REAL-TIME FRAUD DETECTION IN TELECOMMUNICATION NETWORK USING CALL PATTERN ANALYSIS

Kehelwala Gamaralalage Dasun Chamara Kehelwala

(148223L)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2017

REAL-TIME FRAUD DETECTION IN TELECOMMUNICATION NETWORK USING CALL PATTERN ANALYSIS

Kehelwala Gamaralalage Dasun Chamara Kehelwala

(148223L)

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2017

DECLARATION

Candidate:

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my report, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

……..…… …….

K.G.D.C. Kehelwala Date

Supervisor:

The above candidate has carried out research for the Masters Dissertation under my supervision.

……..…… ..…….

Dr. H.M.N. Dilum Bandara Date

Abstract

Telecommunication service providers are losing considerable percentage of their annual revenue due to fraudulent activities. Such activities also deteriorate customer experience. Therefore, real-time detection of such fraudulent activities is required to minimize the revenue loss and to preserve customer experience. Illegal termination of International calls (aka. SIMbox fraud) and extreme usage scenarios related to International revenue share fraud are two major fraudulent activities which make highest impact. While such activities can be detected by identifying behavioral and calling patterns of subscribers, they need to be detected in real time so that subscriber connections linked with an ongoing fraud activity can be terminated to minimize the impact of threat or revenue loss. Call Detail Records (CDRs) produced by telecommunication equipment contains attributes that are specific to a phone call or other communication transactions handled by the device could be used to detect behavioral and calling patterns of subscribers. However, traditional CDR analysis techniques do not facilitate time-sensitive monitoring and analytical requirements. Therefore, we propose a Complex Event Processing (CEP) based solution for the real-time identification of fraudulent and extreme usage subscriber patterns. We identified a rich set of features and set of call patterns, and then combined batch analytics with real-time analytics to increase the detection accuracy. We demonstrated the utility of the proposed solution using a real dataset from a service provider. The proposed solution achieved an accuracy of 99.9% with average latency of 16 call attempts per detection at input event rate of 230 events per second with modest hardware.

Keywords: Complex Event Processing, Data analytics, Call Detail Records, call patterns

ACKNOWLADGEMENT

My sincere gratitude goes to my family members for the continuous support and motivation given to make this thesis a success. I also express my heartfelt appreciation to Dr. Dilum Bandara, my supervisor, for the supervision, advice and valuable feedback given throughout to make this research a success. I also thank to Mr. Ruchira Yasaratne, Mr. Sampath Ilesinghe and Mr. Pradeep De Almeida of the Dialog Axiata PLC, for providing approvals to proceed this project by keeping trust on me. Last but not least I also thank my friends who supported me in this whole effort.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Networks
ASCII	American Standard Code for Information Interchange
BAM	Business Activity Monitor
BI	Business Intelligence
BSC	Base Station Controller
CDR	Call Detail Record
CEP	Complex Event Processor
CLI	Calling Line Identification
CTR	Click-through Rate
CUP	Current User Profile
DAHP	Database-Active Human-Passive
DAS	Data Analytics Server
DBMS	Database Management System
DDOS	Distributed Denial of Service
DOS	Denial of Service
DSMS	Data Stream Management System
EDGE	Enhanced Data rates for GSM Evolution
FDT	Fraud Detection Tool
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GT	Global Title
HADP	Human-Active Database-Passive
HSPA	High Speed Packet Access
HTTP	Hypertext Transfer Protocol
IDD	International Direct Dialing
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
ISC	International Switching Center
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part

LTE	Long Term Evolution
LKR	Sri Lankan Rupee
MCC	Mobile Country Code
MLP	Multi-Layer Perception
MNC	Mobile Network Code
MO	Mobile Originated
MSC	Mobile Switching Center
MSISDN	Mobile Station - ISDN
MT	Mobile Terminated
NFA	Non-Deterministic Finite Automata
NN	Neural Networks
OCS	Online Charging Node
PABX	Private Automatic Branch Exchange
QoS	Quality of Service
RFID	Radio Frequency Identification
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service
SOM	Self-Organizing Map
SVM	Support Vector Machine
TDM	Time Division Multiplexing
TMSC	Tandem Mobile Switching Center
UPH	User Profile History
UTMS	Universal Mobile Telecommunications System
VLR	Visitor Location Register
VoIP	Voice over Internet Protocol

1. INTRODUCTION

1.1. Background

Telecommunication service providers around the world are losing billions of dollars annually due to fraudulent activities [1]. Percentage revenue loss due to frauds is considerable and incur direct impacts on profitability. The effects of these fraudulent activities are felt by end users as well, as it causes quality of service degradations and unexpected tariffs. Illegal termination of International calls (aka. SIMbox fraud), and extreme usage scenarios related to International revenue share fraud are two of the major fraudulent activities [1]. Traditional fraud detection techniques only focus on detection accuracy and barely focus on detection speed. Therefore, fraudsters can survive for substantial time before being detected. Given the lower cost of acquiring a new connection they move on to a new connection once the existing one is blocked. Hence, detection of these fraudulent activities in near real-time is essential to nullify the impact and effectively control the frauds.

Call Detail Records (CDRs) [2] are one of the most valuable data repositories of a telecom operator. CDR is the data record that contains information related to a single instance of telephone call or other related transactions. Mobile Switching Center (MSC) or related telecommunication nodes create a CDR record when a transaction passed through it. Based on the functionality of the telecommunication nodes, level of information in CDR may vary. In most cases, it contains data such as the origination and destination address of the call, the time the call started and ended, and the duration of the call. CDR stream is the best data source which reflects behavioral and calling patterns of subscribers. While several related works use CDR analysis for fraud detection, they follow traditional database reliant store first, process then approach, Moreover, CDRs collected within a large time window are considered for decision making; thus, lack real-time features and unable to effectively control the fraud [3]-[7]. Furthermore, those solutions not focused on the time sensitive call patterns inside the CDR stream, which comprise of valuable information in fraud detection. Therefore, there is a need for a real-time fraud detection tool that can make detections rapidly by effectively utilizing time sensitive call patterns within the CDR stream.

1.2. Motivation

Detecting a fraud after the event has occurred is not nearly as useful as catching it in real time. While CDR is generated in real time, immediately after related transaction is completed, most fraud detection tools use CDR for post analysis. Some of the sophisticated tools available in industry [8] use signaling information in addition to CDRs to make detections in real-time. However, capturing such signaling information incur additional cost as supplementary probing devices are introduced to the network. Also, these probing devices may introduce additional point of failure to the network. Therefore, relying on CDRs is most cost effective and reliable method.

CDR contains all the required data to make near real-time fraud detections. But most of the business analytics tools store this data on static storage and perform batch operations on past data to calculate aggregate values such as sums and averages. Even though such analysis gives useful insight about the past behavior, it is not sufficient in current business world, as it is not capable of exploiting the timeliness value of data and not captures time sensitive call patterns inside the CDR stream. For example, following are two use cases where real-time CDR analytic could be useful.

1.2.1. Grey call detection

Illegal IDD Termination (aka. Grey Calls or Bypass Calls) is a major source of revenue loss for a telecommunication provider, which also deteriorates Quality of Service (QoS) offered to customers [1], [9], [10]. This scenario happens when fraudsters accumulate an incoming international call through the Internet via Voice over Internet Protocol (VoIP), and then inject it back to the destination country's telecom network as a local call using local SIM (Subscriber Identity Module) cards installed in a device called SIMbox. Therefore, the call reaches the called party with a local CLI (Calling Line Identity) and billed as a local call. When the price difference between the international call termination and local call is high, fraudsters can easily gain revenue that exceeds the break-even point with few fraudulent calls. Therefore, detection and termination of such fraudulent phone numbers with minimum delay allows to regain significant portion of revenue and effectively control this kind of fraud.

1.2.2. Extreme usage detection

Under extreme usage detection, we consider the scenarios in which subscribers abnormally originate calls or subscribers abnormally receive calls compared to the normal behavior. These could be due to fraudulent activity or some other incident. Variants of International revenue share fraud and Premium rate service fraud are best examples for this type of fraud. International revenue share fraud and Premium rate service fraud are similar in nature and in either cases fraudsters try to pump up traffic to certain numbers or number ranges belonging to premium rated or high cost destinations. Usually operators need to pay more than LKR 50 per minute to terminate calls to those premium rated or high cost destinations. Once fraudsters have inflated traffic to agreed number levels, they can claim some percentage of the termination revenue from the destination party.

Fraudsters are using various methods to persuade customers to originate calls to these premium rated number levels. *Dial and disconnect scam* is the most frequent method in which fraudsters multicast missed calls to a subset of customers with premium rate number as CLI. After that, some of the customers dial back to that premium rate number as response to missed call and get charged. In some cases, fraudsters multicast fake text messages using SMS (Short Message Service) or messaging facility of over-the-top media service providers like Viber or WhatsApp. Fraudsters include premium number either as message sender's CLI or as a content inside fancy message. Some of the customers dial back those premium rated numbers. Sometimes fraudsters gain unauthorized access to PABX (Private Automatic Branch Exchange) systems belongs to cooperate customers and originate calls premium rated numbers. This is known as PABX hacking fraud. Sometimes fraudsters inject malware to mobile handsets of customers and originate calls to these premium rated number ranges. This scenario is known as *malware originated fraudulent calls*. Sometimes fraudsters register in network as inbound roamer and generate calls to premium rated numbers. This is known as *inbound roamer fraud*. These fraudulent activities reflect calling patterns unique to each scenario in CDR stream.

Situation awareness is important to take timely actions to exploit maximum advantages from opportunities and control the damages in threats. In addition to fraudulent call patterns, CDR contains many other important patterns which require timely attention. As an example, when a popular TV program advertises the hotline number, subscribers try to dial that number and a huge number of attempts reaches the network at an unusual arrival rate. That may cause congestion in network devices and consequently degrades the QoS. Real-time detection is required to take timely actions and minimize the impact. Additionally, some attackers try to make DoS (Denial of Service) and DDoS (Distributed DoS) attacks targeting the voice networks. In this case attackers make large number of call attempts to a set of numbers in the destination network. If the operator can detect it, they can react immediately and drop such calls at network edge without processing.

Extreme usage patterns can be easily identified by analyzing the CDR steam. But most important factor is that those events are time sensitive. Most of the time these cases are identified at root cause analysis. Use of traditional database systems are inefficient in this case, as it takes much time to upload CDRs into a database and perform queries on a huge database. Therefore, it is important to be able to identify these extreme user behaviors by analyzing CDR streams in real time.

1.3. Problem Statement

Traditional systems make detections by generating a set of features or aggregate values by querying static data over a large time window and make decisions based on such values. This is time consuming and by that time a fraudster can make number of successful calls before being detected. Scenarios discussed above highlight the need of a real-time CDR analysis tool, which can snoop CDR streams and generate a recent or real-time view of the telecommunication network. Also, resultant recent view should be able to integrate with past data and produce complete state of the network at a given instance with minimum latency. System should be easily customizable according to varying requirements of the operators. Lower development cost will be an extra advantage. Such a system allows operators to gain maximum advantage by

exploiting timeliness of detected events and save significant amount of revenue by minimizing fraudulent activities.

The solutions available in research literature lack real-time features due to unsuitability of traditional database reliant *store first process then* approach for latency sensitive applications, large time windows for feature generation, shallow feature set, less awareness about context, and inability to detect complex patterns in CDR. Commercial systems are also based on databases, use a proprietary feature set, focuses on specific use case, and are usually expensive. Therefore, operators are unable to afford the cost of such specialized systems.

Therefore, the problem to be addressed by this research can be stated as follows:

How to detect fraudulent call patterns in real-time using CDR?

Our primary focus is to use the power of complex events to support real-time decision making in detection of grey callers and fraudulent activities which involve extreme usage.

1.4. Objectives

Following list of objectives are to be achieved to address the above research problem:

- To research about each use case and its context details and to identify complex call patterns in CDR streams.
- Identify suitable features to support decision making using a short-time window.
- Develop algorithms, CEP queries and batch analytic queries to detect selected use cases.
- Design optimal system architecture to perform fraud detections using real-time and batch analytics together and implement the system using best suited software packages.
- To conduct a comprehensive performance study using real data sets to demonstrate the utility of the proposed solution.

1.5. Outline

The remainder of this report is organized as follows. Chapter 2 describes the theoretical background related to CDR, Grey Calls, and related work. Technologies like Complex Event Processing (CEP) and Business Activity Monitoring (BAM) are also discussed in detail. Proposed design is presented in Chapter 3. Chapter 4 presents about experimental setup and the results obtained by passing real-world dataset through the proposed system. Finally, Chapter 5 summarizes the research findings, and present limitations and future work of this research.

2. LITERATURE REVIEW

This chapter discusses the details about major use cases and presents survey about the possible technologies to address given problem statement. Section 2.1 explains about CDR. Detailed description about Grey calls is presented in Section 2.2. Section 2.3 discusses about fraudulent cases related with extreme usage scenario. Works related to CDR based fraud and abnormal usage detection techniques are presented in Section 2.4. Section 2.5 discusses about complex event patterns comprised in CDR. Section 2.6 discusses about related data processing technologies. Section 2.7 discusses about methods of accessing stored data by CEP engine to execute queries combined with data streams. Section 2.8 presents how both the real time and archived data can be combined. Summary of key findings of literature survey is presented in Section 2.9.

2.1. Call Detail Records (CDR)

Call Detail Record (CDR) [2] is the data record generated by telecommunication equipment. It includes the attributes that are specific to a single instance of a phone call or other communication transaction which is handled by that device. The attributes and level of information inside the CDR vary depending on functionality of the device. The metadata files or instructions manuals of the device typically specify how to extract the information in CDR. CDRs are inherently used for billing purposes. In addition, it is used for troubleshooting, measuring Quality of Service (QoS), fraud detection, gaining Business Intelligence (BI) and forensic investigations.

The most common attributes of a CDR generated for a voice call are listed in Table 2.1. In addition to these attributes, CDR of Global System for Mobile (GSM) telephone call contains additional attributes that represent the information about mobile handset and its location (see Table 2.2). CDRs with these attributes are generated at Class-5 switches to which subscribers are directly connected. Handset and location related attributes of originating and receiving parties of a telephone call are recorded in two separate CDRs at Class-5 switches to which calling and called party subscribers are attached. The telecom switches to which only other switches are connected are known as Class-4 switches. Table 2.3 lists the additional attributes recorded at Class-4 switches in addition to basic attributes mentioned in Table 2.1. These details are

helpful to distinguish telecom operators in signaling interconnection. Tables 2.1 to 2.3 list only the essential attributes of a voice call that are helpful in this research. Many other attributes that represent the QoS parameters, protocol-specific details and device-specific details are included in CDR. Short Message Service (SMS) and Mobile Data (GPRS, EDGE, UTMS, HSPA and LTE) technology transactions also generate CDRs.

Table 2:1: Common attributes in CDR.

Attribute	Description
Origination Date & Time	Date and time when call reached to the system
Calling Party ID (A Number)	Subscriber Identity Number of user who originates the call
Called Party ID (B Number)	Subscriber Identity Number of user intended to receive call
Answer Date & Time	Answered date and time (Only if call answered by B party)
Disconnect Date & Time	Disconnected date and time of call answered call
Release Date &Time	Date and time call released by system
Disconnected Party	Which party has released the call
Call Duration	Duration between answer time and disconnect time
Release Cause	Code that represents the reason for call release
Type of Call	Can be Local, International or National • Local – A and B parties within same telephone operator • National – B party is in different telephone operator but within same country • International – B party is in different telephone operator in different country

Table 2:2: Specific attributes in CDRs generated at Class-5 switches.

Attribute	Description
IMSI – International Mobile Subscriber Identity	Unique number that represents the Subscriber Identity Module (SIM) card
IMEI – International Mobile Equipment Identity	Unique number that identifies particular GSM-enabled device
MCC – Mobile Country Code	Represents the Country to which mobile subscriber belongs to
MNC – Mobile Network Code	Represents the Operator to which mobile subscriber belongs to
LAC – Location Area Code	Represents the BSC (Base Station Controller) to which mobile subscriber is attached to
Cell ID	Represents the sector of a BTS (Base Transceiver Station) to which mobile subscriber is attached to
MSC GT – Global Title Number of Mobile Switching Center	Represents the VLR (Virtual Home Register) and Mobile Switching Center to which subscriber attached to

Table 2:3: Specific attributes in CDRs generated at Class-4 switches.

Attribute	Description
Origination Switch Details	• Origination Point-Code when using Time Division Multiplexing (TDM) technology (e.g., Using ISUP signaling protocol with SS7 stack) • Origination Gateway IP when using Voice over IP (VoIP) technology (e.g., SIP signaling protocol)
Destination Switch Details	• Origination Point-Code when using Time Division Multiplexing (TDM) technology (e.g., Using ISUP Signaling Protocol with SS7 stack) • Origination Gateway IP when using Voice over IP (VoIP) technology (e.g., SIP Signaling Protocol)

Traditional BI techniques and fraud detection techniques only focus about aggregate values like averages, summations, and counts derived using CDRs. However, CDRs contain interesting patterns that reflect customer behavior. Identification of such patterns allows to gain economic advantages by recognizing opportunities, as well as mitigate risks by unmasking threats. Rapid growth of Data Science and Machine Learning fields has enriched the CDR based pattern recognition. Fraud detection is one of the major applications of CDR-based pattern recognition. Section 2.4 focuses on literature that applies CDR-based fraud detection and pattern recognition techniques.

2.2. Grey Calls

Grey call fraud, SIMbox fraud, or Bypass fraud is one of the major source of revenue loss in Telco industry [1]. Its impact is more severe in certain parts of the world, especially in Asia, Africa and North America [9], [10]. This fraud is taken place when the rate of international termination call is considerably higher than rate for local incoming call in that country. Therefore, Fraudsters takeover those international calls and transfer it through Internet to the destination country. Then Voice over IP (VoIP) calls are injected to back to destination network via SIM (Subscriber Identity Module) cards which are installed on device called SIMbox. Because this activity bypasses the legal international interconnections between Telco operators, international calls are billed as national calls and significant revenue leakage is occurred. SIMbox is the device which is capable of converting VoIP calls to GSM network call. With expansion of technology high capacity SIMboxes with advanced technologies like International

Mobile Equipment Identity (IMEI) swapping and intelligent SIM swapping algorithms make the SIMbox behavior closer to actual subscriber [11].

In addition to the revenue loss SIMbox fraud causes QoS loss for customers, as these calls may reach the SIMbox through a low quality, low bandwidth IP routes. Also, a SIMbox may use 100s of SIMs simultaneously, which may overload the base stations in that area. That may cause sudden call drops and loss of QoS observed by the destination customer for IDD calls, as well as a degraded service is offered to actual customers in that area. Since SIMbox spoofs actual international Calling Line Identification (CLI) with a local CLI, SIMbox fraud may lead to privacy issues. So SIMbox fraud may severely affect customer satisfaction and operator's brand name.

We can divide Bypass fraud into two major categories as Onnet bypass and Offnet Bypass. Onnet bypass means fraudsters use the SIM cards of same network of destination number. This is the common case in most of the countries as calls within same network cost much lower than charges for calls to other operators or international calls. Figure 2.1 depicts the Onnet scenario. Instead of using the costly high-quality routes that goes through destination networks International Switching Center (ISC), some wholesale carriers route calls through SIMbox operators connected though the Internet. SIMbox dials that calls via a SIM cards of the same network of the destination number.

In Offnet bypass, fraudsters use the SIM card of a different operator in same country of destination number. Figure 2.2 depicts this scenario. This happens when local interconnection charges are much lower than international interconnection charges. In some countries local interconnection charges are much closer to the international interconnection charges. Therefore, Offnet bypass does not take place in such countries.

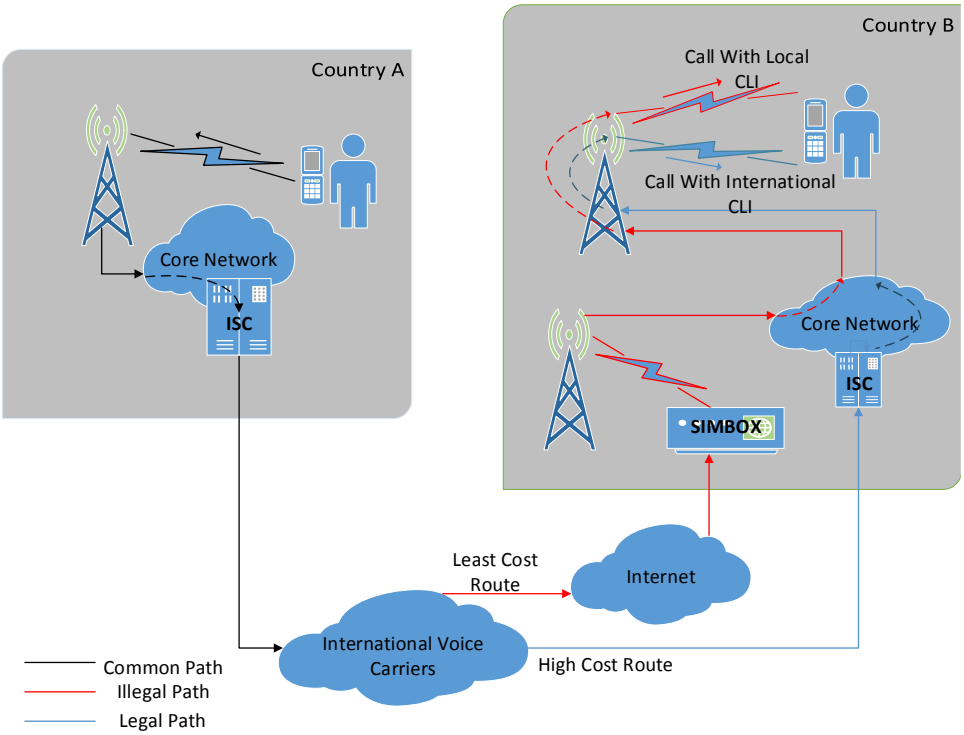

Figure 2:1: Onnet bypass.

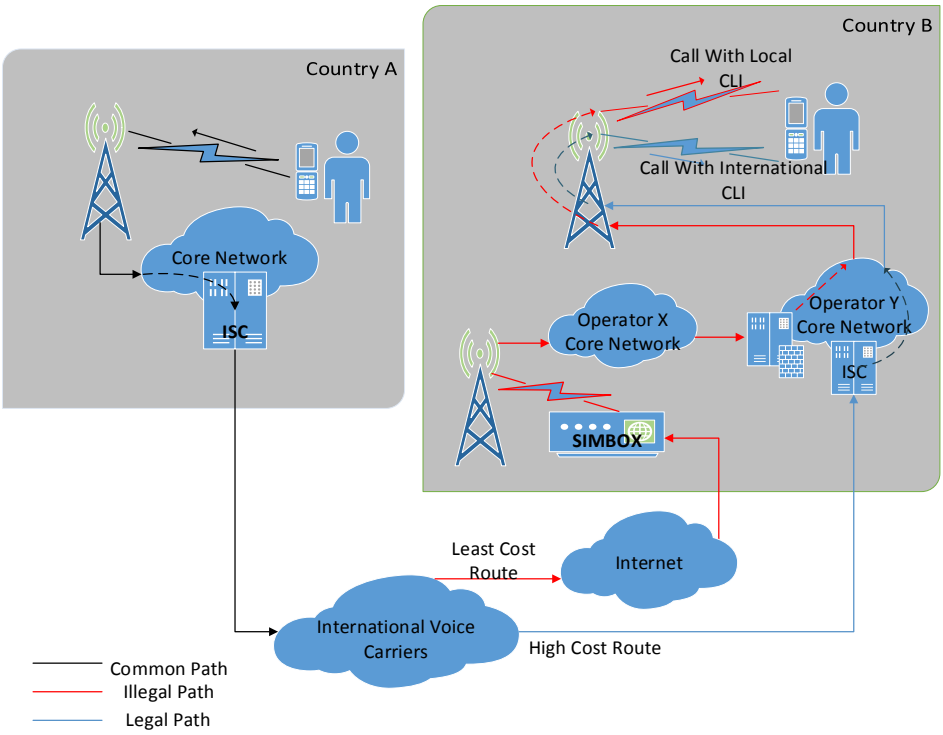

Figure 2:2: Offnet bypass.

Detection of SIM cards used for SIMbox fraud is a challenging task for mobile operators. There are two major approaches. First approach is actively originating calls

to the target network via test units installed in several parts of the world and scan the CLI of those calls. Even though this process detects SIMbox numbers in real time, it is not capable of capturing majority of numbers. Second approach is CDR based analysis. CDRs are loaded into relational database and queries are used to identify fraudulent numbers. Rich set of attributes are needed to derive by summarizing CDRs to achieve effective detection. In context of Onnet bypass, operator has more details including location and owner information. But detection of Offnet bypass has to be performed with limited details and strong pattern mining techniques are required.

Efficient detection process must consist minimum false positive (i.e., detect genuine customers as fraudulent) and false negative (i.e., classify fraudulent numbers as genuine) values. Moreover, number of attempts made by fraudulent SIM card before detection is another important factor. If this value is very high, fraudster can cover the cost before disconnection of SIM card. Because the Telecom industry is highly competitive, in most of the countries fraudsters can buy new SIM cards at a very little cost. Hence, traditional analysis methods fail here as those are based on past CDR analysis. Therefore, when operator has disconnected fraudulent SIM cards, fraudsters use new set of SIM cards as they can cover profit margin. This process continues, and actual task of detection process becomes damage control function.

2.3. Extreme Usage Scenarios

In regular business, Telco operators encounter many scenarios in which normal subscribers show abnormal usage behaviors or excessive usage of network resources, and then reject to pay bills. Subscribers show such kind of behaviors unintentionally or as a result of their reaction to some provocations triggered by fraudsters due to lack of awareness. Fraudsters directly involve overusing network resources only in few cases like inbound roamer fraud.

In majority of extreme usage scenarios fraudsters use premium rate telephone numbers. Some countries have high international incoming call charges to all number ranges belongs to country code or certain number levels in their numbering plan. When the caller in different country has dialed those premium rated numbers, caller's service provider has to charge extremely higher value than standard IDD call charges in order

to cover call termination cost to premium rated number range. Fraudsters misuse those telephone numbers to earn profits through fraudulent activities. Operators publish those premium rated number ranges on their websites to make customers aware about dialing cost to those countries or telephone number ranges. The premium rated numbers used for fraudulent activities are automatically answered and sometimes fraudsters redirect those to fancy recordings to elongate conversation time and make even more profits.

Dial and disconnect scam is the first category of fraudulent activity related to premium rate telephone numbers. In this scenario, fraudsters connected to international voice network multicast call attempts to range of valid telephone numbers in selected network in selected country. Normally those calls are disconnected by fraudster after few rings to make sure customers do not answer those calls. As international voice carriers do not charge for zero duration calls, fraudsters do not need to bear any charges to populate these missed calls. Intention of these missed calls is to arouse natural curiosity of recipient party and persuade them to dial back. When multicasting those missed call attempts fraudsters replace original calling party with premium rated telephone number or telephone number of country to which call termination rate is higher than LKR 50. Sometimes fraudsters wait till called party subscriber answer those call and make some groaning sounds to persuade customers more. Customers who are less aware about this fraudulent activity try to dial back these numbers and get charged. This fraudulent activity is also called *one ring scam* or *ring and run scam*.

