

Institutionen för datavetenskap
Department of Computer and Information Science

Master's Thesis

**Automated Measurement and Change Detection of an
Application's Network Activity for Quality Assistance**

by

Robert Nissa Holmgren

LIU-IDA/LITH-EX-A--14/033--SE

2014-06-16



Linköpings universitet

Master's Thesis

Automated Measurement and Change Detection of an Application's Network Activity for Quality Assistance

by

Robert Nissa Holmgren


LIU-IDA/LITH-EX-A--14/033--SE

2014-06-16

Supervisors: Fredrik Stridsman
Spotify AB

Professor Nahid Shahmehri
Department of Computer and Information Science
Linköping University

Examiner: Associate Professor Niklas Carlsson
Department of Computer and Information Science
Linköping University

	Avdelning, Institution Division, Department Database and Information Techniques (ADIT) Department of Computer and Information Science SE-581 83 Linköping	Datum Date 2014-06-16
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LIU-IDA/LITH-EX-A-14/033—SE Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-107707		
Titel Title Författare Author	Automatisk mätning och förändringsdetektering av en applikations nätverksaktivitet för kvalitetsstöd Automated Measurement and Change Detection of an Application's Network Activity for Quality Assistance Robert Nissa Holmgren	
Sammanfattning Abstract <p>Network usage is an important quality metric for mobile apps. Slow networks, low monthly traffic quotas and high roaming fees restrict mobile users' amount of usable Internet traffic. Companies wanting their apps to stay competitive must be aware of their network usage and changes to it.</p> <p>Short feedback loops for the impact of code changes are key in agile software development. To notify stakeholders of changes when they happen without being prohibitively expensive in terms of manpower the change detection must be fully automated. To further decrease the manpower overhead cost of implementing network usage change detection the system need to have low configuration requirements, and keep the false positive rate low while managing to detect larger changes.</p> <p>This thesis proposes an automated change detection method for network activity to quickly notify stakeholders with relevant information to begin a root cause analysis after a change in the network activity is introduced. With measurements of the Spotify's iOS app we show that the tool achieves a low rate of false positives while detecting relevant changes in the network activity even for apps with dynamic network usage patterns as Spotify.</p>		
Nyckelord Keywords computer networking, software quality assurance, novelty detection, clustering		

Abstract

Network usage is an important quality metric for mobile apps. Slow networks, low monthly traffic quotas and high roaming fees restrict mobile users' amount of usable Internet traffic. Companies wanting their apps to stay competitive must be aware of their network usage and changes to it.

Short feedback loops for the impact of code changes are key in agile software development. To notify stakeholders of changes when they happen without being prohibitively expensive in terms of manpower the change detection must be fully automated. To further decrease the manpower overhead cost of implementing network usage change detection the system need to have low configuration requirements, and keep the false positive rate low while managing to detect larger changes.

This thesis proposes an automated change detection method for network activity to quickly notify stakeholders with relevant information to begin a root cause analysis after a change in the network activity is introduced. With measurements of the Spotify's iOS app we show that the tool achieves a low rate of false positives while detecting relevant changes in the network activity even for apps with dynamic network usage patterns as Spotify.

Sammanfattning

Nätverksaktivitet är ett viktigt kvalitetsmått för mobilappar. Mobilanvändare begränsas ofta av långsamma nätverk, låg månatlig trafikkvot och höga roamingavgifter. Företag som vill ha konkurrenskraftiga appar behöver vara medveten om deras nätverksaktivitet och förändringar av den.

Snabb återkoppling för effekten av kodändringar är vitalt för agil programutveckling. För att underrätta intressenter om ändringar när de händer utan att vara avskräckande dyrt med avseende på arbetskraft måste ändringsdetekteringen vara fullständigt automatiserad. För att ytterligare minska arbetskostnaderna för ändringsdetektering av nätverksaktivitet måste detekteringsystemet vara snabbt att konfigurera, hålla en låg grad av felaktig detektering samtidigt som den lyckas identifiera stora ändringar.

Den här uppsatsen föreslår ett automatiserat förändringsdetekteringsverktyg för nätverksaktivitet för att snabbt meddela stakeholders med relevant information för påbörjan av grundorsaksanalys när en ändring som påverkar nätverksaktiviteten introduceras. Med hjälp av mätningar på Spotifys iOS-app visar vi att verktyget når en låg grad av felaktiga detekteringar medan den identifierar ändringar i nätverksaktiviteten även för appar med så dynamisk nätverksanvändning som Spotify.

Acknowledgments

This thesis was carried out at Spotify in Stockholm and examined at the Department of Computer and Information Science, Linköping University.

I would like to thank my supervisor at Spotify, Fredrik Stridsman, for his support and much appreciated feedback throughout my work. I am also grateful to my examiner, Niklas Carlsson, for going above and beyond on his mission with great suggestions and guidance.

The input and support from my supervisor Nahid Shahmehri and my colleagues at Spotify Erik Junberger and Nils Loodin have been greatly appreciated.

Thanks also to my opponent, Rickard Englund, for his constructive comments.

Last but not least, my fellow thesis student's and all the extraordinary colleagues at Spotify that have inspired me and made my stay at Spotify an interesting and fun experience. Thank you.

Stockholm, June 2014
Robert Nissa Holmgren

Contents

List of Figures	xiii
List of Tables	xv
List of Listings	xvii
Notation	xix
1 Introduction	1
1.1 Mobile App’s Network Activity as a Quality Measure	1
1.1.1 Challenges	2
1.1.2 Types of Network Activity Change	3
1.2 Spotify	3
1.2.1 Automated Testing at Spotify	4
1.2.2 Spotify Apps’ Network Usage	4
1.3 Problem Statement	5
1.4 Contributions	5
1.5 Thesis Structure	6
I Theory	
2 Computer Networks	9
2.1 Internet Protocols	9
2.1.1 IP and TCP/UDP	9
2.1.2 Lower Level Protocols	12
2.1.3 Application Protocols	12
2.1.4 Encrypted Protocols	12
2.1.5 Protocol Detection	12
2.2 Spotify-Specific Protocols	13
2.2.1 Hermes	14
2.2.2 Peer-to-Peer	14
2.3 Content Delivery Networks	14

2.4	Network Intrusion Detection Systems	15
3	Machine Learning	17
3.1	Probability Theory	17
3.2	Time Series	18
3.3	Anomaly Detection	18
3.3.1	Exponentially Weighted Moving Average	19
3.4	k-Means Clustering	20
3.4.1	Deciding Number of Clusters	21
3.4.2	Feature Extraction	22
3.5	Novelty Detection	24
3.6	Evaluation Metrics	25
3.7	Tools	26
3.8	Related Work	27
3.8.1	Computer Networking Measurements	27
3.8.2	Anomaly and Novelty Detection	28

II Implementation and Evaluation

4	Measurement Methodology	33
4.1	Measurements	33
4.1.1	General Techniques	34
4.1.2	Mobile Apps	34
4.1.3	Tapping into Encrypted Data Streams	36
4.2	Processing Captured Data	38
4.2.1	Extracting Information Using Bro	38
4.2.2	Transforming and Extending the Data	38
4.2.3	DNS Information	38
4.2.4	Other Network End-Point Information	39
4.3	Data Set Collection	40
4.3.1	Environment	40
4.3.2	User Interaction – Test Cases	40
4.3.3	Network Traffic	40
4.3.4	App and Test Automation Instrumentation Data Sources	41
4.4	Data Set I - Artificial Defects	42
4.4.1	Introduced Defects	42
4.4.2	Normal Behavior	43
4.4.3	Test Cases	43
4.4.4	Summary	44
4.5	Data Set II - Real World Scenario	45
4.5.1	Test Cases	45
4.5.2	Summary	46
5	Detecting and Identifying Changes	47
5.1	Anomaly Detection Using EWMA Charts	47
5.1.1	Data Set Transformation	48

5.1.2	Detecting Changes	48
5.2	Novelty Detection Using k-Means Clustering	51
5.2.1	Feature Vector	51
5.2.2	Clustering	52
5.2.3	Novelty Detection	53
6	Evaluation	55
6.1	Anomaly Detection Using EWMA Charts	55
6.1.1	First Method ROC Curves	56
6.1.2	Better Conditions for Classifying Defects as Anomalous . .	56
6.1.3	Detected Anomalies	57
6.2	Novelty Detection Using k-Means Clustering – Data Set I	63
6.2.1	ROC Curves	63
6.2.2	Detected Novelties	64
6.3	Novelty Detection Using k-Means Clustering – Data Set II	68
6.3.1	Detected Novelties	68
7	Discussion and Conclusions	71
7.1	Discussion	71
7.1.1	Related Work	72
7.2	Future Work	73
7.2.1	Updating the Model of Normal	73
7.2.2	Keeping the Model of Normal Relevant	73
7.2.3	Improve Identification of Service End-Points	73
7.2.4	Temporal Features	74
7.2.5	Network Hardware Energy Usage	74
7.3	Conclusions	74
A	Data Set Features	79
B	Data Set Statistics	83
B.1	Data Set I - Artificial Defects	83
B.2	Data Set II - Real World Scenario	91
	Bibliography	97

List of Figures

2.1	UDP encapsulation	10
3.1	Example EWMA chart.	20
3.2	k-means clustering example	21
3.3	Clustering silhouette score	23
3.4	Label binarization of categorical feature	24
3.5	ROC curve example	26
5.1	EWMA chart of T3, A2, network footprint.	49
5.2	EWMA chart of T1, A4, network footprint.	50
6.1	ROCs of EWMA, network footprint.	56
6.2	ROCs of EWMA, network footprint, better conditions for positive detection.	57
6.3	ROCs of EWMA, number of packets, better conditions for positive detection	58
6.4	ROCs of EWMA, number of distinct network end-points, better conditions for positive detection	58
6.5	ROCs of EWMA, number of distinct AS/service pairs, better conditions for positive detection.	59
6.6	EWMA chart of T2, A4, ASN-service pairs	61
6.7	EWMA chart of T2, A4, ASN-service pairs, ad-hoc verification data set	61
6.8	ROC curve of k-means clustering novelty detection of stream families.	63
6.9	Identified novelties in data set of defect vs normal.	64

List of Tables

1.1	Thesis chapter structure.	6
3.1	Confusion matrix of an anomaly/novelty detection system.	25
4.1	Number of collected test case runs for each test case and app version for data set I.	44
4.2	Number of collected test case runs for each test case and app version for data set II.	46
5.1	Feature vector for k-means novelty detection.	52
6.1	Detection performance numbers for EWMA on the A1 defect.	59
6.2	Detection performance numbers for EWMA on the A2 defect.	60
6.3	Detection performance numbers for EWMA on the A3 defect.	60
6.4	Detection performance numbers for EWMA on the A4 defect.	62
6.5	Detection performance numbers for k-means novelty detection	67
A.1	Features extracted with Bro from each network packet of the raw network data dump.	80
A.2	Features derived from features in Table A.1.	81
A.3	Features extracted from the test automation tool.	81
A.4	Features extracted from the instrumented client.	82
B.1	Data set statistics for test case T1	83
B.2	Data set statistics for test case T2	86
B.3	Data set statistics for test case T3	88
B.4	Data set statistics for test case T4	91
B.5	Data set statistics for test case T5	92
B.6	Data set statistics for test case T6	94

List of Listings

2.1	Bro script for dynamic detection of the Spotify AP protocol.	13
4.1	Starting a Remote Virtual Interface on a Connected iOS Device (from rvictl documentation).	35
4.2	Algorithm to calculate network hardware active state with simple model of the network hardware.	39
4.3	Command to start tcpdump to capture the network traffic	41
4.4	Login and Play Song (T1)	43
4.5	Login and Play Song, Exit The App and Redo (T2)	43
4.6	Login and Create Playlist From Album, Exit The App and Redo (T3)	44
4.7	Spotify iOS 1.1.0 Release Notes	45
4.8	Artist page biography and related artists (T4)	45
4.9	Display the profile page (T5)	45
4.10	Add an album to a playlist and play the first track (T6)	46

Notation

ABBREVIATIONS

Abbreviation	Meaning
AP	Access Point – In Spotify’s case a gateway for Spotify clients to talk to back-end services.
API	Application Programming Interface – Specifies how one software product can interact with another software product.
AS	Autonomous System – An autonomous network with internal routing connected to the Internet.
ASN	Autonomous System Number – Identifying number assigned to an AS.
CD	Continuous Delivery – Software development practice which requires that the product developed always is in a releasable state by using continuous integration and automated testing. May also use continuous deployment to automatically release a new version for each change that passes testing [8].
CDN	Content Delivery Network – Distributed computer system used to quickly deliver content to users.
COTS	Commercial off-the-shelf – Refers to products available for purchase, and therefore do not need to be developed.
DNS	Domain Name System – Distributed lookup system for key-value mapping, often used to find IP-addresses for a hostname.
EWMA	Exponentially Weighted Moving Average
FPR	False Positive Rate – Statistical performance measure of a binary classification method. Number of correctly identified negative samples over total number of negative samples.

Abbreviation	Meaning
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol – Application level networking protocol used to transfer resources on the World Wide Web.
HTTPS	HyperText Transfer Protocol Secure – HTTP inside a TLS or SSL tunnel.
ICMP	Internet Control Message Protocol – The primary protocol to send control messages, such as error notifications and request for information, over the Internet.
IEEE	Institute of Electrical and Electronics Engineers – A professional association, which among other things create IT standards.
IP	Internet Protocol – Network protocol used on the Internet to facilitate packet routing, etc.
ISP	Internet Service Provider – A company that provides the service of Internet connections to companies and individuals.
KDD	Knowledge Discovery in Databases – The process of selecting, preprocess, transform, data mine and interpret databases into higher level knowledge.
MitM	Man-in-the-Middle attack – Eavesdropping by inserting oneself between the communicating parties and relaying the messages.
NIDS	Network Intrusion Detection System – A system designed to identify intrusion attempts to computer systems by observing the network traffic.
NIPS	Network Intrusion Prevention System – A Network Intrusion Detection System capable of taking action to stop a detected attack.
P2P	Peer-to-Peer – Decentralized and distributed communication network where hosts both request and provide resources (e.g. files) from and to each other.
PCAP	Packet CAPture – Library and file format to capture and store network traffic.
PCAPNG	PCAP Next Generation – New file format to store captured network traffic. Tools compatible with PCAP files does not necessarily handle PCAPNG files.
PSK	Pre-Shared Key – In cryptology: A secret shared between parties prior to encryption/decryption.
PTR	Pointer – DNS record mapping an IP-address to a host name.
SDK	Software Development Kit – Hardware and software tools to aid software development for a platform/system. May include compilers, libraries, and other tools.

Abbreviation	Meaning
SPAN	Switch Port ANalyzer – Cisco’s system for mirroring a switch port.
SPAP	Spotify AP protocol – Notation used in this thesis to denote Spotify’s proprietary AP protocol.
SPDY	(pronounced speedy) – Application level network protocol the World Wide Web. Developed as an alternative to HTTP in an effort to reduce latency of the web. Base for the upcoming HTTP 2.0 standard.
SSH	Secure Shell – An encrypted network protocol for data communication. Often used for remote login and command line access.
SSL	Secure Sockets Layer – An encrypted network protocol for encapsulating other protocols. Superseded by TLS, but is still in use.
SUT	System Under Test – The system being subjected to the test(s) and evaluated.
TCP	Transmission Control Protocol – Transport layer protocol used on the Internet, which provides reliable, ordered and error-checked streams of data.
TLS	Transport Layer Security – An encrypted network protocol for encapsulating other protocols. Supersedes SSL.
TPR	True Positive Rate – Statistical performance measure of a binary classification method. Number of correctly identified positive samples over total number of positive samples.
UDP	User Datagram Protocol – Transport layer protocol used on the Internet, which provide low overhead, best effort delivery of messages.
URL	Uniform Resource Locator – A string used to locate a resource by specifying the protocol, a DNS or network address, port and path.
VPN	Virtual Private Network – An encrypted tunnel for sending private network traffic over a public network.
WiFi	Trademark name for WLAN products based on the IEEE 802.11 standards.
WiP	Work in Progress
WLAN	Wireless Local Area Network
XP	Extreme Programming – An agile software development methodology.

TERMINOLOGY

Term	Definition
Defect	An introduced change leading to unwanted impact on network activity or network footprint.
Network activity	How much the network hardware is kept alive by network traffic.
Network end-point	A unique combination of network and transport layer identifiers, such as IP address and TCP port.
Network footprint	Total number of bytes sent and received for a specific test session.
Service end-point	A service running on any number of networks, physical or virtual machines, IP-address, port numbers and protocols, which is providing clients access to the same functionality and data. Examples: (1) A cluster of web servers serving the same web pages over HTTP (TCP 80), HTTPS (TCP 443) and SPDY from a number of IP addresses, connected to different service providers for redundancy. (2) Spotify's access points, running on a number of machines in various locations. Serving clients with access to Spotify's back-end services over the TCP ports 4070, 443 and 80.
Stream	The same definition as Bro uses for connection: "For UDP and ICMP, 'connections' are to be interpreted using flow semantics (sequence of packets from a source host/port to a destination host/port). Further, ICMP 'ports' are to be interpreted as the source port meaning the ICMP message type and the destination port being the ICMP message code." ¹
Test case	A list of user interactions with the SUT.
Test case run	A run of a test case, which produces log artifacts of the network traffic, test driving tool and the client.

¹Bro script documentation, official site, <http://www.bro.org/sphinx/scripts/base/protocols/conn/main.html>, May 2014.

1

Introduction

Smartphones are becoming more and more common. With higher resolution displays, more media and apps trying to be more engaging the data usage per device and month is increasing quickly [5]. While mobile service providers are addressing the insatiable thirst for more and faster data access with new technologies and more cell towers, the air is a shared and highly regulated medium and therefore expensive to grow capacity in. Having realized that data is becoming the majority load on their networks, the service providers have changed pricing strategies to make SMS and voice calls cheap or free and started charging a premium for data¹ as well as limiting the maximum data packet sizes and moving from unlimited packets to tiered data [5].

1.1 Mobile App's Network Activity as a Quality Measure

As both mobile service providers and users want to minimize network activity there is a clear incentive for developers to minimize wasted traffic and ensure that their app's network usage is essential for the user experience. This can be done in various ways, but pragmatic developers tend to follow the old words of wisdom "to measure is to know"² and find out when, what, why, and with what their app is communicating.

Explicitly measuring, visualizing and automatically regression test the network activity gives several advantages to multiple stakeholders:

¹Article "Telia: Billigare samtal ger dyrare data", published July 2012, <http://www.mobil.se/operat-rer/telia-billigare-samtal-ger-dyrare-data> (In Swedish), February 2014

²Common paraphrase of Lord Kelvin. Full quote in Chapter 4.

- Developers can use this information to implement network-active features with better confidence, knowing the behavior under the tested conditions.
- Testers get tools to test if complex multi-component systems such as caching, network switching strategies and offline mode are working as intended.
- Product owners know when, how much and why the network traffic consumption of the application changes.
- Researchers and curious users can get some insight into the communication patterns of apps and may compare the network footprint of different versions of one app or compare different apps under various configurations and conditions.
- External communicators have reliable and verifiable data on the network footprint of the app, which is highly useful when, e.g., having your app bundled with mobile phone plans and one of the terms is to exclude the app's network consumption from the end-users bill.

Effectively measuring, visualizing and automatically test the network activity is particularly important for larger projects with many developers, stakeholders, and partners. While the ins and outs of a small app project sometimes easily can be handled by a single developer, larger projects often spans developers located across multiple offices, working autonomously on different sub-components. As new components are added and removed, employees or even entire development teams come and go, such large app projects need good tools to maintain knowledge and understanding of the system performance under different conditions.

Manual software testing is generally a tedious and labor-intensive process and therefore costly to use for repeated regression testing. Automated testing can shorten the feedback loop to developers, reduce testing cost and enable exercising the app with a larger test suite more often [10]. Agile development practices such as continuous delivery (CD) and extreme programming (XP) require automated testing as a part of the delivery pipeline – from unit tests to acceptance tests [8, 10]. Network activity regression testing can be considered performance and acceptance tests.

1.1.1 Challenges

Automatically comparing generated network traffic for apps with complex network behavior have some inherit difficulties. Even the traffic for consecutive runs of the same app build under the same conditions is expected to vary in various characteristics, including:

- server end-node, due to load balancing;
- destination port numbers, due to load balancing or dynamic fallback strategies for firewall blocks;
- application layer protocol, due to routing by dynamic algorithms such as A/B testing strategies for providing large quick streaming files; and

- size and number of packets, due to resent traffic caused by bad networking conditions.

There are more characteristics that are expected to vary and more reasons to why than stated above as the Internet and the modern communication systems running over it are highly dynamic.

Comparison can be done by manually classifying traffic and writing explicit rules for what is considered normal. These rules would have to be updated, bug fixed and maintained as the app's expected traffic patterns changes. Perhaps a better strategy would be to construct a self-learning system, which builds a model of expected traffic by observing test-runs of a version of the app that is considered "known good". This thesis will focus on the latter.

