Progress Report Meeting
May 2015

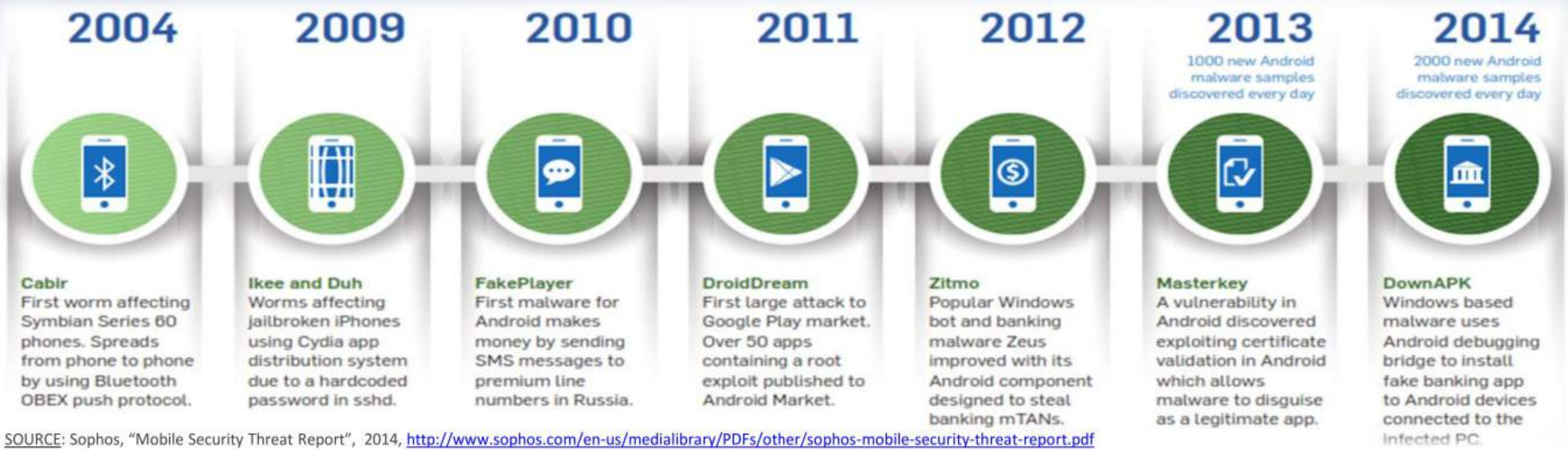
On device Anomaly Detection for resource-limited systems

Maroua Ben-Attia
Chamseddine Talhi, Abdelwahab Hamou-Lhadj, Babak Khosravifar

École de technologie supérieure (ÉTS)
Computer System Architecture Research Lab (LASI)
Department of software engineering and IT
Montreal, QC, Canada

Introduction: *Malware Evolution*

***2 years** of mobile malware evolution <=> **20 years** of Computer malware evolution*



2004

Cabir
First worm affecting Symbian Series 60 phones. Spreads from phone to phone by using Bluetooth OBEX push protocol.

2009

Ikee and Duh
Worms affecting jailbroken iPhones using Cydia app distribution system due to a hardcoded password in sshd.

2010

FakePlayer
First malware for Android makes money by sending SMS messages to premium line numbers in Russia.

2011

DroidDream
First large attack to Google Play market. Over 50 apps containing a root exploit published to Android Market.

2012

Zitmo
Popular Windows bot and banking malware Zeus improved with its Android component designed to steal banking mTANs.

2013

1000 new Android malware samples discovered every day

Masterkey
A vulnerability in Android discovered exploiting certificate validation in Android which allows malware to disguise as a legitimate app.

2014

2000 new Android malware samples discovered every day

DownAPK
Windows based malware uses Android debugging bridge to install fake banking app to Android devices connected to the infected PC.

SOURCE: Sophos, "Mobile Security Threat Report", 2014, http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.pdf

*F-Secure 2014: "**Android devices** are the more popular target for attacks with **294 new threat** families or variants "*

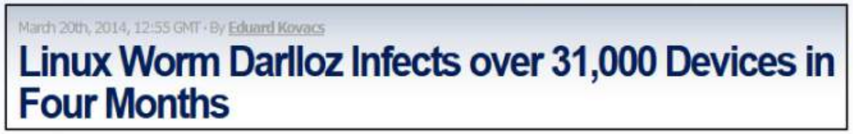
General-purpose small devices

2009:Psyb0t
2010:Stuxnet
2012: DNSChanger
2013:GPS attack
2013:Linux. Darlloz
2014:The Moon

- Linux-based routers, DSL modems
- Industrial control systems (ICS)
- Computers and routers
- GPS based systems
- Cameras, set-top boxes
- Linksys routers

Problem

Security Issue: Just in 2014 !