Outbound dialing due to fake text messages is the second most frequent fraud scenario related to premium rated numbers. Instead of missed calls, fraudsters use fancy messages to persuade normal subscribers to dial back to premium rated numbers. Those text messages are populated using SMS (Short Message Service) or messaging facility of over-the-top media service providers like Viber or WhatsApp. They include premium number either as message sender's CLI or as a content inside fancy message. Some of the customers who lack awareness about these kinds of fraudulent activities dial back these premium rated numbers mentioned in these messages and get charged.

Another possible extreme usage related fraud scenario is *inbound roamer fraud*. Using international or national roaming facility provided by mobile network operators, subscribers belong to other operator's mobile network (foreign or local) can latch to visitor network and use the services provided by host operator. Those subscribers are defined as inbound roamers. According to roaming agreements between two operators, home network operator of roamer needs to pay usage charges to visitor network operator on behalf of their customer. Visitor network operator needs to transfer billing related details to home network operator at earliest. Sometimes fraudsters use this roaming facility for their fraudulent activities. Fraudsters latch guest (foreign or local) SIM card on visitor network and originate calls to premium rated or high cost destinations. Foreign or local operator who is the home network operator of fraudster's SIM card receives usage details once visitor network operator has transferred billing details. If there is any delay in billing file transfer, fraudsters can exploit those delays. Even though foreign operator needs to pay those charges to visitor network operator according to agreements, visitor network operator needs to track those fraudulent incidents and inform foreign operator as there is some associated risk in termination of roaming agreements and not paying excessive charges when usage charges due to fraudulent activity is very high.

In *PABX hacking fraud*, fraudsters acquire the control of PABX (Private Automatic Branch Exchange) system by exploiting vulnerability of PABX system [7] or its connected network. Then fraudsters make large number of calls to premium rated or high cost destination without any intention of actual customer if PABX system has IDD facility. Customer is billed for these calls and more often than not they refuse to pay the bill as calls were originated without their intention. So operator needs to bear the cost. This scenario is encountered only when telecom operator provides trunking solutions to enterprise customers who operate PABX systems. Even though this fraud scenario is less frequent, its impact is high as fraudsters can originate many simultaneous calls up to configured channel capacity of trunk between central office and customer's PABX.

Similar kind of a scenario can happen with malwares installed in smart phones. Fraudsters setup set of premium rated numbers and distribute some malwares that can

remotely or automatically triggered to dial that numbers. When mobile handset is inflected with such malware, it frequently originates long-duration calls to those premium rated numbers and customer get charged. Because these calls are dialed without customer intention customer often refuses to pay the bill and operator has to bear the cost to retain the customer. This scenario is known as *malware originated fraudulent calls*.

All these five cases are well organized frauds and related to premium rate international numbers. This is because fraudsters try to make profits from those activities. Destination party or third-party between premium destination operators and fraudsters advertises these numbers with revenue share model. Fraudsters make agreements with those relevant parties and originate calls into premium rated numbers in fraudulent ways. In all these five cases fraudsters take some percentage of money earned by premium destination operator. *International revenue share fraud* is an umbrella name used to define these five types of fraudulent scenarios. Sometimes these fraud scenarios are defined as *premium rated service fraud*. International revenue share fraud is one of the major frauds and telecommunication service providers around the world are losing billions of dollars [1] due to this. Therefore, under extreme usage detection we will focus on these five fraud scenarios belong to international revenue share fraud.

2.4. CDR-Based Detection Techniques

2.4.1. Grey call detection techniques

Elmi et al. [4] proposed an Artificial Neural Network (ANN) method to address SIMbox fraud. They have used supervised learning method with Multi-Layer Perception (MLP) as classifier. ANN was used because of its generalizing capabilities, ability to learn complex patterns and trends within noisy data and better performance records in this domain. This system comprises nine features derived using CDRs in dataset and corresponding values of those features were calculated for each calling subscriber. Table 2.4 describes the feature set which was used for SIMbox detection.

Table 2:4: Feature set used in ANN based approach [4].

Field Name	Description
Call sub	This is the subscriber identity module (SIM) number which was used as the identity field
Total calls	This feature is derived from counting the total calls made by each subscriber on a single day
Total numbers called	This feature is the total different unique subscribers called by the customer (subscriber) on a single day
Total minutes	Total duration of all calls made by the subscriber in minutes on a single day
Total night calls	The total calls made by the subscriber during the midnight (12:00 to 5:00 am) on a single day
Total numbers called at night	The total different unique subscribers called during the midnight (12:00 to 5:00 am) on a single day
Total minutes at night	The total duration of all calls made by the subscriber in minutes at midnight (12:00 to 5:00 am)
Total incoming	Total number of calls received by the subscriber on a single day
Called numbers to total calls ratio	This is the ratio of the total numbers called/total calls
Average minutes	The is the average call duration of each subscriber

In Multi-Layer Perception the ANN consists of multiple layers of computational units (neurons), connected in feed forward way. So these neurons can be categorized into input, output and hidden neurons based on layer. Connections between neurons are known as edges and which has associated weights. Neurons are only connected to subsequent layers but not to the neurons in same layer. Weighted sum of multiple inputs is taken and it is fed into nonlinear activation function called *Sigmoid function* to generate single output of neuron. This output value is passed as input to the connected nodes in next layer. Back Propagation algorithm is used to train the ANN to minimize the training errors. This algorithm calculates error value for each neuron output (difference between output of neuron and actual value) and weights of network edges are continuously adjusted in a way that minimize errors.

The dataset was divided to ten subsets and average error value was calculated by running experiment for each subset in turn using same model. While using one subset for testing remaining nine was combined and used for training. This is known as 10-fold cross validation. Authors has changed four parameters to find optimum ANN with

highest accuracy. Number of hidden layers, number of neurons per hidden layer, learning rate and momentum are those parameters. Learning rate represents the speed at which the ANN arrives at the global minimum value for Sum Square Error (SSE). The momentum parameter represents the rate at which the ANN approaches neighborhood of optimality at early stages of algorithm. Both momentum and learning rate range its values between zero and one. Altogether they have experimented 240 neural networks and compared them in terms of prediction accuracy, generalization error, time taken to build the model, precision, and recall. So they have unmasked the optimum ANN. They have identified that very high learning rates and momentum rate significantly degrade classification accuracy as it leads algorithm to overshoot the optimal configuration. They have obtained maximum accuracy of 98.7% by using lower momentum value (0.3) and moderately higher Learning rate (0.6) and using two hidden layers. Learning and classification performed in about 17 seconds with considerably lower false positive and false negative detections.

Two years later the same authors came up with a Support Vector Machine (SVM) based approach [5] for the same dataset and compared its results with the ANN based approach. They have used 10-fold cross validation technique while using same features in [4].

Because this is a binary classification problem there are only two classes in the training data, in this case hyperplane is a line. But authors had to use nonlinear SVM as nonlinear curved line was required to separate boundaries. So they have used kernel functions instead of inner product and evaluated performance separately for polynomial kernels, radial basis function kernels, and linear kernels. Additionally, they have taken measurements by changing the C penalty parameter which effectively controls amount of error willing to afford in the training data. Altogether they have evaluated 40 SVM models in terms of accuracy, generalization error, and time taken to build the model, precision and recall using 10-fold cross validation method. Moreover, they have evaluated performance of both methods by changing training and testing set sizes. Authors achieved above 98.5% accuracy in both ANN and SVM based approaches. Finlay, they have located best SVM model and found that it performs better than ANN because of significant reduction in running time. Even

though authors were able to achieve high accuracies and lower running time in both the cases, sustainability of this approach in practical scenario is questionable due to many reasons. When we consider the dataset, its size is much smaller than typical mobile network operator. Dataset contains CDRs of 234,324 calls made by 6,415 subscribers over two months. However, most of mobile operators, especially in Asian countries, have more than 5 million subscribers and generate more than 20 million CDRs per day [12], [13]. Also, they have considered CDRs from one cell id only and ratio between legitimate to fraudulent subscribers is approximately 2:1. But in real cases more than 20,000 cell IDs [14] need to be considered and percentage of SIMbox numbers out of total customer base is very low. So we can conclude that cardinality of dataset is inferior to actual cases. Therefore, this solution's ability to achieve given performances in practical environment is not tested in [5]. Also, they have considered CDR for two months when calculating features. But the actual requirement is SIMbox detection need to perform as early as possible. Fraudsters can cover the cost of buying new SIMs, if they successfully operate over a few hours. So there is no point of performing calculations within a few seconds as long time window is used for feature calculation. Additionally, scalability of these methods with large datasets were not evaluated in both the papers [4], [5].

Murynets et al. [6] have presented a novel classifier for fraudulent SIMbox detection. Comprehensive analysis about SIMbox fraud and its consequences was performed in first part of this paper. As we are focusing on operation of SIMbox fraud and its consequences in previous chapter only important observations are highlighted in this chapter. Table 2.5 shows the attributes of the CDR used by authors, where they have accounted important details including location details, device details and corresponding customer segment of calling party which were absent in the previous cases [4], [5]. Those parameters are directly used as features as described below. Using CDR with a rich set of attributes can be identified as a positive step. By considering IMEI details it is possible to block the confirmed IMEIs or detect new SIM cards that are inserted into a SIMbox with a particular IMEI. But the detection logic cannot too much depend on that since advanced SIMboxes allow changing IMEIs [11].

Table 2:5: CDR fields considered in Classification based approach [6].

CDR Field	Description
Time	Date and time of a call
Duration	Call duration
Originating number phone	Number of a caller
Originating country code	Country of a caller
Terminating number	Phone number of a called party
Terminating country code	Country of a called party
Call type	Mobile originated/terminated call
IMEI	International Mobile Equipment Identity (device identifier)
IMSI	International Mobile Subscriber Identity (user identifier)
LAC-CID	Location area code and cell ID (base station location identifier)
Account age	Time since account activation
Customer segment	Prepaid/postpaid/corporate account

Authors have derived 48 features using mentioned CDR attributes in Table 2.5. It is important to note that the feature set is per IMEI basis and they have targeted to identify SIMbox rather than SIM cards used for IMEI. Even though authors did not give full description about whole feature set, features mentioned in Table 2.6 were highlighted. Based on these features, authors have characterized SIMbox behavior. Authors have demonstrated that SIMboxes have fairly static physical behavior as they connect to a very small number of nearby base stations while a genuine customer is dynamic and moves across many base stations. This is obvious but important observation which were not presented in the previous cases [4], [5]. LAC-CID attribute makes this possible. Because advanced SIMboxes are capable of swapping locations, location information need to be used with care.

Authors have demonstrated SIMboxes have very few Mobile Terminated (MT) calls and generate a huge number of Mobile Originated (MO) calls while genuine customers have same number of initiated and received calls. So usefulness of outgoing calls to incoming calls ratio feature can be highlighted. Also, authors have presented that SIMboxes have very small duration of MT calls over the time than actual customer. Since SIMbox is a machine it cannot lively answer the MT call and maintain a conversation. So it just drops the call or forwards the call to announcement. That is the

reason for this observation. Another observation is SIMbox operators regularly deploy a set of new SIM cards once operator has detected and deactivated existing fraudulent SIM cards. They also tried to filter out the device called Network Probe which is used for quality measurements.

Table 2:6: Feature set used in Classification based approach [6].

Feature Description	Importance
Average Mobile Originated (MO) call duration	Can Compare the ratio between these values which varies significantly for SIMbox and genuine customer.
Average Mobile Terminated (MT) call duration	
Account Age	Allows to identify long stay genuine customers while giving idea about SIM Card replacing activities of fraudsters when operator blocked the detected IMSIs
Customer segment	Pre-paid accounts are more likely to use in SIMbox as those accounts are easier to buy without much authentication. So can assign weights based customer segment.
Total number of Outgoing calls	Grouped according to their corresponding destinations and origins (international and domestic) and counted based on MO and MT time stamps. Further grouped based on originating and terminating country codes. Used to calculate other useful attributes including ratios between these values.
Total number of Incoming calls	
IMSIs operated for IMEI	SIMboxes typically use multiple SIMs
Geo-Location	Allows to compare physical movement of SIMbox vs Mobile Handset of genuine customer.
Ratio of the number of destinations to the total number of calls	Allows to check whether A Party dials many distinct locations abnormally
Ratio of international calls to the total number of calls	Allows to check whether A Party dials international calls regularly

Before we look into classification algorithm it is important to highlight several concerns in feature generation. Per IMEI basis feature calculation has its own set of problems. Advanced SIMboxes can replace IMEIs with dummy values or other genuine IMEIs. So blocking IMEI numbers may block some genuine customers. Additionally, IMEI to MSISDN mapping may give false values. Since choice of device is customer's right, operator has no control on IMEI. So applicability of this system directly in practical environment can be questioned. Better option is feature calculation per MSISDN basis.

Mobile operators disconnect SIMbox connections once they have detected it. So fraudsters insert many new SIM cards to SIMbox frequently. Also, SIMboxes do not move the location on regular basis and attached to limited set of cell IDs. So, there is a high probability that calls originating from those cell IDs to be grey calls. Location details give sense about that. But researchers have not mentioned that they have identified such cell IDs and not presented cell ID wise SIMbox distribution.

When we consider dataset, majority of the features were calculated for data collected over one week period from tier-1 cellular operator in United States. So dataset is considerably larger than the previous cases. But one week period is still higher as operator loses considerable amount of revenue over that period. This dataset contained CDRs of 93,500 subscriber accounts and 500 (or 0.5%) out that were SIMboxes. Since SIMbox user CDRs are mixed inside considerable amount of genuine user's data, this dataset can be considered as good mixing of SIMbox users and Normal user's data than [4],[5]. Features like IMSIs operated per IMEI was calculated for data collected over five months. 66% of labeled accounts were used as the training set while remaining 34% were used for testing. Like in previous cases, cross validation techniques were not used to increase the accuracy.

Classification algorithm which was used in this research is a linear combination of three classifiers associated with weight coefficients. Alternating decision tree, functional tree and random forest are the three classifiers. An alternating decision tree is derived by the combination of single question decision trees which has two types of nodes known as decision nodes and predictor nodes. Decision node contains feature test condition while predicate node has single real number corresponding to negative or positive weight. Root and leaves are predictor nodes and decision node lies between two predictor nodes. So input records are passed through multiple paths and output value is produced based on sign of the weighted sum. Based on training data, Boosting method continuously re-weights the values in predicate node. So ultimate function of boosting method is combining week classifiers into strong classifiers while focusing on majority as well as outliers in the training dataset. In Random forest, multiple decision trees are generated using subset of features and prediction output is generated based majority rule. Functional tree makes decision tests for combination of the

original features at decision nodes, leaf nodes, or both nodes and leaves unlike in standard decision tree in which decision test is done for single feature at decision nodes.

The predictions made by Random forest algorithm provided best false positive rate of 0.0001 while offering comparably higher false negative rate of 0.16. Functional tree algorithm had done predictions with lowest false negative rate of 0.07 but false positive rate was 0.0007 which is comparably higher than value obtained for Random forest. Therefore, to increase the accuracy, multiple regression technique was used. Multiple regression considers prediction output of three classifiers as predictor variables and its linear combination as criterion variable. Prediction error is defined as difference between predicted data label and actual data label. Regression weight coefficients were calculated by locating least value of square error for training dataset. Finally, they have unmasked optimum value for three weight coefficients and classified test data using novel classifier. They were able to minimize the false positive rate up to 0.0001 and false negative rate up to 0.09 and achieve 99.95% accuracy which was higher than previous cases [4], [5]. To enhance practical usage, they have filtered out accounts with less than 10 IMSIs per IMEI, probing devices and well known legitimate accounts and remaining 0.02% of accounts was used for feature generation.

Even though authors gained high accuracy they have not mentioned running time of algorithm and processing requirements. To reduce computational resources, they simply used manual filtering which reduces the size of dataset. But manual filtering is not always possible. Therefore, scalability and running time of this method can be questioned.

Critical evaluation of above approaches reveals many areas that were not focused and thus opens up new research topics. Those facts are summarized below:

- Existing solutions have only targeted the accuracy and running time of classification algorithm while considering large time window for feature calculation. Those solutions did not interpret SIMbox detection as time sensitive operation. So these solutions are incapable of preventing financial losses as fraudsters can make profits easily by operating safely within that time

window before disconnection. Therefore, to make near real-time detections rich set of features should be generated for short-time window and classification algorithm need to be optimized according to that.

- These approaches are only capable of detecting Onnet Bypass and did not pay any attention for Offnet Bypass. Both make similar kind of financial losses for many telecom operators. Features like Location, IMEI, IMSI, and Account type details may not be available for Offnet SIMbox numbers. So a rich set of novel features with additional measures is required to detect Offnet Bypass.
- Features are generated based on calling party behavior only. But by considering called party behavior a valuable set of attributes can be derived. For example, counting the subset of called party numbers which has received IDD calls and belongs to a set of called party numbers dialed by the considered calling party will be valuable feature in context of grey call detection.
- Previous cases have targeted CDR data stored in static databases. But CDRs records are generated in real time and flows as streams of data. Therefore, to gain maximum advantage, a new mechanism that is capable of directly processing the multiple streams is required. Also, that mechanism should support multiple CDR streams generated by Telco nodes, as well as some static data simultaneously.
- A typical mobile service providers have a customer base of more than ten million. So data streams with very high transaction rate are generated at Telco nodes. So highly scalable and fast feature generation method is required to cope with current industry requirements.
- Any of the discussed methods are not capable of identifying complex events masked inside CDR. Detection of complex events allow to exploit maximum situational value. This can be effectively used for SIMbox fraud detection.

2.4.2. Extreme usage detection techniques

Grosser et al. [3] proposed a fraud detection method using ANN. They tried to replace traditional method of fraud detection that is only capable of detecting extreme fraudulent activities. Traditional method is defined as *absolute CDR analysis*, which

generates alerts when calculated summative values of CDR attributes meet the fixed criteria known as Triggers. They suggested the method known as *differential CDR analysis*, which tracks the pattern of behavior of mobile subscriber by comparing the most recent call transactions with historical call transactions and notifies if change of pattern is observed. Two types of user profiles known as Current User Profile (CUP) and User Profile History (UPH) were maintained for each subscriber.

Encoded CDRs which contain IMSI, originate date and time, duration and call type as attributes were used in this analysis. Researchers have grouped incoming CDR entries into three groups based on type of call as local, national, or international. Then they have created three neural networks to recognize patterns in these groups. Call time and its duration were used as inputs to the neural network that determines the pattern to which specific call resembles. ANN type known as Self-Organizing Map (SOM) network was used to generate patterns. This is an unsupervised learning method that is capable to transform an incoming signal pattern of arbitrary dimension into a one or two dimensional discrete map.

Based on analyzed dataset they have identified 144 patterns corresponding to local calls, 64 patterns that represent national calls and 36 patterns corresponding to international calls. The pattern represents most probable call duration of the considered call type (local, national or international) at the given time of the day as per historical stats. So the user profile represents the frequency distribution of each pattern at considered windows for CUP and UPH. When a new encoded CDR of a certain user is generated, corresponding user's CUP is adjusted according to that. Then information related to the oldest entry in the CUP is moved to UPH and information related to the oldest entry in UPH is deleted. To compare CUP and UPH researchers have used Hellinger distance [15], which is used to quantify the similarity between two probability distributions. Suspicious events are triggered when this distance value is large.

Even though this solution takes us one step towards near real-time pattern recognition, it has many drawbacks. Authors have accepted that there is high probability of false positives and this can be used only as tool to narrow down the scope of analysis. Huge

number of genuine customers with random needs will be detected, as customer behavior is varying based on their context. They have not given an indication about the size and cardinality of the dataset which was used in this analysis. Therefore, we are unable to determine the scalability of solution. Maintaining user profiles for millions of customers in a modern mobile operator will exhaust system resources [12], [13]. Number of derived patterns will not be enough to represent behavior of a large customer base and identify abnormalities. Also, usefulness of patterns is lower as they have used only the call duration attribute for profiling. This system can only detect frauds which involve very high usage, thus the method will be ineffective in practice.

Shawe-Taylor et al. [7] presented a mechanism to detect set of fraud scenarios in Global System for Mobile (GSM) network. Authors argue that since fraudsters always seek to beat the system by using new techniques it is impossible to eliminate frauds completely. But the detection approach, which is optimum mix of proactive and reactive approaches is most effective in avoiding the noticeable damages due to such frauds.

They have highlighted many fraud cases including PABX hacking fraud, inbound roaming fraud and Premium rated service fraud that can be categorized into extreme user behaviors. Additionally, they have mentioned about subscription fraud, handset theft and freephone fraud. But we have not focused them as those are not considered in our scope.

Authors have used *Toll Tickets* (or CDRs) as the data source for their Fraud Detection Tool (FDT). Even though CDR is created after call is finished, it is appropriate for fraud detection in real time given that each record is collected immediately after creation. Since the CDRs related to billing are not collected rapidly they have proposed mediation device to support hot billing and minimize delay. Mediation device is responsible to poll the telecommunication switches on a regular basis and collect CDR for their FDT. Signaling data is also a candidate data source, but it is difficult to handle due to sheer volume even though it gives data immediately when a call is setup.

Authors have proposed the *Brutus* tool which tries to detect such frauds in real time by observing usage of subscribers. This tool consists of four major components, namely:

1. An unsupervised learning tool which uses neural networks to monitor called party (aka. B-number) details in CDR
2. An unsupervised learning tool utilizing neural networks looking at calling party details (aka. A-number) in CDR
3. A neural network based tool using supervised learning
4. A rule based tool

These components are arranged in a cascaded manner. But each module can generate the alarms by own. First two parts are unsupervised learning components which observe changes in user behavior without use of prior knowledge about fraud. B-number wise profiling and analysis is placed as first filter to all incoming profiles as it allows to reuse the calculated profile data in future steps. Specifically B-number wise analysis was focused on international B-numbers as most of above-mentioned frauds related to international B-numbers. Then A-number wise profiling and analysis is performed using unsupervised Neural Network (NN) method. This allows the detection of novelty frauds and variant attacks. Collectively unsupervised NN part eliminates most of the normal users from equation and reduces further calculations in great extent. Also, it allows to keep the percentage of true negative at a minimum level.

Supervised NN part is placed at next. These components efficiently pinpoint users whose behavior is similar to previously known fraud patterns. Authors expect to achieve high true positive by optimally tuning this component using training datasets. After that Rule-based tool is placed to examine the CDR based features against fixed criteria. These rules are initialized with manually set parameters based on the past observations. Based on the true alarms raised by other modules new rules can be developed to optimize the rule based tool. Finally, all raised alarms are presented in Monitoring GUI which is overlooked by human operators.

This tool is combination of absolute and differential CDR analysis, as NN-based part performs differential analysis while rule-based part performs absolute analysis. Like [3] this system also uses UPH and CUP profiles when analyzing user behavior changes using NN. The main purpose of differential analysis part is to avoid the possibility of one set of rules appropriate for a subset of customers is being applied to all the

customers. Also, it allows to identify user behavior changes at an early stage. Authors have used CDR of 20,000 users collected within four months. Even though CDR files contained 25 fields they have used A-number, B-Number, call starting time and call duration only. They did not consider location details.

Authors have proposed the function for common alarm level as a weighted sum of alarm levels of each component. So weights can be manually adjusted to achieve optimum results. In order to optimize the function logistic regression approach was used. The system triggers alarm when common alarm level exceeds a certain threshold value. This approach is positive improvement when we consider the previous approach [3]. They were able to achieve 85% of detection accuracy. One of the notable limitations of this tool is each time a CDR entry is received user profiles have to be swapped between disk and main memory. System has been tested with 30 CDRs per second using optimized database tool called GDBM. Since typical mobile operator's CDR generation rate would be thousands per second, application of this tool would be difficult. Also, when considering the customer base of modern mobile operators, maintaining and updating huge number of user profiles will not be practical. Therefore, store and analyze approach need to be replaced. Also, neither of these has not considered complex event sequences contained in CDR stream that can give indications about fraudulent activities. For example, we can identify malicious call back fraud at early stage, if we can detect complex event that involves back to back missed call attempts from premium rate number set of subscribers. Also, Rich set of features are required to detect extreme behaviors discussed in [7] without maintaining resource consuming user profiles.

2.5. Complex Events in CDR

It is important to mention about complex events inside CDRs to understand their power in detection. Figure 2.3 sketches one such example. Imagine the situation where we have three different CDR streams for Local Calls, National Calls and International calls separately. At time *t* a call attempt to Subscriber *B1* from other operator number *A1* is recorded in National CDR stream. But that call attempt is blocked since the telecom operator has already identified *A1* as fraudulent number. After a few seconds

at $t + \Delta t$ the same subscriber *B1* gets a call attempt from *A2* in the local CDR stream. Also, we can find a call attempt by subscriber *B1* to International number *I1* at $t - \Delta t$ in International CDR stream. In such situation there is a high probability that *A2* be a fraudulent number and we can use other attributes to verify it. So it is clear that complex events allows us to narrow down analysis domain without maintaining user profiles as in [3], [7]. Figure 2.4 shows the real-world CDR entries that correspond to the above scenario. All telephone numbers have replaced with non-existing numbers to preserve privacy.

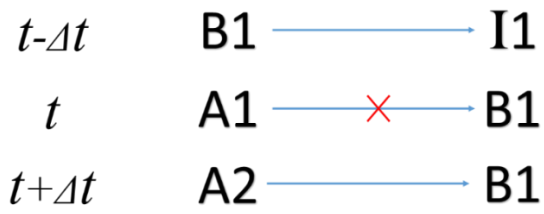

Figure 2:3 : Complex events in CDRs created by SIMbox.

International CDR Stream	2015-02-10 15:08:22	0791234567	+91005637821	
	t-Δt_1	B1	I1	
National CDR Stream	2015-02-10 15:09:42	0739876543	0791234567	blocked
	t	A1	B1	Action
Local CDR Stream	2015-02-10 15:09:53	0790192384	0791234567	
	t+Δt_2	A2	B1	

Figure 2:4: Example complex event in CDRs created by SIMbox.

2.6. Streaming Data Analysis Techniques

All the approaches discussed earlier have derived features based on static data stored on traditional RDBMSs which store and index data before processing. It is apparent that traditional DBMSs alone cannot fulfil timeliness requirements coming from this domain. In this section we will critically evaluate the technologies that can be used for feature generation and complex pattern recognition.

Cugola et al. [16] defined the umbrella term *Information Flow Processing* (IFP) Systems, to represent the tools that can be used to collect information produced by multiple, distributed sources, and to process it in a timely way. Also they have analyzed the capability of each type of system ranging from Active DBMS to CEP to cope with requirements of this domain.

Traditional DBMSs are designed to work on persistent data and deals with infrequent number of queries. When a user enters the command, these systems run that query once and to return a complete answer. Such interactions are known as Human-Active Database-Passive (HADP) interaction. So these systems are not capable to send notifications automatically when predefined patterns or situations are detected. So it is clear that traditional DBMS can hardly fulfil the timeliness requirements and unable to fulfill IFP domain requirements.

Therefore, active database systems were developed as an extension for traditional DBMS to address this limitation by moving reactive behavior totally or in part, from the application layer into the DBMS. Active DBMSs vary based on the kind of active rules that can be expressed on system and the system's runtime behavior. The most common kind of active rule type is called Event-Condition-Action (ECA) rules which breaks active rules into Events, Conditions, and Actions:

- Event describes which sources can be considered as event generators. Some systems only consider internal operators, like a tuple insertion or update, while others also allow external events, like those raised by clocks or external sensors.
- Condition specifies when an event must be taken into account. For example, define condition such that some data can be interested only if it exceeds a predefined limit.
- Action identifies the set of tasks that should be executed as a response to detected event. For example, some systems only allow the modification of the internal database, while others allow the application to be notified about the identified situation.