1.1.2 Types of Network Activity Change

There are a lot of interesting characteristics in the network traffic of mobile apps, which stakeholders would like to be able to regression test. To delimit this thesis we have focused on these characteristics:

- **Network footprint:** Total number of bytes uploaded and downloaded. Mobile network traffic is expensive, a shared resource and unnecessary traffic may cause latency or sluggishness in the app.
- **End-points:** Which service end-points (see definition in Table 2) the app talks to. In many projects new code may come from a number of sources and is not always thoroughly inspected before it is shipped. Malicious or careless developers may introduce features or bugs making the app upload or download unwanted data. This new traffic may be to previously unseen service end-points; since it is possible the developer does not control the original service end-points.
- **Network hardware energy usage:** Network hardware uses more energy when kept in an active state by network traffic. Timing network usage well may reduce an app's negative battery impact.
- **Latency:** Round trip time for network requests.

Latency is not directly considered in this thesis as the author think there are better ways of monitor network and service latency of the involved back-end services and (perceived) app latency than network traffic change analysis of app versions.

1.2 Spotify

Spotify is a music streaming service, which was founded in Sweden 2006. The Spotify desktop application and streaming service was launched for public access October 2008. The company has grown from a handful of employees at launch to currently over 1,000 in offices around the world. A large part of the work force are involved in developing the Spotify software and are working out of

four cities in Sweden and the USA. Spotify is providing over 10 million paying subscribers and 40 million active users in 56 countries³ with instant access to over 20 million music tracks⁴. Spotify builds and maintain clients and libraries that run on Windows, OS X, Linux, iOS, Android, Windows Phone, regular web browsers, and on many other platforms, such as receivers and smart TVs. Some of the clients are built by, or in collaboration with, partners.

Spotify strives to work in a highly agile way with small, autonomous and cross-functional teams called squads, which are solely responsible for parts of the Spotify product or service. This lets the squads become experts in their area, and develop and test solutions quickly. The teams are free to choose their own flavor of Agile or to create one themselves, but most use some modification of Scrum or Kanban sprinkled with values and ideas from Extreme Programming (XP), Lean and Continuous Delivery.

1.2.1 Automated Testing at Spotify

Spotify have a test automation tool used to automate integration and system tests on all the clients. The test automation tool uses GraphWalker⁵ to control the steps of the test that enables deterministic or random walks through a graph where the nodes are verifiable states of the system under test (SUT) and edges are actions [15]. The test automation tool then has some means of interacting with the SUT as a user would: reading text, inputting text, and clicking things. For the Spotify iOS project this is done using a tool called NuRemoting⁶, which opens a network server listening for commands and executing them in the app. NuRemoting also send the client's console log to its connected client.

Automatic tests are run continuously and reported to a central system, which provides feedback to the teams through dashboards with results and graphs.

1.2.2 Spotify Apps' Network Usage

Spotify's client apps have always used a multiplexed and encrypted proprietary protocol connected to one of their access points (APs) in the back-end for all communication with the back-end systems. Nowadays this is supplemented with various side-channels to hypertext transfer protocol (HTTP)-based content delivery networks (CDNs) and third-party application programming interfaces (APIs). The desktop version of the apps also establish a peer-to-peer (P2P) network with other running instances of the Spotify desktop client for fetching music data from nearby computers, which also decreases the load and bandwidth costs of Spotify's servers [14, 9]. Spotify's mobile clients do not participate in this P2P network [13], so P2P will not be a primary concern in this thesis.

³Spotify Press, "Spotify hits 10 million global subscribers", <http://press.spotify.com/us/2014/05/21/spotify-hits-10-million-global-subscribers/>, May 2014

⁴Spotify Fast Facts December 2013, <https://spotify.box.com/shared/static/8eteff2q4tjzpaagi49m.pdf>, February 2014

⁵GraphWalker (official website), <http://graphwalker.org>, February 2014

⁶NuRemoting (official website), <https://github.com/nevyn/NuRemoting>, February 2014

Today the total amounts of uploaded and downloaded data as well as the number of requests are logged for calls routed through the AP. There are ways of having the HTTP requests of the remaining network communication logged as well, but there are no measurements on whether this is consistently used by all components and therefore not enough confidence in the data. Furthermore the logged network activity is submitted only periodically, which means chunks of statistics may be lost because of network or device stability issues.

1.3 Problem Statement

This thesis considers the problem of making the network traffic patterns of an application available to the various stakeholders in its development to help them realize the impact of their changes on network traffic. The main problem is how to compare the collected network traffic produced by test cases to detect changes without producing too many false positives, which would defeat the tool's purpose as the it would soon be ignored for "crying wolf". To construct and evaluate the performance of the anomaly detection system the thesis will also define a set of anomalies that the system is expected to detect.

The primary research questions considered in this thesis are the following:

- What machine learning algorithm is most suitable for comparing network traffic sessions for the purpose of identifying changes in the network footprint and service end-points of the app?
- What are the best features to use and how should they be transformed to suit the selected machine learning algorithm when constructing a network traffic model that allows for efficient detection of changes in the network footprint and service end-points?

1.4 Contributions

The contributions of this thesis are:

- A method to compare captured and classified network activity sessions and detect changes to facilitate automated regression testing and alerting stakeholders of anomalies.

To deliver these contributions the following tools have been developed:

- A tool for setting up an environment to capture the network traffic of a smartphone device, integrated into an existing test automation tool.
- A tool to classify and reduce the captured network traffic into statistics such as bytes/second per protocol and end-point.
- A tool to determine what network streams have changed characteristics using machine learning to build a model of expected traffic, used to highlight the changes and notify the interested parties.

Table 1.1: Thesis chapter structure.

Chapter	Content
1	Introduces the thesis (this chapter).
2	Gives background on computer networking.
3	Gives background on machine learning and anomaly/novelty detection.
4	Describes the proposed techniques to capture an app's network activity and integrating with a test automation tool. It also describes the collected data sets used to design and evaluate the change detection methods.
5	Describes the proposed way to compare captured network traffic to facilitate automated regression analysis.
6	Evaluates the proposed methods for network activity change detection.
7	Wraps up the thesis with a closing discussion and conclusions.

Together these tools form a system to measure network activity for test automation test cases, compare the test results to find changes, and visualize the results.

1.5 Thesis Structure

In **Chapter 1** the thesis is introduced with background and motivations for the considered problems. Then follows a technical background on computer networking in **Chapter 2** and machine learning in **Chapter 3**. **Chapter 4** introduces our measurement methodology and data sets. The proposed methods and developed tools are described in **Chapter 5**. **Chapter 6** evaluates the proposed methods on the data sets. **Chapter 7** wraps up the thesis with discussion and conclusions.

A structured outline of the thesis can be found in Table 1.1.

Part I

Theory

2

Computer Networks

This chapter gives an introduction to computer networks and their protocols.

2.1 Internet Protocols

To conform to the standards, be a compatible Internet host and be able to communicate with other Internet hosts, Internet hosts need to support the protocols in RFC1122 [3]. RFC1122 primarily mentions the Internet Protocol (IP), the transport protocols Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), and the Internet Control Message Protocol (ICMP). The multicast support protocol IGMP and link layer protocols are also mentioned in RFC1122, but will not be regarded in this thesis since IGMP is optional and the link layer protocols does not add anything to this thesis (see Section 2.1.2).

When sending data these protocols work by accepting data from the application in the layer above them, possibly split them up according to their specified needs and add headers to describe to their counterpart on the receiver where the data should be routed for further processing. This layering principle can be observed in Figure 2.1.

2.1.1 IP and TCP/UDP

IP is the protocol that enables the Internet scale routing of datagrams from one node to another. IP is connectionless and packet-oriented. The first widely used IP protocol was version 4 and still constitutes a vast majority of the Internet traffic. IPv6 was introduced in 1998 to among other things address IPv4's quickly diminishing number of free addresses.

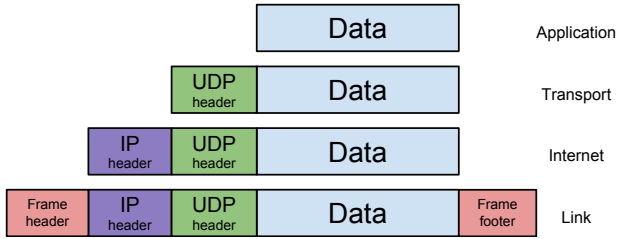


Figure 2.1: Overview of a data message encapsulated in UDP by adding the UDP header. Then IP add the IP header. Finally Ethernet adds its frame header and footer.

IPv4 has a variable header size, ranging from 20 bytes to 60 bytes, which is specified in its Internet Header Length (IHL). IPv6 opted for a fixed header size of 40 bytes to enable simpler processing in routers, etc. To not lose flexibility IPv6 instead defines a chain of headers linked together with the next header field.

There two major transport protocols on the Internet are TCP and UDP. TCP establishes and maintains a connection between two hosts and transports streams of data in both directions. UDP is connection-less and message oriented and transports messages from a source to a destination. TCP provide flow control, congestion control, and reliability, which can be convenient and useful in some applications but come at the price of among other things latency in connection establishment and overhead of transmitted data. UDP is more lightweight and does not provide any delivery information, which can make it a good choice when the application layer protocol want minimize latency and take care of any delivery monitoring it deems necessary.

TCP has a variable header size of 20 bytes up to 60 bytes, contributing to its overhead. UDP has a fixed header size of 8 bytes.

Addressing

Addressing on the Internet is done for multiple protocols in different layers, where each protocol layer's addressing is used to route the data to the correct recipient. To be able to communicate over IP, hosts need to be allocated an IP address. IP address allocation is centrally managed by the Internet Assigned Numbers Authority (IANA), which through regional proxies, allocate blocks of IP addresses (also known as subnets) to ISPs and large organizations.

To know how to reach a specific IP address at any given time the ISPs keep a database of which subnets can be reached through which link. This database is constructed by routing protocol, of which BGP is the dominating on the Internet level. BGP uses Autonomous System Numbers (ASNs) for each network (Autonomous System) to identify the location of the subnets. A bit simplified BGP announces, "ASN 64513 is responsible for IP subnets 10.16.0.0/12 and 172.16.14.0/23" to the world. For each IP address endpoint it is therefore pos-

sible to say what network it is a part of, which can be useful when analyzing traffic endpoint similarity: when the destination IP is randomly selected from a pool of servers it may still be part of the same ASN as the other servers.

Domain Name System (DNS) is a mapping service from hierarchical names, which often are easy for humans to recall, to IP addresses. DNS is also used as a distributed database for service resolution and metadata for a domain. A common technique to achieve some level of high availability and load balance is to map a DNS name to several IP addresses, as can be observed for “www.spotify.com” which as of writing resolves to 193.235.232.103, 193.235.232.56 and 193.235.232.89. An IP address may have multiple DNS names resolving to it and a DNS name may resolve to multiple IPs; DNS name to IP is a many-to-many relation.

DNS also keep a reverse mapping from IP addresses to a DNS name, called a pointer (PTR). An IP can only have one PTR record, whereas a DNS name can have multiple mappings to IP; that is IP to DNS name is a many-to-one relation. PTR records for servers may contain information indicating the domain (sometimes tied to an organization) they belong to and sometime what service they provide. The three IP addresses above resolves through reverse DNS to “www.lon2-webproxy-a3.lon.spotify.com.”, “www.lon2-webproxy-a1.lon.spotify.com.” and “www.lon2-webproxy-a2.lon.spotify.com.” respectively, indicating that they belong to the spotify.com domain, lives in the London data center and perform the www/web proxy service.

DNS information may, similarly to ASN, contribute to determining traffic endpoint similarity. There are high variations in naming schemes, and advanced configuration or even errors occur frequently, so DNS ought to be considered a noisy source for endpoint similarity; even so, it may provide correct association where other strategies fail.

Transport level protocols (UDP and TCP) use port numbers for addressing to know which network socket is the destination. Server applications create listening network sockets to accept incoming requests. Many server application types use a specific set of port numbers so that clients may know where to reach them without a separate resolution service. The Internet Assigned Numbers Authority (IANA) maintains official port number assignments such as 80 for WWW/HTTP, but there are also a number of widely accepted port number-application associations that are unofficial, such as 27015 for Half-life and Source engine game servers. Using the server’s transport protocol port number may be useful in determining the service type for endpoint similarity, but may also be deceitful as a server may provide the same service over a multitude of ports so that compatible clients have a higher probability of establishing a connection when the client is behind an egress (outgoing network traffic) filtering firewall. The source port for traffic from clients to servers is selected in a pseudo random way to thwart spoofing [29, 16].

2.1.2 Lower Level Protocols

There are also underlying layers of protocol that specifies how transmission of traffic is done on the local network (Link in Figure 2.1) and physically on the wire. These protocols are not further described here, as they will not be considered in traffic size and overhead in this thesis, as they varies for wired networks, WiFi and cellular connections. Including the link layer protocols would complicate the network traffic collection for cellular networks as the information is not included with our measurement techniques and complicate comparing network access patterns of test case runs because of differing header sizes, while not contributing to better change detection of the app's network footprint.

2.1.3 Application Protocols

The Internet enabled applications also need standard on how to communicate. Web browsers commonly use the HTTP protocol to request web pages from web servers. HTTP is a line-separated plain-text protocol and therefore easy for humans to analyze without special protocol parsing tools. HTTP's widespread use, relatively simple necessary parts and flexible use-case have made it popular to use for application-to-application API communication as well.

2.1.4 Encrypted Protocols

With the growing move of sensitive information onto the Internet with banking, password services, health-care and personal information on social networks, traffic encryption has become common. HTTP's choice for encryption is the transport layer security (TLS)/secure sockets layer (SSL) suite. TLS for HTTP is often used with X.509 certificates signed by certificate authorities (CA). Which CAs are to be trusted for signing certificates is defined by the operating system or the browser. There are also proprietary and non-standard encryption systems, as well as many more standardized.

Encryption makes classifying and analyzing traffic harder as it is by its very design hard to peek inside of the encrypted packets. This can in some cases be alleviated when controlling one of the end nodes by telling it to trust a middleman (e.g. by adding your own CA to the trusted set) to proxy the traffic or by having it leak information on what it is doing via a side-channel.

2.1.5 Protocol Detection

In the contemporary Internet one can no longer trust the common 5-tuple (protocol type, source IP, destination IP, source ports, destination port) to provide trustworthy information on what service is actually in use [20]. Some of the reasons for this may be for the system generating the traffic to avoid (easy and cheap) detection and filtration of its traffic (e.g. P2P file-sharing) and to handle overly aggressive firewall filtration. There are various suggestions on techniques to classify traffic streams as their respective protocols, including machine learning [20] and matching on known protocol behavior.

Bro

Bro¹ is a network analysis framework that among other things can be used to determine the protocol(s) of a connection [22]. Bro can be run on live network traffic or previously captured traffic in a supported format, and in its most basic case output a set of readable log files with information about the seen traffic. Being a framework it can be extended with new protocols to detect and scripted to output more information.

Listing 2.1: Bro script for dynamic detection of the Spotify AP protocol.

```
# 3 samples of the first 16 bytes of a client establishing a connection,
# payload part. Collected and displayed with tcpdump + Wireshark.
#0000 00 04 00 00 01 12 52 0e 50 02 a0 01 01 f0 01 03 .....R.P.....
#0000 00 04 00 00 01 39 52 0e 50 02 a0 01 01 f0 01 03 .....9R.P.....
#0000 00 04 00 00 01 a3 52 0e 50 02 a0 01 01 f0 01 03 .....R.P.....
signature dpd_spap4_client {
  ip-proto == tcp
  # Regex match the observed common parts
  payload /^\x00\x04\x00\x00..\x52\x0e\x50\x02\xa0\x01\x01\xf0\x01\x03/
  tcp-state originator
  event "spap4_client detected"
}

# 3 samples of the first 16 bytes of server response to above connection,
# payload part. Collected and displayed with tcpdump + Wireshark.
#0000 00 00 02 36 52 af 04 52 ec 02 52 e9 02 52 60 93 ...6R..R..R`.
#0000 00 00 02 38 52 b1 04 52 ec 02 52 e9 02 52 60 27 ...8R..R..R`.
#0000 00 00 02 96 52 8f 05 52 ec 02 52 e9 02 52 60 0d ....R..R..R`.
signature dpd_spap4_server {
  # Require the TCP protocol
  ip-proto == tcp
  # Regex match the observed common parts
  payload /^\x00\x00..\x52..\x52\xec\x02\x52\xe9\x02\x52\x60/
  # Require that the client connection establishment was observed in
  # this connection
  requires-reverse-signature dpd_spap4_client
  tcp-state responder
  event "spap4_server response detected"
  # Mark this connection with service=SPAP
  enable "spap"
}
```

2.2 Spotify-Specific Protocols

Spotify primarily uses a proprietary protocol that establishes a single TCP connection to one of Spotify's edge servers (access points, APs). This connection is then used to multiplex all messages from the client to Spotify's back-end services [14]. This connection is encrypted to protect the messages and the protocol from reverse engineering.

¹Bro (official website), <http://bro.org>, February 2014

Supplementing this primary connection to a Spotify AP are connections using more common protocols like HTTP and HTTP secure (HTTPS).

2.2.1 Hermes

Spotify uses another proprietary protocol called Hermes. Hermes is based on ZeroMQ², protobuf³ and HTTP-like verbs for message passing between the client and the back-end services⁴ [25]. These messages are sent over the established TCP connection to the AP. Hermes messages use proper URIs to identify the target service and path, which is useful in identifying the purpose and source of the message. The Hermes URIs starts with “hm://”, designating the protocol Hermes.

2.2.2 Peer-to-Peer

Spotify’s desktop clients creates a peer-to-peer (P2P) network with other Spotify desktop clients to exchange song data. This serves to reduce the bandwidth load and thereby the cost on Spotify’s back-end servers and in some cases reduce latency and/or cost by keeping user’s Spotify traffic domestic. The P2P mechanism is only active in the desktop clients and not on smartphones, the web client or in libspotify [13].

This thesis focuses on the mobile client and is therefore not further concerned with the P2P protocol. One advantage of excluding P2P traffic from the analysis is that we avoid its probably non-deterministic traffic patterns caused by the random P2P neighbors random cache misses from random song plays.

2.3 Content Delivery Networks

A Content Delivery Network or Content Distribution Network (CDN) is “network infrastructure in which the network elements cooperate at network layers 4 [transport] through 7 [application] for more effective delivery of content to User Agents [web browsers],” as defined in RFC6707 [21]. CDNs perform this service by placing caching servers (Surrogates) in various strategic locations and route requests to the best Surrogate for each request, where best may be determined by a cost/benefit function with parameter such as geographical distance, network latency, request origin network, transfer costs, current Surrogate load and cache status for the requested content.

Different CDNs have different resources and strategies for placing Surrogate. Some observed patterns are (1) leasing servers and network capacity in commercial data centers and use IP addresses assigned by the data center; (2) using several other CDN providers; (3) using their own IP address space(s) and AS numbers;

²ZeroMQ (official website), <http://zeromq.org>, February 2014

³Protobuf (repository), <https://code.google.com/p/protobuf/>, February 2014

⁴Presentation Slides on Spotify Architecture - Press Play, by Niklas Gustavsson <http://www.slideshare.net/protocol7/spotify-architecture-pressing-play>, February 2014

and (4) using their own IP address space(s) and AS numbers, combined with Surrogates on some Internet Service Providers' (ISP's) network, using the ISP's addresses.

The different Surrogate placing strategies and dynamic routing makes determining if two streams belong to the same service end-point hard. It can be especially hard for streams originating from different networks or at different times, as the CDN may have different routing rules for the streams. Spotify utilizes several CDNs and the traffic will therefore show signs of several of the patterns above.

Some data sources that can be useful in determining if two streams are indeed to the same service end-point are (1) the AS number, (2) the DNS PTR for the IP address, (3) the DNS query question string used to find the network end-point IP address, (4) X.509 certificate information for TLS/SSL connections, and (5) the host-field of HTTP requests; (6) content provider hybrid solutions with CDNs and dedicated servers to get lower cost and better customer proximity [6], as these often of legal or best practice reasons contain information related to the service, the content provider and/or the CDN provider.

2.4 Network Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) are systems strategically placed to monitor the network traffic to and from the computer systems it aims to defend. They are often constructed with a rule matching system and a set of rules describing the patterns of attacks. Some examples of NIDS software are SNORT⁵ and Suricata⁶.

Related to NIDS are NIPS – Network Intrusion Prevention Systems – designed to automatically take action and terminate detected intrusion attempts. The termination is typically done by updating firewall rules to filter out the offending traffic.

⁵SNORT official web site, <http://snort.org>, May 2014

⁶Suricata official web site, <http://suricata-ids.org>, May 2014

3

Machine Learning

Arthur Samuel defined machine learning in 1959 as a “field of study that gives computers the ability to learn without being explicitly programmed” [26, p. 89]. This is achieved by running machine learning algorithms on data to build up a model, which then can be used to predict future data. There are a multitude of machine learning algorithms, many of which can be classified into the categories supervised learning, unsupervised learning and semi-supervised learning based on what data they require to construct their model.