March 20th, 2014, 12:55 GMT · By Eduard Kovacs

Linux Worm Darlloz Infects over 31,000 Devices in Four Months

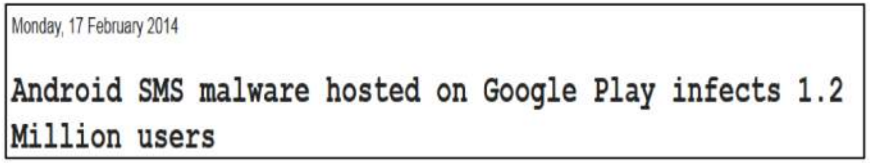
http://news.softpedia.com/news/Linux-Worm-Darlloz-Infects-over-31-000-Devices-in-Four-Months-433242.shtml

"The Moon scans for vulnerable devices as it looks to continue spreading, over 1,000 Linksys routers are already believed to be infected by the malware."

http://www.ubergizmo.com/2014/02/linksys-routers-malware-the-moon-spreading

A Criminal campaign named Windigo Operation has controlled about 25 thousand Unix servers that send millions of fake mails and put 500 thousand computers at risk every day.

http://www.rcoutada.net/2014/03/new-linux-servers-cpanel-backdoor-ebury-a/



Monday, 17 February 2014

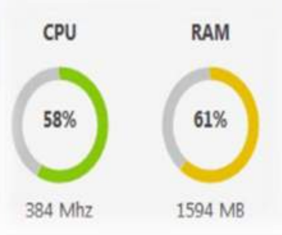
Android SMS malware hosted on Google Play infects 1.2 Million users

http://www.hackleaks.in/2014/02/android-sms-malware-hosted-on-google.html

Resource Limitations

Low power CPUs

- Lightweight processing
- limited multitasking

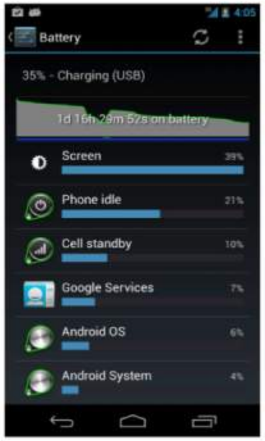


Memory

CPU	600 MHz
RAM	512 MB
Storage	microSD slot
OS	Linux , Android

gumstix®

Battery life



Battery

35% - Charging (USB)