Run time processing model of these rules is known as execution model and consist of five main phases as described in Table 2.7. Active database systems are used in three main contexts as mentioned Table 2.8.

Table 2:7: Phases in execution model of active databases [16].

Phase Name	Description
Signaling	Detection of an event
Triggering	Association of an event with the set of rules dened for it
Evaluation	Evaluation of the conditional part for each triggered rule
Scheduling	Definition of an execution order between selected rules
Execution	Execution of all the actions associated to selected rules.

Table 2:8: Uses of active databases [16].

Context	Description
As a database extension	Active rules refer only to the internal state of the database, e.g., to implement an automatic reaction to constraint violations.
In closed database applications	Active rules can support the semantics of the application but external sources of events are not allowed
In open database applications	Events may come both from inside the database and from external sources.

It is apparent that open database applications are closer to IFP domain. Since active database systems are built using persistence storage inherited from traditional DBMSs there are negative performance impacts when the number of rules expressed exceeds a certain threshold or when the arrival rate of internal or external events is high. Therefore, active databases lack scalability to cope with multiple high speed data streams which is usual case in modern IFP domain applications.

In order to address timeliness and scalability aspects, database community has developed Data Stream Management Systems (DSMSs) which can process large streams of data in a timely way. DSMS deals with unbounded continuous input streams rather than fixed-size stored datasets like tables. In such scenario assumptions which are made in traditional query processing are no longer valid. For example, no assumptions can be made on data arrival order over the stream. So storing the received events from data stream and process it after that is not practical. Such approach also imposes latency and scalability constraints. To address these limitations, DSMSs use

one-time processing by directly dealing with streams. DSMS allows posing new type of queries called standing queries that are deployed once and continue to produce results until removed. Unlike in previous cases standing queries can be triggered by system itself, periodically or continuously, as new stream items arrive without user interaction. Opposite to HADP this type of interaction is called Database-Active Human-Passive (DAHP). In some cases, DSMS produces answer to a query as an append-only output stream while in other cases continuously modifying the entry in storage when new elements comes through stream. It may produce exact answer or approximate value based on available memory to store the required elements of input stream's history. DSMS executes standing queries and produces four main types of outputs as described in Table 2.9.

Table 2:9: Types of outputs produced by DSMS [16].

Type of Output	Description
Stream	Formed by all the elements of the answer that are produced once and remains belong to answer within whole lifespan of query.
Store	Filled with parts of the answer that may be changed or removed at a certain point in the future. The Stream and the Store together define the current answer to queries
Scratch	Represents the working memory of the system that acts as repository where it is possible to store data that is not part of the answer, but that may be useful to compute the answer
Throw	Sort of recycle bin, used to throw away unneeded tuples

Therefore, DSMSs only focus on producing query answers, which are continuously updated to adapt to the constantly changing contents of their input data. Actually it creates modified output stream as an answer. Detection and notification of complex patterns of elements involving sequences and ordering relations are usually out of the scope of these systems. So manual intervention is required to associate a semantics to the data being processed and interpret meaningful complex events in given context.

Unlike previous categories, Complex Event Processing (CEP) systems associate an accurate semantics to the information being processed and detect meaningful situations within that context. CEP Engine is listening to event streams generated by external sources via observers and then perform filtering and combining such notifications to generate higher-level events (aka. composite events or complex events). Detected

events are notified to sinks which act as event consumers. CEP systems can be considered as an extension of traditional publish-subscribe architecture because which allow subscribers to express their interest in composite events. So unlike in traditional publish-subscribe architecture CEP considers the history of already received events or relationships between events.

Ability to detect complex patterns of incoming events (composite events), based on their content, sequencing and ordering relationships is most powerful feature of CEP model. Also, CEPs need to deal with large number of distributed and heterogeneous information sources and sinks.

Cugola et al. [16] also introduced a framework to compare different types of IFPs. This framework includes a set of models that represents the various facets of IFP. From this set of models, Functional model provides the precise description of the functionalities offered by IFP engine and it can be used to describe differences among IFP engines. By considering this model as ideal system, authors have compared different CEP engines with respect to it.

We have studied various possible technologies in previous section and identified that CEP is the closer implementation that satisfies the requirements of IFP Domain. Next, we will evaluate a set of commercial and open source CEP engines in terms of features and performance.

2.6.1. S4

Neumeyer et al. at Yahoo proposed S4 (Simple Scalable Streaming System) [17]. S4's goal was to come with general purpose, easily customizable stream processing platform which allows to use commodity hardware in distributed manner. It has structured to achieve minimum latency by using local memory in each processing node by avoiding use of shared memory across the cluster. S4 considers all the nodes in cluster as identical nodes and there is no centralized control. This symmetry feature was achieved by using ZooKeeper [18] which is an open source cluster management service.

S4 consists of multiple Processing Elements (PEs). PE is the basic unit that performs the computation tasks. PEs interact which each other by emission and consumption of messages exchanged between them in form of data events. S4 framework facilitates for routing events to corresponding PEs and create new PEs when required. PE is uniquely identified by four components, namely functionality of PE and its associated configuration, type of events that it consumes, keyed attribute of those events, and value of keyed attribute.

PE is instantiated for each distinct value of the key attribute. Therefore, PEs consume only the events which have exactly same key value to the keyed attribute value of PE. Keyless PEs are special case which do not have keyed attribute or value. Those are used at input layer and consume all the events of the type they are associated. Also, S4 provides predefined PE types for standard tasks such as count, aggregate, and join.

Processing Nodes (PNs) act as logical hosts of PE and responsible for listening to events, executing selection operations on the incoming events, dispatching events and emitting output events. S4 makes routing decisions on events to corresponding PNs based on hash function of key attribute value of that event. Event listeners running on PN passes incoming events to the processing element controller which invokes appropriate PE in proper order. Every Keyed PE is mapped to exactly one PN of a cluster based on hash function applied on keyed attribute value of that PE. Processing node is functioning at top of the Communication Layer which is responsible for cluster management, automatic failover to standby nodes and maps physical nodes to logical nodes.

Authors have presented the performance of S4 when it is used for streaming click through rate (CTR) computation. Click through rate (CTR) is the ratio of number of clicks divided by number of ad impressions. Users have used S4 grid to calculate CTR in real-time and system performed CTR computation with 0.2% relative error at input event rate of 7,268 events per second. Beyond that event rate, relative error was increased since S4 grid was not able to process the event stream fast enough.

2.6.2. SASE

Gyllstrom et al. proposed SASE [19] to fulfil real-time analytical requirements of RFID-based applications. SASE can also be used for general purpose applications as well. SASE targets to perform complex logics involving filtering, pattern matching, aggregation and recursive pattern matching over express-rate data streams with minimum latency and acquire meaningful actionable information. SASE is available as an open source system for stream processing and pattern matching.

Authors introduce SASE Complex Event Language which is user friendly and expressive. High-level structure of SASE language is similar to SQL even though the design of language is centered on event pattern matching.

Implementation of SASE is based on query plan-based approach which uses a dataflow paradigm with pipelined operators like in relational query processing. This approach provides greater flexibility in query execution. Native sequence operators are formulated on Non-deterministic Finite Automata (NFA) based model. NFA is state machine concept, in which the state machine in one state can have zero, one or more choices for the next state for particular input symbol. As per definition, we can say that such a state machine has accepted the string of symbols, if there is at least one sequence of state transitions on an input that leaves the machine in an accepting state. This concept allows to detect complex event patterns efficiently from continuously arriving event streams and used in most CEP systems. Authors were able to deal with lengthy sliding windows and large intermediate result sets using new abstractions of query processing mentioned above.

Architecture of SASE based application consists four main layers. Bottom layer is known as physical layer which represents RFID readers, tags and antennas in RFID based scenario. But in general case that can be Sensor, Telecommunication node or any other data source. The cleaning and association layer lies above physical device layer which accepts data and performs cleaning and event generation. Within this layer, raw data is subjected to data preprocessing functions including anomaly filtering, temporal smoothing, time conversion and deduplication before generating events. Complex Event Processor is the main component in third layer and supports

long-running queries written in SASE language. It also has the capability of handling complex continuous queries that integrate database lookup. At complex query execution, Event processor first detects events and then sends subquery to database and combines the results to give final output. Event Database lies parallel to the CEP. This is the persistence storage component which allows historical data access and join that with resultant events. Finally, UI is the topmost layer which allows user to issue both continuous queries over the Streams and ad-hoc queries over database. It also visualizes results.

2.6.3. Esper

Esper is an open source library for CEP and event stream analysis [20], [21]. Esper is available under GNU General Public License v2. There is also a commercial version with high availability features. Esper uses DBMS, DSMS, and CEP concepts and can be used in data stream based and CEP applications. Esper engine allows applications to store queries and evaluate them against the data stream running through System. It generates real-time response when events match to conditions specified in continuous queries. Esper is based on the foundation of Event-Driven Architectures (EDA) and can be considered as natural extension to Service Oriented Architectures (SOA).

Esper allows writing complex queries using the language called EPL (Event Processing Language), which is quite similar to SQL. EPL allows to express filtering, aggregation, grouping, sorting, counting unique events and join functions, possibly over sliding windows of multiple event series. It also supports same operations over batch windows. Sliding or batch window can be length window or time window. EPL provides the concept of named window. Named windows are data windows which are globally visible. So, *inserted-into* or *deleted-from* operations executed by one or more query statements can be applied on that. Also, Named windows can be queried by one or more statements. Esper statements can also be combined together with “followed by” conditions thus deriving complex events from more simple events.

Esper subscribes to source event publishers through event stream connector and adapters. Esper supports a wide variety of event representations, such as Java beans,

XML document, legacy classes, or simple (key, value) pairs (*java.util.Map*), which promotes reuse of existing systems acting as messages publishers [21]. Also, it supports connecting relational databases as data sources through historical data access layer. Esper has the capability to join event streams against these historical data sources. Esper can be easily embedded in an existing Java application or middleware to add event-driven capabilities to existing platforms without paying high serialization cost or network latency for every message received and action triggered. Once event queries and pattern statements are registered in the Esper core container, event data gets analyzed and can trigger arbitrary logic bound to the engine in the form of Plain Old Java Objects.

2.6.4. Siddhi CEP

Suhothayan et al. performed a comprehensive evaluation of design decisions associated with CEP [22], [23] and suggested several approaches to improve CEP performance by using more stream processing style pipelines. Siddhi CEP is the final result of their research which implements the aforementioned suggestions. Later WSO_2 developed it and made it freely available under Apache Software License v2.0 [24]. Siddhi uses design decisions such as multi-threading, queues and use of pipelining, nested queries and chaining streams, and query optimization and common sub query elimination to improve the performance.

Siddhi receives events through input adapters, then Siddhi core performs processing, and finally emits output through output adapters. Query compiler is connected to Siddhi core to deploy queries on it. Input adapters are responsible to receive events and convert them to a common data model known as tuple. Tuple data structure is similar to row in relational database table and it contains Stream ID and other data items belong to columns. Tuple allows to process events faster by minimizing overhead while allowing to use SQL like queries with traditional database optimization techniques. Siddhi core only accepts tuples for internal processing, so events received as XML or POJO (Plain Old Java Object) are converted to tuples at input adapters. Query compiler is responsible for converting the user submitted queries to its runtime representation and deploy it on the Siddhi core. Runtime representation of Siddhi

query is defined as Processor. Processing tasks are performed by Siddhi core which consists of processors and event queues. After converting to tuples, input events are placed at input queues and processors fetch those events for processing. Then those events are evaluated against the query conditions and corresponding output events are produced for matching events. Output events are placed at output queues and based on requirement those events can be sent to external subscribers through output adapters or can be consumed by another processor for further processing. Siddhi architecture allows on the fly query manipulation. This feature allows users to add or remove queries while siddhi engine is running.

Siddhi queries are represented using query object model which follows SQL like structure. These queries are in line with relational algebra and allowed to use optimization techniques used in relational databases. Query produces a stream as an output and that can be recursively passed to another query as an input stream to make complex queries. Since the query objects are loosely coupled, users can easily compose nested queries while allowing to eliminate common sub-queries to achieve better overall performance. Initial implementation of Siddhi allows users to create queries via the Java interface. Later releases of Siddhi supports Siddhi Query Language (SiddhiQL) which is more user friendly. More details will be discussed later in this chapter.

Siddhi uses pipelined architecture in query execution. Query execution is broken into different stages through processors which are connected through event queues. Data is moved through the pipeline using publication-subscription model in which queries at downstream subscribes to interested upstream query outputs. Each processor is composed of several executers which are responsible to evaluate events against single query condition and produce Boolean output to indicate whether event has matched or not. Matching events are passed to logical executers at downstream while non-matching events are simply discarded.

In Siddhi all processing tasks are performed by multiple treads and pipelined architecture along with transparent query object model allow to ensure that common sub-query is executed at only one point in the system. This approach helps to

outperform both single tread query processing and one thread per single query approaches where former is suffered by less parallelism while latter is incapable of eliminating duplicate sub-query execution. Siddhi uses single input queue to feed processors. This is achieved by multiplexing multiple streams into a single queue. Stream ID field of tuple facilitates to distinguish types of events. This approach makes intermediate query handling much simpler and improve Siddhi's performance.

Siddhi processor evaluates the conditional or temporal conditions against each incoming events. Executors and Event Generator are two major components of processor. Executors are generated by query parser by parsing the query object model constructed by the user and responsible to evaluation of conditions. Executers are arranged in tree-like structure and event is passed to the root of the tree and evaluated according to depth first search order. This structure helps to eliminate non-matching events early and enhances Siddhi performance.

Siddhi state machine is the major part in processing complex queries. State machine is used to handle two type of queries named as Sequence queries and Pattern queries. Sequence queries allow to define Siddhi to fire an event when series of conditions are satisfied one after the other. Since Siddhi stops capturing same type of event sequences once first instance of particular event sequence is detected, we need to use *Every* operator to instruct Siddhi to continuously capture such event sequences. Pattern Queries allows to fire an event when series of conditions are satisfied one after the other in consecutive manner. Kleene star operator is used to define infinite number of intermediate conditions in pattern queries. That means when we use "*" operator with event type, Siddhi looks for zero or more events from that event type in event sequence.

Another useful feature of Siddhi is Sliding window and Batch window based queries which allows to reason about collection of events. *Sliding windows* allow to analyze events come in given time or length window including statistical analysis of the arrived events such as average and sum of attributes. So these windows keep sliding for each new event arrived to the stream. This can be divided into time based and length based windows. Time based sliding windows consider past events received within given amount of time from current event. Length based sliding windows consider given

value of event count from current event to backwards. *Batch window* provides similar functionalities, but it performs the analysis batch wise. So it releases the event batch for processing after given amount of time has elapsed after starting timestamp (Time Batch Window) or specified maximum number of events has received from given starting point (Length Batch Window) and commence to collect new batch. Multiple Siddhi queries can utilize same windows as it implements the windows within event queues rather than event processors. This enables effective memory consumption and allows to get better performance.

Duplicate event detection is another useful feature of Siddhi. Duplicity can be defined by specifying the set of event attributes that needs to match. Siddhi provides two ways to deal with duplicate events. First approach, UNIQUE option only considers last arrived events and old duplicate events are discarded. Second option, FIRSTUNIQUE works vice versa and it accounts only first arrived event while discarding newly arrived duplicate events.

It is important to compare Siddhi performance against Esper because those are two major candidates for our application. For simple filter query (without time or length window) and time window query for average calculation of a given symbol, siddhi performs about 20-30% better than Esper [22]. Also, for pattern query with state machine Siddhi performs significantly better than Esper [22].

2.6.5. CEP evaluation

We have studied in detail about four CEP systems available in the industry. Cugola et al. performed the full analysis about existing Information Flow Processing (IFP) systems as of 2011 with respect to a set of models defined by them [16]. Esper [21], SASE [19], and S4 [17] included in that list. Even though we only discussed about functional model which helps to categorize IFP systems, we will directly refer other models in [16] to compare CEP systems.

Even though S4 supports massive scale processing of data streams it still cannot be categorized as an effective CEP engine as it cannot handle complex events. Also, it does not support basic temporal event processing capabilities over time or length

windows. So SASE, Esper and Siddhi can be categorized as full CEP systems. All these systems use variations of Non-Deterministic Finite Automata (NFA) model to provide complex event detection. Also, Etalis [25], Cayuga [26], and ODE [27] are the open source systems which can be categorized as CEPs [22]. Additionally, commercial CEP systems like Coral8 [28], Oracle CEP [29], and Streambase [30] are available.

Suhothayan et al. also compared CEP systems including SASE and Esper [22], [23]. When we consider SASE it performs considerably well due to NFA model to capture the sequencing events. SASE reports both satisfaction of query and the event caused to fulfil the satisfaction condition. Even though this can be viewed as advantage it significantly increase complexity of query processing. The most significant drawback of SASE is output of one query cannot be used as input to another query. So it is incapable of handling hierarchy of complex event types. Since this feature is required in our system SASE cannot be categorize as eligible candidate. Cayuga is a general-purpose CEP system, which can be used to detect event patterns in multiple unrelated event streams and capable of handling hierarchy of queries. Its single treaded nature imposes limitations.

In our analysis we found that Esper also has the ability to detect sequence of patterns in unrelated event streams while supporting temporal windows, joining, sorting and various other functions. Esper is multi-threaded and its architecture predominantly depends on observer pattern. Siddhi and Esper behave same in terms of functional, processing, data, time and rule models defined by Cugola et al. [16]. Siddhi is also multi-threaded. Both provides required functionality to implement our system. Since complex pattern detection plays major role in our solution, Siddhi's performance in complex event detection hugely motivated us to select it as best candidate. But, Esper has rich documentation and many successful deployments.

2.7. Accessing Persistent Data within CEP

Accessing persistent data in addition to real-time processing is common requirement of complex event processing systems. According to the above analysis Esper and SASE allows to define RDBMS as data sources. Event Tables option in WSO_2 CEP [24] supports for using historical data in real-time processing.

Event tables allow to store, retrieve and process events in a database table-like structure. This option is primarily used in use cases where events need to be extracted from the stream and accumulated over a long period of time for real-time (or non-real-time) batch processing such as performing comparisons with the incoming event stream or feeding it to Business Activity Monitor (BAM). Event tables can have more sophisticated storage and retrieval criteria unlike in event windows which can be considered as predefined tables in WSO_2 CEP. A single event table can be used in multiple SiddhiQL expressions. Event table can be defined either in-memory or in a relational database. In Memory Database Event Tables are fast and easier to define hence it is created in memory. Relational Database Event Tables allow to link RDBMS table to Siddhi CEP. Current version of WSO_2 CEP supports event tables for widely used databases such as MySQL and H2.

2.8. Combining Real-time View with Historical View

The real-time view need to be combined with historical view to build the complete state of the network at a given instant. CEP is specialized to perform real-time event analysis while BAM is specialized to build historical view by executing batch operations. Therefore, standardized architecture is required to combine these tools in optimum manner. Marz et al. [31] proposed Lambda Architecture which allows to unite set of tools used in Big Data Analysis and come with most complete solution. Their main idea is to build Big Data systems as a series of layers. Each layer satisfies a subset of the properties and builds upon the functionality provided by the layers beneath it. Figure 2.5 depicts the layers of the Lambda architecture and its functionality.

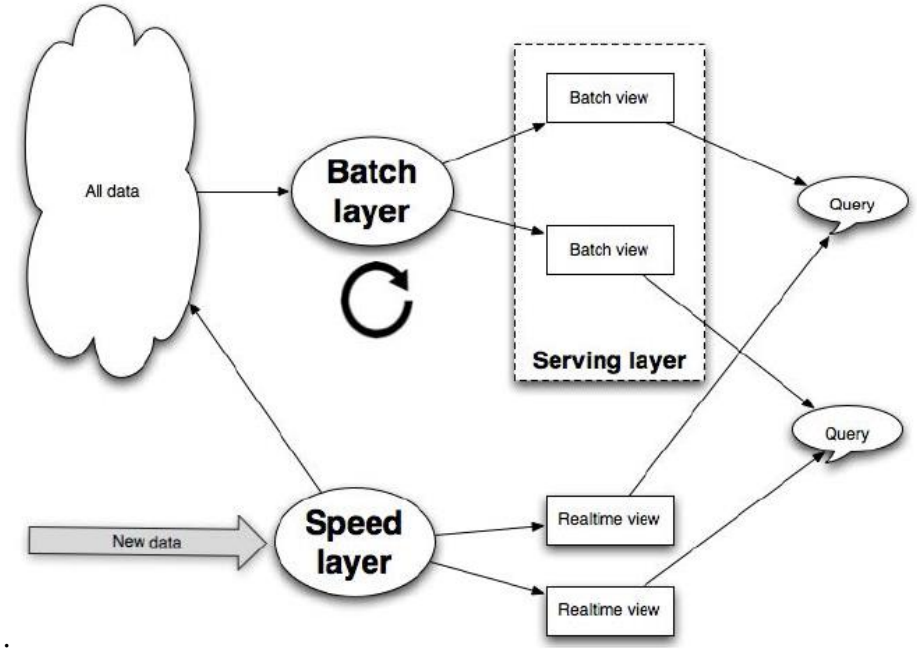

Figure 2:5: Lambda architecture for Big Data [31].

The batch layer stores the master copy of the dataset and precomputes batch views on that master dataset. It should be capable to deal with storage requirements of master dataset which is immutable, constantly growing very large list of records. Also, it should have enough processing power to compute arbitrary functions on that dataset. Batch processing system such as Hadoop [32] is best suited for implementation of this layer. The serving layer saves the resultant batch views emitted by batch layer and allows to be queried when required. The serving layer is a specialized distributed database that loads in batch view and makes it possible to do random reads on it. Support for random writes is not essential as it may cause complexity in databases. When new batch views are available, the serving layer automatically swaps those in so that more up-to-date results are available. Authors suggested ElephantDB for this layer.

Batch layer and Serving layer satisfy key properties of big data system such as robust and fault tolerant, scalable, general, extensible, allowing ad-hoc queries, minimal maintenance, and debuggable.

The remaining most important feature is low latency updates and computation of real time view. Speed layer provides this functionality. The serving layer is updated whenever the batch layer has finished precomputing a batch view. This means that, the data not represented in the batch views is the data that came in while the pre-computation was running. In order to provide full real-time functionality, speed layer calculates required views on recent data that was not accounted by last batch view and compensates for high latency of updates from batch layer to serving layer. Therefore, the goal is to make new data represented in query results as fast as needed for the application requirements. The big difference is that the speed layer only looks at recent data and computes incremental updates, whereas the batch layer looks at all the data at once and provide batch updates. Finally, serving layer provides full real-time view by merging both real time and batch view. The speed layer uses databases that support random reads and random writes. But more sophisticated systems like stream engines and CEPs easily provide this functionality.

We have evaluated possible technology options for real-time event stream analysis in section 2.6.5. WSO_2 Siddhi CEP is an optimum candidate according to the facts presented. [33], and [34] present how WSO2 products can fit to lambda architecture. Authors have suggested to use WSO_2 Business Activity Monitor (BAM) to implement the batch layer and WSO_2 CEP to implement speed layer. Incoming data is sent to both BAM and CEP using high performance data transport called "Data Bridge" that can achieve throughput up to 300,000 events/second. This functionality is provided by Data Bridge feature in the WSO_2 feature repository. BAM runs user defined Hive queries to calculate the batch views and CEP runs user defined CEP queries to calculate the runtime views. Then both the views can be combined using *Event tables* in WSO_2 CEP, which map the batch views in a database into CEP windows, to answer the queries posed by the users.

2.8.1. WSO_2 BAM

WSO_2 Business Activity Monitor (BAM) addresses a wide range of monitoring requirements in business activities and processes [35]. It achieves this level of flexibility, while facilitating technologies such as big data storage, analytics, and high-

performance data transfer. WSO_2 BAM is designed to be significantly scalable to handle large amounts of data loads when aggregating, analyzing and presenting data.

BAM can be divided into four main parts. Those are Data Agents, Data receiver, Analyzer engine and Dashboard. First two parts perform data aggregation function and Analyzer engine performs Data analysis part. Finally, Dashboard and Report server perform data presentation functions. Data that needs to be monitored goes through these modules in order.

The BAM analytics framework runs summarization and data analytics on collected data. WSO_2 BAM implements data analysis using an Apache Hadoop-based Big Data analytics framework, which uses the highly-scalable, MapReduce technology [36] underneath it. BAM allows to write data processing queries and analytic jobs in integrated Apache Hive query language. So BAM users are released from the burden of writing complex Hadoop jobs to process data using underneath MapReduce technology. Hive is a simple query language similar to SQL, and provides the right level of abstraction from Hadoop engine while internally submitting the analytic jobs to Hadoop.

2.8.2. WSO_2 DAS

WSO_2 Data Analytics Server (DAS) [37] is a successor of WSO_2 BAM. WSO_2 DAS supports all the features provided by WSO_2 BAM. So, DAS facilitates to aggregate events through data receivers, store those events in persistent storage, analyze those data using high speed large-scale data processing platform, and present information. In addition to BAM features, DAS is capable of performing real-time analytic tasks because all the features of WSO_2 CEP integrated within DAS.

Figure 2.6 depicts high-level architecture of WSO_2 DAS. DAS is considered as complete revamp of old BAM because of major architectural level differences between two products. Instead of Apache Hadoop based Big Data analytics framework used in BAM, WSO_2 DAS uses Apache Spark based analytics engine. Also, WSO_2 DAS allows to execute SQL queries on the underlying data-sources as specified in Data Access Layer of the DAS. To provide this functionality, DAS uses Spark SQL as the

query engine which is Apache Spark's module for working with structured data. WSO_2 DAS users can execute Spark SQL queries interactively through Batch Analytics Console. Batch Analytics Scripts allow to run Spark queries in a sequence and WSO2 DAS allowed to schedule those queries as per user's requirement.

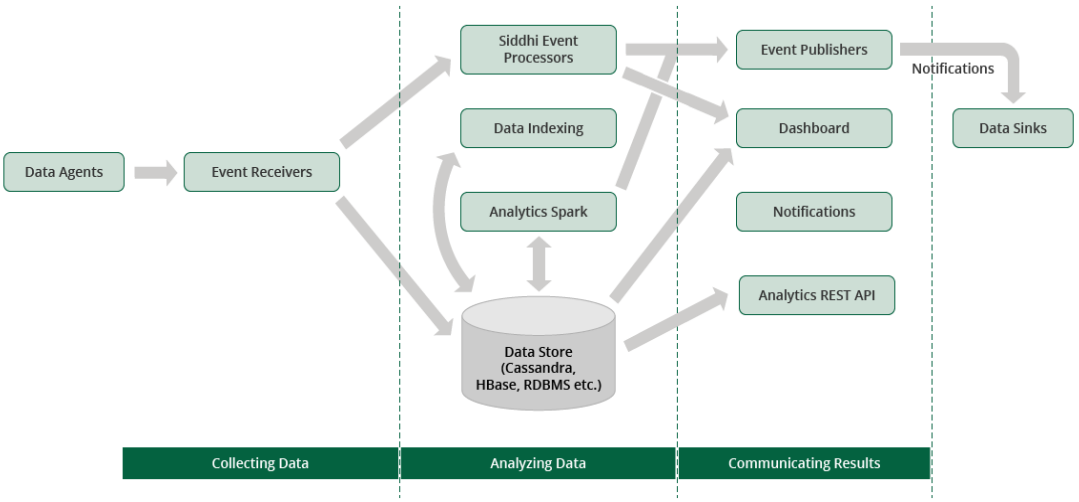

Figure 2:6: WSO_2 DAS Architecture [37].