Supervised learning algorithms take labeled data: samples of data together with information on how the algorithm should classify each sample. Unsupervised learning algorithms take unlabeled data: samples without information on how it is supposed to be classified. The algorithm will then need to infer the labels from the data itself. Semi-supervised learning algorithms have multiple definitions in literature. Some researchers define semi-supervised learning as having a small set of labeled data combined with a larger set of unlabeled data to boost the learning. Other, especially in novelty detection, defines semi-supervised learning as only giving the algorithm samples of normal class data [11].

3.1 Probability Theory

A stochastic variable is a variable that takes on values by chance, or random, from a sample space. The value of a stochastic variable is determined by its probability distribution.

The mean of a stochastic variable X is denoted μ_X and is for discrete stochastic

variables defined as:

$$\mu_X = \mathbf{E}[X] = \sum_{i=1}^N x_i p_i,$$

where p_i is the probability of outcome x_i and N the number of possible outcomes. For a countable but non-finite number of outcomes $N = \infty$.

The variance of a stochastic variable X is the expected value of the squared deviation from the mean μ_X :

$$\text{Var}(X) = \mathbf{E}[(X - \mu_X)^2].$$

Standard deviation is defined as the square root of the variance:

$$\sigma_X = \sqrt{\text{Var}(X)}.$$

A stochastic process is a collection of stochastic variables. Stochastic processes where all stochastic variables have the same mean and variance are called stationary processes. Non-stationary processes' stochastic variables can have different mean and variance, meaning the process probability distribution can change over time.

3.2 Time Series

A time series is a sequence of data points where each data point correspond to a sample of a function. The sampling interval is usually uniform in time and the function can be the number of bytes transmitted in since the last sample.

In this thesis we make use of data in ordinary time series data. We also consider another format where the sampling is not done with uniform time interval, but triggered by an event, e.g. the end of a test case. This will form a series of data points, which can be treated as a time series for some algorithms, like Exponentially Weighted Moving Average (EWMA).

3.3 Anomaly Detection

Anomaly detection is the identification of observations that do not conform to expected behavior. Anomaly detection can be used for intrusion detection, fraud detection and detection of attacks on computer networks, to name a few applications. In machine learning anomaly detection is among other things used to automatically trigger an action or an alarm when an anomaly is detected, which enables manual or automatic analysis and mitigation of the cause. Because of the typically non-zero cost associated with manual/automatic analysis and mitigation the anomaly detection a low rate of false positives is desirable, and since there is often a value in the observed process working as expected the anomaly detection also need a low rate of false negatives.

There are a lot of anomaly detection algorithms, each with their strengths, weaknesses and suitability to different domains. Since the unit of measurement and definition of anomaly is domain specific, the selection of the anomaly detection algorithm and pre-processing of observations also often is domain specific and may require knowledge of the domain.

3.3.1 Exponentially Weighted Moving Average

A basic method to detect anomalies in time series of interval or ratio values is exponentially weighted moving average (EWMA).

The EWMA series is given by

$$z_t = \alpha x_t + (1 - \alpha)z_{t-1},$$

where x_t is the observed process value at time t , and α is the decay factor, typically selected between 0.05 and 0.25. Upper and lower thresholds are set as

$$UCL = \mu_s + T\sigma_s,$$

$$LCL = \mu_s - T\sigma_s,$$

where μ_s is the mean of the series, σ_s the standard deviation, and T is the number of tolerated standard deviations before considering an EWMA value, z_t anomalous. In general [12] the parameter T is determined by the decay value α as

$$T = k\sqrt{\frac{\alpha}{2 - \alpha}},$$

where k typically chosen as $k = 3$ from the “three sigma” limits in Shewhart control charts [12]. The process is considered to produce anomalous values at times where $z_t < LCL$ or $UCL < z_t$, that is the EWMA value passes outside the upper or lower thresholds.

A different way to define how quickly the EWMA value should change with new values is by specifying a span value. Span is related to the decay value α as: $\alpha = \frac{2}{span+1}$ and is meant to describes the number of samples which contributes a significant amount of their original value. To get meaningful EWMA charts, $span$ should be selected ≥ 2 . To see this, note that as $span = 1$ means $z_t = x_t$ (no historic samples), $span = 0$ gives $z_t = 2x_t - z_{t-1}$, $span = -1$ is undefined and $span \leq -1$ gives sign inverted z_t . The typically selected α values 0.05 and 0.25 correspond to $span = 7$ and $span = 39$, respectively. As $span$ approaches infinity α approaches 0, that is the EWMA takes infinitesimal regard to current values and is therefore biased towards historic values, which will decay slowly.

EWMA Chart

An example EWMA chart is shown in Figure 3.1. The line annotated “defect client” denotes where a defect was introduced in the application. Data points 0 to 68, before the line, are measurements from the normal version of the app; data points 69 to 76, after the line, are measurements from the application with a defect. The mean μ_s and variance σ_s are calculated from the data points from

the normal version. Note how sample 74 and 76 are detected as anomalies as the EWMA line (dashed) crosses the upper threshold (UCL).

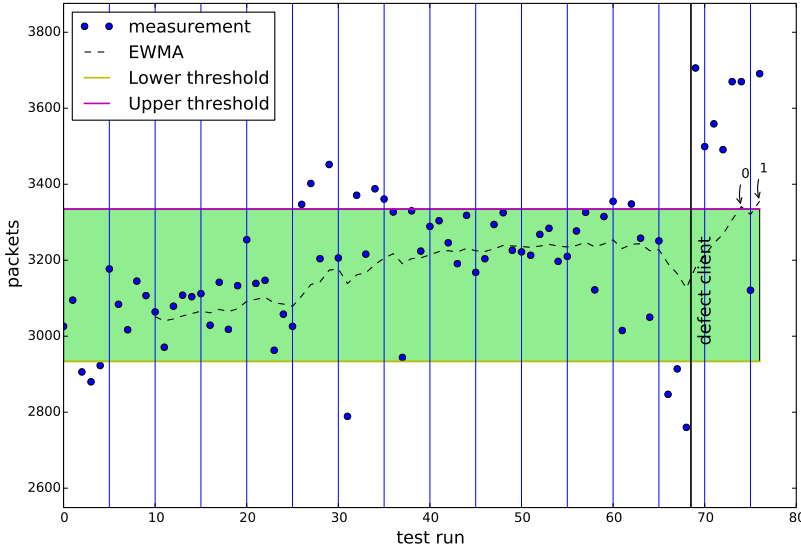


Figure 3.1: Example EWMA chart. $span = 20$ and threshold according to equations in Section 3.3.1. Data set is the number of packets for test case T3, with normal app and app with defect A3, both further described in Section 4.4.

3.4 k-Means Clustering

Cluster analysis is a method of grouping similar data points together in order to be able to conclude things from the resulting structure. Cluster analysis, or clustering, is used as or as a part of many machine learning algorithms. Running a cluster analysis of the data and labeling the clusters can construct an unsupervised classifier.

k-means is “by far the most popular clustering tool used in scientific and industrial applications.” [2]. The k-means algorithms require the number of clusters, k , as input and finds k clusters such that the sum of the squared distance from each point to its closest cluster center is minimized. That is find k centers so as to minimize the potential function,

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2,$$

where μ_i is the center point for cluster i .

Solving this problem optimally is NP-hard, even for $k = 2$ [7, 19]. There are more computationally efficient heuristic algorithms, which are often used in practice.

A widely used heuristic algorithm for k-means is Lloyd's algorithm [18]. Lloyd's algorithm finds a local minimum for the cost function by (1) selecting k center candidates arbitrarily, typically uniform at random from the data points [1]; (2) assign each data point to each nearest center; and (3) re-compute the centers as the center of mass for all data points assigned to it. As the Arthur and Vassilvitskii [1] explains, the initial center point candidates can be chosen in a smart way to improve the both the speed and accuracy.

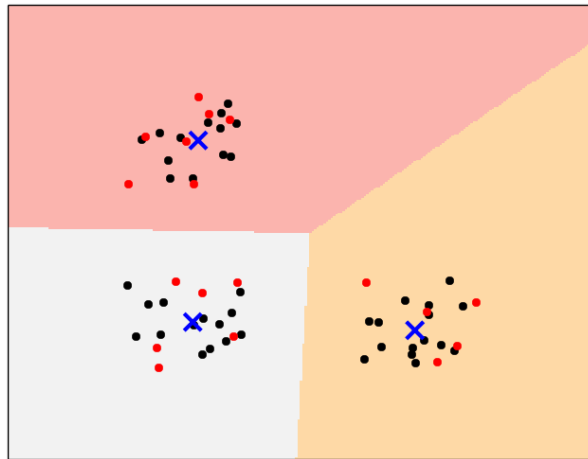


Figure 3.2: Example of k -means clustering of 20 observations each of three stochastic processes with Gaussian distribution and mean $(0,0)$, $(2,0)$ and $(0,2)$ respectively, $k=3$. The observations are split into learn and verify as 90%/10%. The learn set is used to train the model, that is decide where the cluster centers are. Observations from learn is black, verify red and the cluster center is a blue X.

3.4.1 Deciding Number of Clusters

k -means need the number of clusters as input. Finding the true number of clusters in a data set is a hard problem which many times is solved by manual inspection by a domain expert to determine what is a good clustering. Automated analysis must solve this in another way. One way to automatically select a reasonable number of clusters in a data set is to run the k -means clustering algorithm for a set of values for k and determine how good the clustering turned out for each one.

The silhouette score, S_k , is a measurement of how well data points lie within their clusters and are separated from other clusters [24], where k is the number of clusters (parameter to k-means). It is defined as

$$S_k = \frac{1}{k} \sum_{i=1}^k \frac{b_i - a_i}{\max(a_i, b_i)},$$

where a_i is the average dissimilarity of data point i with the other data points in the same cluster, b_i the lowest average dissimilarity of i to any other cluster where i is not a member, and k is the number of clusters (parameter to k-means). Silhouette scores are between -1 and 1 , where 1 means that the data point is clustered with similar data points and no similar data points are in another cluster.

The average silhouette score, S_k , gives a score of how good the clustering fits the total data set and can therefore be used to decide whether the guess for number of clusters, k , is close to the actual number of clusters. The k value giving the highest silhouette score S_k is denoted as k^* , and calculated as

$$k^* = \operatorname{argmax}_{k_{min} \leq k \leq k_{max}} S_k,$$

where k_{min} and k_{max} is the upper and lower limits of the range of tested k .

In Figure 3.3 we show an example of using silhouette scoring to decide the number of clusters in the data set from Figure 3.2. With $k_{min} = 2$ and $k_{max} = 14$, the silhouette score analysis gives

$$k^* = \operatorname{argmax}_{2 \leq k \leq 14} S_k = 3,$$

which is what we intuitively expected from the data set.

3.4.2 Feature Extraction

Good selection and transformation of features is vital in constructing a serviceable model using data mining or machine learning.

Features can be classified in different measurement classes depending on how measured values of a feature can be compared. Stanley Smith Stevens introduces the scale types nominal, ordinal, interval and ratio [27]. Nominal values can be evaluated if they are the same or if they differ; one example is male vs. female. Nominal is also known as categorical, which is the term used in this thesis. Ordinal values can in addition be ordered; one example is healthy vs. sick. Interval values can in addition be added and subtracted; one example is dates. Ratio values can in addition be compared in ratios, such as a is twice as much as b ; one example is age.

Often collected data need processing to be usable in algorithms used for data mining and machine learning. The standard k-means clustering algorithm for example computes the Euclidean distance between data point vectors to determine their likeness and therefore need features defined as meaningful numbers. Cat-

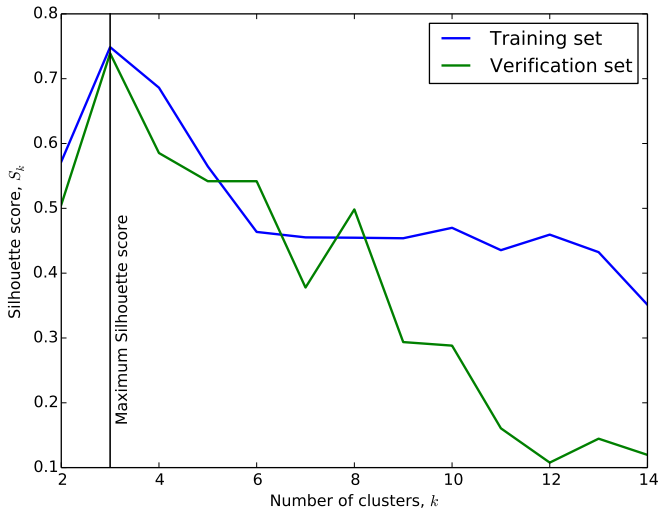


Figure 3.3: Silhouette scores for learn and verify set in Figure 3.2 for cluster sizes 2 to 14. In this example, the maximum Silhouette score is achieved for $k^* = 3$.

egorical features such as colors can be mapped to a binary feature vector, where each color in the domain is mapped to its dimension.

Ordinal features that are not numerical need to be encoded in order to be used to determine distance with classic distance metric such as the Euclidean distance metric. One example of ordinal features is “too small”, “too big” and “just right”, which may be ordered as “too small”, “just right”, “too big” and encoded as -1, 0 and +1, respectively. Binary features may be encoded with this method without being ordinal features, as it then essentially mimics the label binarization introduced above. This kind of encoding is called label encoding.

Normalization

Normalization is needed to avoid single features dominating others in distance measurement between data points by having larger values. Normalization can be done for feature j with values $x_{i,j}$ ($1 \leq i \leq N$), by calculating the mean μ_j and

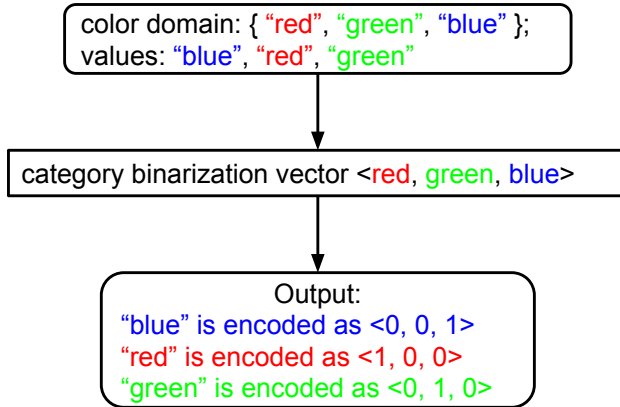


Figure 3.4: Transformation of red, green and blue from the color domain to a binary vector, where each color is its own dimension.

standard deviation σ_j of feature j as:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j},$$

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2},$$

where N is the number of observations and $x_{i,j}$ is the value of feature j in observation i . The normalized feature vector is then calculated as

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j},$$

for each instance i in feature j . This makes the features comparable to each other in terms of their deviation from the mean.

3.5 Novelty Detection

Novelty detection is the act of classifying observations as similar to previously seen observations and thereby is “normal”, or if they constitute deviations from the previously seen observations and thereby is “novel”.

One method of performing novelty detection is by training a clustering algorithm on observations considered normal, which will form clusters of observations with

Table 3.1: Confusion matrix of an anomaly/novelty detection system.

	Sample anomalous/ novel	Sample normal
Classified as anomalous/novel	True positive (t^+)	False positive (f^+)
Classified as normal	False negative (f^-)	True negative (t^-)

cluster centers and distances of observations to cluster centers. The maximum distance from an observation from the normal data set to its cluster center can be considered the outer boundary of normal values for each cluster. New observations are then considered normal if they fall inside the normal boundary, i.e. have an equal or shorter distance to the cluster center. Observations that fall outside the normal boundary are considered novel.

3.6 Evaluation Metrics

To know if our machine learning algorithms are performing acceptably and to compare against other algorithms, we define some performance metrics. Anomaly and novelty detection systems are a type of binary classification systems; determining if a sample is novel/anomalous or normal. In the context of anomaly and novelty detection a classification of a sample as a novelty or anomaly is denoted as positive, while a classification as normal is denoted as a negative. The performance is often based on the four rates of true/false positive/negative classification, visualized as a confusion matrix in Table 3.1.

Some common metrics to evaluate the performance of a classification system are precision, true positive rate (TPR) and false positive rate (FPR), defined as follows:

$$Precision = \frac{t^+}{t^+ + f^+},$$

$$TPR = \frac{t^+}{t^+ + f^-},$$

$$FPR = \frac{f^+}{f^+ + t^-},$$

where t^+ is the number of true positive, t^- is the number of true negative, f^+ is the number of false positive, and f^- is the number of false negative. Precision

is the rate of detections that are correct. True Positive Rate (TPR) is the rate of detection of anomalous/novel samples, also called recall. False Positive Rate (FPR) is the rate of miss-classification of normal samples as anomalous/novel.

Ideally, a system should have high precision, high true positive rate and a low false positive rate.

The true positive and false positive rates for a range of values of a system threshold setting is often visualized in a graphical plot called a Receiver Operating Characteristics (ROC) curve. ROC curves are generated by varying the threshold setting for the evaluated system to find the achievable pairs of TPR and FPR ; ideally this is done by running the system once to determine a score for each data point and use that to calculate all achievable pairs of TPR and FPR . Figure 3.5 shows an example of a ROC curve. The area under the ROC curve should ideally be equal to 1, which is achieved when the true positive rate is 1 for all values of the false positive rate, especially false positive rate = 0. The ROC curve can be an aid in comparing classification systems and choosing a threshold with acceptable rates of true and false positives.

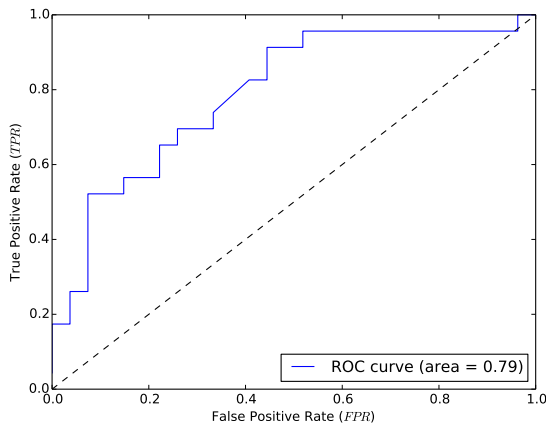


Figure 3.5: Receiver Operating Characteristic (ROC) curve example showing the relation between the true positive rate and the false positive rate for a binary classification system. The dashed line constitutes the ROC of a system which making random guesses. Above and to the left of the curve is better than random guess and below and to the right is worse.

3.7 Tools

There are a lot of tools and software suites for doing data mining and machine learning. Python based libraries have the advantage of being widely accepted for deployment in production, which is a necessity for an automated regression

testing system. For the machine learning part of this thesis the following are used:

- **Python** – dynamically typed programming language;
- **Pandas Data Analysis Library**¹ – storing features, data manipulation and calculating statistics;
- **Scikit-learn** for Python [23] – machine learning and calculating statistics;
- **matplotlib**² – for plotting.

3.8 Related Work

This section describes identified previous work related to this thesis.

3.8.1 Computer Networking Measurements

Mobile developing software suites offer ways to profile apps in the simulator/emulator or on the device (e.g. instruments for iOS³ and DDMS for Android⁴). One of the measure categories is network I/O and often includes bytes total, bytes per second, packets total and packets per second; all both for incoming and outgoing. This is a good way for a developer to get an instantaneous overview of how much the app is communicating, but often does not give details on with what and why, and does not provide an automatic way to compare and find changes between test case runs.

Jimenez et al. [13] investigate the consequences of integrating the Spotify Android client in Spotify's P2P network. A part of this study was to measure the correlation of network activity and energy consumption, which affects battery life, of the Spotify application for Android. This serves as a good highlight of one of the reasons to why monitoring an app's network activity changes over time is a good idea. The paper does not bring up any solutions on how to automate this testing, as the main focus is verifying the impact of using P2P on mobile.

As mentioned in Section 1.2.2 the Spotify client saves some aggregated statistics on network usage. While this may have been enough at the time, the growing complexity of the clients together with partner demands means Spotify need a new solution.

¹<http://pandas.pydata.org/>, May 2014

²<http://matplotlib.org/>

³Instruments User Guide, <https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/MeasuringIOActivityiniOSDevices/MeasuringIOActivityiniOSDevices.html>, February 2014

⁴Using DDMS, <https://developer.android.com/tools/debugging/ddms.html#network>, February 2014

3.8.2 Anomaly and Novelty Detection

There is a lot of work on automatically identifying anomalies and novelties in the traffic of computer networks in order to augment the static rule-based detection of Network Intrusion Detection Systems (NIDS) with detection of yet unknown types of attacks.

Lee et al. [17] describe “data-flow” environment as applications that involve real-time data collection, processing and analysis. They build a data-flow environment to automatically create rules for network intrusions detection by building a model of the normal traffic from network traffic collect over a few days, and a model of intrusions with network traffic from simulated or real and manually labeled attacks. The features used are the start time, duration, source host, source port number, destination host, destination port number (service), bytes sent from the source, bytes sent from the destination, and a flag denoting the state of the connection. Methods for efficient storage and evaluation of the created rules are also detailed. As the rules are constructed using explicit samples of network traffic corresponding to intrusions, it is likely this method will not successfully identify novel intrusion types; this is also noted by the authors in future work.