1d 16h 29m 57s on battery

Screen 39%

Phone idle 21%

Cell standby 10%

Google Services 7%

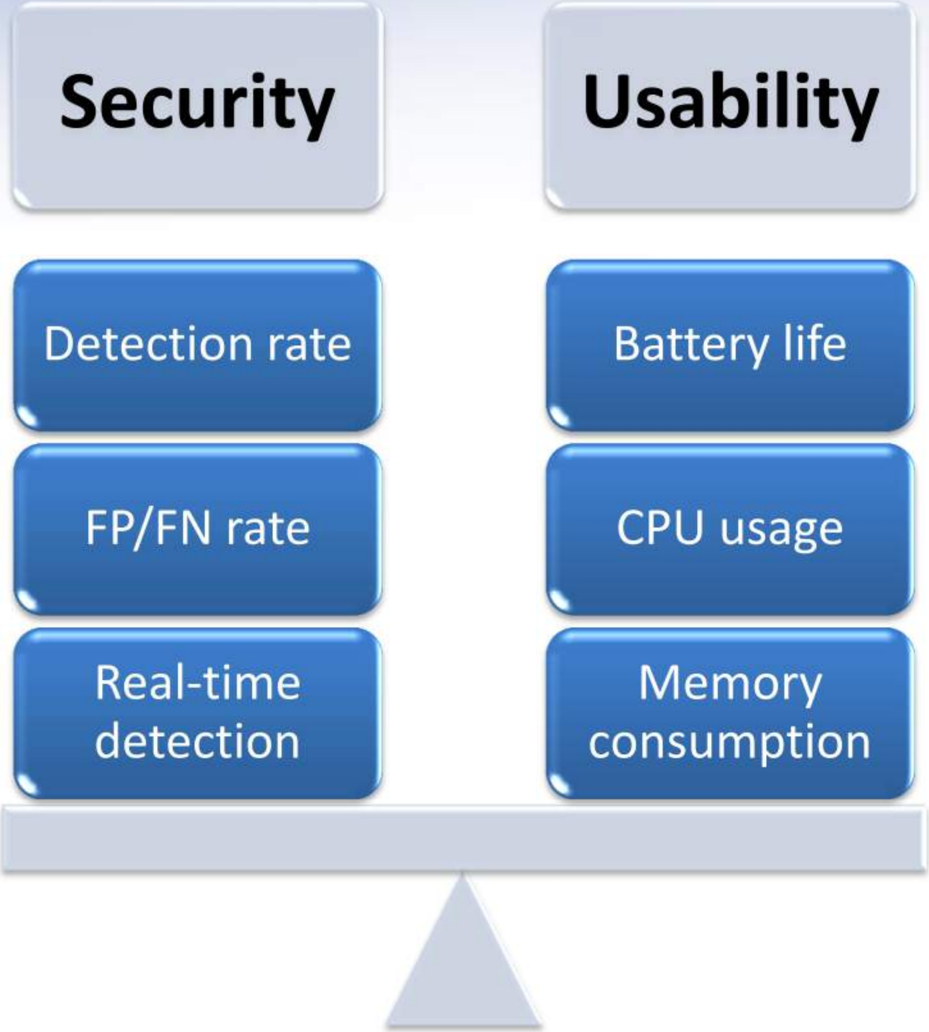
Android OS 6%

Android System 4%

CPU	720MHz
RAM	256 MB
Storage	4GB microSD
OS	Android, Linux

beaglebone

Objective



Security

Detection rate

FP/FN rate

Real-time detection

Usability

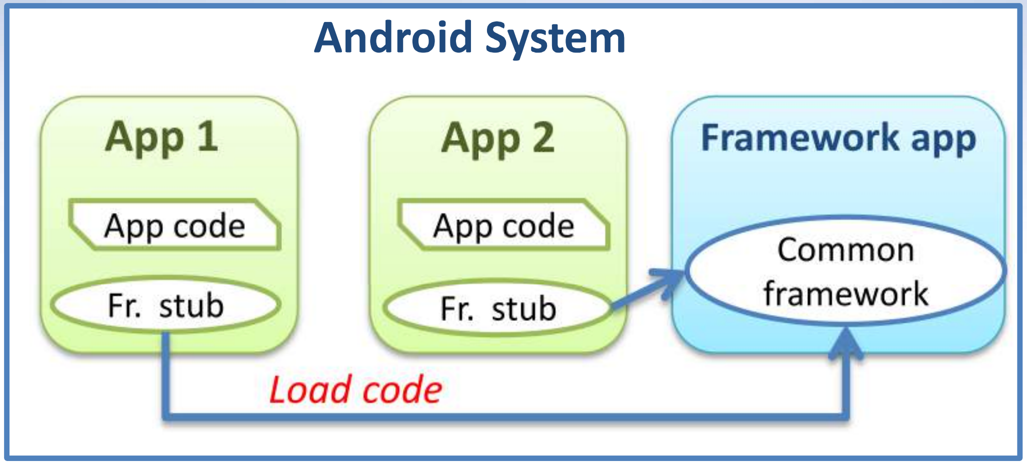
Battery life

CPU usage

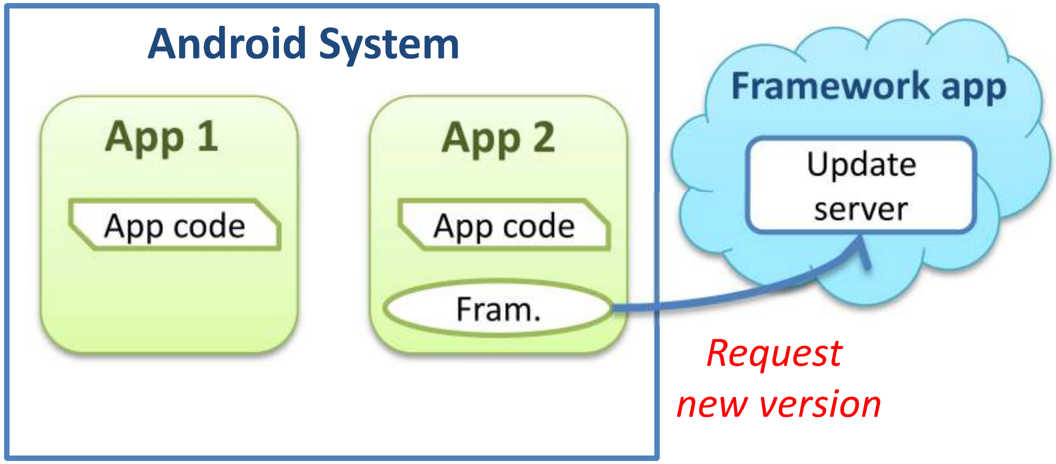
Memory consumption

Attack scenario - Android

1. Benign applications that loads, for benign reasons, additional code that can be replaced with malicious ones by the attacker.