Instead of Cassandra centric storage in BAM, DAS provides pluggable storage architecture and allows to choose underline storage mechanism based on requirement. For low to medium scale enterprise deployments, RDBMS data storage mechanisms such as MySQL, MSSQL or Oracle can be used. For Big Data enterprise deployments NoSQL storage mechanisms like for HBase or Cassandra can be used. The WSO_2 CEP module integrated in WSO_2 DAS version 3.1.0 supports Siddhi Query Language 3.0 and allows to perform real time analytics.

Jayawardhana et al. proposed custom CDR analyzer "Kanthaka" for near real-time telecom promotions in their research [38]. In "Kanthaka", batch of CDRs is stored in Hash-maps in memory module and batch processing is done on that data. After that, increments are sent to Cassandra for each batch. Mentioned system has performed simple filter query on 600k events in 18 seconds. This method is less scalable and limited in functionality. Also, more complex queries need to be implemented in our scenario. Our preference for batch layer is WSO_2 DAS instead of approach in "Kanthaka" because DAS provides generalized architecture with more functionalities while it can be easily integrated with WSO_2 CEP and other products in suit. DAS is

highly scalable and provides high performance data capture framework with abilities to monitor, collect, and store Big Data. Sirbiladze et al. analyzed the available commercial and open source BAM solutions. They were compared Oracle BAM vs WSO_2 BAM and found that WSO_2 BAM has almost all the dashboard and data integration features in commercial BAM. Since WSO_2 DAS is successor of WSO_2 BAM with enhanced performance and real-time analytics capability, DAS is the best suited platform for this scenario.

2.9. Summary

In first part of this chapter we have discussed about CDRs and the value of information available in CDRs. Next we have presented details about two main use cases of our project, namely grey call fraud and extreme usage scenarios. According to provided details it is evident that detecting those scenarios in real time is worthwhile. Based on literature, it is clear that available approaches are unable to meet the requirement of real-time fraud detection due to traditional database reliant store first process then approach for latency sensitive applications, depending only on large time windows for feature generation, shallow feature set, less awareness about context, and ignoring complex patterns in CDR in decision making. Then we have discussed about complex events which can be identified in CDR stream and value of those complex events or call patterns in decision making. We have evaluated streaming data analysis techniques and identified that WSO_2 CEP as the most suitable candidate for CDR stream analysis, and call pattern detection due to Siddhi's enhanced performance in complex event detection. We further studied possible ways of accessing persistent data from CEP and discovered that WSO_2 CEP supports easier ways to access persistent data within CEP queries. We have identified that Lambda architecture is suitable baseline architecture for systems which perform both real time and batch analytics. Also, we have identified that WSO_2 DAS as candidate platform to build our system due to its capability of performing high-speed batch analytics. According to this discussion, it is evident that there is a requirement of real-time fraud detection tool and open-source tools available in industry can be used as platform for such a tool.

3. PROPOSED DESIGN AND IMPLEMENTATION

This chapter presents the architecture of the proposed real-time fraud detection system for telecom operators, its design, and implementation. Section 3.1 presents the high-level architecture of proposed design and describes its components in detail. Section 3.2 discusses the selection of features and steps in designing algorithms for each use case.

3.1. High-Level Architecture

To build the complete behavioral view of the customer base of telecommunication network, both historical and real-time views are required. Therefore, we followed the Lambda architecture and developed a system architecture that comprises batch, speed, and serving layers. Figure 3.1 depicts the high-level architecture of the proposed real-time pattern detection platform. This architecture consists of three major layers (Speed, Batch, and Serving) similar to Big Data Lambda architecture, as it is well suited for application which performs both real-time and batch analytics. System receives events through data receivers, then perform analytics operations and output can be obtained from serving layer. Based on the use case, the output can be directly used or can be passed through a classifier. WSO_2 DAS is used in batch layer due to its ability to perform high-speed batch processing. Siddhi CEP is used at speed layer due to its enhanced performance in complex event detection. Both WSO_2 DAS and CEP are used at serving layer as application needs to get real-time, batch, or combined output when required. Because Siddhi CEP is integrated within WSO_2 DAS package, there is added advantage in using these packages together.

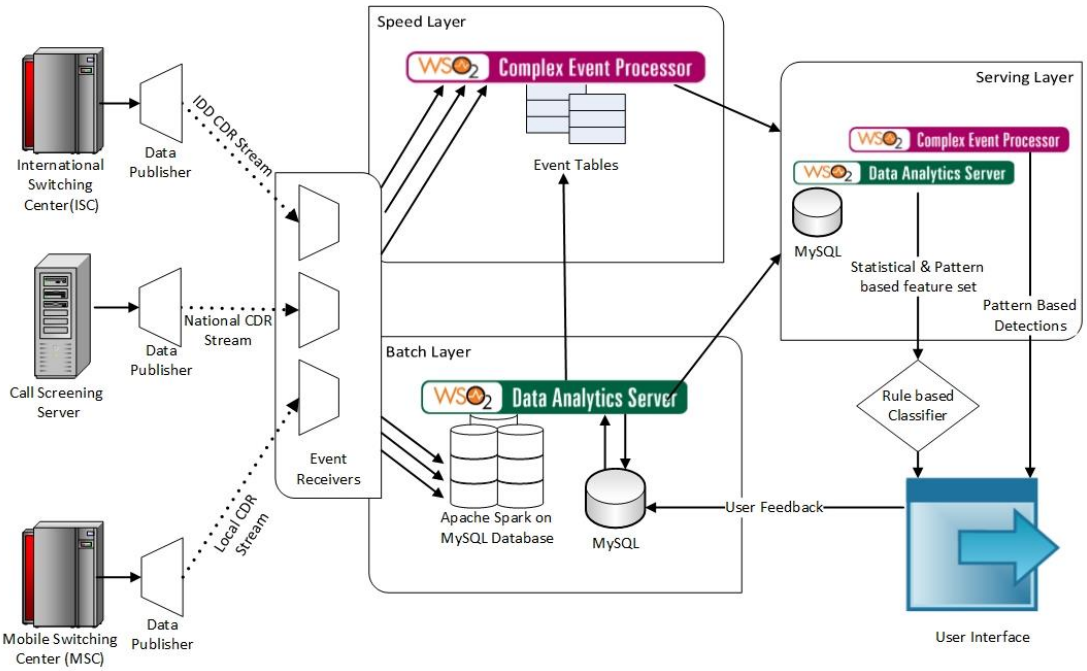

Figure 3:1: High-level system architecture.

To support the two chosen use cases, namely grey call detection and extreme usage detection (see Section 2.2 and 2.3 for details), three data sources are utilized. As seen in Figure 3.1 those data sources are Local CDRs, National CDRs, and International CDRs. These data were acquired from different nodes of the telecommunication core network. First, data from each data source need to be mediated by removing unnecessary characters and only the required fields need to be filtered. Then mediated data need to be send as event streams using data publishers. Event receivers listen to event streams published by external sources and direct that to processing layer.

Incoming data is sent to both batch and speed layers, where batch layer pre-calculates a historical view of the system and speed layer calculates the most recent view of the system. Major component of the speed layer is the CEP engine. CEP calculates real-time view of the data streams forwarded by Event Receiver and then feeds the calculated views into serving layer. Real-time view of the call patterns is calculated using simple aggregation queries and complex pattern queries deployed on the CEP engine. We selected the WSO_2 Siddhi CEP engine for speed layer, as it suitable for complex event detection and its high performance comparable to other CEP engines. While implementing CEP queries to detect some of the use cases, we need to join

persistent or other short-term data tables with streams. Siddhi CEP achieves this by providing option to define RDBMS and In-Memory event tables in CEP Query plans. These event tables can be pointed to RDBMS table (e.g., MySQL) or defines short term In-Memory table and those tables can be directly accessed via CEP queries. Even though this design decision slightly varies from the original Lambda architecture, it helps to make more meaningful detections.

WSO_2 DAS produces historical view by performing batch processing at batch layer while CEP performs real-time processing. Therefore, events need to be fed into both the CEP and DAS. DAS can perform high-speed batch processing using Apache Spark Engine integrated with it. WSO_2 CEP is integrated into to WSO_2 DAS; hence, data receivers in WSO_2 DAS receive the events published by Data Publishers and first fed them to CEP and then only, the data is persisted through WSO_2 DAS. DAS allows using either a RDBMS or NoSQL storage as the underline database for event store. Also, system updates context data related to subscriber behavior using current data, detections made by system, and user feedback stored within database. These context data are used at feature generation as discussed in Section 3.2.1.1.

The output of batch and speed layers are directed to serving layer and stored within it. Based on use case, serving layer facilitates to emit batch processing output, real-time view, or combined view. Therefore, when real-time view is required, we need to implement filtering queries to produce output of speed layer. When historical view is required, serving layer should facilitate to get it by using Spark Query. Also, real-time and historical view need to be combined at serving layer to provide combined view when required. Therefore, both DAS and CEP modules used at the serving layer. Even though we have depicted serving layer with separate DAS and CEP components to demonstrate our architecture clearly, the same CEP that was used in speed layer and same DAS in batch layer was used for serving layer.

Some of the fraud instances can be detected by considering real-time behavior only and such instances are captured by filtering implementing queries on CEP. Combining real-time view and batch view is required to detect remaining fraud instances with minimum delay and higher accuracy. This combination produces a rich set of features

that represent statistics of dataset, context information, and complex events. Therefore, calculated feature values are passed through a Rule-based classifier to filter those fraud instances and final output is presented to the user. It helps to fully automate the system and make decisions without human intervention. Classifier module can be replaced with a suitable machine-learning based classifier without violating original system architecture. The classifier introduced in [6] is a possible option. As our primary focus was come up with a rich set of features which enables decision making based on a short-time window, selection of optimal classifier is left as future work. Finally, user feedback is fed back into batch layer and it is used to update context data.

3.1.1. Data sources, Publisher, Receiver, and Event streams

Three main data sources namely, Local, National, and International CDRs are used in this design. *Local CDR* means transaction logs for calls originated by operator's own subscribers. These records are generated at Mobile Switching Centers (MSCs) in ASN.1 (Abstract Syntax Notation One) format and typically new file is generated in less than one minute. *National CDR* stands for transaction logs for calls terminated by other operators within the same country to the operator under study. Detailed version of these Call logs for answered calls are generated at TMSCs (Tandem or Gateway MSC). However, the telecom operator in our case use Call Screening Server as a firewall between other operators and home network. This node can blacklist numbers which need to be blocked and generate transaction logs in real time with the corresponding action (e.g., Blocked or Passed). Inclusion of action field makes these logs more useful and allows us to exploit CEP to locate the patterns easily. This server creates a log file once a minute with calls logged within the last minute. Therefore, Call Screening Server log was used to feed National CDR Stream. *International CDR* means records corresponding to calls originated to or terminated from foreign operators and detailed version of these CDRs are generated at ISC (International Switching Center). CDRs are originally generated as ASCII formatted files and then converted to CSV format.

Data Publisher module is responsible for sending CDR entries to DAS and CEP by means of event stream. Event receiver, which is accompanied with DAS, listens to

event streams published by external sources. The format and the attributes of streams need to be defined in DAS and CEP prior to sending data. System only accepts published data complied with defined stream format through event receivers. Event adapters in DAS and CEP define network protocol and listening ports to receive events. Therefore, event receiver binds event adapter type together with stream definition to receive events properly. Data publisher should publish data using the protocol defined in event adapter and complying with tuple formats defined in event stream definitions.

3.1.2. Batch layer

Main component of batch layer is WSO_2 DAS. Apache Spark analytics engine integrated in DAS and the underline persistent data storage performs batch layer functionality. WSO_2 DAS provides predefined data-source named as *WSO2_ANALYTICS_EVENT_STORE_DB* to persist input stream data. This data-source can be pointed to separate RDBMS or NoSQL database by modifying backend configurations of WSO_2 DAS. Once Stream persistence is enabled, WSO_2 DAS creates separate Spark table mapped to a Stream and the table can be accessed through Spark SQL batch analytic queries. For large-scale deployment it is desirable to use a NoSQL database. Persisted data is then analyzed using DAS batch analytics engine which is powered by Apache Spark. To access and create tables on Spark Analytics engine, CarbonAnalytics relation provider was used [37]. These tables can be accessed only through Data Access Layer using spark Queries. Carbon JDBC relation provider in Apache Spark was used to access already defined MySQL tables. Using this option, we were able to update feedback and context data externally and accessing those tables through Spark Analytics engine when required. Output data is then stored on Processed Data Store which is also can be pointed to RDBMS or NoSQL Database from DAS configurations.

In Spark Query language used in WSO_2 DAS, users need to define temporarily table mapped to each actual table to access data in actual table. Within query scripts, users need to refer to temporarily table instead of directly referring to the actual table. Temporarily table definitions are dropped from memory after Spark queries reach the

end. But data in actual table remain unchanged and if new temporarily table which mapped to actual table is defined again, old data can be accessed. In batch operations, we have requirement of truncating data in Spark actual tables which are used to store intermediate calculations. Because those tables are used periodically to fill fresh data, old data need to be removed. Spark Query language does not include inbuilt function to delete actual Spark tables. DAS only provides separate shell script to delete those tables. But in our case, we have a requirement of truncating or completely deleting some intermediate tables at the end of a query. Therefore, a new user defined function called *deleteTable* was implemented to achieve this.

3.1.3. Speed layer

WSO_2 CEP module is the main component of speed layer. In actual implementation, CEP module integrated in WSO_2 DAS can be used to achieve speed-layer functions. Three main input event streams correspond to Local, National, and International CDRs are subjected to real-time analytic queries in this layer. So complex patterns detection is performed at this layer. Set of related Siddhi query expressions, and relevant input and output stream definitions are included within the entity called Execution Plan. To implement intended goals of each use case, one or more execution plans was used. Figure 3.2 depicts overall event flow within the CEP engine. First CEP receives events corresponding to Local (Onnet), National (Offnet), and International CDR Streams through event receivers. Then events are fed into CEP through defined streams and subjected to set of CEP queries included in execution plan. Then output streams are published to serving layer using Data Publishers. As we need to store those resultant streams on MySQL tables, *rdbms* was used as Output Event Adapter Type. So, output data is directly inserted form output stream into MySQL table defined in output data publisher.

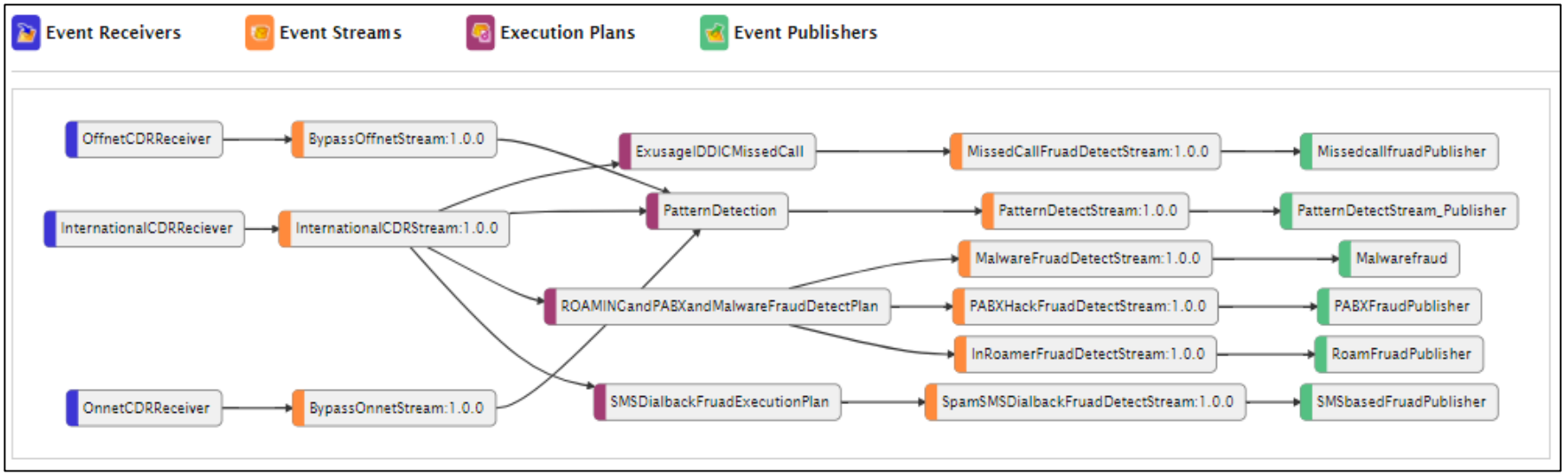

Figure 3:2: Overall event flow through CEP.

3.1.4. Serving layer

Functionality of serving layer is to store and merge output of Batch Layer and Speed Layer and produce output by applying certain filter queries on the output. The CEP that is used in speed layer and DAS which is used in batch layer were used for serving layer as well, even though it is shown as a separate architectural component in Figure 3.1.

Within this layer, a set of features derived by aggregating the complex events detected on speed layer is combined with the feature set derived on batch layer by performing statistical calculations and then resultant views are stored. Thus, users can apply certain filter queries on this final dataset within this layer or can input to external classifier to derive final output. Extreme usage related use-cases can be directly identified by considering only the real-time view. Those events are captured using filter queries on CEP at serving layer. Additionally, complex patterns which reflects grey callers with high confidence can be directly routed to output through serving layer.

3.1.5. Rule-based Classifier

Rule-based classifier is used for Grey call detection use case only. This module consists of set of classification rules used in bypass detection use case. Actually, classification rule is the filtering query. Serving layer just builds the subscriber profile by combining features derived on real-time and batch analytics and sends output to classifier. After that, within classifier module, fraud instances are filtered by applying filtering rules on subscriber profile. Even though same can be done on serving layer, classifier is used as separate component to make it a pluggable component. In future rule-based classifier could be replaced by supervised-learning based classifier. Rule-based classifier was implemented as Java program comprising set of filter queries.

3.2. Feature Selection and Algorithm Design

3.2.1. Grey call detection

This subsection describes the research approach used to address grey call detection problem. Fraudulent and non-fraudulent numbers are the only two classes available in Grey call detection. Also, there are significant differences in grey caller's behavior from country to country. Therefore, we approached grey call detection as supervised learning problem. After gathering datasets, first step is to identify complex patterns within the CDR stream. We studied past CDR pattern for verified grey callers in training dataset and identified six patterns with significant decisive power in grey call detection. Once complex patterns are identified those can be captured by executing CEP queries.

Second step is to identify a rich set of features that can be used to make detections within a short-time window. To support near real-time decision making by considering the caller behavior within short time-window, a rich feature set is essential. In our research we have identified a novel feature set by studying about both called-party stats and calling party behavior. Novel feature set composed of three main components. First component is a set of features derived based on the identified complex events. Next, component is the feature set calculated for short-time window by aggregating CDR. We have the option of calculating this in the speed or batch layer based on data rates of streams and length of the sliding window. Third component is a feature set calculated on batch layer based on by aggregating past CDR. These feature set was identified by observing calling party subscriber's behavior in training dataset and called party subscriber's context data.

Even though some of the fraud instances can be identified using complex events and stats for short-time window, in some cases past data is required to support decision making and to improve accuracy of the system. Thus, three parts of the feature set mentioned above is combined at the serving layer. Rules are then developed based on combined feature set to make detections effectively. These set of rules have been developed by studying verified fraud instances in the training data set.

3.2.1.1. Data sources and context data

To detect complex patterns three types of CDRs, namely Local, National, and International CDRs were obtained from different nodes in telecommunication network as discussed in Section 3.1. Table 3.1 describes the data fields available in Local CDR Stream. As this CDRs belong to calls originated by subscribers belong to the network under study, location and device related data are available in this stream. Table 3.2 presents fields in national CDR stream. As these calls originated by customers belongs to other operators, location details are not available. Table 3.3 describes data fields in international CDR stream.

Table 3:1 : Fields in Local CDR Stream.

Field Name	Field Type	Description
calling_party_id	String	Subscriber identity number of the user who originates the call
called_party_id	String	Subscriber identity number the user who is intended to receive call
originating_date_time	Integer	Date and time when call attempt reached to the system
duration	Double	The duration between answer time and disconnect time
location	String	Cell location of subscriber when originating call. Composed by combining Location Area Code (LAC) and Cell ID
imei	String	International Mobile Equipment Identity (IMEI) which is unique to device

Table 3:2: Fields in National CDR Stream.

Field Name	Field Type	Description
calling_party_id	String	Subscriber identity number of the user who originates the call
called_party_id	String	Subscriber identity number the user who is intended to receive call
originating_date_time	Integer	Date and time when call attempt reached to the system
opc	String	Origination Point Code
dpc	String	Destination Point Code
action	String	Action taken by Call Screening Server for attempt – “blocked” or “passed”

Table 3:3: Fields in International CDR Stream.

Field Name	Field Type	Description
calling_party_id	String	Subscriber identity number of the user who originates the call
called_party_id	String	Subscriber identity number the user who is intended to receive call
release_dir	String	Indicate which party disconnected the call initially
time	Integer	Date and time when call attempt reached to the system
duration	Double	The duration between answer time and disconnect time
call_dir	String	Indicates whether mentioned transaction is for incoming call to local network or outgoing call from local network.

We have used training dataset to identify complex patterns within CDRs, discover feature set, and design algorithms. Training dataset consists of 7,241,372 local CDRs belong to 284,351 distinct callers and there are 51 verified Onnet bypass instances within that. Also, there are 8,559,106 National CDRs within training data set belong to 1,153,409 other operator numbers and 328 numbers out of that was categorized as offnet bypass numbers. Additionally, we have included 5,217,259 international incoming and outgoing CDRs to identify complex patterns.

In addition to above dataset we have used the following context data as support data to generate the feature set:

- The IMEI numbers of handsets which was used by verified grey caller numbers within last year.
- Location Area Code (LAC) and Cell ID of the locations where verified grey caller numbers were operated within last year.
- The subscriber numbers who have received at least one call from verified grey caller numbers within last 3 months.
- Subscribers who have received at least one IDD call in last 30 days.
- Subscribers who have originated at least one IDD call in last 30 days.
- Total answer duration and maximum call duration of subscriber for incoming answered calls from network under study. Onnet or Offnet Subscribers who have received at least one call from network under study within last 30 days were considered.

- First calling date of Onnet subscribers.
- The first date on which Offnet number has terminated a call on considered network.
- Number of days subscriber has originated at least one call to network since first call date.
- Number of days offnet number has terminated at least one call to network since first call date.

3.2.1.2. Locating complex patterns and design CEP queries

Locating complex patterns in CDR is essential to identify fraudulent behaviors in near real time. Identifying such patterns is one of the major contributions in our research. Once complex patterns are identified those can be captured by executing Siddhi QL pattern queries on WSO_2 CEP. After analyzing three CDR streams we have identified six complex patterns that can be used in grey-call detection.

Figure 3.3 depicts complex pattern Type 1. In this scenario called party *B1* receives call from previously identified grey caller *AN1* at time *t* from another operator network, but it was blocked by operator at firewall. In this case relevant attempt recorded in the National CDR stream as calling party number *AN1* belongs to another operator. Immediately after Δt time same called party receives call from different calling party ID *AN2* which passed through firewall. *AN2* is also belongs to another operator and relevant event recorded in National CDR stream. According to training dataset, there is a fair chance that *AN2* being a grey caller as most of SIM boxes operate as cluster of SIMs and if one call failed from one SIM card in cluster, they would try through another SIM in the same cluster.

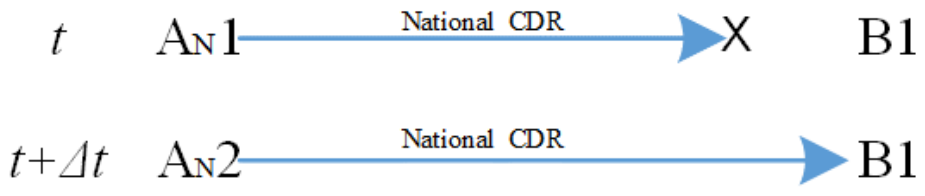

Figure 3:3: Complex Event Type 1.

Figure 3.4 depicts real-world example for Type 1 pattern. According to above details, we have implemented the siddhi query shown in Figure 3.5 to detect complex event Type 1. We have set Δt to 10 minutes in this case.

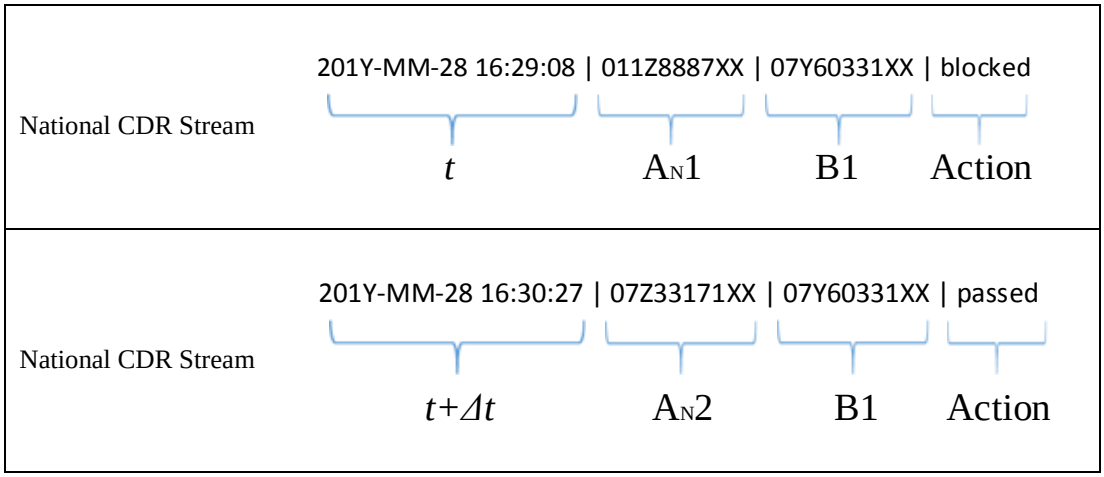

Figure 3:4: Sample Type 1 Complex event in CDR Stream.

```
from every a11 = offnetCDRStream[action == "blocked"]
-> b11 = offnetCDRStream[called_party_id == a11.called_party_id
and action == "passed"] within 10 min

select b11.calling_party_id as calling_party_id,
a11.called_party_id as called_party_id,
"01" as patternID,
b11.originating_date_time as detect_time

insert into patternStream_temp;
```

Figure 3:5: Siddhi Query to detect Complex Pattern Type 1.

Figure 3.6 depicts CDR event flow of complex event Type 2. First event happens at time *t* is similar to the first event in complex event Type 1. However, in this case second event comes through local CDR stream after Δt time as calling party number A_L2 who has originated the call belongs to same network under study. There is a fair chance that A_L2 to be a grey caller, if cluster of SIMs in SIMbox contains the SIMs belong to many networks. Figure 3.7 shows real-world example for Type 2 complex event located in training dataset. After studying this behavior, we have designed the Siddhi Query shown in Figure 3.8. In this case also we set Δt to be 10 minutes.