Yeung et al. [30] propose using a novelty detection, semi-supervised learning approach to identify network intrusions, which is trained only using normal data, and not data from observed/simulated intrusions. They create a model of the normal traffic as a density estimation for the probability density function, using Parzen-window estimators with Gaussian kernels. New samples are tested against the normal model by thresholding the log-likelihood that they are drawn from the same distribution as the samples that created the normal model. The threshold is also corresponding to the expected false detection rate. The proposed method is evaluated using the KDD Cup 1999 network data set, with the categorical features (called “symbolic” in the paper) encoded as binary features. It compares favorably to the winner of the KDD Cup 1999, considering the winner used supervised learning to explicitly learn to identify the different attacks present in the data set. A drawback of using the Parzen-window method is that the window width parameter σ need to be specified by the user or an expert.

Tarrassenko et al. [28] use a clustering analysis approach to design a novelty detection system for identifying unusual jet engine vibration patterns in an effort to highlight possible failures before they happen. Their data set consists of measured vibration spectra of 52 healthy engines to build a model of normal. The measured spectra are encoded as 18-dimensional feature vectors. To give each feature equal importance the authors use two transformations: (1) component-wise normalization, as described in Figure 3.4.2; and (2) whitening, removing correlation between features. Cluster analysis with the k-means method assigns each transformed feature vector to a cluster of similar vectors. To avoid having to select a global threshold that is assuming the same probability distribution for each cluster, the authors calculates the average distance from feature vectors to their respective centers for all clusters. The average distances are then used to normalize, or weight, the distance of a test sample to a cluster center, giving its

novelty score. The threshold is set to classify all training samples as normal. This method achieved good metrics, identifying all unusual vibration patterns of 33 engines and managed to identify on average 8.8 of 10 verification samples from the set of normal engines. The component-wise normalization performed far better than the whitening, which only identified on average 5.4 of the verification samples as normal, as the whitening transform lead to model overfitting on the training samples.

The k-means clustering with component-wise normalization approach described is interesting as it uses a well known technique to group data together with intuitive add-ons to perform novelty detection with a good results. The intuitiveness should make it easy to reason about the output of the process, which may be needed when alerting stakeholders.

Part II

Implementation and Evaluation

4

Measurement Methodology

"I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be."

LORD WILLIAM KELVIN

This chapter describes the tools and techniques used to capture and process the network traffic. The resulting data sets used in the evaluation are also described in this chapter.

4.1 Measurements

There are various tools and processes to capture information about network traffic. For internet service provider's (ISP's) backbone networks, companies' edge nodes or even busy servers storing all network traffic would be prohibitively expensive compared to the business advantages (at least if the business is not spying on the world by any means necessary). These kinds of networks often only stores aggregated statistics on usage such as number of bytes and packets for periods of e.g. one minute or since last restart. There are also commercial solutions to sample the network traffic and extract particularly useful features for capacity planning and maintenance, such as Cisco's NetFlow¹.

¹Cisco IOS NetFlow, <http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html>, March 2014

Since our purpose is to monitor the network activity of a single application under conditions that are expected to work on smartphones connected via mobile networks, the total network traffic will be comparatively small. Furthermore, capturing all of the raw network traffic allows for more involved offline analysis at a later date, which may come in handy.

Capturing network traffic on a local interface is often done with some tool using a port of libpcap², usually tcpdump, which is capable of capturing all traffic matching a set of rules on a specified interface and stores its dumps in the PCAP³ or PCAPNG⁴ format.

4.1.1 General Techniques

To capture traffic you first need to have access to it. There are generally three ways to get access to all network traffic destined for or originating from another host (A) on a network:

- Gatekeeper – Tap the network where the traffic must pass to reach (A), e.g. the default gateway, the nodes ISP. Where the tap is placed will determine what portion of the traffic you capture – tapping at ISP level will not grant you access to the host’s local network traffic.
- Snitch – Use something in the know to send you a copy of the traffic. This may be realized by using a so called mirror or switch port analyzer (SPAN) port on a switch - a port which will send a copy of all traffic on one switch port out on the mirror port.
- Sniff – Some network topologies send all hosts’ traffic to all other directly connected hosts. Unencrypted wireless networks, wireless networks encrypted with a pre-shared key (PSK)-scheme⁵, and old-style wired networks using hubs are some examples of such network topologies. This arrangement of course makes it trivial for all connected hosts to capture other hosts’ data.

The Gatekeeper method allows for manipulation of packets before they are transferred, facilitating man in the middle attacks to extract more information from the protocols (see Section 4.1.3). The other two methods generally only allow observation of the traffic streams.

4.1.2 Mobile Apps

There are many ways of capturing the network traffic of apps running on a mobile platform. Often the software development kit (SDK) includes tools to perform network activity measurement for the platform, either the simulator/emulator

²tcpdump & libpcap official web page, <http://www.tcpdump.org>, March 2014

³PCAP man page, <http://www.tcpdump.org/manpages/pcap.3pcap.html>, March 2014

⁴Wireshark: PcapNg documentation, <http://wiki.wireshark.org/Development/PcapNg>, March 2014

⁵Wireshark guide: How to Decrypt 802.11 <http://wiki.wireshark.org/HowToDecrypt802.11>, March 2014

running on the developer's computer or on the physical mobile device, or both. The output from network activity measurement tools varies, some only output aggregated statistics, and some give access to the actual network traffic for more detailed analysis.

Other ways are related to the general techniques described in Section 4.1.1, like setting up an ad-hoc WiFi network on a computer running tcpdump and connecting the device to the WiFi.

Not all techniques are usable to capture network traffic on both WiFi and cellular connections, which can be necessary to get a complete view of the app's behavior in the different environments it is commonly used.

IOS

Apple provides the testing tool `rvictl`⁶ to configure a mirror interface of a connected iOS device. A remote virtual interface (rvi) is configured by connecting a device with USB to a host computer and providing the rvi control tool `rvictl` with the target device id:

Listing 4.1 : Starting a Remote Virtual Interface on a Connected iOS Device (from `rvictl` documentation).

```
$ # First get the current list of interfaces.
$ ifconfig -l
lo0 gif0 stf0 en0 en1 p2p0 fw0 ppp0 utun0
$ # Then run the tool with the UDID of the device.
$ rvictl -s 74bd53c647548234ddcef0ee3abee616005051ed

Starting device 74bd53c647548234ddcef0ee3abee616005051ed [SUCCEEDED]

$ # Get the list of interfaces again, and you can see the new virtual
$ # network interface, rvi0, added by the previous command.
$ ifconfig -l
lo0 gif0 stf0 en0 en1 p2p0 fw0 ppp0 utun0 rvi0
```

This method is of the `snitch` type as the iOS device mirrors all packets to the virtual interface.

The virtual interface represents the entire network stack of the iOS device, and there is no way to distinguish between traffic over the cellular link and traffic over WiFi. This also means that the rvi may be used to capture 3G network data. Measuring over 3G could otherwise be hard as devices are often directly connected to the telephone companies network so the vanilla Gatekeeper technique will not work. The Sniff technique relies on weak encryption, and `snitch` requires privileges to run a network sniffer on the device, which could be achieved through jailbreaking⁷. Jailbreaking is not always feasible as it is not possible at all times

⁶https://developer.apple.com/library/mac/qa/qa1176/_index.html#//apple_ref/doc/uid/DTS10001707-CH1-SECIOSPACKETTRACING

⁷The iPhone wiki, general description of jailbreak and availability matrices, <http://theiphonewiki.com/wiki/Jailbreak>, May 2014

for all device and software versions, may be illegal in some regions, violate business agreements, and may affect the System Under Test (SUT) as measurement is not done in the same environment as the app will run in later. Another way to capture 3G network data would be routing all traffic via a catchall proxy such as a VPN. Drawbacks with routing through a VPN to capture traffic is that it requires managing the VPN system and that it could affect the SUT by changing the latency with the extra hop, worse/better network routes to destination, increasing/decreasing data size by adding VPN data or compressing payloads. Existing receiver side methods to detect and route mobile traffic may also be disrupted. Using a VPN to capture network traffic may also raise a suspicion that the operating system selects what traffic to route through the VPN and what to route directly through the Internet.

One drawback of the `rvictl` approach is that it requires the measurement computer to run an OS X environment, but since that is required to build iOS apps that requirement is often automatically fulfilled. The virtual interface is using a custom data link type and Apple's modified `libpcap` is required to capture traffic, making tools such as vanilla `tcpdump` and `Bro`, built with vanilla `libpcap` fail. Traffic dump files written with the `tcpdump` library in OS X is compatible with vanilla `libpcap`, making analysis possible.

We have observed that intermittently the dump files written by Apple's `tcpdump` are corrupt and not possible to read. This occurred in 20 of the test case runs for a set of 61 test case runs done 2014-05-20. The corrupt files were tested with vanilla `tcpdump`, Apple's `tcpdump`, `Wireshark`, and `Bro`, all rendering the same or similar error messages and no further information about the captured traffic, hindering investigation. We perceived the error messages to be related to corrupt or missing `libpcap` metadata for the network interfaces and suspect a bug in one or more of `rvi`, `libpcap` or `tcpdump`, possibly Apple's modified versions. No pattern in time of measurement or `tcpdump` output file size was observed between the corrupt and normal files. As we missed tools to do further analysis, no pattern was observed for the corruptions, and the error message hinted at a problem in writing interface metadata to the files, the corrupt files were not thought to have any specially interesting network traffic and were excluded from the data set.

4.1.3 Tapping into Encrypted Data Streams

Many modern applications use encryption protocols such as SSL/TLS, SSH, or something else to tunnel much or all of the applications traffic. In Spotify's case a great part of the app's traffic is sent over the multiplexed and encrypted TCP stream to their AP, as described in Section 1.2.2. This hampers attempts to extract information from the payload, which could have been useful for analysis, such as what resource is accessed. If all of the app's traffic is routed through a single encrypted tunnel we will only be able to analyze traffic flow patterns such as bytes/second, packets/second, etc. This chapter describes some techniques that can be used to get access to richer information in such cases.

Man-in-the-Middle Techniques

A popular technique to see behind the veil of encryption for one of the most used encrypted protocols, HTTPS, is to set up a program that injects itself into the flow by intercepting traffic from the two communicating parties, decrypt it, and re-encrypt and retransmit it to the other, acting like the original source, i.e. a man-in-the-middle attack (MitM). HTTPS+X.509's solution to this problem is to verify the sender's identity by cryptographic measures based on a certificate issued by a trusted third party, the certificate authority (CA). This system can be abused by having the MitM software act as a CA and issue itself certificates for every party it which to impersonate. Since we in this case have access to the operating system on the device running the app under test, we are able to setup our MitM CA as trusted by the device by installing the MitM CA's certificate on the device.

There are many commercial off-the-shelf (COTS) products with CA to do a MitM attack of HTTPS, e.g. mitmproxy⁸, Fiddler⁹ and Charles proxy¹⁰. There are however ways for the app developers to hinder these kind of MitM attack called certificate pinning, where the app verifies that the server's certificate used to establish the connection is the expected (hard coded) one and not just one signed by a CA trusted by the operating system.

However, not all encryption protocols work the same way HTTPS+X.509 does. Examination of Spotify's AP protocol show that using these COTS designed for HTTPS will not be fruitful, as it seem to implement some other standard. There may very well be a way to MitM attack this protocol as well, but since we have access to instrument the app to send meta data about the traffic via a side-channel no more time was put into this effort. There is also the risk of affecting the SUT's behavior when using these kinds of semi-active measurements instead of just passively listening and recording the data.

Instrumenting the App and Test-automation Tool

The test automation tool and the app itself may be instrumented to provide information on interactions, state and network requests. One example for iOS is implementing NSURLProtocol¹¹ and subscribing to relevant URL requests, logging it, and then pass the request on to let the actual protocol handler take care of it. The Graphwalker test-automation driver system can be configured to output time-stamped information about visited vertices and traversed edges, describing the intended state and performed user interactions.

⁸mitmproxy, official website, <http://mitmproxy.org/>, March 2014

⁹Fiddler official website, <http://www.telerik.com/fiddler>, March 2014

¹⁰Charles proxy official website, <http://www.charlesproxy.com/>, March 2014

¹¹Apple NSURLProtocol Class Reference, https://developer.apple.com/library/mac/documentation/cocoa/reference/foundation/classes/NSURLProtocol_Class/Reference/Reference.html, May 2014

4.2 Processing Captured Data

This section describes how the captured raw traffic is processed to extract statistics and features for the anomaly and novelty detection algorithms described in Chapter 5.

4.2.1 Extracting Information Using Bro

Bro is a network analysis framework, introduced in Section 2.1.5. It can be scripted to output statistics and information on captured traffic. In this thesis Bro is used to convert the binary PCAP network traffic capture files to plain text log files with information about each packet, DNS request and HTTP request.

Several output formats are used to enable analysis of various features: aggregate statistics for each stream, statistics for each network packet, aggregate statistics for plain text HTTP and DNS requests. The data format for the used output is further described in Appendix A.

4.2.2 Transforming and Extending the Data

To import the test case run artifacts in the change detection tools, the logs from Bro, the test automation tool and the app are read to extract information we wish to use.

All timestamps are transformed to be relative to the time of the first observed network packet ($t = 0$). The measurement computer's clock is the time source for all logs, so no synchronization is needed.

4.2.3 DNS Information

Bro's standard log output for connections does not include information about the DNS name of the network end-points. As described in Section 2.3, this information could be relevant in determining if two streams are to the same service end-point, even though the network end-point (IP and port) is different.

We hypothesize that a very useful data point for determining service end-point likeness is the DNS named used to establish a stream, which hold true if a canonical DNS name is used to establish all connections to a service end-point. The original DNS name is the easily retrieved with a recursive lookup from the stream end-point's IP address in a map of observed DNS queries and responses, which is one of the standard outputs of Bro.

In some instances a DNS query matching the stream network end-point address cannot be found. This may be because of at least two distinct reasons: (1) the stream is not established using a DNS name, meaning none of the test case runs will have a DNS name for the IP address; and (2) the DNS response is cached, so the app/operating system does not have to resolve it. For (1) an alternative DNS name can be derived from the IP address; the PTR record for the IP address may tie the stream to the organization/service as discussed in Section 2.1.1. For (2) using the DNS name for some test case runs and PTR or some other information

derived from the IP address will be problematic since they may very well be different (see “www.lon2-webproxy-a2.lon.spotify.com” vs. “www.spotify.com” example in Section 2.1.1). As (1) and (2) is indistinguishable in this analysis the same method must be used for both. Because of this the usefulness of the DNS name as a data point for service end-point similarity of streams will be different for different apps and operating systems, and must be evaluated for each case.

4.2.4 Other Network End-Point Information

The log processing system also add some other information about the network end-point of the responding party: numeric IP address, whether the IP address belongs to a subnet reserved for private (non-Internet routable) or multicast networks and the PTR record. Details described in Table A.2, which contain a mix of features added by Bro and the log processing system.

Estimate Network Activity

Since keeping the network hardware in a non-idle state by uses a lot of energy [13] it is preferable to use the network in bursts and let the network hardware idle in between. We do not explicitly measure the energy consumption, but using the captured network traffic characteristics and a simple model to emulate the network hardware’s energy state we can calculate a number that can be used to compare how much the app let the network hardware idle.

We propose the following algorithm to calculate the time a test run’s network access have kept the network hardware active, based on the simplified model that there are two energy states for the network hardware, active and idle and that the network hardware is kept in the active state “tail” seconds after a packet transmission is started:

Listing 4.2: Algorithm to calculate network hardware active state with simple model of the network hardware.

```
def calculate_network_active_time(packet_timestamps, tail=0.2):
    """
    Calculate the time the network hardware is in a non-idle state by
    using a model where the hardware transition to the idle state after
    tail seconds.
    Input:
    - packet_timestamps: sorted list of timestamps in seconds of sent
      packets.
    - tail: number of seconds the hardware is kept non-idle after a
      packet is sent.
    """
    active_time = 0
    start = packet_timestamps[0]
    end = start + tail
    for ti, ts in enumerate(packet_timestamps):
        if ts > end:
            active_time += end - start
            start = ts
        end = ts + tail
    if ti == len(timestamps) - 1:
```

```
    awake += end - start
return active_time
```

4.3 Data Set Collection

This section describes how the data sets were collected.

4.3.1 Environment

All data sets were collected from an iPhone 5c running iOS 7.0.4, connected to a computer running the `rvictl` tool (Section 4.1.2) to capture the phone's network traffic and Spotify's test automation tool to perform the test cases defined in Section 4.4.3 and Section 4.5.1 in a predictable manner. The phone was configured to minimize iOS network traffic by disabling iCloud, iMessage and location services.

For data set I the phone was connected to the Internet over the Spotify's office WiFi connection. For data set II the phone was connected to a WiFi provided with Internet access through Comhem. Data set I and II are used in separate evaluations and no direct comparison is done between them.

Measurements were exclusively done over WiFi to avoid the extra cost associated with cellular data connections during the design and evaluation period. The results are expected to translate well to measurements over cellular data connections as none of the methods are tailored to the traits of a WiFi connection.

4.3.2 User Interaction – Test Cases

To make comparisons of the network activity of various versions of the app, some way to minimize the variability between runs of a test case. To achieve this a graphical user interface (GUI) automation tool is used, as a computer is much better at performing tasks in a deterministic manner than a person. This also helps in generating the large amount of samples that may be necessary to train the algorithms, with minimal manual intervention. The network activity measuring system is integrated with Spotify's system for automated testing to make it easy and accessible to write new tests on a familiar platform and reuse suitable existing tests.

Each test case starts with clearing the app's cache, setting up the network traffic recording as described in Section 4.1.2 and ends by stopping the network recording and collecting recorded network and log data.

4.3.3 Network Traffic

The collected network traffic was saved to `pcapng` files using `tcpdump` to save all network traffic on the `rvictl` interface to a file by running the command in Listing 4.3. Traffic from/to transport protocol port 8023 was filtered out as it is used by the test automation tool to remote control the application and receive

log and state information. No non-test automation remote control traffic could be observed on port 8023 when investigating, so no negative impact on the quality of the data set is expected.

Listing 4.3: Command to start tcpdump to capture the network traffic

```
/usr/sbin/tcpdump -i rvi0 -s 0 -w tcpdump.pcapng port not 8023
```

Features are extracted from the network dumps with Bro [22] (partially described in Section 2.1.5) to identify protocols and connections and extract payload data from the application level protocols DNS and HTTP. Bro by default produces an abundance of log files with miscellaneous information. For our analysis and visualization we required some custom features, so we defined a custom Bro log format to capture information for each network packet by subscribing to Bro's new_packet event¹². This logging format enables time-based analysis. Description of the features can be found in Table A.1 in Appendix A.

In a later preprocessing step the following features are added, derived from the features above. Description of the derived features can be found in Table A.2 in Appendix A.

Although the phone was configured to minimize non-app network traffic some intermittent traffic to Apple's servers was detected. Since it is possible that changes to the app will lead to a change in network traffic to or from Apple's servers, it was decided to avoid filtering attempts of this traffic at this stage.

4.3.4 App and Test Automation Instrumentation Data Sources

The test automation tool and app produces logs, which can include relevant data to correlate features from the captured network traffic with. To enable time synchronization and to give the context in which anomalies/novelties were detected the state and actions of the GraphWalker model based testing system that drives the test automation is extracted from the logs. The app have been instrumented to output information about its view state, observed HTTP/HTTPS requests and when adding network usage statistics to the app-internal RequestAccounting system.

The information obtained using instrumentation is:

1. The Graphwalker (test-automation driver) vertices and edges, corresponding to state and action of the test automation system.
2. "Breadcrumbs" from the app, detailing what view of the app is entered/exited; representing the visual state of the app.
3. Calls to the internal network activity statistics module, detailing the endpoint, bytes uploaded, bytes downloaded and the network type. This is the

¹²http://bro.icir.org/sphinx/scripts/base/bif/event.bif.html#id-new_packet, May 2014

main method to extract more detailed information about the service end-point traffic for the long-lived, multiplexed and encrypted connection to Spotify's access point.

4. HTTP/HTTPS requests by implementing NSURLProtocol and subscribing to requests for the relevant protocols to log it and then pass it to the real protocol handler.

The app log data will not be available in situations where it is not possible to instrument the app, like when missing access to the source code. Test run state and actions should be available even when testing a prebuilt binary as it is assumed the examiner is in control of the test-driving tool, whether it is manual or automated. Because of this and the utility to others than developers of the system we will avoid relying on the information from the app log to construct the basic anomaly/novelty detection system. It will be used to augment the analysis or construct a complementary, more detailed, detection system.

4.4 Data Set I - Artificial Defects

This section describes the dataset with manually introduced defects used to evaluate the algorithms' ability to find the effects of some types of defects.

Data set I was collected 2014-04-07 - 2014-04-14.

4.4.1 Introduced Defects

To verify the novelty detection and visualization algorithms and system some artificial defects were introduced into the app that was used to generate the normal base traffic. These defects are created as examples of some of the anomaly/novel types we aim to detect, affecting the network activity in different ways to facilitate algorithm tweaking and detection evaluation.

