, 2014.

[8] P. H. Las-Casas *et al.*, "A Big Data architecture for security data and its application to phishing characterization," 2016.

[9] C. Wheelus, E. Bou-Harb, and X. Zhu, "Towards a big data architecture for facilitating cyber threat intelligence," *NTMS*, 2016.

[10] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, pp. 1690-1700, 2014.

[11] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets," *Conference on privacy, security and trust*, 2005.

[12] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, 2015.

[13] G. T. Heineman, and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, 2001.

[14] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, 2010.

[15] J. Kramer, and J. Magee, "Self-managed systems: an architectural challenge," *Future of Software Engineering*, 2007.

[16] J. Cámara *et al.*, "Evolving an adaptive industrial software system to use architecture-based self-adaptation," *Software Engineering for Adaptive and Self-Managing Systems*, 2013.

[17] R. Asadollahi, "Starmx: A framework for developing self-managing software systems," University of Waterloo, 2009.

[18] G. Edwards *et al.*, "Architecture-driven self-adaptation and self-management in robotics systems," *SEAMS'09*, 2009.

[19] S.-W. Cheng *et al.*, "Software architecture-based adaptation for pervasive systems," *Architecture of Computing Systems*, 2002.

[20] A. Hadoop, "Hadoop. https://goo.gl/GLWG9Q," 2009.

[21] P. Clements *et al.*, "Documenting software architectures: views and beyond," *ICSE*, 2003.

[22] J. Mazel *et al.*, "Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection," *International Journal of Network Management*, 2015.

[23] L. Zeng *et al.*, "QoS-aware middleware for web services composition," *IEEE Transactions on software engineering*, 2004.

[24] C. Lu *et al.*, "Performance specifications and metrics for adaptive real-time systems," *Real-Time Systems Symposium*, 2000.

[25] A. Mahout, "Apache Mahout. https://goo.gl/uUVJER," 2017.

[26] KDD, "KDDcup99 Knowledge discovery in databases DARPA archive. https://goo.gl/Jz2Un6," 1999.

[27] MIT, "DARPA intrusion detection evaluation data set. Available at https://goo.gl/jYBYNe," 1998.

[28] M. Ring *et al.*, "Flow-based benchmark data sets for intrusion detection," *ECCWS*, 2017.

[29] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," *ICISSP*, 2018.

[30] S. Marchal, X. Jiang, and T. Engel, "A big data architecture for large scale security monitoring," *Congress on Big Data*, 2014.

[31] L. Fetjah *et al.*, "Toward a big data architecture for security events analytic," *CSCloud*, 2016.

[32] R. Woolson, "Wilcoxon Signed‐Rank Test," *Wiley encyclopedia of clinical trials*, 2008.

[33] M. J. Van Der Donckt *et al.*, "Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application," *ENASE*, 2018.

[34] V. Nallur, and R. Bahsoon, "A decentralized self-adaptation mechanism for service-based applications in the cloud," *IEEE Transactions on Software Engineering*, 2013.

[35] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing*, no. 9, 2016.

[36] J. Breier, and J. Branišová, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wireless Personal Communications*, 2017.

[37] S. Zhao *et al.*, "Real-time network anomaly detection system using machine learning," *Design of Reliable Networks*, 2015.

[38] Z. Chen *et al.*, "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Research,* vol. 3, pp. 10-23, 2016.

[39] T. Chen *et al.*, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications*, 2014.

[40] J. Xiang *et al.*, "Using extreme learning machine for intrusion detection in a big data environment," *Artificial Intelligent and Security Workshop*, 2014.

[41] W. Yu *et al.*, "A cloud computing based architecture for cyber security situation awareness," *Communications Security*, 2013.

[42] K.-D. Kang, and S. H. Son, "Towards security and qos optimization in real-time embedded systems," *ACM SIGBED Review,* vol. 3, no. 1, pp. 29-34, 2006.

[43] T. Chen, R. Bahsoon, and G. Theodoropoulos, "Dynamic QoS optimization architecture for cloud-based DDDAS," *Procedia Computer Science,* vol. 18, pp. 1881-1890, 2013.

[44] C.-G. Lee, C.-S. Shih, and L. Sha, "Online QoS optimization using service classes in surveillance radar systems," *Real-Time Systems,* vol. 28, no. 1, pp. 5-37, 2004.

[45] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *IEEE CVPR*, 2017.

APPENDIX A

Table V lists the workflows considered in our evaluation – The component are: (C1) Removal of Duplicates (C2) Handling Missing Values (C3) Feature Selection (C4) Signature-based Detection (C5) Anomaly-based Detection (C6) Alert Ranking.

TABLE V. WORKFLOWS

S#	Workflow	S#	Workflow	S#	Workflow
1	C5	11	C2-C3-C5	21	C2-C3-C4-C5
2	C1-C5	12	C2-C4-C5	22	C2-C3-C5-C6
3	C2-C5	13	C2-C5-C6	23	C2-C4-C5-C6
4	C3-C5	14	C3-C4-C5	24	C3-C4-C5-C6
5	C4-C5	15	C3-C5-C6	25	C1-C2-C3-C4-C5
6	C5-C6	16	C4-C5-C6	26	C1-C2-C4-C5-C6
7	C1-C2-C5	17	C1-C2-C3-C5	27	C1-C3-C4-C5-C6
8	C1-C3-C5	18	C1-C2-C4-C5	28	C1-C2-C3-C5-C6
9	C1-C4-C5	19	C1-C3-C4-C5	29	C2-C3-C4-C5-C6
10	C1-C5-C6	20	C1-C4-C5-C6	30	C1-C2-C3-C4-C5-C6