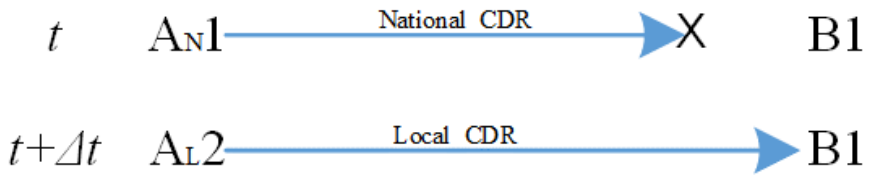

Figure 3:6 : Complex event Type 2.

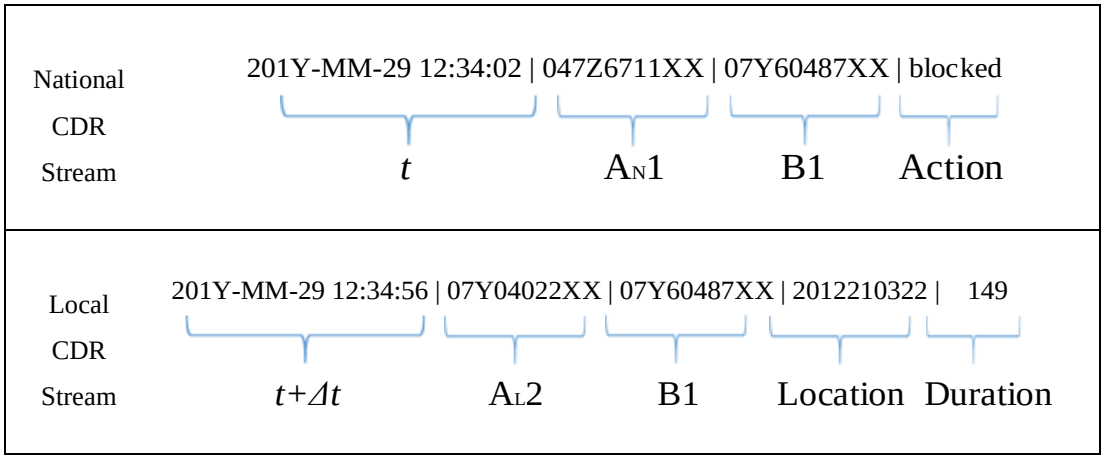

Figure 3:7: Sample Type 2 Complex event in CDR Stream.

```
from every a21 = offnetCDRStream[action == "blocked"]
-> b21 = onnetCDRStream[called_party_id == a21.called_party_id]
within 10 min

select b21.calling_party_id as calling_party_id,
a21.called_party_id as called_party_id,
"02" as patternID,
b21.originating_date_time as detect_time

insert into patternStream_temp;
```

Figure 3:8: Siddhi Query to detect Complex Pattern Type 2.

Figure 3.9 depicts complex pattern Type 3. In this scenario called party *B1* receives call from overseas number *AI1* at time *t*, but call was not answered by called party or disconnected intentionally. Since this is international incoming call corresponding event recorded at international CDR stream. Immediately after *⊿t* same called party receives call from different calling party number *AN2* which is belongs to another operator network. Therefore, the second event is recorded in national CDR stream.

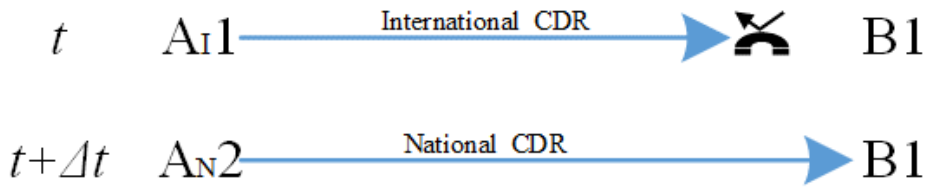

Figure 3:9: Complex event Type 3.

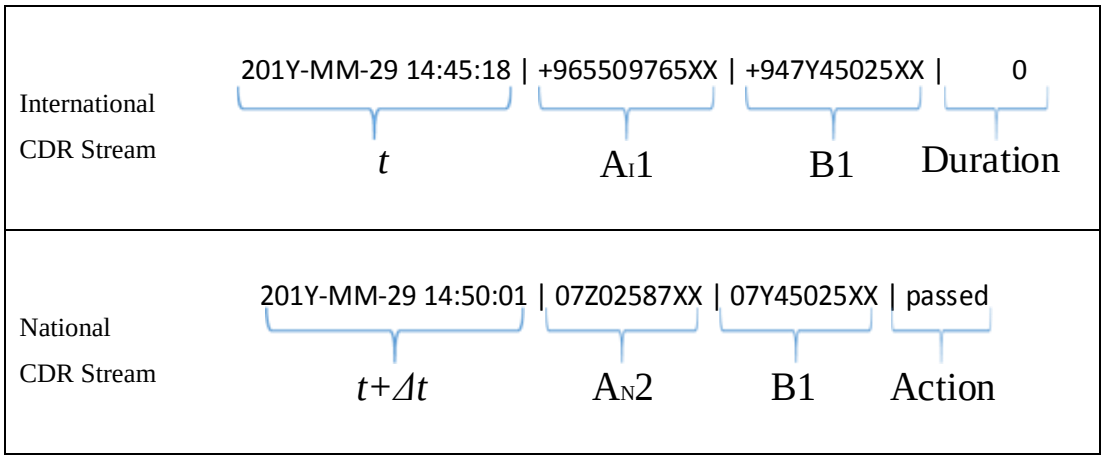

Figure 3:10: Sample Type 3 Complex event in CDR Stream.

According to our analysis of the training dataset, there is fair chance that *AN2* to be a grey caller. According to domain experts, most of international voice carriers who are utilizing these grey routes use mix of genuine and grey routes. If one call comes through proper international call route, next call may reach to destination SIM box route. Figure 3.10 demonstrates sample instance of complex event Type 3.

After studying the mentioned behavior, the Siddhi query shown in Figure 3.11 was developed. Compared to first event in complex event Type 1 and 2, first event of complex event type 3 and 4 happen more frequently as receiving international call attempt with zero duration is more probable. When Δt becomes smaller, decisive power of complex event is increased. Therefore, in this case, we have reduced Δt and set to 5 minutes.

```
from every a31 = intlCDRStream[duration == 0.0 and call_dir=="1"]
-> b31 = offnetCDRStream[called_party_id == a31.called_party_id
and action == "passed"] within 5 min

select b31.calling_party_id as calling_party_id,
a31.called_party_id as called_party_id,
"03" as patternID,
b31.originating_date_time as detect_time

insert into patternStream temp;
```

Figure 3:11: Siddhi Query to detect Complex Pattern Type 3.

Figure 3.12 depicts CDR event flow of complex event Type 4. First event happens at time t is similar to first event in complex event Type 3. But in this case, second attempt comes after Δt time from calling party number A_L2 which belongs to same operator's network under study. Thus, second event is recorded in local CDR stream. A_L2 could be a grey caller due to genuine and grey route mixing like in Type 3 complex event.

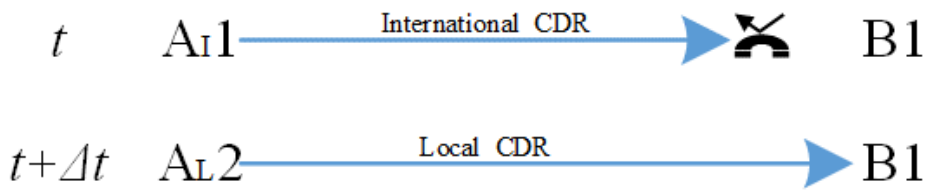

Figure 3:12: Complex event Type 4.

Figure 3.13 shows real-world example for Type 4 complex event located in training dataset. After studying this behavior, we have implemented the Siddhi query shown in Figure 3.14. In this case also, we have set Δt to 5 minutes.

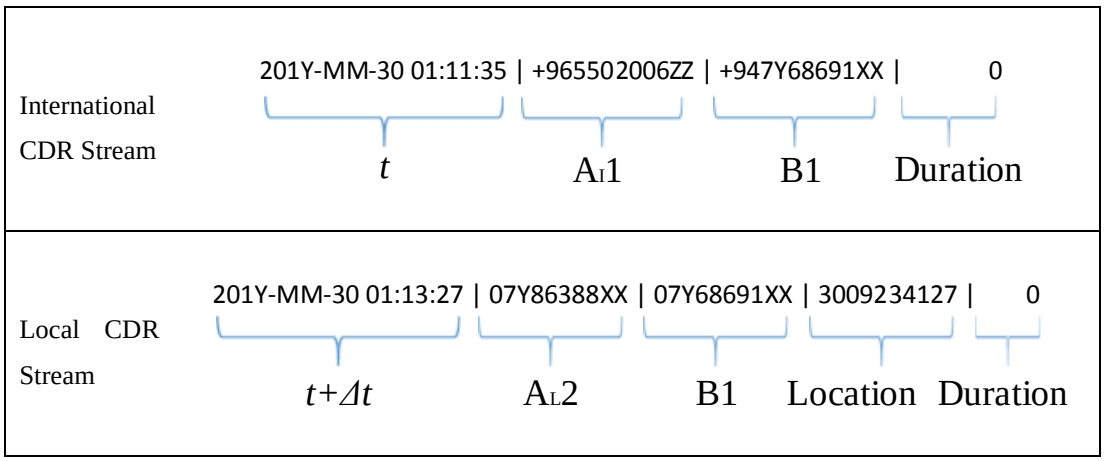

Figure 3:13: Sample Type 4 Complex event in CDR Stream.

```
from every a41 = intlCDRStream[duration == 0.0 and call_dir=="1"]
-> b41 = onnetCDRStream[called_party_id == a41.called_party_id]
within 5 min

select b41.calling_party_id as calling_party_id,
a41.called_party_id as called_party_id,
"04" as patternID,
b41.originating_date_time as detect_time

insert into patternStream_temp;
```

Figure 3:14: Siddhi Query to detect Complex Pattern Type 4.

Figure 3.15 depicts complex pattern Type 5. In this scenario, subscriber *B1* dials overseas number B_I1 at time *t*, but call was not answered by B_I1. This attempt generates event in International CDR stream. Immediately after Δt time *B1* receives call from different calling party number A_N2 belongs to another operator network and event is generated on national CDR stream. There is some probability to second call being a call from B_I1 to *B1* as a response to missed call, but that call may reach through SIM Box number A_N2 due to route mixing. Figure 3.16 shows sample instance of Type 5 complex event located in the training dataset.

Figure 3.17 represents Siddhi query to detect Type 5 complex event within real data. We set Δt to 5 minutes considering the frequency of first raw event of this complex event.

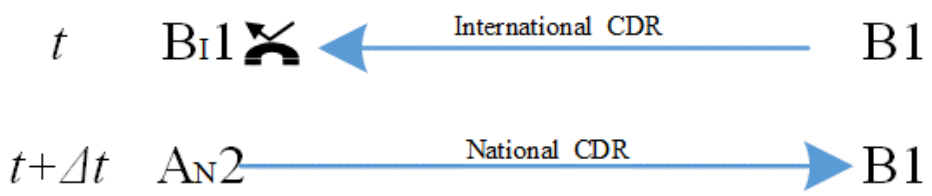

Figure 3:15: Complex event Type 5.

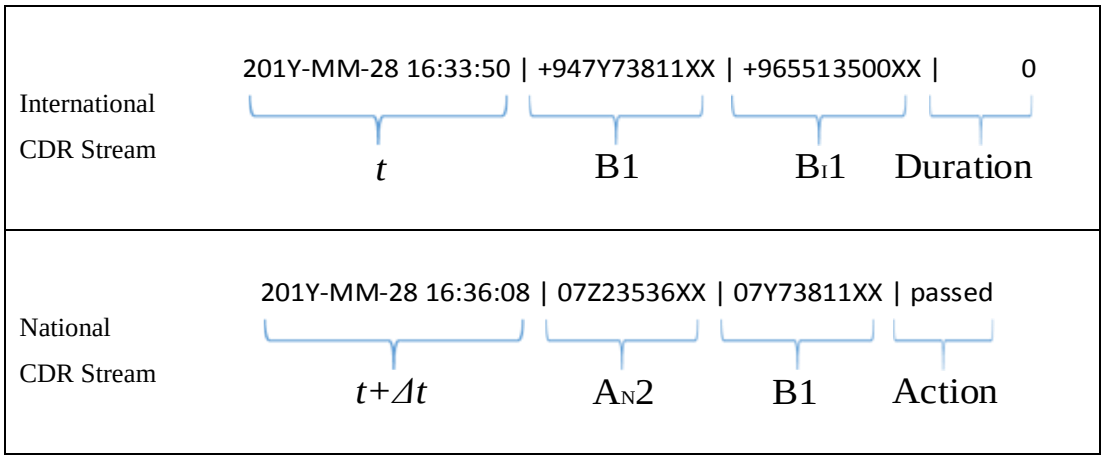

Figure 3:16: Sample Type 5 Complex event in CDR stream.

```
from every a51 = intlCDRStream[duration == 0.0 and call_dir=="0"]
-> b51 = offnetCDRStream[called_party_id == a51.calling_party_id
and action == "passed"] within 5 min

select b51.calling_party_id as calling_party_id,
a51.calling_party_id as called_party_id,
"05" as patternID,
b51.originating_date_time as detect_time

insert into patternStream_temp;
```

Figure 3:17: Siddhi Query to detect Complex Pattern Type 5.

Figure 3.18 shows complex pattern Type 6, where the first event in this scenario is similar to Type 5. Subscriber *B1* originate call to overseas number $B_{I}1$ at *t*, but was not answered by called party. This attempt generates event in international CDR stream. Immediately after Δt, *B1* receives call from different calling party number $A_{L}2$ belongs to same operator under study and event is recorded on local CDR stream. There could be possibility of receiving call from $B_{I}1$ to *B1* as a response to missed call and that call may reach through SIM box due to route mixing. Figure 3.19 shows sample instance of Type 6 complex event located in training dataset, and the corresponding query is shown in Figure 3.20.

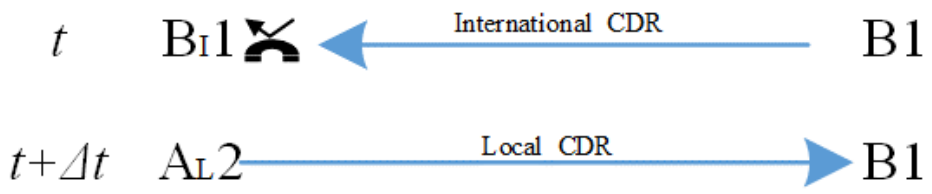

Figure 3:18: Complex event Type 6.

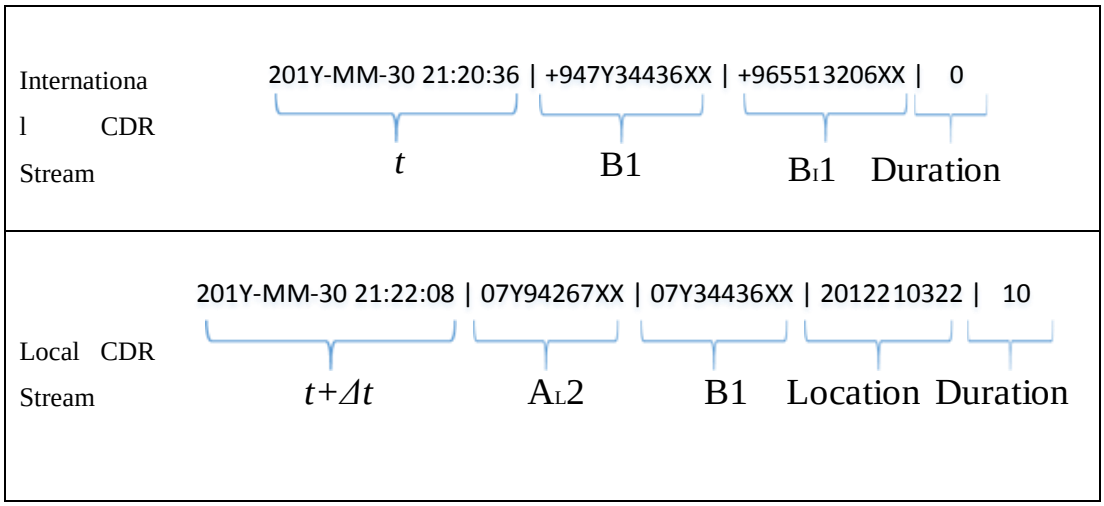

Figure 3:19: Sample Type 6 Complex event in CDR Stream.

```
from every a61 = intlCDRStream[duration == 0.0 and call_dir=="0"]
-> b61 = onnetCDRStream[called_party_id == a61.calling_party_id]
within 10 min

select b61.calling_party_id as calling_party_id,
a61.calling_party_id as called_party_id,
"06" as patternID,
b61.originating_date_time as detect_time

insert into patternStream_temp;
```

Figure 3:20: Siddhi Query to detect Complex Pattern Type 6.

Figure 3.21 depicts event flow inside the execution plan which was deployed on WSO2 CEP to detect six patterns mentioned above. Local, National, and International CDR streams are exported to execution plan as *onnetCDRStream*, *offnetCDRStream*, and *intlCDRStream*, respectively. After subjecting to pattern queries, resultant pattern detections are directed to MySQL database through stream names as *PatternDetectStream*.

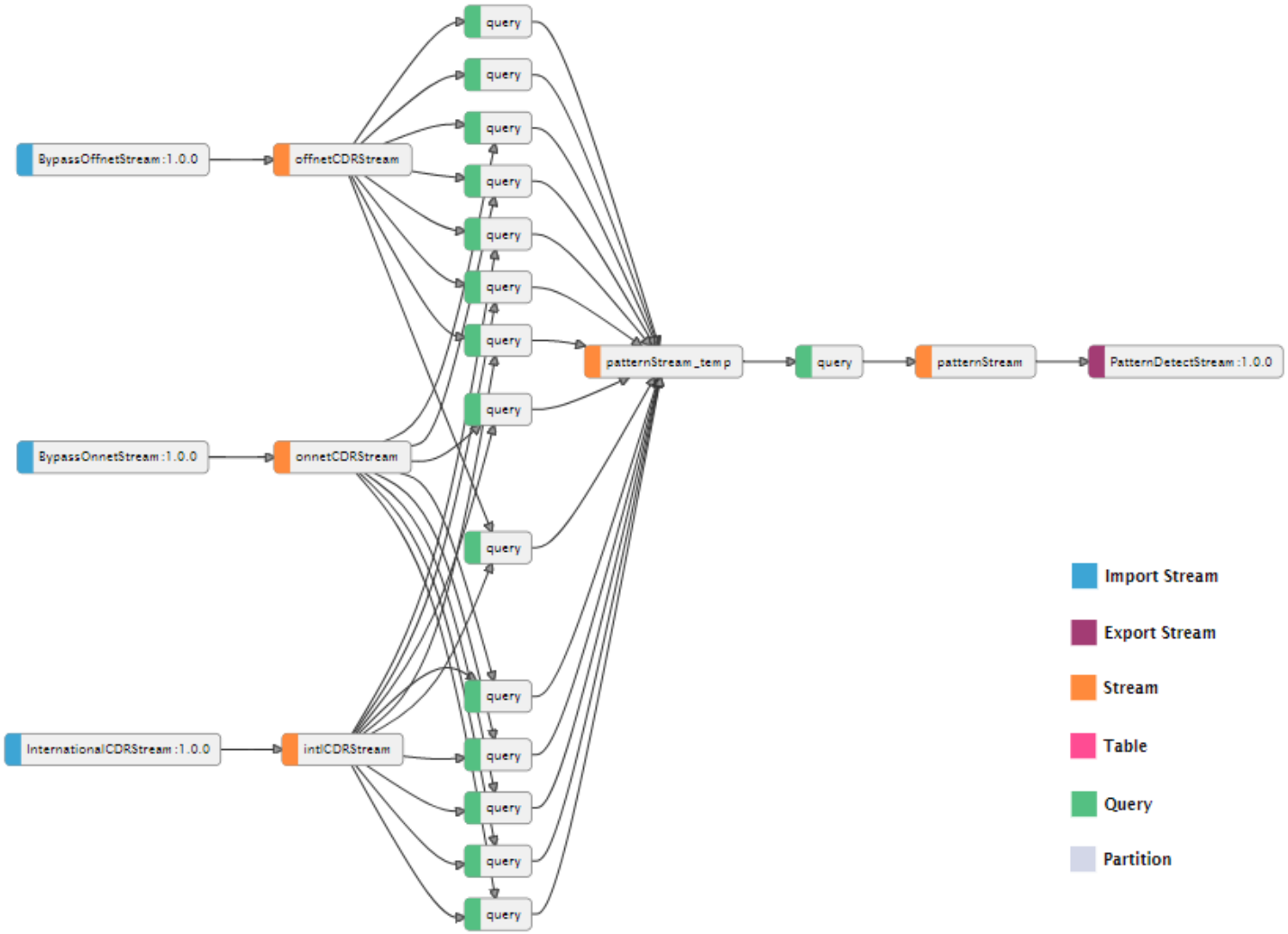

Figure 3:21: Overall event flow in execution plan used for pattern detection.

3.2.1.3. Feature set and detection rules for Onnet bypass detection

Next part of this research is identifying feature set which supports near real-time detections and derive detection rules based on that feature set. Since selection of features and detection rules is slightly different in Onnet and Offnet bypass scenarios, this section only focuses about Onnet Bypass Detection. Section 3.2.1.4 discuss about Offnet bypass detection.

Feature set for bypass detection is divided into three parts as complex events, short-time window, and past data based features. Table 3.4 shows the complex-event-based feature set. These values are generated by aggregating complex event count for the considered time span. Since second event of Type 2, 4, and 6 complex patterns are generated by onnet numbers, we have used those complex event types for onnet bypass detection.

Table 3:4: Pattern based feature set for Onnet bypass detection.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
P2	Number of Type 2 complex events generated by calling party number
P4	Number of Type 4 complex events generated by calling party number
P6	Number of Type 6 complex events generated by calling party number

Table 3.5 lists the feature set generated by focusing on one hour sliding window. Initial plan was to calculate one hour sliding window based stats on WSO_2 CEP. But based on available hardware resources and CEP performance, these stats may need to be calculated on WSO_2 DAS using Spark SQL Scripts in most of the cases. Along with CDRs for considered one-hour window, context data also used to calculate these set of attributes. Attributes which were calculated with help of context data mentioned in Section 3.2.1.1 is denoted in Table 3.5 using *"*"* mark at end of the attribute name. Using DAS's functionality to schedule Spark SQL scripts, stats for one hour sliding window were repeatedly calculated once per every 30 minutes and joined with pattern based features to come with feature set for short-time window.

Table 3:5: Feature set used in Onnet bypass detection based on short-time window.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
og_cnt_hour	Total outgoing call count originated by given Subscriber
og_dcnt_hour	Different numbers dialed by Subscriber
max_cell_hour	The cell id in which subscriber was stayed while taking most number of calls.
og_tot_dur_hour	Total outgoing call duration by subscriber
cell_count_hour	Total number of distinct cell ids from which subscriber has originated at least one call
imei_count_hour	Number of different IMEI numbers used by calling party.
grcell_hour *	Does this subscriber has originated calls from cell location from which previously identified grey callers also originated calls? Value is set to 1 if answer is yes otherwise value is set to 0.
grey_imei_hour *	Does this subscriber has originated calls from the device with IMEI number which was previously used by verified grey caller? Value is set to 1 if answer is yes otherwise value is set to 0.
grb_dcnt_in_hour *	Number of distinct called party numbers dialed by this customer which has received call from verified grey caller's numbers previously.
iddb_dcnt_in_hour *	Distinct called party numbers dialed by this number who have received IDD calls in past.
iddb_dcnt_out_hour *	Distinct called party numbers dialed by this number who have dialed IDD calls in past
ic_tot_dur_hour	Total call duration of subscriber for incoming answered calls from network under study within one hour
ic_max_dur_hour	Maximum call duration of subscriber for incoming answered calls from network under study within one hour

Some of the fraud instances can be directly identified by focusing on complex pattern based and one-hour sliding window based features. But fraudsters use advanced techniques to simulate normal users' behavior. So along with short-term data, past data also need to be used to make detections at earliest with high accuracy. So using the event data persisted in WSO_2 DAS, we have built user behavior recent for past 24 hours. Table 3.6 shows feature set built for 24-hours sliding window. Attributes which were calculated with help of context data is denoted in Table 3.6 using *"*"* mark at end of the attribute name.

Table 3:6: Feature set calculated using past data for Onnet bypass detection.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
og_cnt	Total outgoing call count originated by given Subscriber
og_dcnt	Different numbers dialed by Subscriber
og_cnt_other	Total outgoing call count by Subscriber to other operator numbers
og_dcnt_other	Distinct other operator numbers dialed by Subscriber
og_tot_dur	Total outgoing call duration by subscriber
cell_count	Total number of distinct cell ids from which subscriber has originated at least one call
max_cell	The cell id in which subscriber was stayed while taking most number of calls.
imei_count	Number of different IMEI numbers used by calling party.
grey_cell *	Does this subscriber has originated calls from cell location from which previously identified grey callers also originated calls? Value is set to 1 if answer is yes otherwise value is set to 0.
grb_dcnt_in *	Number of distinct called party numbers dialed by this customer which has received call from verified grey caller's numbers previously.
grey_imei *	Does this subscriber has originated calls from the device with IMEI number which was previously used by verified grey caller? Value is set to 1 if answer is yes otherwise value is set to 0.
iddb_dcnt_in *	Distinct called party numbers dialed by this number who have received IDD calls
iddb_dcnt_out *	Distinct called party numbers dialed by this number who have dialed IDD calls
ic_tot_dur *	Total call duration of subscriber for incoming answered calls from network under study
ic_max_dur *	Maximum call duration of subscriber for incoming answered calls from network under study
og_idd_dcnt *	Distinct IDD numbers dialed by subscriber
ic_idd_dcnt *	Number of distinct IDD numbers which dialed this subscriber
day_count *	Number of days subscriber has originated calls since subscriber's first call date.
first_call *	The date of the first call originated by subscriber

Context data was stored in MySQL tables and imported to Spark script by creating temporarily tables using *CarbonJDBC* as provider [37]. Persisted events stored in DAS was imported to Spark script by creating temporarily tables using *CarbonAnalytics* as provider [37]. Even though WSO_2 DAS stores persisted data on MySQL as per our configuration, those data stored using its own format. As context data was in typical MySQL format, we had to use two analytic providers to implement Spark script.

Figure 3.22 shows sample Spark queries which were included in Spark script on WSO_2 DAS. As mentioned earlier to access data through Spark script creation of temporarily table which points to permanent table is required. Then users can insert data to temporarily table which automatically updates pointed permanent table.

```
CREATE TEMPORARY TABLE ONNET_OG_SUMMARY USING CarbonAnalytics
OPTIONS
(tableName "ONNETOGSUMMARY",schema "calling_party_id STRING,
og_cnt INT, og_dcnt INT, og_ans_count INT, og_max_dur DOUBLE,
og_tot_dur DOUBLE, cell_count INT,og_cnt_other INT,og_dcnt_other
INT", primaryKeys "calling_party_id");

INSERT OVERWRITE TABLE ONNET_OG_SUMMARY SELECT
calling_party_id,
count(called_party_id) as og_cnt,
count(distinct(called_party_id)) as og_dcnt,
sum(CASE WHEN duration!=0 THEN 1 ELSE 0 END) as og_ans_count,
sum(duration) as og_tot_dur,max(duration) as og_max_dur,
count(distinct(location)) as cell_count,
count(CASE WHEN called_party_id not like "77%"  and
called_party_id not like "76%" THEN called_party_id ELSE Null
END) as og_cnt_other,
count(distinct(CASE WHEN called_party_id not like "77%"  and
called_party_id not like "76%" THEN called_party_id ELSE Null
END)) as og_dcnt_other
from BYPASS_ONNET_DAILY group by calling_party_id;
```

Figure 3:22: Sample Spark Query used to calculate attributes.