The introduced defects are:

- A1 On login (starting the app, not resuming from background) downloading a 935 kB large jpeg image¹³ over HTTP from a service end-point not previously observed in the app's traffic. This anomaly deviate from the normal traffic on several dimensions and ought to be easily detectable by suitable machine learning algorithms, and may as such be used as a sanity check.
- A2 Partially breaking the app's caching mechanism by removing cache folders on when the test automation tool is restarting the app.
- A3 Sending ping messages to the Spotify AP 100 times more often. Ping messages are small with only 4-byte payload.
- A4 On login (starting the app, not resuming from background) downloading a 25 kB large cover art image over HTTP from one of Spotify's CDNs for

¹³Spotify press logo for print, <http://spotifypresscom.files.wordpress.com/2013/01/spotify-logo-primary-vertical-light-background-cmyk.jpg>, March 2014

media metadata¹⁴. Downloads from the media metadata CDN is usually done over HTTPS. This cover art is normally downloaded as part of the test case in Listing 4.6. As this file is small relative to the total size of the network traffic of a test case run and from a source that is used in normal traffic it should be a challenge to detect.

The introduced defects are selected to create network traffic changes we would like to be able to find, and to create varyingly difficult changes to detect to benchmark the detection methods.

4.4.2 Normal Behavior

The algorithms are taught what traffic patterns are considered normal from a set of test case run artifacts for each test case running on the Spotify iOS client 0.9.4 to. Note that some traffic activity of this app version may not be actually desirable, and may therefore contain traffic that ought to be classified as anomalies. This problem is not considered in this thesis; we solely focus on novelty detection and therefore need some traffic patterns to use as a baseline for what is normal.

4.4.3 Test Cases

The test cases T1, T2 and T3 (detailed in Listing 4.4, Listing 4.5, and Listing 4.6) are used to generate the data for the normal and defect versions of Spotify iOS 0.9.4.25. These test cases are created to trigger one or more of the introduced defects and to produce varying amount of network activity with different characteristics to enable detection performance analysis for the algorithms for different situations.

Listing 4.4: Login and Play Song (T1)

0. Start with cleared cache.
1. Login.
2. Search for "Infected Mushroom Where Do I Belong".
3. Touch the first song to start it.
4. Verify that the song is playing.
5. Pause the song.

Listing 4.5: Login and Play Song, Exit The App and Redo (T2)

0. Start with cleared cache.
1. Login.
2. Search for "Infected Mushroom Where Do I Belong".
3. Touch the first song to start it.
4. Verify that the song is playing.
5. Pause the song.
6. Exit the app (triggering removal of metadata if defect A2 is active).
7. Search for "Infected Mushroom Where Do I Belong".
8. Touch the first song to start it.
9. Verify that the song is playing.

¹⁴<http://d3rt19901pmkn.cloudfront.net/640/9c0c3427b559f5cae474f79119add480544e58d5>, April 2014 over HTTP

10. Pause the song.

Listing 4.6: Login and Create Playlist From Album, Exit The App and Redo (T3)

```

0. Start with cleared cache.
1. Login.
2. Search for "Purity Ring Shrines".
3. Touch the first album to go to the album view.
4. Add the album as a new playlist.
5. Go to the new playlist and verify its name.
6. Remove the new playlist.
7. Exit the app (triggering removal of metadata if defect A2 is active).
8. Search for "Purity Ring Shrines".
9. Touch the first album to go to the album view.
10. Add the album as a new playlist.
11. Go to the new playlist and verify its name.
12. Remove the new playlist.

```

4.4.4 Summary

Defects (Section 4.4.1) were introduced by modifying the source code of the Spotify iOS 0.9.4.25 app and building one binary per defect, which was installed on the phone before collecting measurements for these app types. The measurements were performed on all combinations of app types {normal, A1, A2, A3, A4} and test cases {T1, T2, T3}. The numbers of test case runs for each app type/test case combination can be found in Table 4.1.

The normal version has many runs as it need to capture the possibly different network activity patterns of the app. The numbers of test case runs of the defect versions are low to enable manual inspection of each as necessary. This also correspond to the expected use case, where many historic measurements of versions deemed normal are available while a small number of samples a new app is preferred to minimize time to detection and maximize throughput. Differences in the numbers of test case runs between test cases are due to the corrupt network captures in Section 4.1.2.

Table 4.1: Number of collected test case runs for each test case and app version for data set I.

	T1	T2	T3
normal	87	72	69
A1	5	4	3
A2	22	5	8
A3	9	10	8
A4	3	10	7

4.5 Data Set II - Real World Scenario

In an effort to evaluate the network pattern change detection performance in a real world scenario data sets were collected from instrumented versions of the Spotify iOS client version 1.0.0 and 1.1.0.

Data set II was collected 2014-05-18.

The release notes for 1.1.0 can be found in Listing 4.7 and may include clues about what network traffic pattern changes can be expected to be found or lead to an explanation of why a change was observed.

Listing 4.7: Spotify iOS 1.1.0 Release Notes

First, an apology from us.

We know there were some issues introduced in the last release.

We've been getting through a lot of coffee trying to fix them, and things should get a lot better with this release! Thanks for bearing with us.

- New: Introducing Your Music, a better way to save, organise and browse your favourite music.
- New: Play the same song over and over with Repeat One. Now available for Premium users and free users on iPad.
- Fixed: Smoother track skipping.
- Fixed: We've banished some crashes.
- Fixed: You can now delete Radio stations.

4.5.1 Test Cases

The test cases T4, T5 and T6 (detailed in Listing 4.8, Listing 4.9, and Listing 4.10) are used to generate the data set for the Spotify iOS version 1.0.0 and 1.1.0 in this thesis. These test cases were taken from the Spotify iOS client test automation set, but slightly modified to behave deterministically.

Listing 4.8: Artist page biography and related artists (T4)

0. Start with cleared cache.
1. Login.
2. Search for "David Guetta".
3. Touch the first artist to go to the artist page.
4. Go to the artist biography.
5. Go back to the artist's page.
6. Go to related artists.
7. Touch the first artist to go to its artist page.
8. Go back two time, ending up on the first artist's page.

Listing 4.9: Display the profile page (T5)

0. Start with cleared cache.
1. Login.
2. Go to the profile page.

Listing 4.10: Add an album to a playlist and play the first track (T6)

0. Start with cleared cache.
1. Login.
2. Search for "Purity Ring Shrines".
3. Touch the first album to go to the album view.
4. Add the album as a new playlist.
5. Go to the new playlist.
6. Play the first track in the playlist.
7. Pause the track.
8. Remove the new playlist.

4.5.2 Summary

The number of test case runs for each app type/test case combination can be found in Table 4.2. The differing number of test case runs are due reasons analogue to the ones given in Section 4.4.4.

Table 4.2: Number of collected test case runs for each test case and app version for data set II.

	T4	T5	T6
1.0.0	59	52	32
1.1.0	22	22	18

5

Detecting and Identifying Changes

We have implemented and evaluated two change detection systems: (1) an anomaly detection system using the EWMA chart method described in Section 3.3.1, and (2) a novelty detection system using the k-means clustering algorithm described in Section 3.4, Section 3.5 and Section 3.8.

5.1 Anomaly Detection Using EWMA Charts

A classic method for anomaly detection in statistical quality control is Exponentially Weighted Moving Average (EWMA) charts; see Section 3.3.1 for a theoretical introduction. An EWMA chart defines a target value and an allowed multiple of standard deviations from the target value as the upper and lower bound. EWMA uses the inertia from the “tail” of weighted prior values to smooth the signal, which means that occasional data points outside the bound will not set off the alarm, but a systematic drift will drag the weighted EWMA function outside and trigger the alarm.

The inertia is dictated by the decay factor α . This thesis instead defines *span*, which is related to α as $\alpha = \frac{2}{span+1}$ and approximately describes the number of historic data points that influence the EWMA in (see discussion in Section 3.3.1). Span is commonly selected as $7 \leq span \leq 39$, corresponding to $0.05 \leq \alpha \leq 0.25$, to strike a balance between new and historic values. Larger *span* values gives more smoothing and more resiliency to random noise, but slower reaction to change.

Setting *span* to different values within the interval $[7, 39]$ was not observed to impact the ROC curves or performance measures *precision*, *FPR* or *TPR* in an apparent way for the data set I. We believe this is due to that the threshold T

changing to allow a larger threshold when the EWMA is less smoothed because of lower *span*, and vice versa. As no superior *span* value was found, $span = 20$ is used for the rest of this thesis.

The upper and lower thresholds for anomaly detection, UCL and LCL , is selected as

$$\begin{aligned} UCL &= \mu_s + T\sigma_s, \\ LCL &= \mu_s - T\sigma_s, \end{aligned}$$

where μ_s is the mean and σ_s the standard deviation of the time series and T the a tolerance factor giving a trade off between false positives and false negatives.

5.1.1 Data Set Transformation

For each test case run four values are calculated as features for the EWMA chart analysis:

1. Number of network level bytes (*ip_len*) – the network footprint.
2. Number of packets – related to network footprint and network hardware activity.
3. Number of unique network end-points (feature names: *resp_h*, *resp_p*).
4. Number of unique (ASN, services) pairs, where services is the protocol detected by Bro's protocol identification.

These features are selected from the available features in Appendix A in an effort to make regression testing possible for the characteristics in Section 1.1.2.

5.1.2 Detecting Changes

EWMA analysis is good at identifying changes when the deviation is sufficiently large and persistent to drive the EWMA outside the UCL/LCL thresholds, as can be seen in Figure 5.1, where the data points from the defect client is correctly identified as anomalous after five data points of false negatives.

Each transformed data set from Section 5.1.1 may be described as a stochastic process (see Section 3.1). The data sets are separately subjected to EWMA chart analysis, as the basic method only treat one process at a time. As the observed outcomes from the stochastic variables depend on unknown states of the whole communication system, such as the time of the day, the current routing for music streams and current available download bandwidth from different sources, the stochastic process is non-stationary. EWMA chart analysis can fail to detect changes in non-stationary processes, since a single mean and variance is calculated, not taking into account the (possibly) different properties of the stochastic variables. In an EWMA chart of such a process the mean will be the mean of means and the variance will be larger than any of the process' single stochastic variable variance.

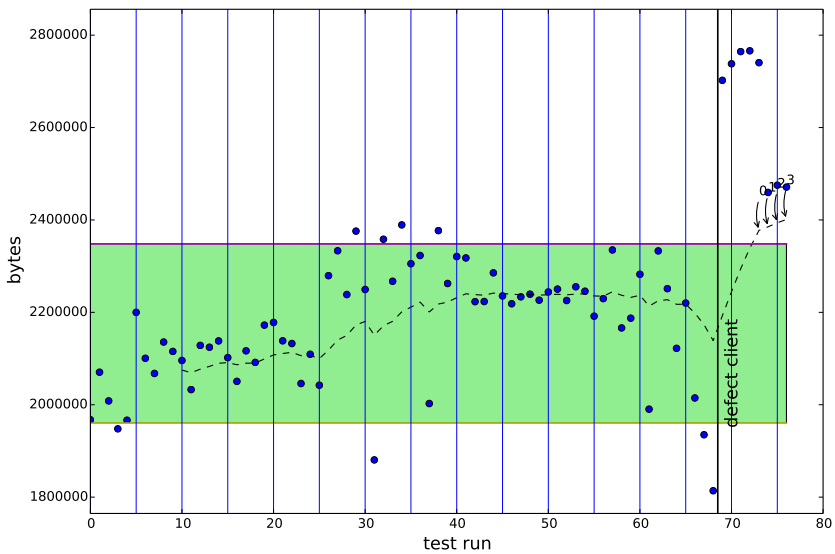


Figure 5.1: EWMA chart of the A2 (metadata) anomaly using the T3 (album playlist) test case and total number of IP bytes – the network footprint. $span = 20$ and threshold according to equations in Section 3.3.1.

If the anomaly is small enough to not deviate sufficiently from the mean with regards to the variance it will hide in the large variation of the normal traffic. In Figure 5.2, three levels of total network traffic can be observed (disregarding the single sample 5), eliminating any chance that the anomaly will be detected. The reason for the three distinct levels of the normal traffic in Figure 5.2 is twofold:

1. Test case run artifacts collected on 2014-04-08 have a 500 kB large meta-data request which have not been observed any other day. This corresponds to the 4 MB level of the first 24 artifacts and the two last artifacts on the anomalous client side.
2. When starting a non-cached song the first small chunk will be requested from two sources in an effort to minimize click-to-play latency. If the first request completes before the second, the second is cancelled and a new request is issued for the rest of the song. If instead the second request completes first enough data have been downloaded to satisfy the short play session and no more request is issued.

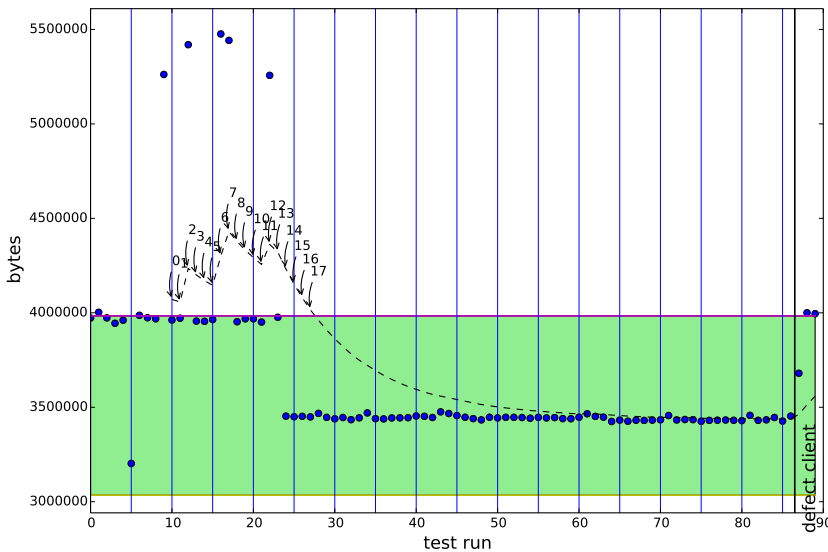


Figure 5.2: EWMA chart of the A4 defect using the T1 (play song) test case and total number of IP bytes. Note the number of false positives (marked with a number and arrow). $span = 20$ and $threshold$ according to equations in Section 3.3.1.

5.2 Novelty Detection Using k-Means Clustering

Novelty detection can be used to find when sampled network traffic exhibit patterns dissimilar to a learn model of normal traffic. Novelty detection is done by building a model of the system's normal behavior and comparing new data points, similarity to the model to determine if it should be classified novel or normal. With careful selection of features and methods this detection system can be used to identify in what way the network traffic patterns have changed compared to earlier data points, if any. Measurements transformed into vector space – a vector of selected features for each measurement data point – are called vectors.

5.2.1 Feature Vector

In an effort to detect and highlight in what traffic type a change have been found we wish to build the normal model as traffic characteristics for meaningful traffic types. Traffic types can be defined in a lot of ways; including more information gives higher resolution, which means higher precision in the change detection report, helping analyzers to pinpoint the problem. However, including too much or noisy information has the drawback of creating patterns that are not useful in determining if a new sample is novel or not.

As discussed in Section 1.1.1, Section 2.1.1 and Section 2.3, some of the collected network traffic features (see Appendix A) are problematic to use directly to establish if two different connections are to the same service end-point. Especially load balancing routing to different end-points, cloud computing creating real-time dynamic sized clusters with different addresses, and the non-contiguous allocation of IP-addresses make the end-point IP-address resp_h problematic for classifying the network traffic of dynamic applications. Our hypothesis is that the Autonomous System (AS) number (explained in Section 2.1.1) can be used to classify network traffic stream into meaningful *stream families* with all end-points belonging to the same cluster of machines in the same family.

To increase the resolution further, as the AS numbers can contain a lot of IP addresses and machines, a feature that describe the type of service the traffic belongs to would be suitable. As mentioned in Section 1.1.1 and Section 2.1.1 the transport protocol port numbers may not be the best way of determining the service type any longer. We therefore elect to use the feature provided by Bro's protocol identification system, which is available in the services field of the network capture data set.

Many streams are asymmetrical in regards of traffic amount sent/received, such as downloading files; the traffic direction is added to discern such patterns.

The network traffic features are grouped on the categorical features described above (asn, services, direction) into stream families; streams that have the same ASN, identified service, and direction. This feature vector is described in Table 5.1.

This transformation gives multiple dimensions in cluster space for each test run,

Table 5.1: Feature vector for *k*-means novelty detection.

label	transformation	description
asn	Label binarization (described in Section 3.4.2)	ASN identified by Bro.
services	Label binarization (described in Section 3.4.2)	Service protocol identified by Bro.
direction	Label encoding 0, 1	Direction of the stream – to or from the app.
ip_len	Component-wise normalization (described in Figure 3.4.2)	Number of IP-level bytes for the stream family.
count	Component-wise normalization (described in Figure 3.4.2)	Number of packets for the stream family.

typically 28-33 dimensions, which means the novelty detection is performed on stream families and not the test case runs. The relationship between a test case run and a stream family is identified from a cluster space vector by keeping track of indices in the data structures.

5.2.2 Clustering

Assigning features vectors to clusters based on similarity allows the algorithm to estimate the center and size of the regions where vectors from the normal data set are in vector space. The *k*-means algorithm identifies *k* clusters of similar feature vectors, finding an assignment of vectors to clusters that minimizes the dissimilarity between the points and their respective cluster center. We use the Euclidean distance as the dissimilarity metric.

One problem with using the *k*-means algorithm to automatically find clusters in previously unknown data is that the algorithm need the number of clusters as an in parameter; that is, we need to know how many clusters there are in the data. The silhouette score introduced in Section 3.4.1 is a measurement of how good the clustering is based on how similar vectors in the same cluster are and how dissimilar they are to vectors in other clusters. Running the *k*-means algorithm iteratively with a range of values for *k* and calculating the silhouette score for each let us determine which value k^* gives the most meaningful clustering with regards to cluster separation. This silhouette score ranking is used to automatically determine the value for *k* for the set of vectors from the normal app when building the model of normal.

To counteract overfitting the model to the normal data set, the data set is split into a 90 % learning set and a 10 % verify set selected at random.

5.2.3 Novelty Detection

When the vectors from the learn data set is divided into clusters, novelty detection can be done on new vectors by determining by how much they deviate from the clusters.

The deviation for vectors to their closest cluster center is based on the Euclidean distance. For vector x_j the distance to its closest cluster center, i is $d_{i,j}$:

$$d_{i,j} = \|x_j - \mu_i\|,$$

where μ_i is the vector for the cluster center i .

The verify set is used in combination with the learn set to determine the maximum distance a normal vector can have to its cluster center – the normal boundary for the cluster. The normal boundary is determined for each cluster and is selected such that all vectors in the learn and verify set falls within it. S_i is the set of all vectors from the learn and verify data set with closest cluster i . The normal boundary for cluster i , b_i , is calculated as:

$$b_i = \max_{x_j \in S_i} d_{i,j}.$$

Each *new* vector z_j , with closest cluster S_i is assigned a novelty score n_j based on its distance to the closest cluster center, weighted with the cluster's normal boundary to account for different variations for different clusters:

$$n_j = \frac{\|z_j - \mu_i\|}{b_i + \xi},$$

where ξ is a small tolerance term to account for the clusters with non-existent variance. No variance of the vectors in cluster i means $b_i = 0$. Setting $\xi = 0$ and $b_i = 0 \implies n_j = \infty$, when $z_j \neq \mu_i$. $\xi = 0.05$ is determined as a good balance to not exaggerate the novelty score for deviations from a zero-variance cluster with a too small value while keeping it small enough to not have the normal boundary cover novelties. The testing to determine ξ was performed on data set I.

The normal boundary is not used as a hard limit for classifying novelties, but for normalization of the distances to establish a novelty score. Vectors with novelty score ≥ 1 have a larger distance to their cluster center than vectors from the learn- and verify set. This means 1 is a feasible start threshold for classifying novelties, but higher or lower thresholds may be selected to decrease the false positive rate or increase the rate of true positives, respectively.

6

Evaluation

In this chapter the proposed methods from Chapter 5 are evaluated using the data sets from Chapter 4.

6.1 Anomaly Detection Using EWMA Charts

The performance of the EWMA anomaly detection system to detect network activity change is evaluated by measuring the number of correct and incorrect classification of the data points from the normal and defect app. Since all introduced defects increase all the metrics used for EWMA (see Section 5.1.1), the lower bound, LCL , is set twice the distance from the mean compared to UCL ; that is, $LCL = \mu_s - 2T\sigma_s$. This eliminates some of the instances where the data points from the defect happen to be slightly below LCL , which together with the repetition of each value in Section 6.1.2 made them classified as anomalies of the wrong reason. As the numbers of samples for the defect apps are low the wrong classifications have a big impact on the performance number, so adjusting the threshold make the evaluation more clear. Taking this to the extreme and setting $LCL = 0$ would be less acceptable in a real world scenario; great deviations toward lower values are still interesting and $LCL = \mu_s - 2T\sigma_s \gg 0$ for the considered features.

Ideally there should be possible to find a tolerance factor, T , where all data points from apps with a defect which is triggered by the current test case are marked as anomalous, while none of the data points from the normal app are. The measures precision, true positive rate TPR and false positive rate FPR introduced in Section 3.6 are used as performance measurements for the classification system.

6.1.1 First Method ROC Curves

The first attempt at measuring the classification rates of the EWMA anomaly detection system is to use the mechanism described above to classify data points from a series of data points, where the data points from the defect app are placed after the data points from the normal. The data points are tagged as from the normal or defect version of the app to determine if the classification is true/false positive/negative.

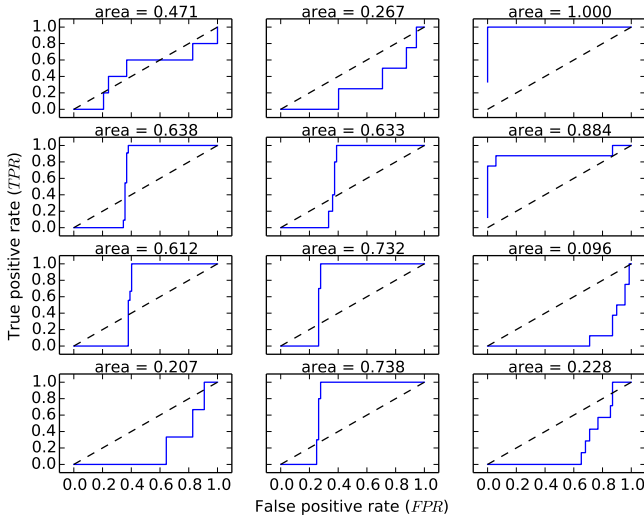


Figure 6.1: ROC curve of EWMA chart anomaly detection of the **feature network footprint (total number of bytes sent/received)**, first method. Sub-plots on x-axis left to right: $T1$, $T2$, $T3$; y-axis top to bottom: $A1$, $A2$, $A3$, $A4$.

As a core property of EWMA is the delayed reaction, the first data points from a defect app is unlikely to be classified as anomalous, which drives down the true positive rate artificially as some of these data points would eventually be detected. An alternative way of establishing whether a sample would be classified as anomalous is described below.

6.1.2 Better Conditions for Classifying Defects as Anomalous

Determining if defect app data point is an anomaly or not will be done by repeating the defect app data point $span$ times and count the sample as anomalous only if the last repeated sample of it is marked as an anomaly. The ROC curve for the EWMA anomaly detection system under these ideal conditions for anomaly detection can be observed in Figure 6.2.

Comparing the ROC curves in Figure 6.2, generated with the higher probability

of detecting anomalies in the data set from the defect app, to the ROC curves in Figure 6.1 it is clear that the new conditions detect more data points from the defect app and therefore have a higher TPR . The true TPR to FPR relation lies somewhere in between the two sets of ROC curves, but the latter will be used in this analysis, keeping in mind that it is under artificial conditions.

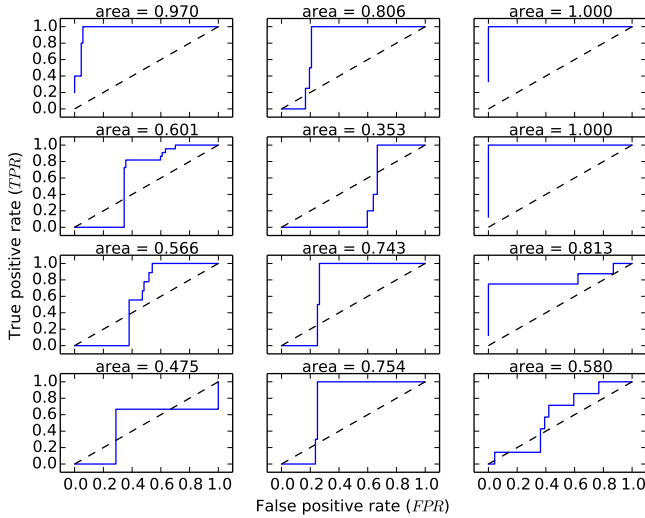


Figure 6.2: ROC curve of EWMA chart anomaly detection of the feature **network footprint (total number of bytes sent/received)**, ideal EWMA conditions for true positives. Subplots on x-axis left to right: T1, T2, T3; y-axis top to bottom: A1, A2, A3, A4.

6.1.3 Detected Anomalies

In this section we for each introduced defect discuss the EWMA charts analysis' detection ability and false positives for the typical setting for UCL as introduced in Section 3.3.1 and the LCL double the distance of UCL as discussed in Section 6.1. The referenced ROC curves are created with varying UCL and LCL to map the balance between achievable TPR and FPR with ideal detection. The ROC curves can be found in Figure 6.2 through Figure 6.5, where test cases T1-T3 are represented on the horizontal x-axis left to right, and the introduced defects A1-A4 on the vertical y-axis top to bottom.

A1 Defect

The A1 defect was expected to be easy to detect since it adds 935 kB network traffic and uses a new network and service end-point. However, for the T1 and T2 test cases with large variance in network footprint, the FPR is high and TPR and precision is low for all data representations except the number of distinct AS/service pairs where T2 achieves good and T1 ok scores. T3 achieves good

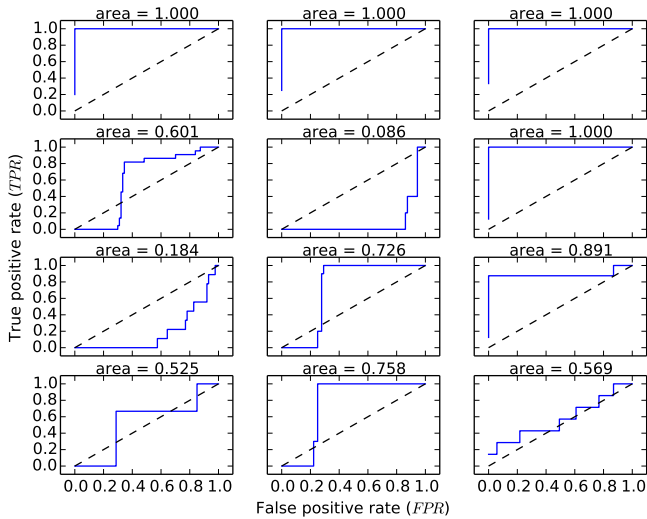


Figure 6.3: ROC curve of EWMA chart anomaly detection of the feature **number of packets**, ideal EWMA conditions for true positives. Subplots on x-axis left to right: T1, T2, T3; y-axis top to bottom: A1, A2, A3, A4.

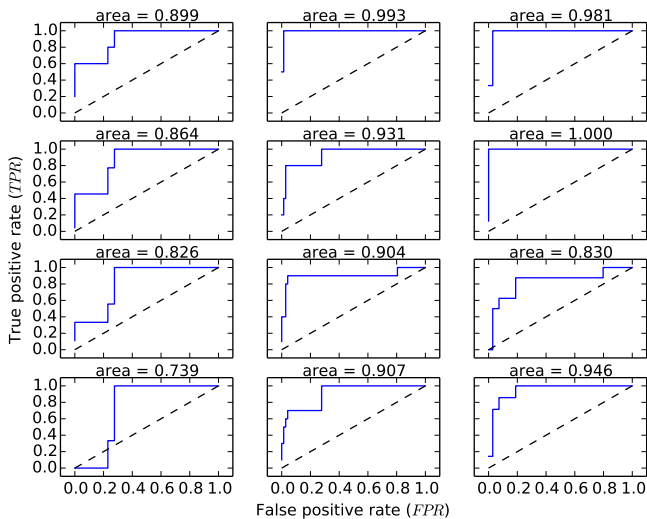


Figure 6.4: ROC curve of EWMA chart anomaly detection of the feature **number of distinct network end-points**, ideal EWMA conditions for true positives. Subplots on x-axis left to right: T1, T2, T3; y-axis top to bottom: A1, A2, A3, A4.

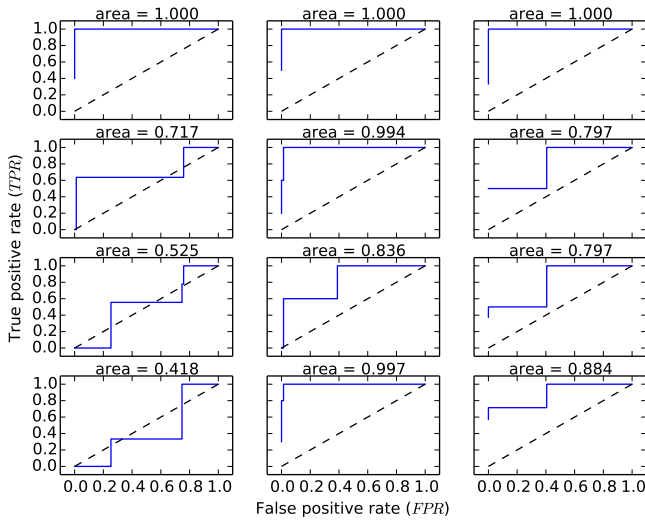


Figure 6.5: ROC curve of EWMA chart anomaly detection of the feature number of distinct (AS, service) pairs, ideal EWMA conditions for true positives. Subplots on x-axis left to right: T1, T2, T3; y-axis top to bottom: A1, A2, A3, A4.

scores for the network footprint because of the test case's smaller variance in network footprint.

Table 6.1: Detection performance numbers for EWMA on the A1 defect.

feature	test case	precision	TPR	FPR
IP bytes	T1	0.12	0.40	0.17
	T2	0.00	0.00	0.13
	T3	1.00	1.00	0.00
Packets	T1	0.16	0.60	0.18
	T2	0.00	0.00	0.11
	T3	1.00	1.00	0.00
Network end-points	T1	0.00	0.00	0.00
	T2	0.00	0.00	0.00
	T3	0.00	0.00	0.04
ASN-service pairs	T1	0.25	0.60	0.10
	T2	1.00	1.00	0.00
	T3	1.00	0.33	0.00

A2 Defect

The A2 defect is triggered when the app is restarting, which only occurs in T2 and T3; thus the T1 test case should be equally probable to detect data points from normal as the defect. This “random guess” can be observed in Figure 6.1 as the ROC curve tracking the dashed line and $area \approx 0.5$. The pattern can be observed for the ROC Figures 6.2 - 6.5, as well as the curves bias toward higher

TPR. The T3 test case is best in classifying the defect combined with the network footprint features.

Table 6.2: Detection performance numbers for EWMA on the A2 defect. (*) Defect not triggered by test case, no true positive detection possible.

feature	test case	precision	TPR	FPR
IP bytes	T1*	-	-	0.17
	T2	0.00	0.00	0.13
	T3	1.00	0.75	0.00
Packets	T1*	-	-	0.18
	T2	0.00	0.00	0.11
	T3	1.00	0.75	0.00
Network end-points	T1*	-	-	0.00
	T2	1.00	0.20	0.00
	T3	0.57	0.50	0.04
ASN-service pairs	T1*	-	-	0.10
	T2	1.00	0.60	0.00
	T3	0.00	0.00	0.00

A3 Defect

The A3 defect increases the rates of pings in the Spotify AP protocol by 100. It is expected to be primarily detectable in the packet features, since the ping messages are only 4 bytes each. As can be seen in Table 6.3, only T3 with network packets detected the defect.

Table 6.3: Detection performance numbers for EWMA on the A3 defect.

feature	test case	precision	TPR	FPR
IP bytes	T1	0.00	0.00	0.17
	T2	0.00	0.00	0.13
	T3	0.00	0.00	0.00
Packets	T1	0.00	0.00	0.18
	T2	0.00	0.00	0.11
	T3	1.00	0.63	0.00
Network end-points	T1	0.00	0.00	0.00
	T2	0.00	0.00	0.00
	T3	0.00	0.00	0.04
ASN-service pairs	T1	0.00	0.00	0.10
	T2	0.00	0.00	0.00
	T3	0.00	0.00	0.00

A4 Defect

The A4 defect downloads a small image file from one of the metadata CDN resources already used by the app, but using HTTP instead of the usual HTTPS. It is expected to be hard to detect using EWMA and the set of features because it should only cause a relatively small deviation for some of the features. T2 have a high *TPR* and low *FPR* for the ASN-service pair feature; see the EWMA chart in Figure 6.6. As no clear explanation of why just this combination of test and feature should be able to find the defect, a second data set captured 2014-05-20 was analyzed for this defect. The EWMA chart for the same defect, test case and feature on the secondary data set can be found in Figure 6.7 and indicates that the test case and feature is not able to detect the A4 defect.

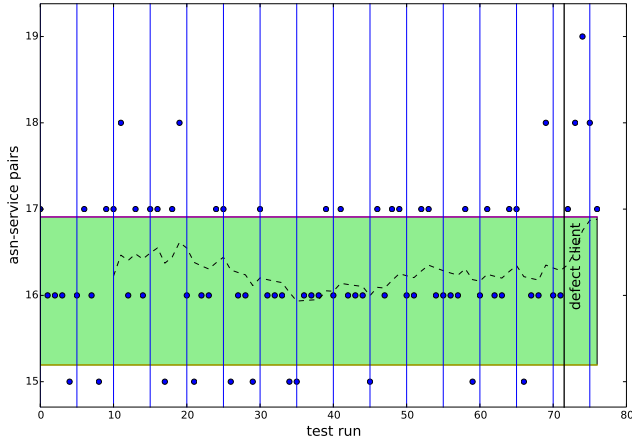


Figure 6.6: EWMA chart of the A4 http-cover defect using the T2 (play song, exit, play song) test case and total number of distinct ASN-service pairs. $span = 20$ and threshold according to equations in Section 3.3.1.

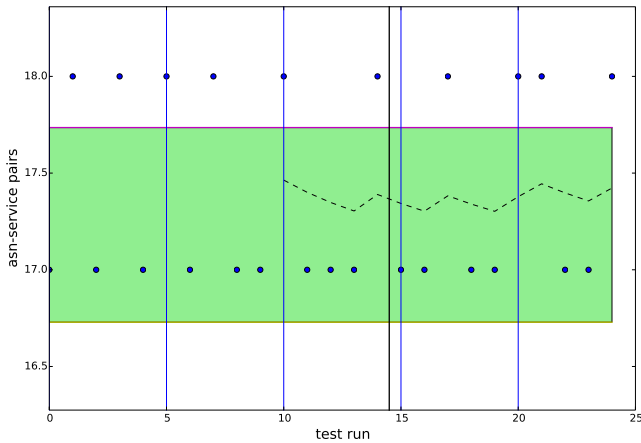


Figure 6.7: EWMA chart of the A4 http-cover defect using the T2 (play song, exit, play song) test case and total number of distinct ASN-service pairs. $span = 20$ and threshold according to equations in Section 3.3.1. Ad-hoc data set for this verification, as described in “The A4 defect” in Section 6.1.3.

Table 6.4: Detection performance numbers for EWMA on the A4 defect.

feature	test case	<i>precision</i>	<i>TPR</i>	<i>FPR</i>
IP bytes	T1	0.00	0.00	0.17
	T2	0.00	0.00	0.13
	T3	0.00	0.00	0.00
Packets	T1	0.00	0.00	0.18
	T2	0.00	0.00	0.11
	T3	0.00	0.00	0.00
Network end-points	T1	0.00	0.00	0.00
	T2	0.00	0.00	0.00
	T3	0.00	0.00	0.04
ASN-service pairs	T1	0.00	0.00	0.10
	T2	0.00	0.00	0.00
	T3	0.00	0.00	0.00

6.2 Novelty Detection Using k-Means Clustering – Data Set I

6.2.1 ROC Curves

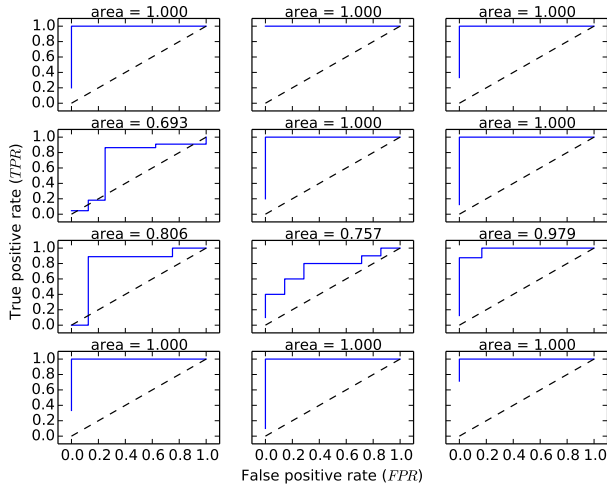


Figure 6.8: ROC curve of k-means clustering novelty detection of stream families. Subplots on x-axis from left to right: T1, T2, T3; y-axis from top to bottom: A1, A2, A3, A4.

To evaluate the false positive rate of the novelty detection system 10% of the test runs of the normal app were randomly selected and removed before training. Vectors from this test set were evaluated against the model of normal in the same way that vectors from runs of a defect app are and the classification performance recorded. Random selection of the test data set and retraining of the model is done several times and the average computed.

For the ROC curve in Figure 6.8 test case runs having at least one stream family vector detected as novel are marked as detected/positive. The true positive and false positive rates are in other word calculated on test runs and not stream families, to make the ROC curves comparable to the ones generated for the EWMA chart.

Since the proposed novelty detection method is detecting novelties in feature vectors based on stream families, an alternative evaluation graph have been added to compare the number of identified novelties for the test data set versus the data set from a defect app (Figure 6.9). A dashed horizontal line is added to mark the number of expected novelties for each app defect type. The number of expected novelties is approximate based on observations and deduction on what is reasonable for a given defect. We cannot determine the exact number of nov-

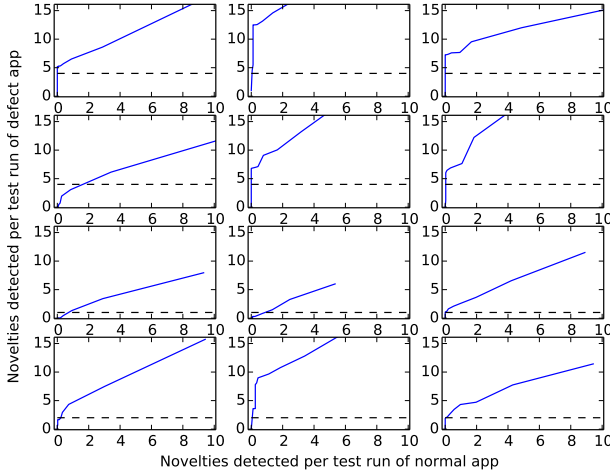


Figure 6.9: Number of identified novelties in the data set from the app with defect versus the normal app. Subplots on x-axis from left to right: T1, T2, T3; y-axis from top to bottom: A1, A2, A3, A4. The dashed line corresponds to the expected number of detected novelties.

elties to be detected for a defect-test-pair, since the defect may have unforeseen consequences and the dynamic network nature of the app makes some requests only occur intermittently. Other parts of the app/back-end communication may also change in between measurements, which may lead to vectors that should be classified as novelties but are not caused by the introduced defect.

One example of the latter is in the test-defect-pairs T2-A1 and T2-A4 where a request for configuration for a module was routed to one AS during the data collection from the normal app and another AS during the data collection for the defect apps. The corresponding high number of average detected novelties for the defect apps can be seen in the subplot (2, 1) and (2, 4) of Figure 6.9.

6.2.2 Detected Novelties

Novelty detection is done for each test case on the data set from each defect app and a test set which consist of 10 % randomly selected from the normal set and removed before training the model. Vectors are classified as novel if their novelty score are above 1. The performance numbers *precision*, *TPR* and *FPR* can be found in Table 6.5.

The silhouette analysis found $k = 39$ for T1, $k = 34$ for T2, and $k = 39$ for T3.

A1 Defect

The A1 defect downloads an image from “spotifypresscom.files.wordpress.com”, which resolve to a set of IP addresses located in two different AS (2635 and 13768), the effects of the defect may have different values, with the app selecting the request target at random during run time. The discovered t^+ novelties are expected to be any or all of the classes: ASN={13768, 2635}, service={UNKNOWN, HTTP}, direction={send, receive}.

No novelties are detected in the test data set for any of the test cases. For T1 5.2 novel vectors on average are detected, for T2 8.5 and T3 6.7. For T1 there are two f^+ in the set of five test case runs: Spotify AP protocol (SPAP), receive (score 4.3 and 5.1), because they are in the middle of two clusters representing two of the three levels in the captures of the play test case discussed in Section 5.1.2. All expected vectors from the defects are classed as novelties and the minimum novelty score of the t^+ novelties is 12.5. For the threshold of novelty score = 1 the performance values are $precision = 0.92$, $TPR = 1.00$, $FPR = 0.01$, and selecting the threshold above 5.1 but below 12.5 would yield a good balance between TPR and FPR .