Feature set based on for 24-hours sliding window was calculated once per hour and joined with real-time view to apply filtering rules. We developed 14 filtering rules based on behavior of verified grey caller numbers on training data set. Three of these rules are derived based on real-time view, seven rules build based on 24-hour stats, and remaining four rules are derived using both real time and 24-hour stats. Table 3.7 shows sample rule used in Onnet bypass detection. It is important to note that in some cases we have used composite attributes, which was derived by subjecting raw attributes to simple mathematical operations.

Table 3:7: Example filtering criteria in detection rule used in Onnet bypass detection.

Attribute/Composite Attribute	Condition
og_cnt_hour	>0
P2	>0
P4	>0
P6	>0
(P2+P4+P6)/og_cnt	>0.01
grcell_hour	>0
ic_tot_dur_hour	<1000

These rules were developed after observing the behavior of verified bypass numbers in training dataset. Once initial filtering criteria has developed, we applied those to training dataset and obtained results. If accuracy level and detection delay are not met expected levels, we adjusted thresholds of rules and applied again for training dataset. Same procedure is repeated until comprehensive set of logics which meets expected accuracy and detection delay were obtained. Finally, derived set of rules were applied on the test dataset and the results are discussed in Chapter 4.

3.2.1.4. Feature set and detection rules for Offnet bypass detection

Similar to Onnet bypass detection, feature set is divided into three parts named as complex events, short-time window, and past data based features. Table 3.8 shows the complex event based feature set. These values are generated by aggregating complex event count within the considered time span. Since second event of Type 1, 3 and 5 complex patterns are generated by offnet number, we have used those complex event types for offnet bypass detection.

Table 3:8: Pattern based feature set for Offnet bypass detection.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
P1	Number of Type 1 complex events generated by particular calling party number
P3	Number of Type 3 complex events generated by particular calling party number
P5	Number of Type 5 complex events generated by particular calling party number

Table 3.9 describes the feature set generated by focusing on one hour sliding window. These stats were calculated on WSO_2 DAS using Spark SQL script by accessing persisted event data. Main data source was the stored events from National CDR stream. Along with CDRs for considered one-hour time window, context data also used to calculate these set of attributes. Attributes which were calculated with help of context data is denoted in Table 3.9 using *"*"* mark at end of the attribute name. Similar to Onnet bypass scenario, stats for one-hour sliding window were repeatedly calculated once per every 30 minutes and joined with pattern based features to come with feature set for short-time window. It is important to note that that, because operator do not have location details and handset details of subscriber of another operator network, feature set generated for Offnet bypass detection is relatively smaller compared to Onnet bypass detection.

Table 3:9: Feature set used in Offnet bypass detection with one-hour time window.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
og_cnt_hour	Total incoming call count originated by offnet number to network under study
og_dcnt_hour	Distinct number count dialed by offnet number
grb_dcnt_in_hour*	Number of distinct called party numbers dialed by this Offnet number which has received call from verified grey caller's numbers previously.
iddb_dcnt_in_hour*	Distinct called party numbers dialed by this number who have received IDD calls previously
iddb_dcnt_out_hour*	Distinct called party numbers dialed by this number who have dialed IDD calls previously
ic_tot_dur_hour	Total call duration of incoming answered calls by this offnet number from network under study within considered time span
ic_max_dur_hour	Maximum call duration of incoming answered calls by this offnet number from network under study within considered time span

In this case also some of the fraud instances can be directly identified by focusing on complex patterns and one hour sliding window based features. Because of advanced techniques used by fraudsters to simulate normal user's behavior, decision making process cannot only rely on real-time data. So along with near real-time data, past data was also required to make detections at earliest with high accuracy. So, using the event data persisted in WSO_2 DAS we have built user behavior for recent 24 hours. Table

3.10 shows feature set built for 24-hours sliding window. Attributes which were calculated with help of context data is denoted in Table 3.10 using "*" mark at end of the attribute name. Since location and device information not available for offnet numbers, limited number of features are available for Offnet bypass detection. Similar to Onnet bypass scenario, 24-hour behavior was calculated on WSO_2 DAS using Spark scripts. Feature set based on for 24-hours sliding window was calculated once per hour and joined with real time view to apply filtering rules.

Table 3:10: Feature set calculated using past data for offnet bypass detection.

Attribute	Description
calling_party_id	Calling party Number (Primary Key)
og_cnt	Total incoming call count originated by offnet number to network under study
og_dcnt	Distinct number count dialed by offnet number
grb_dcnt_in*	Number of distinct called party numbers dialed by this offnet number which has received call from verified grey caller's numbers previously.
iddb_dcnt_in*	Distinct called party numbers dialed by this offnet number who have received IDD calls
iddb_dcnt_out*	Distinct called party numbers dialed by this number who have dialed IDD calls
ic_tot_dur*	Total call duration of subscriber for incoming answered calls from network under study
ic_max_dur*	Maximum call duration of subscriber for incoming answered calls from network under study
day_count*	Number of days Offnet number has originated calls since first call termination date to network under study.
first_call*	The date of first incoming from offnet number received to network under study

We have developed 16 filtering rules based on the behavior of verified grey caller numbers on training data set. Six of these rules were derived based on real-time view, eight were build based on 24-hour stats, and he remaining two rules derived using both real time and 24-hour stats. We have used composite attributes which was derived by subjecting raw attributes to simple mathematical operations within some rules.

These rules were developed after observing the behavior of verified bypass numbers in training dataset. Once initial filtering criteria has developed, we have applied those to training dataset and obtained results. When the accuracy level and detection delay are not within the expected levels, we adjusted thresholds of rules and applied again

for training dataset. Same procedure is repeated until comprehensive set of logics that meet expected accuracy and detection delay is obtained. Finally, derived set of rules were applied on test dataset.

When we consider offnet bypass scenario, summarized dataset consists of behavior of numbers belongs to another operator's mobile network as well as fixed telephony network. Since fixed telephone network operators provide PABX, hotline and call center solutions, sometimes generates higher amount of calls to distinct numbers. So, we have applied different set of rules to fixed telephone operators numbers and mobile network operator numbers. Also, numbers belong to one wireless fixed network operator have shown abnormal behavior in some instances. So, we had to use specified set of rules to capture the numbers belongs to particular operator.

3.2.2. Extreme usage detection

This section describes the method which was used to identify different kind of fraudulent activities related to extreme usage scenarios related to premium rated numbers. We have focused on five major scenarios and in each subsection, we describe the CDR pattern observed within each scenario and how those are captured in our system.

3.2.2.1. Dial and disconnect scam

This is most frequent category of fraudulent activity related to premium rate telephone numbers. In this scenario, fraudsters connected to international voice network multicast call attempts to range of valid telephone numbers in selected network in selected country. When multicasting those missed call attempts, fraudsters replace original calling party with premium rated telephone number or telephone number of country to which call termination rate is higher than LKR 50. Fraudsters normally use premium rated or high cost destination numbers for these kind of fraudulent activities as their profit can be maximized when termination cost per one minute become higher. In all the sample insistences we have found, fraudsters used CLI belongs to destination to which calling cost per one minute is higher than LKR 50. To nullify the effect of this fraudulent activity, these incoming fraudulent call attempts toward local network need to be identified before customers start to respond those in large scale.

We have studied similar fraud cases within one month and identified the incoming call pattern in this scenario. Table 3.11 shows hourly statistics of two sample fraud instances in first three hours of their operation. These statistics are calculated by grouping incoming IDD calls with respect to calling party number. So, we can clearly see that these fraudulent numbers dial considerably higher amount of distinct numbers than normal users. Also, these two calling party numbers belongs to countries such as Surinam and Somalia and call cost per one minute is LKR 80 to both countries. Therefore, using this information we developed execution plan in WSO_2 CEP to detect these scenarios. First, we took rate sheet from operator websites. Table 3.12 shows sample entries from rating table. Action value is set to one when a particular destination is required to consider in number level analysis.

Table 3:11: Instances of Dial and Disconnect Scam.

	Calling Party	**Hour**	**No of distinct called party nos dialed by this no**
Case 1	5977619782	2017-11-04 13	30
	5977619782	2017-11-04 14	118
	5977619782	2017-11-04 15	99
Case 2	252800778114	2017-11-04 11	40
	252800778114	2017-11-04 13	131
	252800778114	2017-11-04 15	153

Table 3:12: Rating table with destination number prefixes.

Country Code	**Destination Digits**	**Cost (LKR)**	**Country**	**Destination Name**	**Action**
87	87	900	INMARSAT	INMARSAT	1
46	4674	900	SWEDEN	SWEDEN SPECIAL SERVICE	1
355	3554249	500	ALBANIA	ALBANIA PREMIUM	1
355	35534606	500	ALBANIA	ALBANIA PREMIUM	1
355	35534608	500	ALBANIA	ALBANIA PREMIUM	1
355	35535505	500	ALBANIA	ALBANIA PREMIUM	1
597	597	80	SURINAM	SURINAM	1
252	252	80	SOMALIA	SOMALIA	1

Next, we developed Siddhi query on WSO_2 CEP to analyze International CDR stream. This query analyzes all the incoming attempts toward the considered network and calculates distinct numbers dialed by each calling party number within one hour sliding window. Also, we have extracted leftmost digits of calling party number for different lengths to use those prefixes in the next step. Figure 3.23 shows sample code snippet used. Then we have matched output stream of the query mentioned in Figure 3.23 with rating table. Figure 3.24 shows the Siddhi query used to match intermediate stream with rating table. Even though rating table is stored in MySQL table, we need to access it from CEP query in this scenario. So prior to executing this query, we have defined event table named as *HighCostDestTable* and pointed that to rating table because event tables can be directly access from CEP query.

Finally, filtering query in Figure 3.25 was used to filter out fraudulent numbers. Using the past cases, we identified that receiving calls from premium rated or high cost calling party number to more than 10 distinct subscribers could be considered as suspicious situation. Figure 3.26 shows the overall event flow of execution plan deployed to detect dial and disconnect scam.

```
from intlCDRStream[call_dir=='1' and duration<10 and
str:length(calling_party_id)>7]#window.time( 60 min )

select calling_party_id,
distinctcount(called_party_id) as dst_callednum_count,
min(time) as firstattempttime,max(time) as lastattempttime,
str:substr(calling_party_id,0,2)as firstdigit2,
str:substr(calling_party_id,0,3)as firstdigit3,
str:substr(calling_party_id,0,4)as firstdigit4,
str:substr(calling_party_id,0,5)as firstdigit5,
str:substr(calling_party_id,0,6)as firstdigit6,
str:substr(calling_party_id,0,7)as firstdigit7,
str:substr(calling_party_id,0,8)as firstdigit8
group by calling_party_id

insert into tmpMFSummary8;
```

Figure 3:23: Query used for event aggregation to detect Dial and Disconnect Scam.

```
from tmpMFSummary8#window.unique(calling_party_id) as
unqAttemptSummary8 join HighCostDestTable on

unqAttemptSummary8.firstdigit2==HighCostDestTable.Dest_Digits or
unqAttemptSummary8.firstdigit3==HighCostDestTable.Dest_Digits or
unqAttemptSummary8.firstdigit4==HighCostDestTable.Dest_Digits
or unqAttemptSummary8.firstdigit5==HighCostDestTable.Dest_Digits
or unqAttemptSummary8.firstdigit6==HighCostDestTable.Dest_Digits
or unqAttemptSummary8.firstdigit7==HighCostDestTable.Dest_Digits
or unqAttemptSummary8.firstdigit8==HighCostDestTable.Dest_Digits

select unqAttemptSummary8.calling_party_id,
      unqAttemptSummary8.dst_callednum_count,
      unqAttemptSummary8.firstattempttime,
      unqAttemptSummary8.lastattempttime

insert into TmpMissedCallFruadStream;
```

Figure 3:24: Query used to join Rating table with aggregated data.

```
from TmpMissedCallFruadStream[dst_callednum_count>10]

select calling_party_id,dst_callednum_count,firstattempttime,
lastattempttime

output every 30 sec

insert into MissedCallFruadStream;
```

Figure 3:25: Filtering Query used to detect Dial and Disconnect Scam.

3.2.2.2. Outbound dialing due to fake text messages

In this scenario, fraudsters use fancy messages instead of missed calls to persuade normal subscribers to dial back to premium rated numbers. These scenarios cannot be detected by relaying on incoming text messages as those messages can reach to customer in many ways. So, this scenario need to be captured using outgoing call attempts at early stage before significant number of users dial those premium numbers. Table 3.13 shows instances for Outbound dialing fraud due to fake text messages received in form of SMS. In these cases, distinct calling party numbers dials the same high-cost destination number or number range. Country code 248 and 291 belongs to Seychelles and Eritrea, respectively, where cost per one minute is LKR 80 for both the countries.

Table 3:13: Instances for Outbound Dialing due to fake Text Messages

	Called party id	**Hour**	**No of distinct calling party ids dialed this number**
Case 1	2486427007	2017-11-04 17	742
	2486427007	2017-11-04 18	495
	2486427007	2017-11-04 18	79
	2486427007	2017-11-04 19	104
Case 2	2917185957	2017-11-04 12	18
	2917185957	2017-11-04 15	14
	2917185957	2017-11-04 16	14

We have implemented CEP Execution plan to detect this scenario after studying past instances. Implementation is similar to the methodology in Section 3.2.2.1. However, instead of incoming international call attempts toward local network in International CDR stream, outgoing international call attempts originated from local network is considered. Also, distinct calling party number count is obtained with respect to each called party number. Then resultant intermediate stream is joined with rating table and cases related to high cost destinations are filtered. Then simple filtering query is applied, and the output is obtained. According to past cases, we have identified that more than ten distinct numbers dials same premium rated or high cost calling party number within one hour, such event could be fraudulent activity. Figure 3.27 shows overall event flow of execution plan deployed to detect Outbound dialing scenarios due to fake text messages.

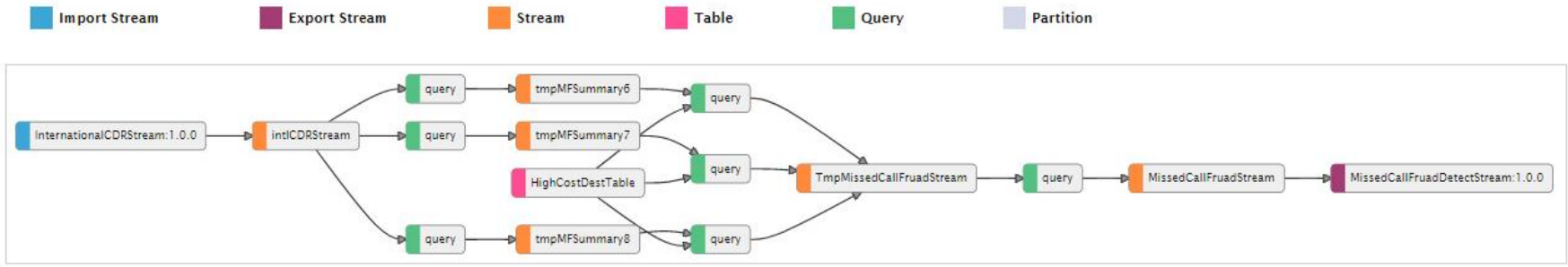

Figure 3:26: Event flow of execution plan used to identify Dial and Disconnect Fraud.

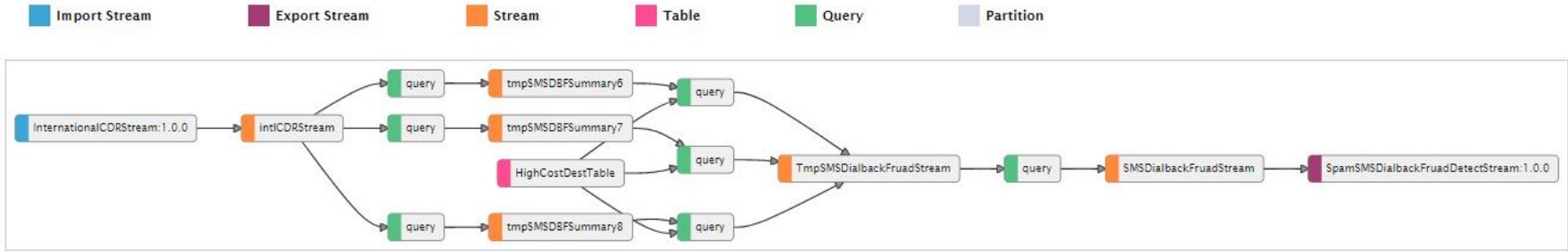

Figure 3:27: Event flow of execution plan used to detect Outbound dialing due to fake text messages.

3.2.2.3. Inbound roamer fraud

Inbound roamer fraud is another important scenario in which we can observe extreme usage behavior. Fraudsters latch foreign SIM card on local network and originate calls to premium rated or high cost destinations. Even though foreign operator need to pay those charges to local operator according to agreements, local operator need to track those fraudulent incidents and inform foreign operator as there is some associated risk in termination of roaming agreements and not paying excessive charges when usage charges due to fraudulent activity is very high.

Table 3.14 shows example case of roaming fraud. Calling party id means MSISDN of foreign network subscriber. Within few hours those inbound roamers have made significant number of calls to three high cost destinations. Country code 224, 371, and 232 respectively belongs to Guinea, Latvia, and Sierra Leone and to each country call termination charges for one minute is LKR 80. As roaming call charges are normally higher than normal call charges, normal roaming customers do not take long duration calls to costly destinations. But in this case within few hours fraudster has originated more than 100 minutes via each SIM card and average call duration is above 10 minutes in eight of those cases.

Table 3:14: Sample instances of Inbound Roamer Fraud.

Calling Party ID	Number Level	Attempt Count	Total Call Duration	Average Call Duration
66X493032YY	224	12	148.7627	12.3969
66X491347YY	224	11	195.359	17.7599
66X493153YY	224	10	166.724	16.6724
66X489321YY	224	10	182.2201	18.222
66X493135YY	224	10	174.7128	17.4713
66X489175YY	224	10	186.3567	18.6357
66X182460YY	224	9	163.502	18.1669
66X491302YY	224	9	147.5956	16.3995
66X493153YY	371	9	0	N/A
66X493153YY	232	8	0.4713	0.2356

Then, we developed CEP query to analyze International CDR stream. This query analyzes all the outgoing international call attempts from considered network and calculated attempt count, answer call count, and sum of duration dialed by each calling party number to each distinct called party number within 90 minutes sliding window. Also, we have extracted leftmost digits of called party number for different lengths to use those prefixes in next step. Figure 3.28 shows sample code snippet used.

```
from intlCDRStream[call_dir=='0' and
str:length(called_party_id)>7]#window.time( 90 min )

select calling_party_id,called_party_id,
str:concat(calling_party_id,called_party_id) as cus_symbol,
count(time) as attempt_count,
sum(duration>0) as ans_count,sum(duration) as tot_duration,
min(time) as firstattempttime,max(time) as lastattempttime,
str:substr(called_party_id,0,2)as firstdigit2,
str:substr(called_party_id,0,3)as firstdigit3,
str:substr(called_party_id,0,4)as firstdigit4,
str:substr(called_party_id,0,5)as firstdigit5,
str:substr(called_party_id,0,6)as firstdigit6,
str:substr(called_party_id,0,7)as firstdigit7,
str:substr(called_party_id,0,8)as firstdigit8
group by calling_party_id,called_party_id

insert into tmpRoamPABXFSummary8;
```

Figure 3:28: Aggregation query used in execution plan used for inbound roamer fraud detection.

Then rating table was exported as an event table and joined with output stream of query mentioned in Figure 3.28 to obtain result of prefix matching and acquire only calling party numbers which has dialed called party number ranges defined in rating table. Figure 3.29 shows the CEP query used to match intermediate stream generated as output of query mentioned in Figure 3.28 with rating table.

After that intermediate query shown in Figure 3.30 was used to calculate attempt count, answer call count, and sum of duration dialed by each calling party number to distinct called party number ranges. Three leftmost digits were considered when summarizing usage with respect to destination number level. Finally, filtering query mentioned in Figure 3.31 used to filter out fraudulent numbers. Figure 3.34 shows the event flow within execution plan deployed to detect inbound roamer fraud.

```
from tmpRoamPABXFSummary8#window.unique(cus_symbol) as
unqRMOutSum8 join HighCostDestTable on

unqRMOutSum8.firstdigit2==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit3==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit4==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit5==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit6==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit7==HighCostDestTable.Dest_Digits or
unqRMOutSum8.firstdigit8==HighCostDestTable.Dest_Digits

select
unqRMOutSum8.calling_party_id,unqRMOutSum8.called_party_id,
unqRMOutSum8.attempt_count,unqRMOutSum8.ans_count,
unqRMOutSum8.tot_duration,unqRMOutSum8.firstattempttime,
unqRMOutSum8.lastattempttime

insert into TmpRoamPABXFruadStream;
```

Figure 3:29: Siddhi query used to match intermediate stream with rating table used to detect inbound roamer fraud.

```
from TmpRoamPABXFruadStream

select calling_party_id,
str:substr(called_party_id,0,3) as dialed_range,
str:concat(calling_party_id,str:substr(called_party_id,0,3)) as
cus_symbol_rng,
attempt_count,ans_count,tot_duration,
firstattempttime,lastattempttime

insert into TmpRoamPABXFruadStream2;


from TmpRoamPABXFruadStream2#window.unique(cus_symbol_rng)

select calling_party_id,dialed_range,
sum(attempt_count) as tot_attempt_count,
sum(ans_count) as tot_ans_count,
sum(tot_duration) as final_tot_duration,
min(firstattempttime) as firstattempttime1,
max(lastattempttime) as lastattempttime1
group by calling_party_id,dialed_range

insert into TmpRoamPABXFruadStream3;
```

Figure 3:30: Intermediate query used to calculate usage of each calling party number to distinct premium number levels.

```
from TmpRoamPABXFruadStream3[str:substr(calling_party_id,0,2)
!="94" and ((tot_attempt_count>9 and  final_tot_duration>3600)
or final_tot_duration>(tot_ans_count*300))]

select calling_party_id,dialed_range,
tot_attempt_count as attempt_count,
tot_ans_count as ans_count,
final_tot_duration as tot_duration,
firstattempttime1 as firstattempttime,lastattempttime1 as
lastattempttime

insert into inRoamFruadStream;
```

Figure 3:31: Siddhi query used to detect inbound roamer fraud and high usage scenarios.

3.2.2.4. PABX hacking fraud

In this scenario hackers gain the access to the PABX system using system vulnerability and generate large number of calls to premium rated destinations without any intention of actual customer. Table 3.15 shows sample instance of PABX hacking fraud. Fraudsters has originated huge number attempts to high cost number level belongs to country called Serbia to which call termination cost for one minute is LKR 80. Fraudster tries 81 attempts within one-hour time span and average call duration is comparably high. In some cases, like call center solutions, normal PABX customer makes higher number of calls similar to this but average call duration is comparably low.

Table 3:15: Sample instance of PABX hacking fraud.

Calling Party ID	Number Level	Attempt Count	Total Duration	Average Call Duration
9411Y4441XX	381	81	453.7167	7.438

Therefore, Siddhi QL execution plan can be deployed to identify fraudulent behavior. Instead of creating different execution plan, this scenario was detected by adding another filtering query to execution plan mentioned in Section 3.2.2.3 as detection can be made using the same attributes. Figure 3.32 shows the filtering query used to detect PABX hacking fraud.

```
from TmpRoamPABXFruadStream3[(str:substr(calling_party_id,0,2)
=="94" and str:length(calling_party_id)==11 and
str:substr(calling_party_id,0,4) !="947Y" and
str:substr(calling_party_id,0,4) !="947Z") and
((tot_attempt_count>9 and  final_tot_duration>3600) or
final_tot_duration>(tot_ans_count*300))]

select calling_party_id,dialed_range,
tot_attempt_count as attempt_count,
tot_ans_count as ans_count,
final_tot_duration as tot_duration,
firstattempttime1 as firstattempttime,
lastattempttime1 as lastattempttime

insert into PABXFraudStream;
```

Figure 3:32: Filtering query used to detect PABX hacking fraud

3.2.2.5. Malware originated fraudulent calls

In this scenario, malicious software installed on subscriber's handset originate calls to premium rated numbers automatically without user's intention. These calls typically span for more than 30 minutes duration, sometimes till maximum allowed call duration within mobile network operator is met. Two cases mentioned in Table 3.16 provide examples for malware fraud.

In these cases, called party ID belongs to the country called Ascension and international call termination cost for this number level is LKR 380 per minute. So, a typical mobile network subscriber is highly unlikely to intentionally originate calls with such long duration to those countries. Instead of one long-duration calls, malware may originate series of comparably short duration calls also. Operators target is to identify such fraud numbers at earliest to avoid further damage to same customer or other customers who could get affected due to same malware. So, key attribute in this scenario is sum of call duration to premium or high cost destinations originated by given subscriber. This scenario also can be detected by just adding another filtering rule to same execution plan mentioned in Section 3.2.2.3. Figure 3.33 shows the filtering query used to detect Malware fraud.

Table 3:16: Instances of Malware fraud.

	Originate Date Time	Calling party	Called Party	Duration (Seconds)	Cost (LKR)
Case 1	201Y-1M-03 1H:41:58	7Y74867XX	247050000	3430	21723.33
Case 2	201Y-1M-26 1H:30:01	7Y35499XX	24793741	5400	34200.00

```
from TmpRoamPABXFruadStream3[(str:substr(calling_party_id,0,2)
=="94" and str:length(calling_party_id)==11 and
(str:substr(calling_party_id,0,4) =="9477" or
str:substr(calling_party_id,0,4) =="9476")) and
(tot_attempt_count>4 or final_tot_duration>600)]

select calling_party_id,dialed_range,
tot_attempt_count as attempt_count,
tot_ans_count as ans_count,
final_tot_duration as tot_duration,
firstattempttime1 as firstattempttime,
lastattempttime1 as lastattempttime

insert into MalwareFraudStream;
```

Figure 3:33 : Filtering Query used to detect Malware fraud.