For T2 there are 4 f^+ in the set of 4 test case runs: SPAP receive (score 1.2, 1.4, 1.8, 1.9). All expected vectors from the defects are classed as novelties and the minimum novelty score of the t^+ novelties is 24.3.

For T3 there were no f^+ . All expected vectors from the defects are classed as novelties and the minimum novelty score of the t^+ novelties is 1.0, but the novelties with the big file download had a minimum novelty score of 2.2.

A2 Defect

The A2 defect is triggered when restarting the app and there should not be detectable by the T1 test case. The expected effects of the A2 defect is harder to judge than for the A1 defect since it affects all parts of the system it removes the cache for. We expect there will be some extra metadata fetching from Spotify’s AP over the SPAP protocol and some images from one or more of the CDN providers.

T1 finds 0 novelties, as expected. No performance numbers involving true positives are specified, as there is nothing to detect.

T2 find on average 2.4 novelties. Only two are deemed true positive, caused by the defect: receive over SPAP with novelty score 1.1 and 1.2 respectively. 10 false positives are found, two for each test case run: send/receive to ASN 1299 over HTTP, score 27.4. The reason for the false positives Requests to a third party API operating on a hybrid type CDN (discussed in Section 2.3) routed requests to a IP on an AS which is not in the normal model. As the requests to AS 1299 are to the same service end-point they are deemed false positives. Adding a test case run of the normal version of the app when the CDN is routing to AS 1299 would eliminate these false positives. There is no ground truth on which vectors are false negatives, but from the detections in T3 we can surmise that the send and

receive vectors for the SPAP service and send/receive for the SSL service to AS 14618 should be considered novel.

T3 find on average 4.1 novelties for the A2 defect. There are 8 test case runs for this test/defect combination. Vectors for AS 14618 SSL send and receive vectors are detected for all test case runs, the cause being the re-download of configuration because of the cache removal. SPAP receive is detected for the five first test case runs and send four of them. For test case runs 3, 4, 6, 7 and 8 the vector of the CDN resource AS 16509, SSL, receive is classified as novel, with novelty score 1.4 for 6-8 and 1.05 for 3 and 4. A possible explanation for this would be a gradual shift to download metadata from the CDN instead of SPAP. Due to the uncertainty of what should be detected performance numbers are left out.

A3 Defect

The A3 defect is expected to be detectable as an increase in the number of packets and a minor increase in number of bytes sent over the SPAP protocol. No novelties are detected for this defect using the T1 test case.

With the T2 test case two false positives are detected: SPAP receive in both cases. The reason the novel SPAP receive are not determined as true positives, even though the A3 defect's extra ping messages is likely to have a minor effect on the receive stream as well in the form of ACK messages, is that no change is detected for SPAP send which ought to be relatively larger and that SPAP receive is not marked novel with the T3 test case, which marks 7/8 of the SPAP send as novel.

With T3 the method identifies one false positive vector (AS 16509, SSL, receive) with score 1.00, just over the threshold. It also correctly identifies 7 of the 8 expected SPAP send, with novelty scores 1.13 to 1.54.

A4 Defect

The A4 defect is expected to deviate in the dimensions service (HTTP), number of bytes and number of packets. It is however a small download (25 kB) relative the other traffic levels.

T1 identifies two false positives for one test case run: send and receive from AS 16509 SSL. It correctly identified three true positives for receive over HTTP from 16509, but misses the send vectors.

T2 suffers the same problem with false positives from AS 1299 as for the test case runs for defect A2, see above. In total 30 false positives are found, of which 20 are AS 1299 vectors and 10 are SPAP receive. The effects of the defect are found as HTTP receive for 8 of the 10 test case runs. The expected AS 16509, HTTP, send vectors are not classified as defect and are therefore false negatives.

T3 correctly identifies 14 vectors of send and receive, AS 16509, HTTP, with scores above 2.4 for the receive. No false positives are detected.

Table 6.5: Detection performance numbers for novelty detection using cluster analysis. This table is NOT directly comparable with Table 6.1 through Table 6.4 as this method considers change detection in stream families whereas the EWMA considers change detection of whole test case runs. (*) Defect not triggered by test case, no detection possible. (**) Left out due to uncertainties, see discussion in the T3 paragraph of the A2 section above.

Defect	test case	precision	TPR	FPR
A1	T1	0.92	1.00	0.01
	T2	0.88	1.00	0.03
	T3	1.00	1.00	0.00
A2	T1*	-	-	0.00
	T2	0.17	0.10	0.06
	T3**	-	-	-
A3	T1	0.00	0.00	0.00
	T2	0.00	0.00	0.01
	T3	0.88	0.88	0.00
A4	T1	0.60	0.50	0.02
	T2	0.21	0.40	0.09
	T3	1.00	1.00	0.00

6.3 Novelty Detection Using k-Means Clustering – Data Set II

In this section the performance of the clustering novelty detection method is evaluated for data set II – the comparison of Spotify iOS 1.0.0 and Spotify iOS 1.1.0 with test cases T4, T5 and T6.

The test case runs for version 1.0.0 are used as the baseline to train the model and the test case runs for version 1.1.0 is compared against the model for 1.0.0 to identify changes in the traffic patterns.

10 % of the baseline test case runs are randomly selected split and off into a test dataset used to verify that the model is not overfitted to the training vectors, making unseen vectors of the same app appear as novelties.

The silhouette analysis found $k = 30$ for T4, $k = 32$ for T5, and $k = 34$ for T6.

Please note that it is likely that the found network activity increases for metadata from 1.0.0 to 1.1.0 described below only occur when starting the app with an empty cache; that is, no general increase in network activity for ordinary usage is established.

6.3.1 Detected Novelties

No vectors from the test data set are classified as novelties.

Test Case T4

The T4 test case detects two novelties for each test case run: SPAP receive with novelty scores 2.8 - 3.3 and SPAP send with novelty scores 1.2 - 2.0, corresponding to an increase in network footprint of 167 kB or 26 %. Since the change is in the encrypted SPAP connection further identification of the root cause are done in the client logs (see Section 4.1.3). The client logs reveals that data increase is due to an added metadata request, possibly due to the introduction of your music (Listing 4.7). The source code revision control system enables assisted binary search in the history for the commit that changed the behavior.

Test Case T5

The T5 test case also detect two novelties for each test case run: SPAP receive with novelty scores 6.0 - 6.7 and SPAP send with novelty scores 1.7 - 2.9, corresponding to an increase in network footprint of 167 kB or 29 %. Manual inspection of the client logs reveals that the detected novelty for the T5 test case is the same as the one found in T4.

Test Case T6

The T6 test case detects, for one test case run, the vector (AS 0, service “-DNS”, direction send) as a novelty due to the unique service signature “-DNS” caused by a protocol misclassification by Bro for one of the local services multicast messages. Two true positive novelties are detected for each test case run: SPAP receive with

scores 9.0 - 9.7 and send with scores 3.5 - 4.5, corresponding to an increase in network footprint of 419 kB or 82 %. Inspection of the client logs reveals an increase in the payload communication with the metadata service of 397 kB.

7

Discussion and Conclusions

This chapter sum up the thesis with a discussion and conclusions.

7.1 Discussion

The methods for network activity change detection proposed in Chapter 5 both have advantages and disadvantages. In an effort to save space the EWMA-chart anomaly detection method introduced in Section 5.1 will be called EWMA, and the k-means clustering novelty detection introduced in Section 5.2.2 called clustering.

EWMA can detect small systematic changes over long time (trends) caused by concept drift. Clustering have difficulties detecting these small drifts because the change detection requires a vector to have a larger distance to its cluster center than any of the vectors in the normal set belonging to the cluster. The drift problem is exacerbated for the clustering method if the model, in an effort to keep it up to date, is automatically updated with all vectors not classified as novel.

As for quick detection after an introduced change EWMA is held back by its intrinsic delay caused by the averaging. This is configurable by lowering the *span* value (affecting the α decay value), but that could lead to a higher rate of false positives due to quicker reaction to change even for the normal data points. The clustering method does not suffer this problem and will detect the change immediately.

When it comes to detection performance the clustering method performs better for some of the artificially introduced defects, see for example the A4 defect where the clustering method with test case T3 achieved *precision* = 1.00, *TPR* =

1.00, $FPR = 1.00$ and the EWMA method achieved $precision = 0.00$, $TPR = 0.00$ for all test cases and features while getting an average false positive rate (FPR) of 0.06.

The clustering method is able to provide more details about the stream set that deviated from normal than the EWMA method, since the EWMA analysis use aggregate statistics from all streams making it impossible to distinguish which have changed. This problem could possibly be mitigated by performing EWMA-chart analysis on the features of rational type for each combination of values for categorical features, making it possible to identify which combinations of categorical feature values the anomaly is detected in.

With the arguments above, a case can be made for using both systems in collaboration – EWMA chart analysis with a long span and a wide threshold to keep track of the long term trends and the cluster analysis novelty detection method for detailed feedback quickly after something has changed.

7.1.1 Related Work

Zhong et al. [31] investigate multiple clustering algorithms for unsupervised learning network intrusion detection using the data set from the 1998 DARPA off-line intrusion detection project. Under the assumption that the normal traffic constitutes $\eta\%$ of the data set (total size N), they introduce a novel technique to label vectors as normal or attack: Find the largest cluster with cluster center μ_0 , sort remaining clusters in ascending order of center-to-center distance from μ_0 and the instances in each cluster the same way, mark the first ηN as normal and the rest as attack. They find that the Natural-Gas algorithm performs best with regard to mean square error and average cluster purity, but that an online version of the k-means algorithm achieves comparable numbers and is superior in execution time. They also find that 200 clusters achieve better accuracy, false positive rate, and detection rate than 100 clusters for the considered clustering algorithms; the greater amount of clusters also incurs a penalty in the execution time.

We believe the proposed method and knowledge from the comparison is applicable on the problem of network activity change detection, seeing change as attack. The assumption that normal traffic dominates the data set holds as long as the data set to be tested is smaller than the data set for normal. As ηN vectors will be classified as attack/change, seldom occurring but normal traffic patterns will be marked as change. This could be positive as seldom occurring patterns could be unwanted and a notification could lead to an investigation, but the total false positive rate could prove too high for alerting without mitigation techniques.

Chakrabarti et al. [4] introduce a framework for handling the problem of evolutionary clustering – producing consistent clustering over time, when new data is incorporated. A modification to the k-means clustering algorithm is presented, which updates the cluster centers with a weighted mean of the new and historic cluster centers. The suggested algorithm for k-means evolutionary clustering may

be useful to address the problems of updating the normal case and keeping the model of normal relevant, discussed in future work (Section 7.2).

7.2 Future Work

In this section we discuss ideas for future work to improve the detection capabilities of the proposed methods.

7.2.1 Updating the Model of Normal

When the novelty detection algorithm have discovered a novel data point and a stakeholder have determined that the data point is benign and should be considered normal from now on, there need to be a mechanism to update the model of normal.

Identified alternatives: (1) Wipe the old model and re-learn using the new app. Drawbacks: time consuming to re-learn the normal model; may miss cases that only occur once in a while, potentially leading to reintroduced false positives later. (2) Add the data point to the normal model, leading to a new cluster or a redefinition of the normal boundary. May eventually lead to a cluttered normal model where every data point is considered normal because it is always inside the normal boundary of some cluster. (3) Combine (2) with a strategy to keep M latest test case runs or data points.

7.2.2 Keeping the Model of Normal Relevant

A problem with the proposed methods, and especially the novelty detection clustering analysis method, is how to decide what test case runs to include in the data set defining normal. Using too few risks that the only a part of the actual variance of the normal behavior is represented, leading to false positives. Using too many/old test case runs of a often changing system risks that the vector space is crowded with regions defined as normal, leading to nothing being detected. There is also the concern of the run time complexity of the involved algorithms slowing down the detection process too much to be usable if the data set is too large.

This will need to be discovered over time. One initial approach is to just keep the last M test case runs.

7.2.3 Improve Identification of Service End-Points

In the proposed method and set of features service end-points are approximated by AS number and detected protocol. This identification method is coarse since all service end-points served over e.g. HTTP hosted at a common cloud provider will be classified as the same end-point. It also fails when a service end-point is served over multiple protocols or from multiple networks with different AS numbers.

Features to better represent service end-points and get more stable segmentation and precise detections should be investigated. Some features that may be used separately or in combination are discussed in Section 2.3.

7.2.4 Temporal Features

Further segmenting measured network activity features with some temporal feature, like timestamp or test-automation tool actions, would increase the change sensibility for regions with lower network activity in the normal case. It would also increase the identification precision for notifications to stakeholders as the detected change can be pinpointed in the time domain. This is probably necessary for the change detection system to be useful in longer test cases, simulating typical real user behavior.

Unfortunately this proves challenging as test-automation tool and the way it controls the client introduces varying delays, so some sort of multipoint synchronization would be needed. Using the test-automation log for state changes improves the segmentation somewhat compared to using the time elapsed since the first packet, but suffers from different levels of residual network traffic for some actions like pausing a track.

Two possible ways forward are: (1) synchronize test case runs on test-automation or client state changes and sample with some partially overlapping windowing function; and (2) using explicit synchronization states with delays before and after in the test-automation system to avoid state-overlapping network traffic.

7.2.5 Network Hardware Energy Usage

Network change detection could include detection of changes in the usage levels of network hardware, which affects the device battery drain. The feature of simulated or measured network hardware uptime both for total and per stream family could be added to the methods proposed in this thesis.

7.3 Conclusions

Our main research questions (stated in Section 1.3) were:

(1) *What machine learning algorithm is most suitable for comparing network traffic sessions for the purpose of identifying changes in the network footprint and service end-points of the app?*

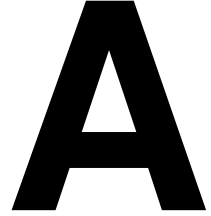
Using clustering analysis and novelty detection enable quick stakeholder notification when changes occur. It is capable of identifying network footprint changes exceeding the extreme values of the training set and if there are distinct local extremes, even values between a the maximum of one peak and the minimum of another. Detection of service end-points change depends on the used features' abilities to describe a service end-point such that they can be clustered together.

(2) *What are the best features to use and how should they be transformed to*

suit the selected machine learning algorithm when constructing a network traffic model that allows for efficient detection of changes in the network footprint and service end-points?

The communication data size is a necessary feature to enable detection of changes in the network footprint. We have further shown that segmenting the network traffic metrics into buckets of related streams improves the detection likelihood of small deviations in streams. The segmentation also adds the value of an initial identification of what kinds of traffic have changed, enabling quicker root cause analysis. The evaluated method of segmentation on AS number, detected protocol and flow direction work well when comparing traffic measurements from the same source network and approximately at the same time, due to the dynamic routing problem mostly observed for CDNs.

Appendix



Data Set Features

Bro defines the following data types relevant this thesis:

- **port**: integer;
- **count**: integer;
- **addr**: network layer address in grouped format (ipv4 format: 172.16.13.21, ipv6 format: ff02::1);
- **string**: string of characters

Table A.1: Features extracted with Bro from each network packet of the raw network data dump.

name	data type	description
conn	string	unique connection identifier.
ts	time	unix timestamp with μs resolution.
direction	string	R (receive) for traffic to the device running the SUT and S (send) for from the device running the SUT.
transport_protocol	string	transport level protocol: icmp, tcp, udp.
orig_p	port	originating (phone) transport protocol port for tcp/udp, type for icmp.
resp_h	addr	IP address of the receiving party.
resp_p	port	destination transport protocol port for tcp/udp, type for icmp.
services	string	identified protocol by Bro's protocol identification system.
eth_len	count	size of the ethernet frame in bytes.
ip_len	count	size of the IP packet in bytes.
transport_len	count	size of the transport protocol packet in bytes.
payload_len	count	size of the payload data in bytes.

Table A.2: Features derived from features in Table A.1.

name	data type	description
country_code	string	two character country code for resp_h according to MaxMind's IP-to-country database of 2014-04-28.
asn	count	AS number for resp_h according to MaxMind's IP-to-ASN database of 2014-04-28.
port_group	string	dns (53/udp), http (80/tcp), https (443/tcp), spap (4070/tcp), low (<1024), high (\geq 1024).
ip_net_twelve	string	CIDR /12 subnet of resp_h, example: 193.176.0.0/12.
ip_net_twenty	string	CIDR /20 subnet of resp_h, example: ff02::/20.
ptr	string	DNS PTR record for resp_h or the PTR query (1.1.168.192.in-addr.arpa.) if no PTR record is returned.
is_multicast	count	1/0 denoting if resp_h belongs to a multicast subnet defined by IANA or global broadcast address. IPv6: ff00::/8, IPv4: 224.0.0.0/4, 255.255.255.255/32.
is_private_net	string	1/0 denoting if resp_h belongs to a private net as defined by RFC1918 or equivalent. IPv6: fc00::/7, IPv4: 10.0.0.0/8, 172.16.0.0/16, 192.168.0.0/16.

Table A.3: Features extracted from the test automation tool.

name	data type	description
vertex_change	string	Describe transitions from one vertex to another in the Graphwalker test model graph.

Table A.4: Features extracted from the instrumented client.

name	data type	description
RequestAccountingAdd	string, integer, integer, integer	Sent everytime a network request is completed by an app module supporting the RequestAccounting logging. Specifies SPAP endpoint URI, network type, bytes downloaded, bytes uploaded.
HttpRequestAdd	string, integer, integer	Sent everytime a HTTP(s) request detected by the NSURLProtocol (implementation described in Section 4.1.3) is completed by the app. Specifies URL, bytes downloaded, bytes uploaded.

B

Data Set Statistics

B.1 Data Set I - Artificial Defects

Table B.1: Data set statistics for test case T1

App type	Measurement	Statistic	Value
Normal	Number of packets	mean	3,774.60
		std	370.25
		min	3,345.00
		25%	3,587.00
		50%	3,622.00
		75%	3,954.50
		max	5,165.00
	Aggregate IP size	mean	3,659,696.06
		std	476,921.73
		min	3,202,439.00
		25%	3,436,810.00
		50%	3,446,860.00
		75%	3,947,839.50
		max	5,475,984.00
	Aggregate payload size	mean	3,464,279.85
		std	457,470.20
		min	3,029,779.00
		25%	3,251,562.00
		50%	3,261,184.00
		75%	3,742,833.00
		max	5,208,924.00
	Unique network end-points	mean	22.78
		std	1.86
		min	19.00
		25%	22.00
		50%	22.00
		75%	23.50
		max	30.00
	Unique streams	mean	46.17
		std	3.89
min		34.00	
25%		43.00	

Table B.1 – continued from previous page

App type	Measurement	Statistic	Value			
A1		50%	46.00			
		75%	49.00			
		max	55.00			
	Number of packets		mean	5,003.20		
			std	464.23		
			min	4,636.00		
			25%	4,654.00		
			50%	4,704.00		
			75%	5,509.00		
			max	5,513.00		
			Aggregate IP size		mean	4,957,549.40
					std	679,264.77
					min	4,458,671.00
					25%	4,462,958.00
					50%	4,463,658.00
	75%	5,669,200.00				
	Aggregate payload size		mean	4,704,898.80		
			std	664,600.72		
			min	4,217,826.00		
			25%	4,218,947.00		
			50%	4,222,982.00		
			75%	5,395,516.00		
	Unique network end-points		max	5,469,223.00		
			mean	24.20		
			std	1.92		
			min	22.00		
			25%	23.00		
			50%	24.00		
			75%	25.00		
	Unique streams		max	27.00		
mean			45.20			
std			3.27			
min			40.00			
25%			45.00			
50%			46.00			
75%			46.00			
A2	Number of packets		max	49.00		
			mean	3,475.68		
			std	92.40		
			min	3,374.00		
			25%	3,418.50		
			50%	3,442.50		
			75%	3,479.75		
	Aggregate IP size		max	3,700.00		
			mean	3,338,687.00		
			std	59,066.10		
			min	3,302,273.00		
			25%	3,308,176.75		
			50%	3,311,814.00		
			75%	3,323,333.00		
	Aggregate payload size		max	3,472,004.00		
			mean	3,159,219.95		
			std	54,354.65		
			min	3,125,681.00		
			25%	3,131,278.75		
			50%	3,133,714.00		
			75%	3,146,877.50		
	Unique network end-points		max	3,286,488.00		
			mean	21.82		
			std	1.22		
			min	20.00		
			25%	21.00		
			50%	22.00		
			75%	22.75		
	Unique streams		max	25.00		
			mean	43.05		

Table B.1 – continued from previous page

App type	Measurement	Statistic	Value
A3		std	3.93
		min	37.00
		25%	40.25
		50%	42.00
		75%	47.00
		max	49.00
		Number of packets	mean
	std		52.38
	min		3,615.00
	25%		3,649.00
	50%		3,669.00
	75%		3,717.00
	max		3,768.00
	Aggregate IP size	mean	3,435,826.33
		std	8,089.57
		min	3,420,274.00
		25%	3,435,016.00
50%		3,436,256.00	
75%		3,440,139.00	
max		3,445,837.00	
Aggregate payload size	mean	3,245,870.89	
	std	6,563.57	
	min	3,231,142.00	
	25%	3,245,936.00	
	50%	3,247,863.00	
	75%	3,248,847.00	
	max	3,252,118.00	
Unique network end-points	mean	22.56	
	std	1.24	
	min	21.00	
	25%	22.00	
	50%	23.00	
	75%	23.00	
	max	25.00	
Unique streams	mean	46.11	
	std	2.71	
	min	42.00	
	25%	45.00	
	50%	46.00	
	75%	47.00	
	max	52.00	
A4	Number of packets	mean	3,928.00
		std	210.27
		min	3,686.00
		25%	3,859.00
		50%	4,032.00
		75%	4,049.00
		max	4,066.00
		Aggregate IP size	mean
	std		183,488.20
	min		3,680,043.00
	25%		3,837,730.50
	50%		3,995,418.00
	75%		3,997,826.50
	max		4,000,235.00
	Aggregate payload size		mean
		std	172,386.29
		min	3,490,107.00
		25%	3,638,770.50
		50%	3,787,434.00
		75%	3,788,681.00
		max	3,789,928.00
		Unique network end-points	mean
	std		0.58
	min		22.00
	25%		22.50
	50%		23.00

Table B.1 – continued from previous page

App type	Measurement	Statistic	Value
		75%	23.00
		max	23.00
	Unique streams	mean	47.00
		std	5.29
		min	43.00
		25%	44.00
		50%	45.00
		75%	49.00
		max	53.00

Table B.2: Data set statistics for test case T2

App type	Measurement	Statistic	Value
Normal	Number of packets	mean	6,917.28
		std	755.12
		min	5,912.00
		25%	6,344.50
		50%	6,565.00
		75%	7,372.00
		max	8,709.00
		Aggregate IP size	mean
	std		990,003.08
	min		5,840,768.00
	25%		6,280,937.00
	50%		6,500,155.00
	75%		7,637,116.75
	max		9,349,237.00
	Aggregate payload size		mean
		std	950,816.52
		min	5,535,412.00
		25%	5,952,611.00
		50%	6,160,272.00
		75%	7,257,475.75
		max	8,898,729.00
		Unique network end-points	mean
	std		2.46
	min		24.00
	25%		26.00
	50%		28.00
	75%		29.00
	max		36.00
	Unique streams		mean
		std	4.69
		min	53.00
		25%	59.75
50%		63.50	
75%		67.25	
max		75.00	
A1		Number of packets	mean
	std		98.68
	min		7,870.00
	25%		7,880.50
	50%		7,945.00
	75%		8,023.25
	max		8,075.00
	Aggregate IP size		mean
		std	26,410.87
		min	7,795,363.00
		25%	7,797,178.00
		50%	7,811,243.00
		75%	7,831,424.25
	max	7,851,588.00	
	mean	7,405,198.25	

Aggregate payload size

Table B.2 – continued from previous page

App type	Measurement	Statistic	Value	
		std	21,326.76	
		min	7,387,131.00	
		25%	7,389,183.00	
		50%	7,400,345.00	
		75%	7,416,360.25	
		max	7,432,972.00	
	Unique network end-points	mean	30.50	
		std	0.58	
		min	30.00	
		25%	30.00	
		50%	30.50	
		75%	31.00	
	Unique streams	max	31.00	
		mean	64.75	
		std	4.99	
		min	61.00	
		25%	61.75	
		50%	63.00	
	A2	Number of packets	75%	66.00
			max	72.00
			mean	6,843.00
std			51.45	
min			6,783.00	
25%			6,795.00	
Aggregate IP size		50%	6,866.00	
		75%	6,869.00	
		max	6,902.00	
		mean	6,621,587.40	
		std	5,765.82	
		min	6,612,471.00	
Aggregate payload size		25%	6,620,802.00	
		50%	6,622,081.00	
		75%	6,624,750.00	
		max	6,627,833.00	
		mean	6,268,332.60	
		std	4,025.42	
Unique network end-points		min	6,262,447.00	
		25%	6,266,338.00	
		50%	6,269,952.00	
	75%	6,270,053.00		
	max	6,272,873.00		
	mean	30.60		
Unique streams	std	4.28		
	min	27.00		
	25%	29.00		
	50%	29.00		
	75%	30.00		
	max	38.00		
A3	Number of packets	mean	71.80	
		std	4.66	
		min	66.00	
		25%	69.00	
		50%	71.00	
		75%	76.00	
	Aggregate IP size	max	77.00	
		mean	6,342.20	
		std	77.31	
		min	6,210.00	
		25%	6,314.25	
		50%	6,348.00	
		75%	6,397.75	
		max	6,437.00	
		mean	6,037,592.80	
		std	41,824.09	
		min	5,956,502.00	
		25%	6,033,182.75	
		50%	6,047,295.50	

Table B.2 – continued from previous page

App type	Measurement	Statistic	Value
	Aggregate payload size	75%	6,060,140.00
		max	6,083,968.00
		mean	5,709,798.80
		std	38,374.58
		min	5,635,822.00
		25%	5,706,092.75
		50%	5,715,913.50
		75%	5,732,610.00
	Unique network end-points	max	5,754,559.00
		mean	29.10
		std	2.28
		min	25.00
		25%	28.25
		50%	29.00
		75%	31.00
		max	32.00
	Unique streams	mean	63.40
		std	2.76
		min	59.00
		25%	61.50
50%		63.00	
75%		64.75	
max		68.00	
A4		Number of packets	mean
	std		76.50
	min		5,899.00
	25%		5,988.50
	50%		6,060.00
	75%		6,110.25
	max		6,117.00
	Aggregate IP size	mean	5,864,168.30
		std	53,503.95
		min	5,798,759.00
		25%	5,819,417.00
		50%	5,863,606.00
		75%	5,903,693.25
		max	5,951,664.00
	Aggregate payload size	mean	5,551,616.00
		std	50,025.02
		min	5,494,047.00
		25%	5,509,197.00
		50%	5,549,710.00
		75%	5,588,434.50
max		5,635,960.00	
Unique network end-points	mean	29.90	
	std	3.73	
	min	26.00	
	25%	27.00	
	50%	29.50	
	75%	30.75	
	max	38.00	
Unique streams	mean	61.80	
	std	7.66	
	min	53.00	
	25%	55.50	
	50%	60.00	
	75%	65.75	
	max	77.00	

Table B.3: Data set statistics for test case T3

App type	Measurement	Statistic	Value
Normal	Number of packets	mean	3,160.36

Table B.3 – continued from previous page

App type	Measurement	Statistic	Value
		std	156.38
		min	2,760.00
		25%	3,058.00
		50%	3,191.00
		75%	3,284.00
		max	3,452.00
	Aggregate IP size	mean	2,170,137.67
		std	127,404.95
		min	1,813,872.00
		25%	2,095,819.00
		50%	2,191,681.00
		75%	2,251,307.00
	Aggregate payload size	mean	2,006,822.90
		std	119,506.13
		min	1,671,992.00
		25%	1,937,204.00
		50%	2,025,713.00
		75%	2,084,790.00
	Unique network end-points	mean	27.22
		std	2.36
		min	23.00
		25%	26.00
		50%	27.00
		75%	29.00
Unique streams	mean	63.32	
	std	4.34	
	min	52.00	
	25%	60.00	
	50%	63.00	
	75%	66.00	
A1	Number of packets	mean	5,042.00
		std	248.80
		min	4,822.00
		25%	4,907.00
		50%	4,992.00
		75%	5,152.00
	Aggregate IP size	mean	4,079,885.00
		std	269,431.86
		min	3,802,473.00
		25%	3,949,549.50
		50%	4,096,626.00
		75%	4,218,591.00
	Aggregate payload size	mean	3,945,812.67
		std	113,913.99
		min	3,837,903.00
		25%	3,886,266.00
		50%	3,934,629.00
		75%	3,999,767.50
	Unique network end-points	mean	31.33
		std	4.04
		min	29.00
		25%	29.00
		50%	29.00
		75%	32.50
Unique streams	mean	70.67	
	std	4.04	
	min	66.00	
	25%	69.50	
	50%	73.00	

Table B.3 – continued from previous page

App type	Measurement	Statistic	Value
		75%	73.00
		max	73.00
A2	Number of packets	mean	3,802.00
		std	113.72
		min	3,621.00
		25%	3,730.50
		50%	3,830.50
		75%	3,868.25
		max	3,963.00
		Aggregate IP size	mean
	std		143,030.40
	min		2,459,521.00
	25%		2,474,209.75
	50%		2,720,123.00
	75%		2,746,446.00
	max		2,766,137.00
	Aggregate payload size		mean
		std	137,710.65
		min	2,266,256.00
		25%	2,285,070.75
		50%	2,521,829.00
		75%	2,545,257.25
		max	2,566,117.00
		Unique network end-points	mean
	std		3.07
	min		30.00
25%	31.75		
50%	32.00		
75%	33.25		
max	40.00		
Unique streams	mean		76.88
	std	3.18	
	min	73.00	
	25%	75.00	
	50%	75.50	
	75%	78.75	
	max	82.00	
	A3	Number of packets	mean
std			193.91
min			3,121.00
25%			3,497.00
50%			3,614.50
75%			3,675.25
max			3,706.00
Aggregate IP size			mean
		std	177,368.29
		min	1,827,135.00
		25%	2,216,456.25
		50%	2,302,374.50
		75%	2,330,542.75
		max	2,392,168.00
		Aggregate payload size	mean
std			167,596.19
min			1,665,887.00
25%			2,036,466.00
50%			2,115,624.00
75%			2,140,559.25
max			2,201,232.00
Unique network end-points			mean
		std	1.96
		min	24.00
		25%	25.75
		50%	26.50
		75%	29.00
		max	29.00
		Unique streams	mean
std			5.01

Table B.3 – continued from previous page

App type	Measurement	Statistic	Value
A4	Number of packets	min	54.00
		25%	61.25
		50%	64.00
		75%	66.25
		max	70.00
		mean	3,014.00
	std	86.40	
	min	2,856.00	
	25%	2,980.00	
	50%	3,029.00	
	75%	3,067.00	
	max	3,119.00	
	Aggregate IP size	mean	2,040,448.43
		std	37,739.89
		min	1,982,132.00
		25%	2,027,757.50
		50%	2,032,429.00
		75%	2,055,363.00
	max	2,102,337.00	
Aggregate payload size	mean	1,884,651.71	
	std	33,941.31	
	min	1,835,302.00	
	25%	1,871,855.50	
	50%	1,877,495.00	
	75%	1,897,536.00	
max	1,940,982.00		
Unique network end-points	mean	28.29	
	std	2.06	
	min	25.00	
	25%	27.50	
	50%	29.00	
	75%	29.00	
max	31.00		
Unique streams	mean	62.43	
	std	4.47	
	min	56.00	
	25%	60.50	
	50%	62.00	
	75%	64.00	
max	70.00		

B.2 Data Set II - Real World Scenario

Table B.4: Data set statistics for test case T4

App type	Measurement	Statistic	Value
1.0.0	Number of packets	mean	3,970.64
		std	134.08
		min	3,723.00
		25%	3,886.00
		50%	3,953.00
		75%	4,058.00
	max	4,348.00	
	Aggregate IP size	mean	2,927,543.52
		std	112,101.24
		min	2,719,011.00
		25%	2,858,263.75
		50%	2,913,247.00
		75%	2,987,192.25
	max	3,283,753.00	

Table B.4 – continued from previous page

App type	Measurement	Statistic	Value
	Aggregate payload size	mean	2,722,386.02
		std	105,349.54
		min	2,526,623.00
		25%	2,658,417.50
		50%	2,709,089.00
		75%	2,776,299.25
		max	3,058,857.00
	Unique network end-points	mean	28.62
		std	2.40
		min	25.00
		25%	27.00
		50%	28.00
		75%	30.00
		max	35.00
	Unique streams	mean	54.00
		std	5.54
		min	43.00
		25%	50.00
		50%	53.50
		75%	58.00
		max	68.00
1.1.0	Number of packets	mean	4,056.00
		std	120.36
		min	3,862.00
		25%	3,956.00
		50%	4,063.00
		75%	4,138.00
		max	4,266.00
	Aggregate IP size	mean	3,054,163.05
		std	106,375.67
		min	2,897,609.00
		25%	2,938,301.00
		50%	3,062,319.00
		75%	3,121,189.00
		max	3,244,857.00
	Aggregate payload size	mean	2,844,554.86
		std	100,205.51
		min	2,697,513.00
		25%	2,734,585.00
		50%	2,852,099.00
		75%	2,908,513.00
		max	3,025,157.00
	Unique network end-points	mean	28.95
		std	1.47
		min	27.00
		25%	28.00
		50%	29.00
		75%	30.00
		max	32.00
	Unique streams	mean	52.57
		std	5.88
		min	44.00
		25%	48.00
		50%	52.00
75%		55.00	
max		66.00	

Table B.5: Data set statistics for test case T5

App type	Measurement	Statistic	Value
1.0.0		mean	1,505.30
		std	51.08
		min	1,433.00

Number of packets

Table B.5 – continued from previous page

App type	Measurement	Statistic	Value	
		25%	1,477.50	
		50%	1,490.50	
		75%	1,526.75	
		max	1,683.00	
	Aggregate IP size	mean	872,930.42	
		std	35,829.69	
		min	852,906.00	
		25%	858,727.75	
		50%	861,949.50	
		75%	868,165.75	
	Aggregate payload size	max	995,980.00	
		mean	795,508.26	
		std	33,497.48	
		min	777,782.00	
		25%	781,447.25	
		50%	785,559.50	
	Unique network end-points	75%	790,394.75	
		max	909,764.00	
		mean	26.30	
		std	1.74	
		min	24.00	
		25%	25.00	
	Unique streams	50%	26.00	
		75%	27.00	
max		32.00		
mean		45.44		
std		3.65		
min		40.00		
1.1.0	Number of packets	25%	43.00	
		50%	45.00	
		75%	47.00	
		max	57.00	
		Aggregate IP size	mean	1,654.43
			std	26.20
	min		1,611.00	
	25%		1,635.00	
	50%		1,651.00	
	75%		1,664.00	
	Aggregate payload size	max	1,710.00	
		mean	1,026,279.29	
		std	5,152.99	
		min	1,013,151.00	
		25%	1,024,224.00	
		50%	1,025,300.00	
	Unique network end-points	75%	1,028,630.00	
		max	1,038,303.00	
		mean	941,079.67	
		std	4,549.05	
		min	927,523.00	
		25%	939,746.00	
	Unique streams	50%	940,435.00	
		75%	941,519.00	
max		950,846.00		
mean		26.05		
std		1.83		
min		23.00		
	25%	25.00		
	50%	26.00		
	75%	27.00		
	max	32.00		
	mean	44.19		
	std	3.08		
	min	40.00		
	25%	42.00		
	50%	43.00		
	75%	46.00		
	max	54.00		

Table B.6: Data set statistics for test case T6

App type	Measurement	Statistic	Value
1.0.0	Number of packets	mean	3,896.35
		std	215.18
		min	3,494.00
		25%	3,777.50
		50%	3,827.00
		75%	3,960.50
		max	4,673.00
	Aggregate IP size	mean	3,345,209.26
		std	187,475.09
		min	2,714,340.00
		25%	3,300,056.00
		50%	3,309,643.00
		75%	3,393,413.00
		max	4,079,685.00
	Aggregate payload size	mean	3,169,288.74
		std	179,762.14
		min	2,553,508.00
		25%	3,130,193.50
		50%	3,138,611.00
		75%	3,212,407.00
		max	3,872,425.00
	Unique network end-points	mean	28.42
		std	2.33
		min	26.00
		25%	27.00
		50%	28.00
		75%	29.50
		max	36.00
Unique streams	mean	57.84	
	std	5.81	
	min	47.00	
	25%	53.00	
	50%	57.00	
	75%	61.50	
	max	72.00	
1.1.0	Number of packets	mean	4,553.65
		std	296.70
		min	4,237.00
		25%	4,312.00
		50%	4,473.00
		75%	4,589.00
		max	5,098.00
	Aggregate IP size	mean	3,982,639.41
		std	277,835.89
		min	3,743,046.00
		25%	3,812,335.00
		50%	3,871,313.00
		75%	3,911,745.00
		max	4,486,828.00
	Aggregate payload size	mean	3,773,910.47
		std	266,199.69
		min	3,548,342.00
		25%	3,614,551.00
		50%	3,661,229.00
		75%	3,699,937.00
		max	4,256,736.00
	Unique network end-points	mean	28.94
		std	2.73
		min	26.00
		25%	27.00
		50%	28.00
		75%	29.00
		max	35.00

Table B.6 – continued from previous page

App type	Measurement	Statistic	Value
	Unique streams	mean	59.12
		std	10.22
		min	50.00
		25%	53.00
		50%	55.00
		75%	59.00
		max	82.00

Bibliography

- [1] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. In *Proc. ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. (Cited on page 21.)
- [2] Pavel Berkhin. *A survey of clustering data mining techniques*. In *Grouping Multidimensional Data*, pages 25–71. Springer, 2006. (Cited on page 20.)
- [3] R Braden. *Rfc 1122. Requirements for Internet Hosts—Communication Layers*, 1989. (Cited on page 9.)
- [4] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. *Evolutionary clustering*. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006. (Cited on page 72.)
- [5] Cisco. *Cisco visual networking index: Global mobile data traffic forecast update, 2012–2017*. Technical report, Cisco, February 2013. (Cited on page 1.)
- [6] György Dán and Niklas Carlsson. *Dynamic content allocation for cloud-assisted service of periodic workloads*. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2014. (Cited on page 15.)
- [7] Sanjoy Dasgupta. *The Hardness of K-Means Clustering*. Department of Computer Science and Engineering, University of California, San Diego, 2008. (Cited on page 21.)
- [8] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007. (Cited on pages xix and 2.)
- [9] Mikael Goldmann and Gunnar Kreitz. *Measurements on the spotify peer-assisted music-on-demand streaming system*. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 206–211, 2011. (Cited on page 4.)

- [10] Børge Haugset and Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile 2008 Conference*, pages 27–38, 2008. (Cited on page 2.)
- [11] Victoria J Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004. (Cited on page 17.)
- [12] J Stuart Hunter. The exponentially weighted moving average. *Journal of Quality Technology*, 18(4):203–210, 1986. (Cited on page 19.)
- [13] Raul Jimenez, Gunnar Kreitz, Björn Knutsson, Marcus Isaksson, and Seif Haridi. Integrating smartphones in spotify’s peer-assisted music streaming service. 2013. Draft. (Cited on pages 4, 14, 27, and 39.)
- [14] Gunnar Kreitz and Fredrik Niemelä. Spotify – large scale, low latency, p2p music-on-demand streaming. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10, 2010. (Cited on pages 4 and 13.)
- [15] Erik Kurin and Adam Melin. Data-driven test automation: Augmenting gui testing in a web application. Master’s thesis, Linköping University, 2013. (Cited on page 4.)
- [16] Michael Larsen and Fernando Gont. Rfc 1122: Recommendations for transport-protocol port randomization. 2011. (Cited on page 11.)
- [17] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. Mining in a data-flow environment: Experience in network intrusion detection. In *Proc. ACM international conference on Knowledge discovery and data mining (SIGKDD)*, pages 114–124. ACM, 1999. (Cited on page 28.)
- [18] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. (Cited on page 21.)
- [19] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *WALCOM: Algorithms and Computation*, pages 274–285. Springer, 2009. (Cited on page 21.)
- [20] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008. (Cited on page 12.)
- [21] B. Niven-Jenkins, F. Le Faucheur, and N. Bitar. Rfc 6707: Content distribution network interconnection (cdni) problem statement. 2012. (Cited on page 14.)
- [22] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Proc. USENIX Security Symposium*, 1998. (Cited on pages 13 and 41.)
- [23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Ma-

- chine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (Cited on page 27.)
- [24] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. (Cited on page 22.)
- [25] Vinay Setty, Gunnar Kreitz, Roman Vitenberg, Maarten van Steen, Guido Urdaneta, and Staffan Gimåker. The hidden pub/sub of spotify (industry article). In *Proc. ACM international conference on Distributed Event-Based Systems (DEBS)*, pages 231–240, 2013. Arlington, TX. (Cited on page 14.)
- [26] Phil Simon. *Too Big to Ignore: The Business Case for Big Data*. John Wiley & Sons, 2013. (Cited on page 17.)
- [27] Stanley Smith Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946. (Cited on page 22.)
- [28] Lionel Tarassenko, Alexandre Nairac, Neil Townsend, and P Cowley. Novelty detection in jet engines. *IEE Colloquium on Condition Monitoring: Machinery, External Structures and Health*, 034:4/1–4/5, 1999. (Cited on page 28.)
- [29] Paul Watson. Slipping in the window: Tcp reset attacks. Technical report, 2003. (Cited on page 11.)
- [30] Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. In *Proc. International Conference on Pattern Recognition*, volume 4, pages 385–388. IEEE, 2002. (Cited on page 28.)
- [31] Shi Zhong, Taghi M Khoshgoftaar, and Naeem Seliya. Clustering-based network intrusion detection. *International Journal of Reliability, Quality and Safety Engineering*, 14(02):169–187, 2007. (Cited on page 72.)



Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>