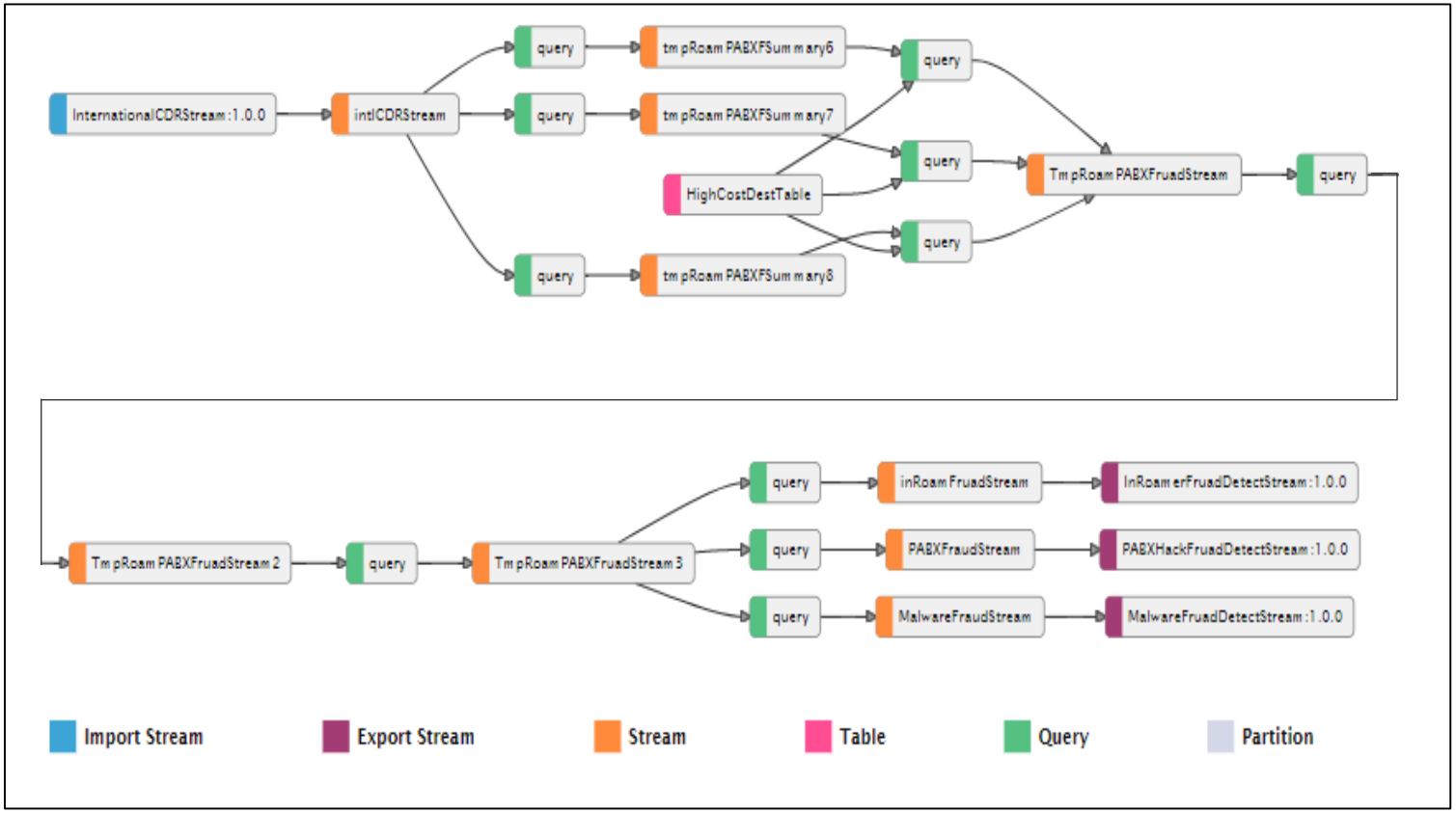

Figure 3:34: Event flow inside siddhi execution plan used to detect Inbound Roamer, PABX Hacking, and malware fraud scenarios.

4. PERFORMANCE EVALUATION

This chapter explains the experiments we performed to evaluate real-time fraud detection system using a real-world dataset. Section 4.1 presents the experimental setup and the dataset used to evaluate real-time fraud detection system. Section 4.2 presents the results and performance of bypass detection use case while Section 4.3 presents extreme usage detection use case. Finally, Section 4.4 presents the resource utilization of the proposed system.

4.1. Experimental Setup

Figure 4.1 depicts the experimental setup used to evaluate the performance of the system. Stream simulator was used to convert static CDR data stored in files into event streams and then data publisher published those events. System receives the published events through event receivers. Then, system performs intended real time and batch calculations on input data and deliver output. In Grey call detection use case, final output obtained by passing summarized data through rule-based classifier. In extreme usage detection, system directly outputs results.

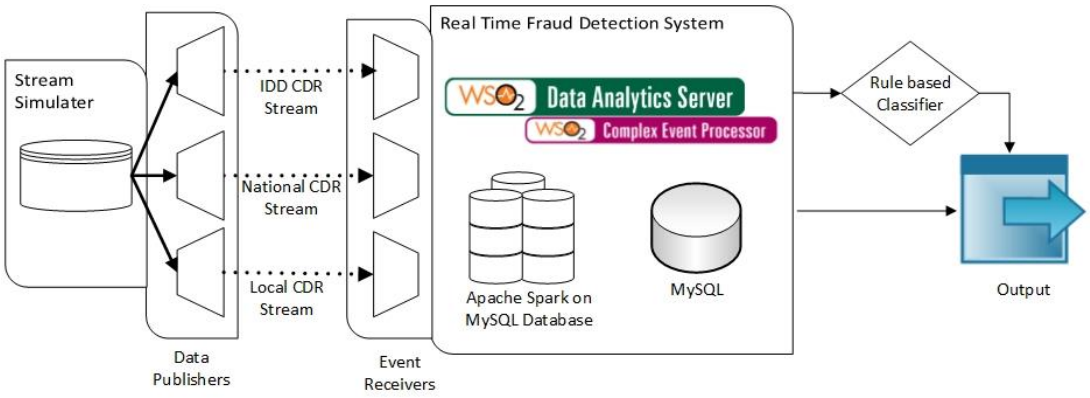

Figure 4:1: Experimental setup.

WSO_2 DAS version 3.1.0 was the main software tool used for this implementation. As per documentation, CEP integrated in WSO_2 DAS version 3.1.0 supports Siddhi query language 3.0 which is the equivalent to CEP version 4.1. But in actual implementation it supports some functions supported in Siddhi query language 3.1 which is the query language used in WSO_2 CEP version 4.2. MariaDB server version 5.5.52 was the

database server used in this setup. MariaDB is the successor of MySQL. In this proof of concept design, we have pointed relevant data sources in DAS to MySQL databases by modifying backend configurations, as it simplifies the implementation. Oracle JDK 1.8.0_121 was the Java platform used.

Due to privacy concerns, we did not gain access to live data feed in real time. Instead operator provided recent dataset in CSV formatted files. So, we have developed stream simulator software module to simulate event stream in experimental setup and coupled that module with data publisher. First, stream simulator converted timestamp of events into a Unix timestamp. Then normalized timestamp values with respect to common base were obtained by deducting base timestamp from each timestamp in dataset. Then we stored each CDR objects with mapping (key, value) in special kind of Hash-Map called Multi-Map. Normalized timestamp was used as hash key. We have used Multi-Map for storing events as it allows to store multiple objects mapped to same hash key and stored in memory while program is running. So, we can retrieve all CDR events originated at given second using hash key within minimum delay. When timer triggered, the stream data simulator retrieves the CDR and send data through stream using data publisher. Similar approach is followed in all three types of streams. We used the same base value for all three data sources and same timer was used to trigger events to make sure three streams are synchronized and real environment is replicated in experimental setup.

We used server with hardware specifications mentioned in Table 4.1. Server consists two processors which contains 8-cores of Intel Xeon 2.40GHz. CentOS Linux release 7.3.1611 64-bit version was the operating system in server when this experiment is conducted. Maximum possible heap size for JVM was increased up to 14GB to provide sufficient memory for application.

Table 4:1: Hardware specifications of experimental server.

System Resource	Specification
Processing	16 cores × Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Cache Size	20,480 KB
Memory	16 GB

Since grey call detection is supervised learning problem we have used two different datasets for training and testing. First, we have selected training and test datasets with enough positive fraud instances. Since CDR reflects privacy of subscriber, we were allowed to access data after the operator applied certain types of filtering to dataset and after proper approvals were granted.

Table 4.2 describes the distribution of training dataset which was taken between 2016-10-28 15:00:00 to 2016-10-31 00:00:00 GMT+0530. Table 4.3 describes the distribution of test dataset which was between 2017-09-01 03:00:00 to 2017-09-02 09:00:00 GMT+0530. Feature set was calculated with respect to each subscriber number. To derive feature set for Onnet bypass scenario, Local CDRs were used as main data source and International CDRs were used as support dataset. Support data was used to derive additional context data to calculate some of the features. After the calculation of feature set, it was realized that the training dataset contained 284,351 distinct onnet subscriber profiles which is 29.6% of total subscriber profiles in whole dataset. Test set population size was 677,046 which is 70.4% of total onnet subscriber profiles in whole dataset.

Table 4:2 : Details of training dataset.

	Local CDR	**National CDR**	**International CDR**
Total CDR Count	7,241,372	8,559,106	5,217,259
Distinct Number Count	284,351	1,153,409	859,718
Time Span (Hours)	57	57	57
Average Data Rate (events per second)	35.29	41.71	25.43
Maximum Data Rate (events per second)	108	145	71
Average Rate at Peak Hours (events per second)	66.76	70.19	41.77

Table 4:3: Details of test dataset.

	Local CDR	**National CDR**	**International CDR**
Total CDR Count	3,702,113	7,332,382	1,929,613
Distinct Number Count	677,046	1,373,124	481,888
Time Span (Hours)	30	30	30
Average event rate (events per second)	34.28	67.89	17.87
Maximum Data Rate (events per second)	110	187	60
Average Rate at Peak Hour (events per second)	69.72	131.72	33.58

To derive feature set for Offnet bypass scenario, National CDRs were used as main data source while using International CDRs and Local CDRs as support data. After the calculation of feature set for Offnet bypass scenario, it was realized that the training set population size was 1,153,409 entries which is 45.6% of total offnet caller profiles. Test set population size was 1,373,124 which is 54.4% of total offnet caller profiles. As this is time sensitive application, we have to periodically calculate feature values by considering 24-hour sliding window and one-hour sliding window over the time span. Each periodic calculation gave snapshot of each subscriber's behavior within the considered time window. Same subscriber may reflect significantly different behavior than earlier calculation when considered time windows are not overlapping.

The training and test datasets which were used for grey call detection use case did not contain any instance of extreme usage related frauds. Extreme usage scenarios are relatively infrequent compared to grey call fraud instances. Therefore, we have taken CDRs for past instances of extreme usage related fraud cases and used those data to build our logics. Then, as test data set, we have obtained CDRs of incoming and outgoing International calls for two 24-hour time windows and fed into the system through international CDR stream. First dataset comprises of 637,904 CDR entries and while second one had 658,982 CDR entries.

It is important to mention about the system used by operator in fraud detection as we compare performance of proposed system with operator's fraud detection system. Operator has made detections using MySQL-based batch processing system running on virtual server which contains 16-cores of Intel Xeon 2.27GHz and 24 GB RAM. They have used 24-hour and 3-hour sliding windows in calculating features. Also, operator's system is making detections at 50% false-positive rate. Then manual analysis is performed to filter out false positives. They are following this approach to minimize the detection time.

4.2. Grey Call Detection Results

This section describes the results of grey call detection use case while comparing the accuracy and efficiency of the system with respect to labeled data provided by operator. Because we focus on efficiency of CDR based feature generation and effective use of complex event patterns for grey caller detection, simple rule-based classifier was used for decision making.

4.2.1. Onnet bypass

The training dataset which was used to derive set of detection rules for Onnet bypass detection consists of 284,351 distinct subscribers and 51 verified fraud instances. The set of rules which was derived upon the feature set obtained in this research covers all the 51 fraud instances after several cycles of fine tuning. But system has located one false positive. Further fine-tuning of rules was not effective as it resulted more false negatives and model tended to over fit to training dataset. Table 4.4 shows Confusion Matrix for Onnet bypass detection system after applying final set of rules on training dataset. We have considered Fraud as positive class and Genuine as negative class. This convention is used in rest of the document.

The test dataset contained 677,046 different subscribers and out of which 45 connections are fraudulent connections. Proposed system detected 44 fraud instances correctly with one false negative. Table 4.5 shows the Confusion Matrix for Onnet bypass detection after applying final set of rules on test dataset.

Table 4:4: Confusion Matrix for Onnet bypass detection with training dataset.

		System Detected Class	
		Genuine	**Fraud**
Actual Class	**Genuine**	284,299	1
	Fraud	0	51

Table 4:5: Confusion Matrix for Onnet bypass detection system with test dataset.

		System Detected Class	
		Genuine	**Fraud**
Actual Class	**Genuine**	677,001	0
	Fraud	1	44

Inherently grey call detection problem has uneven or unbalanced class distribution as a few grey callers are operating within massive customer base. So, accuracy value easily exceeds 99% as majority of genuine customers were correctly classified by our system in Onnet scenario. But accuracy does not give correct view in bypass detection as class distribution is uneven. Therefore F-measure or F1-score gives more reasonable view.

Table 4.6 presents the performance measures of classification job performed on training and test datasets. We have fine-tuned rules to obtain 0.9903 F1 score for training dataset. Once same set of rules applied on test set, we could obtain 0.9888 F-score for Onnet bypass detection. In [4], [5], and [6] accuracy is above 98% and F-score of those instances were in 0.9 range. But in each case, the dataset which was used consists of more even distribution of grey callers and normal customers. Whereas the proposed system was evaluated against real dataset, which contained a more skewed class distribution.

Bypass detection is a time sensitive use case, as we can minimize revenue loss, if we can make detections earlier. Number of attempts, number of answer attempts, and minutes of usage before detection are other most important parameters when we look at system from telecom operator's view. Whereas in [4], [5] and [6] authors do not use such important parameters when measuring performance.

Table 4:6: Performance measures of classification job performed in Onnet bypass detection.

	Training Set	Test Set
True Positives	51	44
True Negatives	284,299	677,001
False Positives	1	0
False Negatives	0	1
Accuracy	0.9998	0.9999
Precision	0.9808	1
Recall	1	0.9778
F1 Score	0.9903	0.9888

Table 4.7 presents how fast Onnet bypass detection can be performed. It can be seen that while the operator took 23.16 answered calls in average to detect Onnet bypass within the considered 30-hour time window, the proposed system detected it after 6.55 answered calls in average. Therefore, proposed system made detection 16.6 answered attempts earlier than operator detections while reducing the revenue loss. Moreover, while fraudsters were successful in using onnet bypass number for 65.2 minutes in average per number in operator's existing detection system, the proposed system detected it within 14.53 minutes in average. Therefore, the proposed system makes detections with considerable accuracy without compromising detection speed. This was possible due to the use of features based on both real-time and past data, as well as using CEP.

Table 4:7: Speed of Onnet bypass detection with test dataset.

	System Detections		Operator Detections	
	Total	**Average**	**Total**	**Average**
Usage Duration (Minutes)	639.35	14.53	2251.92	51.18
Attempt Count	802	18.23	2869	65.20
Answer Attempt Count	288	6.55	1019	23.16

The chart shown in Figure 4.2 represents the contribution of different types of rules for Onnet bypass detection. Out of 44 detected instances, 32 instances were detected using real-time rules type. Real-time rules were derived based on complex pattern

detections and accounting stats for recent one-hour sliding window. 24-hour type represents rules which were derived based on stats for 24-hour sliding window. Nine detections were made using 24-hour type rules. Remaining three detections were made using hybrid rules that represent rules derived based on 24-hour sliding window, complex pattern detections, and recent one-hour stats. In Onnet bypass scenario, highest percentage (73%) of detections made by real-time rules.

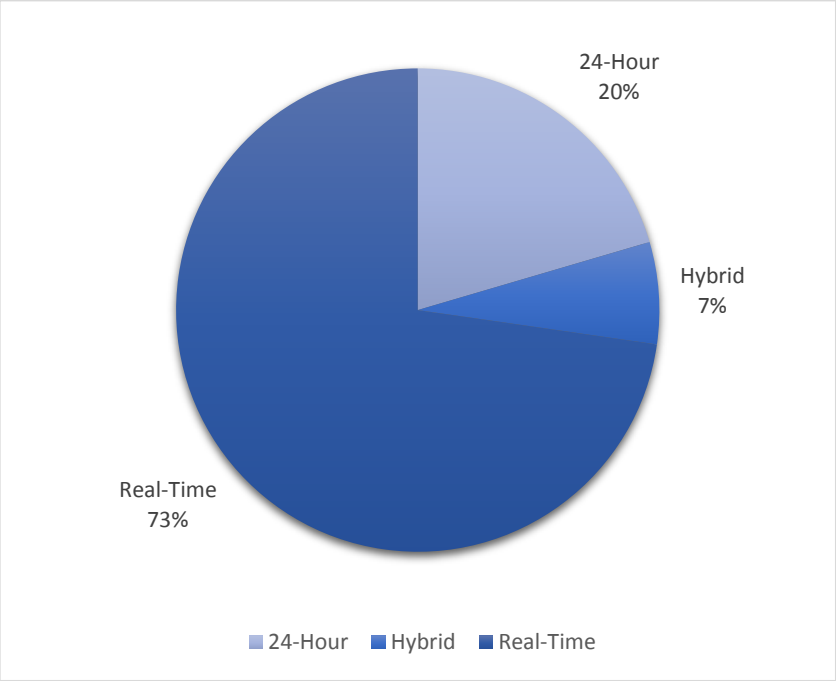

Figure 4:2: Contribution of different types of detection rules for Onnet bypass detection.

4.2.2. Offnet bypass

The training dataset which was used to derive set of detection rules for Offnet bypass detection consisted of 1,153,409 customers belonging to other operators. Out of which there were 328 verified fraud instances. The set of rules which was derived upon the feature set obtained in this research has located 311 fraud instances after several cycles of fine tuning. But system has detected 83 false positives and there were 17 false negatives. Further fine-tuning of rules was not effective as it resulted more false negatives. Table 4.8 shows Confusion Matrix for Offnet bypass detection system after applying final set of rules on training dataset.

Table 4:8: Confusion Matrix for Offnet bypass detection system with training dataset.

		System Detected Class	
		Genuine	**Fraud**
Actual Class	**Genuine**	1152998	83
	Fraud	17	311

The test dataset contains 1,373,124 different other operator subscriber profiles and out of which 233 connections are fraudulent connections. System detected 219 fraud instances correctly. But there were 44 false positives and 14 false negatives in this case. Table 4.9 presents the Confusion Matrix of Offnet bypass detection.

Table 4:9: Confusion Matrix of Offnet bypass detection with test dataset.

		System Detected Class	
		Genuine	**Fraud**
Actual Class	**Genuine**	1372847	44
	Fraud	14	219

Similar to Onnet bypass detection, offnet bypass detection also has uneven class distribution. In this case also, system delivered 99% accuracy level as majority of true negatives were classified correctly. Therefore, we need to consider F-score to get correct idea about classification job. Table 4.10 presents the performance measures of classification job performed on training and test datasets. We have fine-tuned rules for training dataset to obtain 0.8615 F1-score. Once the same set of rules was applied on the test set, we could obtain 0.8831 F-score for Offnet bypass detection. In [4], [5], and [6] authors did not addressed offnet bypass problem. But for the operator considered for the analysis, Offnet bypass number imposes a higher threat than Onnet bypass numbers.

Table 4:10: Performance measures of classification performed for Offnet bypass detection.

	Test Set	Training Set
True Positives	219	311
True Negatives	1,372,847	1,152,998
False Positives	44	83
False Negatives	14	17
Accuracy	1	0.9999
Precision	0.8327	0.7893
Recall	0.9399	0.9482
F1 Score	0.8831	0.8615

Like in Onnet bypass detection, detection speed is important aspect of performance for Offnet bypass scenario as well. But National CDR Stream, which was used for Offnet bypass detection does not contains call duration related information. Therefore, we have considered attempt count when measuring detection speed. Table 4.11 presents detection speed related performance measures. Proposed system has detected 219 fraud instances in test set with 16.09 average attempts per number. But Operator has spent 26.02 average attempts per number within considered 30-hour time window before being detected. Therefore, the proposed system detected fraud instances 9.93 call attempts earlier than operator's existing solution.

Table 4:11: Detection speed related performance measures for Offnet bypass detection with test set.

	System Detections		Operator Detections	
	Total	**Average**	**Total**	**Average**
Attempt Count	3523	16.09	5698	26.02

We have analyzed contribution of each type of rules for detecting 219 Offnet bypass fraud instances in test set. 24-Hour type rules detected 118 of those instances. Real-time rules type contributed by detecting 12 instances. Remaining 89 detections were made using Hybrid rules. The chart shown in Figure 4.3 represents the contribution of different types of rules for Offnet bypass detection. In offnet bypass scenario, 24-hour

type rules made highest contribution by detecting 54% of total detections. Hybrid rules contributed next by making 41% of detections. Contribution of real-time rules was 5% in this case.

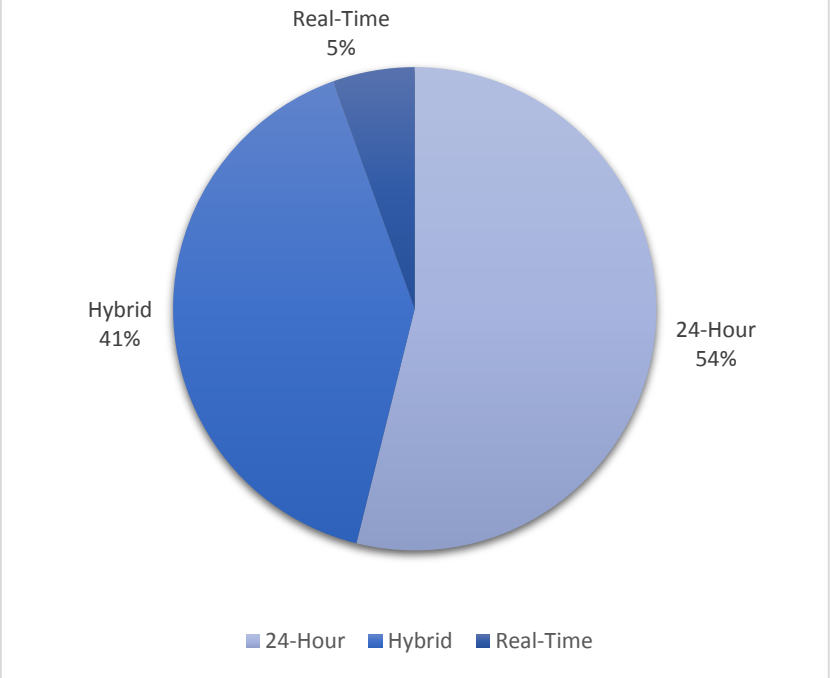

Figure 4:3: Contribution of different types of detection rules for Offnet bypass detection.

4.3. Extreme Usage Detection Results

We focused on five types of extreme usage related fraud scenarios in this research, where all the cases are associated with premium number dialing. These fraud instances are detected relatively infrequently compared to Grey call fraud. Because behavior of these fraud scenarios are straight forward, fraudulent numbers can be detected by using CEP execution plan directly without considering past data. In this scenario, telco operators are more interested about alerting suspected fraudulent activity at earliest by compromising accuracy to certain level. Therefore, we do not discuss about accuracy measures in detail related to this case.

Table 4.12 shows instances of Dial and Disconnect Scam detected by the proposed system. Once particular premium number terminated more than 10 customers of telecom network under the study, system immediately notifies those instances. As per

configured threshold value, system detects fraud instances after 11th subscriber received a call from given premium number. Therefore, effect of this fraudulent activity for more customers can be eliminated. In this case no false positives or false negatives were detected.

Table 4:12: Dial and Disconnect Fraud instances detected by System.

Calling Party Number	No of distinct subscribers received calls from this no when detected by system	No of distinct subscribers received calls from this no
5068687800	11	89
27230040092	11	61

Table 4.13 shows instances of fraud scenario associated with Outbound Dialing due to fake SMS messages detected by System. Once more than 10 distinct customers have dialed a specific premium number, system immediately notifies those instances. As per in Table 4.13, it is clear that many more customers can be affected, if these instances are not detected near real time.

Table 4:13: Instances of Outbound Dialing due to fake text messages detected by system.

Called Party Number	No of distinct subscribers dialed this no when detected by system	No of distinct subscribers dialed this no
240555903098	11	229
261344693077	11	555
23786395002	11	120
2917326679	11	696
22997947006	11	104
17673162598	11	21

Within the considered time span, system has detected one instance of inbound roamer's high usage scenario. Table 4.14 presents details about suspicious inbound roamer high usage instance which was detected by system. Given customer has dialed 43720 number range which belongs to Austria special service and call termination cost to this level is LKR 500 per minute. So, operator need to be alerted about this case and make sure High Usage Report (HUR) is sent to home network operator of detected

inbound roamer. But in this case roamer may not involve in organized inbound roamer fraud. Within this dataset, no instances of malware originated calls or PABX hacking fraud was detected.

Table 4:14: Instance for inbound roamer's extreme usage.

Calling Party Number	Dialed number range	Answer attempt count before detection	Total duration before detection (seconds)
4917328727XX	43720	2	3681

4.4. Resource Utilization

Figure 4.4 shows CPU load on server when Bypass detection solution is running. The snapshot is taken by including time span in which system operated with peak event rate. In our case, according to statistics of test and training datasets mentioned in Section 3.2.1.1, possible peak rate is around 233 events per seconds. System operates below 30% of CPU usage when pattern queries are running. But we can see hikes in processing when Spark scripts are running on BAM.

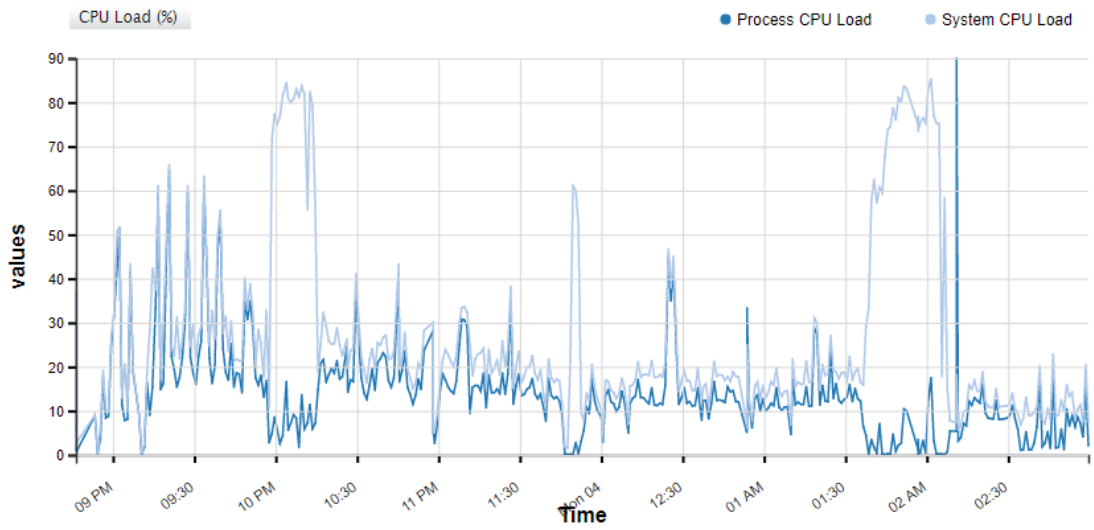

Figure 4:4: CPU utilization of server with bypass detection application.

Figure 4.4 shows memory utilized by JVM when bypass detection is running. We have set maximum possible Heap size to 14 GB from Java options. According to the graph, heap utilization is lower than allowed maximum memory level throughout run time.

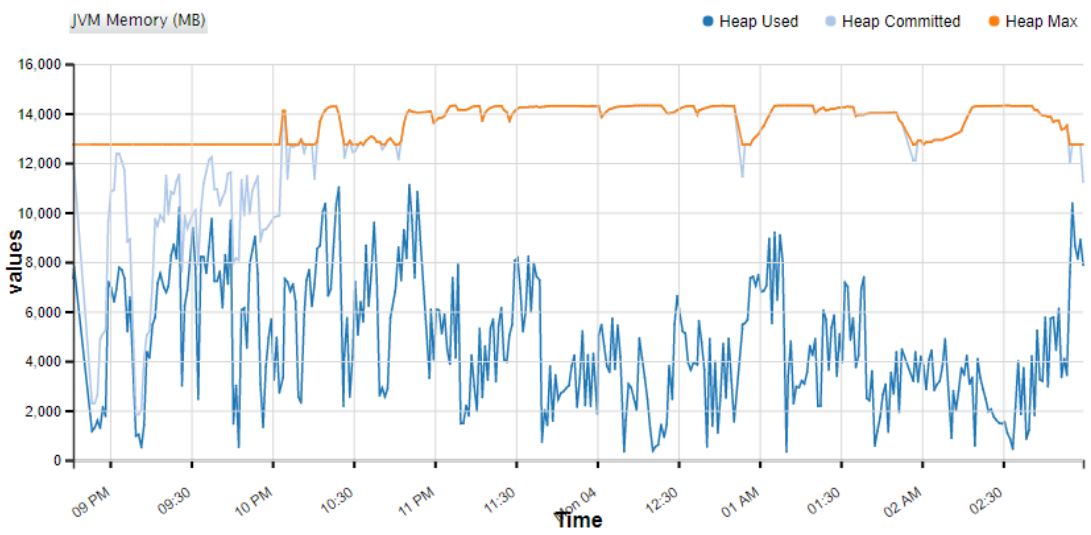

Figure 4:5: Memory utilization of Java virtual machine with bypass detection.

Figure 4.6 presents a CPU load and Heap utilization when CEP queries used to bypass detection is executed at varying event rates. The CEP queries used to bypass detection is the critical point of proposed system which defines the maximum operable input event rate. If any event stream is lagging relative to other input streams, proposed system could not produce correct results for pattern queries. Therefore, we have disabled batch analytic scripts and CEP queries used in extreme usage detection while conducting this experiment. At phase 1, we have loaded system with input rate around 125 events per second. We can see CPU load is below 8% in this case. Heap utilization is below 3 GB and it started at lower value and slightly grows with time as more events are feed into CEP. In second phase, we loaded CEP with 250 events per second and CPU load was below 12%. Heap utilization was below 5.5 GB and it starts at lower value and slightly grows with time as more events are feed into CEP. At third phase, we loaded CEP with input event rate of around 375 events per second. In this stage, CPU load was below 14% and Heap utilization was below 6 GB. Then we tested with 500 events per second. In this phase CPU and Heap utilizations are below 15% and 6 GB, respectively. It is important to note that when experiment ran around 90 minutes, growth of heap utilization is stabilized. Beyond 500 events per second input rate, we have observed that the CDR stream with highest input event rate was lagged relative to remaining two streams when experiment goes on. The reason for this observation

could be increase in transaction time of receiving new event via event receiver due to queued events at CEP. Data publisher publish events to event receiver using TCP (Transmission Control Protocol). Therefore, reliable transaction need to be completed between publisher and receiver at arrival of each event to CEP. When events queued at CEP input beyond certain level, this transaction time could be increased and sending events on event stream could be delayed.

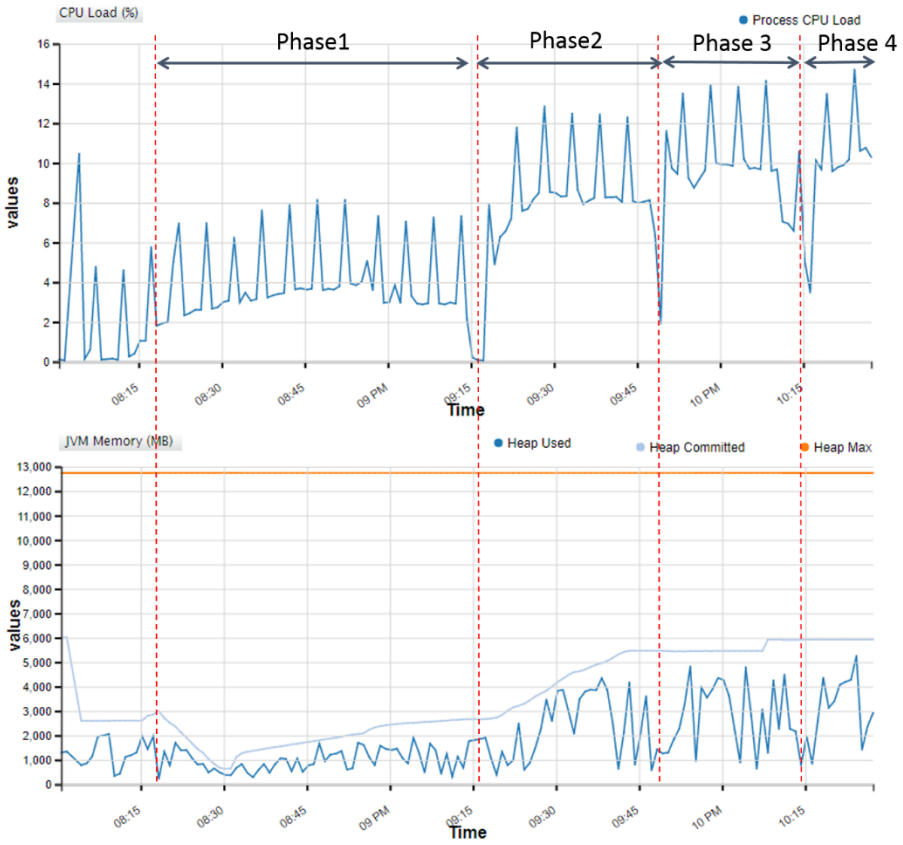

Figure 4:6: CPU and Heap utilization of CEP queries used for Bypass detection at varying event rates.

Figure 4.7, and 4.8 show CPU load and JVM memory utilization when extreme usage detection application is running. In this case CPU utilization is below 15% and memory utilization is below 7 GB.

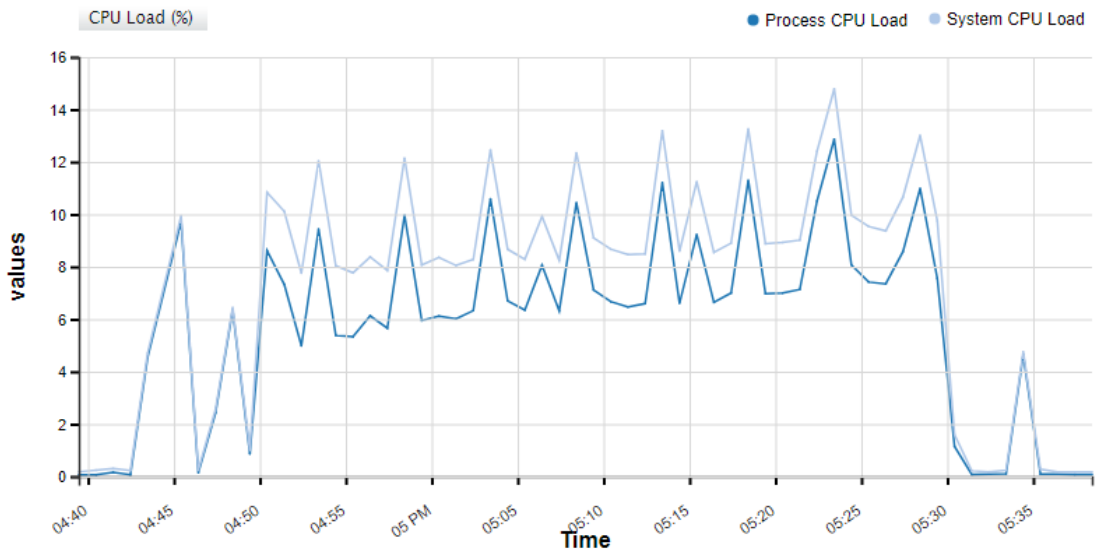

Figure 4:7: CPU utilization of server with extreme usage detection.

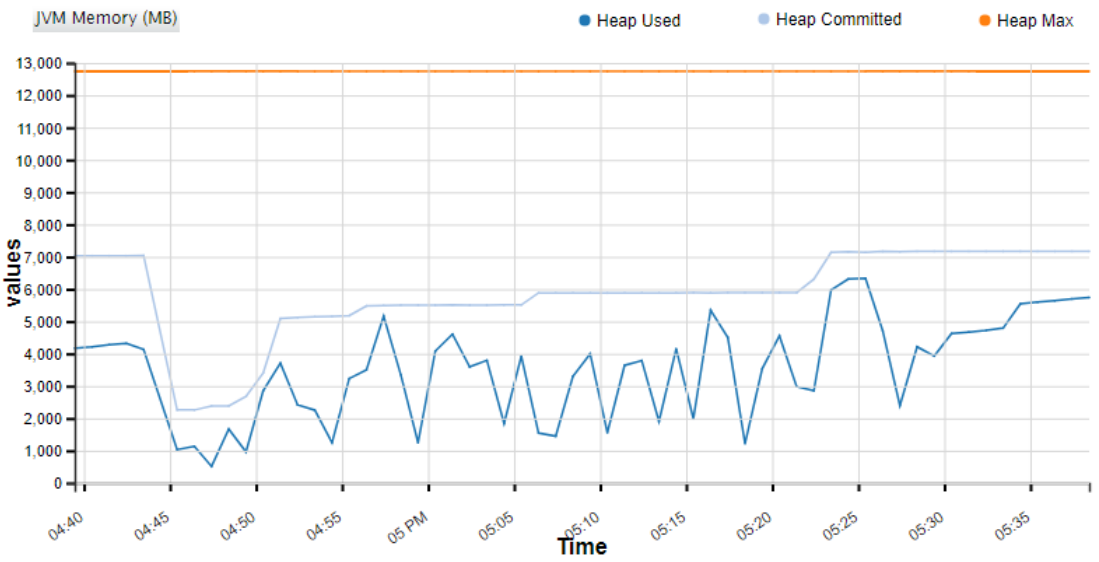

Figure 4:8: Memory utilization of Java virtual machine with extreme usage detection.

In extreme usage detection use case, we fed 658,982 events in 45 minutes. Thus, the event rate is around 244 events per second. According to resource utilization graphs mentioned above, extreme usage detection application uses lower system resources than bypass detection application. Possible reason for this observation is calculations performed in bypass detection is more complex than extreme usage detection. Also, bypass detection application is dealing with three different streams and

synchronization of receiving events from three streams is important for efficient detection of patterns. If one stream is lagged than others proper complex event detection may not happen. Also, DAS queries used in bypass detection also causes higher CPU utilization.

4.5. Summary

In first part of this chapter, we have discussed about the experimental setup and the dataset used to evaluate real-time fraud detection system. Then we have discussed about the results of grey call detection use case. We were able to achieve 99.9% accuracy in both onnet and offnet bypass detection while significantly increasing detection speed. Next, we have presented results of extreme usage detection and we have detected such instances in real-time before any impact is felt to operator. Finally, we have demonstrated resource utilization when proposed system is utilized in each use case. As per our observations, proposed system has performed intended functionality without performance degradations at input event rate of 500 events per second with modest hardware.

5. CONCLUSION AND FUTURE WORK

This chapter concludes dissertation by summarizing the findings, research limitations, and possible future enhancements. Section 5.1 summarizes the problem statement, design, and outcomes of the research. The problems and limitations encountered during this research are described in Section 5.2. Section 5.3 presents the potential future developments of this project.

5.1. Summary

Primary focus of this research is to detect fraud scenarios in telecom network in near real-time by using call patterns reflected in CDR stream. Grey call detection and detection of extreme usage scenarios related to international premium numbers are two major use cases of this system. We followed the Big Data Lambda architecture and developed a system architecture that comprises batch, speed, and serving layers, as it is well suited for application which performs both real-time and batch analytics. WSO_2 DAS was used in batch layer due to its ability to perform high-speed batch processing and Siddhi CEP was used at speed layer due to its enhanced performance in complex event detection.

First, we started with the goal of detecting grey call numbers in near real-time. Related work such as [4], [5], and [6] do not support real-time detection due to unsuitability of traditional database reliant store first process then approach, dependence on large time windows for feature generation, shallow feature set, and ignoring complex patterns in CDRs in decision making. Therefore, we have primarily focused on generating rich set of features in near real time. Real-time feature set consists two components. First component is feature set derived based on complex patterns in CDR stream by using CEP engine. Second component is feature set generated by aggregating CDRs within recent one hour sliding window. We were able to detect some of the fraud instances by directly focusing on real-time behavior. But integration of past behavior is also important to minimize false positives and false negatives in bypass detection. So, we have generated set of features by considering past 24-hours behavior of subscribers. Additionally, we have used context data related to this fraud scenario to enrich feature set. Finally, we were able to come with a rich feature set to facilitate near real-time

grey call detection. Based on this feature set we built a set of classification rules to locate grey call instances.

Grey call detection problem is divided to two sub-problems called Onnet bypass detection and Offnet Bypass detection. Onnet bypass detection considers grey callers who are using connections of same operator under study. Therefore, operator has visibility to location and device details of those connections and decision making is relatively easier. All the approaches mentioned in [4], [5], and [6] focused about Onnet grey call detection. Also, they have used dataset with more even class distribution. But our system has obtained results with similar F-Score for dataset with more uneven class distribution. Also, those approaches did not consider about detection speed of system. Average number of answer attempts and average duration conceded for number are important metrics used in industry to measure performance of bypass detection system. When considering those parameters our system clearly outperformed existing methodologies used by telecom operator.

Offnet bypass detection considers grey callers that are using connections belong to other operators in country to terminate international calls to operator under study. In this case operator does not have cell information and device information. So Offnet bypass detection is more challenging and [4], [5], and [6] do not support Offnet bypass detection. In our case complex patterns and using context data were more useful and we were able to make offnet detections with considerable accuracy where we were able to reach 0.88 F-Score. Also, the proposed solution was able to detect offnet bypass calls about 10 attempts earlier than the operator. Thus, the proposed system can provide significant savings to telecom operator by reducing impact of grey call fraud.

Grey call detection is complex fraud scenario as fraudsters tend to replicate normal user's behavior to mislead grey call detection systems. Even though fraudsters can emulate normal user's behavior by changing calling party behavior, there are some invariants unique to this fraud scenario which can be located by complex events generated within CDR stream and considering context of called party numbers. We had a challenge of including those invariants when building the feature set. The six complex event based features named as *P1, P2, P3, P4, P5,* and *P6* capture invariant

call patterns unique to grey call fraud (see Section 3.2.1.3 and 3.2.1.4 for details). Fraudsters cannot avoid generating those event sequences while fraudulent activity is on course as fraudsters do not have control in all the individual events in that sequence. Also, features named as *grb_dcnt_in_hour, iddb_dcnt_in_hour, iddb_dcnt_out_hour, grb_dcnt_in, iddb_dcnt_in* and *iddb_dcnt_out* capture invariant behaviors unique to grey call fraud. Those features were built by considering context of called party numbers and fraudsters cannot influence those features by manipulating calling party behavior. System continuously updates context data based on user feedback and fresh data feed. Even though this is a rule based system, system is able to detect grey call fraud instances with new behaviors due to these tactics.

We further worked on detecting extreme usage scenarios related to premium rated international destinations. These fraud instances are relatively infrequent compared to grey calls. So, we have obtained CDRs for past instances of these frauds and started to model Siddhi QL queries on CEP. We have identified behavior of five types of extreme usage scenarios and developed CEP execution plans to detect those. Unlike grey call detection problem, these fraud instances can be detected directly by considering real-time view. Finally, we have fed international CDR for two days into system and detected instances of three types of extreme usage scenarios. Instances of PABX hacking fraud and Malware fraud were not available in test dataset as occurrence of those types of frauds are comparably rare. System had made detections before those fraudsters make noticeable effect to customers.

5.2. Research Limitations

We faced many challenges when labeling both training and test datasets for grey call detection. In Onnet bypass detection operator labeling was correct. But in offnet case both operator, and ourselves faced the challenge of determining class labels. We have observed some Offnet numbers were labeled as genuine subscribers even though those numbers reflected highly suspicious call pattern. These connections belong to one of the wireless fixed-line networks in the country and that operator claimed those as genuine subscribers. In deeper analysis, we observed that fraudsters have used real subscriber connections to fraudulent activity in these cases and those instances need to

be labeled as fraudulent. This issue imposed great challenge in developing rules as issue was severe in the time we acquired training dataset. So, we have used different set of rules to different operators to address this issue. Also, some of the fraudsters have used call forwarding and many other advanced techniques to replicate genuine usage behavior and mislead detection systems. Therefore, we had to go through series of verifications to decide class labels for offnet bypass detection.

When we consider server resource utilization, grey call detection application consumed more resources than extreme usage detection. Initial plan was to implement aggregation for recent one-hour sliding window on CEP itself. But that approach was not feasible due to complexity of join queries and number of records involved in join queries. When system ran on this configuration, processing of some streams were lagging related to others. Pattern queries on CEP were affected due to this lagging. So, we have performed calculations based on one-hour sliding window on DAS. Calculations based on 24-hour sliding window was also done in DAS. But in this case, we have done those 24-hour calculations offline and merged with real-time view. This is same as performing 24-hour based calculations in separate DAS server. Ideally this can be done in separate physical server in parallel to real-time calculations. Calculations related to extreme usage scenario were done on CEP only and resource utilization is comparably lower in this case.

Due to privacy concerns, operators were not willing to expose these data to outside parties, so we identify the limitation of reproducibility. Also, we were not authorized to bring CDR details outside and experiment the system with better computing resources due to privacy concerns of operator. System was tested on server available in operator premises. With better computing resources we may able to test this system on higher data rate and evaluate system performance. Also, operator did not provided CDRs for full customer base due to privacy reasons. We have gained access to the CDRs of subset of customer base after operator has made some precautionary actions to preserve privacy. Therefore, we were not able to perform analysis on full customer base. But, the dataset provided was sufficient to implement comprehensive solution.

Average call duration attribute was not used in initial model developed to detect inbound roamer fraud, PABX hacking fraud and malware fraud. When we observe inbound roamer fraud detection results for test set we observed that new attribute is required to reduce false positives. So we have collected more training instances and decided to use average call duration attribute.

Additionally, we have not considered subscription fraud [7] in this research, as detection of those frauds requires sensitive privacy information of customer in addition to CDRs. Also, subscription fraud is relatively rare within the network under study due to operational policies followed by operator. We did not consider handset theft scenarios in this research even though such instances could be detected by using this solution. Within this country, customers hold full ownership of their mobile handsets, so operators do not have interest in detecting handset theft fraud. During this period, DoS and DDoS attacks on voice network were not observed. Therefore we did not have sample data to study about those scenarios.

5.3. Future Work

Inclusion of machine-learning techniques and using Neural Network or Tree-based classifier on derived feature set is interesting future work of this project. But this will be challenging task as some of the grey call instances replicate genuine behavior and that may corrupt learning process. Using machine-learning approach for Offnet bypass detection will be more challenging as numbers belong to different operators show different behaviors. So, hybrid method of rule-based and machine-learning based classification will be a fitting approach. Integration of WSO_2 Machine learner which is a WSO_2 module for predictive analytics will be another interesting future enhancement of this project.

Also, we can expect significant performance enhancements if this system can be run one clustered environment with high processing power. With high computing resources, more calculations can be moved to CEP and detection speed can be further increased. Also, scaling the proposed system to handle CDRs of full customer base of the operator is another challenging future work. Additionally, this system can be extended to detect handset theft scenario in future based on operator’s requirement.

Developing CEP queries to detect DoS and DDoS attacks on voice network could be value addition to the proposed solution. However, this system landmarks the good initiative in near real-time fraud detection in telecom operators by deviating from traditional database reliant approach.

REFERNCES

[1] R. Arnoff, "Global Fraud Loss Survey 2013 by Communications Fraud Control Association," 2013. [Online]. Available: http://www.cvidya.com/media/62059/global-fraud_loss_survey2013.pdf. [Accessed 4 Jun 2015].

[2] Wikipedia, "Call detail record," [Online]. Available: http://en.wikipedia.org/wiki/Call_detail_record. [Accessed 5 Jun 2015].

[3] H. Grosser, P. Britos and R. García-Martínez, "Detecting fraud in mobile telephony using neural networks," in *Springer-Innovations in Applied Artificial Intelligence Lecture Notes in Computer Science*, vol. 3533, 2005, pp. 613-615.

[4] A. H. Elmi, S. Ibrahim, and R. Sallehuddin, "Detecting SIM Box fraud using neural network," *IT Convergence and Security-2012. Springer*, vol. 215, pp. 575-582, 2013.

[5] R. Sallehuddin, S. Ibrahim, A. M. Zain, and A. H. Elmi, "Classification of SIMbox fraud detection using support vector machine and artificial neural network," *International Journal of Innovative Computing, Universiti Teknologi Malaysia,* vol. 4, no. 2, 2014.

[6] I. Murynets, M. Zabarankin, R. P. Jover, and A. Panagia, "Analysis and detection of SIMbox fraud in mobility networks," in Proc. *IEEE INFOCOM '14,* pp. 1519-1526, Apr. 2014.

[7] J. Shawe-Taylor, K. Howker and P. Burge, "Detection of Fraud in Mobile Telecommunications," *Information Security Technical Report,* vol. 4, no. 1, pp. 16-28, 1999.

[8] LATRO Services, "Advanced Analytics Solution for Telecom Fraud - VERSALYTICS," [Online]. Available: http://www.latroservices.com/products/versalytics-analytics-solution-for-bypass-fraud/. [Accessed 21 December 2017].

[9] E. Okutoyi, "SIM Box Fraud - New Headache for Africa's Mobile Operators," 2012. [Online]. Available: http://www.humanipo.com/news/142/sim-box-fraud-new-headache-for-africas-mobile-operators/. [Accessed 13 Jun 2015].

[10] F. Kornbo, "Carrier bypass: No drastic surgery required to protect revenue," 2012. [Online]. Available: http://www.telecomasia.net/pdf/CSGI/CSG_AsiaConnectionsJuly2012_CarrierBypass.pdf. [Accessed 4 Jun 2015].

[11] Etross Telecom Co. Ltd., "GSM Modem Pool 8 Ports 32Sims ETS-8132 with SIM Rotation," 2013. [Online]. Available: http://www.etross.com/products_ys/&productId=30&comp_stats=comp-FrontProducts_list01-1364547681813.html. [Accessed 5 Jun 2015].

[12] Bangladesh Telecommunication Regularity Commission, "Mobile Phone Subscribers in Bangladesh January 2014," 2014. [Online]. Available: http://www.btrc.gov.bd/content/mobile-phone-subscribers-bangladesh-january-2014. [Accessed Mar 2015].

[13] Wikipedia, "List of mobile network operators," [Online]. Available: http://en.wikipedia.org/wiki/List_of_mobile_network_operators. [Accessed 5 Mar 2015].

[14] OpenCellID Community, "OpenCellID Database," [Online]. Available: http://opencellid.org/#action=statistics.cells&type=2&dateFrom=&dateTo=&mcc=&mnc=&sortBy=1. [Accessed 7 Jun 2015].

[15] Wikipedia, "Hellinger distance," [Online]. Available: http://en.wikipedia.org/wiki/Hellinger_distance. [Accessed 7 Jun 2015].

[16] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, Jun 2012.

[17] L. Neumeyer, S. Clara, B. Robbins, A. Nair and A. Kesari, "S4: Distributed Stream Computing Platform," *IEEE Int. Conf. on Data Mining Workshops '10,* pp. 170-177, Dec 2010.

[18] Apache Software Foundation, "Apache ZooKeeper," [Online]. Available: https://zookeeper.apache.org/. [Accessed 7 June 2015].

[19] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg and G. Anderson, "SASE: Complex Event Processing over Streams," *CIDR,* Jan 2007.

[20] Esper Team and EsperTech Inc., "Esper Reference Documentation Version 5.1.0," 2014.

[21] EsperTech Inc., "Esper: Event Processing for Java," 2015. [Online]. Available: http://www.espertech.com/products/esper.php.

[22] S. Suhothayan, K. Gajasinghe, I. L. Narangoda and S. Chaturanga, "Siddhi: A second look at complex event processing architectures," *ACM GCE Workshop,* 2011.

[23] S. Suhothayan, K. Gajasinghe, I. L. Narangoda and S. Chaturanga, "Siddhi-CEP, B.Sc. Project Report," Dept. of Computer Sci. and Eng, Univ. of Moratuwa, Moratuwa, Sri Lanka, 2011.

[24] WSO2 Inc., "WSO2 Complex Event Processor Documentation Version 3.1.0," 2015.

[25] D. Anicic, S. Rudolph, P. Fodor and N. Stojanovic., "Stream reasoning and complex event processing in ETALIS," *Semantic Web Journal,* 2012.

[26] Cornell Database Group, "Cayuga: Stateful publish/subscribe for event monitoring," [Online]. Available: http://www.cs.cornell.edu/bigreddata/cayuga/ . [Accessed 7 Jun 2015].

[27] N. Gehani, H. Jagadish and O. Shmueli, "Composite event specification in active databases: Model & implementation," in Proc. *Int. Conf. on Very Large Data Bases*, 1992, p. 327–327.

[28] J. Morrell and S. D. Vidich., "Complex Event Processing with Coral8," [Online]. Available: http://download.microsoft.com/download/5/6/6/566AEA2A-C50E-47B8-890E-BCF4E0EC5D0B/ Complex_Event_Processing_with_Coral8_Final.pdf. [Accessed Jun 2015].

[29] Oracle Corporation, "Oracle Event Processing," [Online]. Available: http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/oepdatasheet12c-2226352.pdf. [Accessed Jun 2015].

[30] TIBCO Software, "TIBCO StreamBase," [Online]. Available: http://www.tibco.com/products/event-processing/complex-event-processing/streambase-complex-event-processing. [Accessed Jun 2015].

[31] N. Marz and J. Warren, "A new paradigm for Big Data," in *Big data. Principles and best practices of scalable real-time data systems*, Manning Publications, 2014.

[32] Apache Software Foundation, “Apache Hadoop,” [Online]. Available: https://hadoop.apache.org/. [Accessed 5 Jun 2015].

[33] S. Perera, “Implementing Bigdata Lambda Architecture using WSO2 CEP and BAM,” 2014. [Online]. Available: http://srinathsview.blogspot.com/2014/03/implementing-bigdata-lambda.html. [Accessed 27 May 2015].

[34] D. De Silva, “Lambda Architecture Demo for CEP [Online],” 2014. [Online]. Available: http://wso2-oxygen-tank.10903.n7.nabble.com/Lambda-Architecture-Demo-for-CEP-td107942.html. [Accessed 27 May 2015].

[35] WSO2 Inc., “WSO2 Business Activity Monitor Documentation Version 2.5.0,” 2015.

[36] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in Proc. *6th Symp. on Operating System Design and Implementation (OSDI)*, 2004, p. 137–150.

[37] WSO2 Inc., “WSO2 Data Analytics Server Documentation Version 3.1.0,” 2017.

[38] P. Jayawardhana, A. Kumara, D. Perera and A. Paranawithana, “Kanthaka: Big Data Caller Detail Record (CDR) Analyzer for Near Real Time Telecom Promotions,” Proc. *Fourth Int. Conf. on Intelligent Systems Modelling & Simulation (ISMS)*, pp. 534-538, 2013.