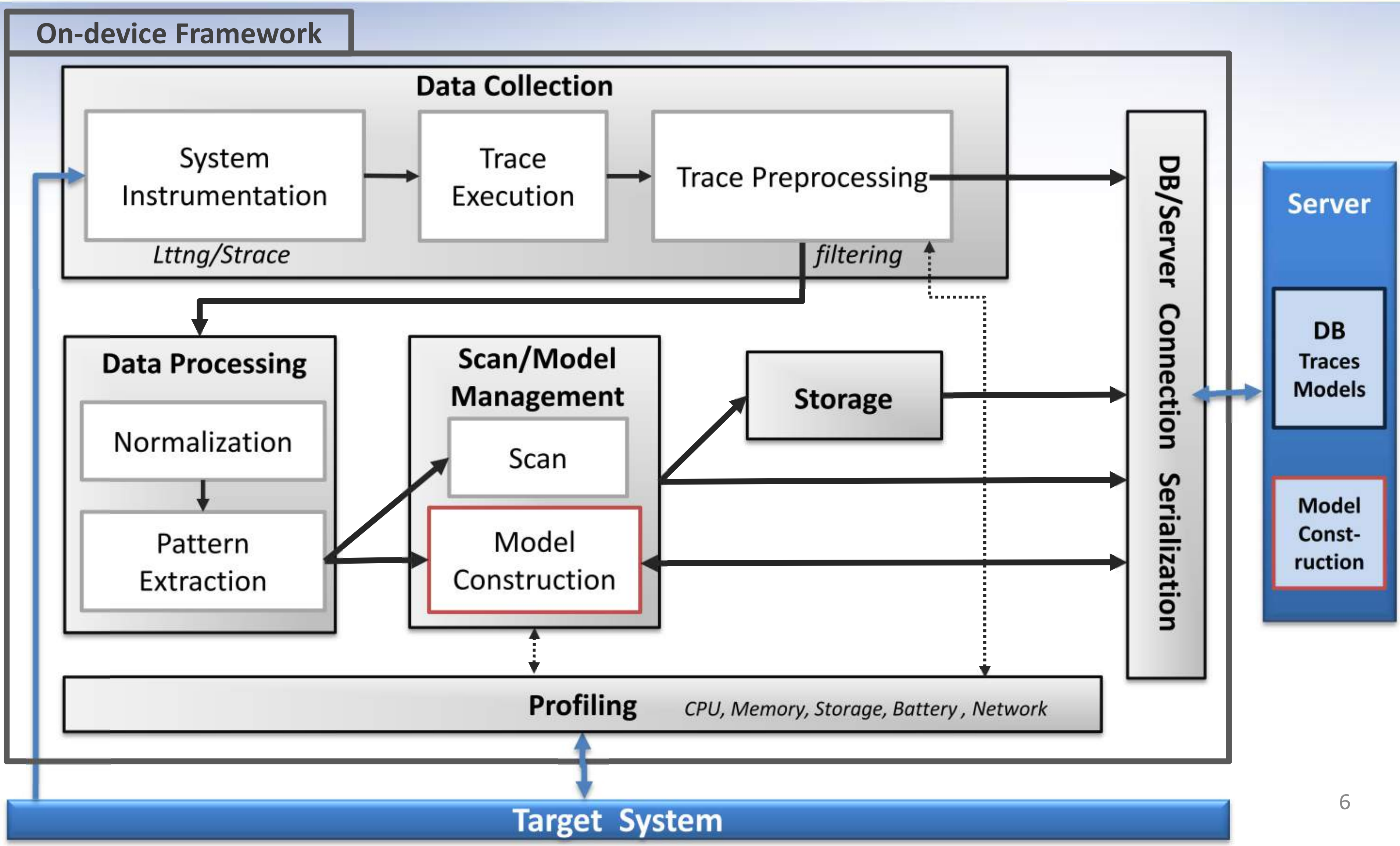


2. Malicious application that does not contain initially any clearly malicious code, but downloads additional faked code after being installed on a device.



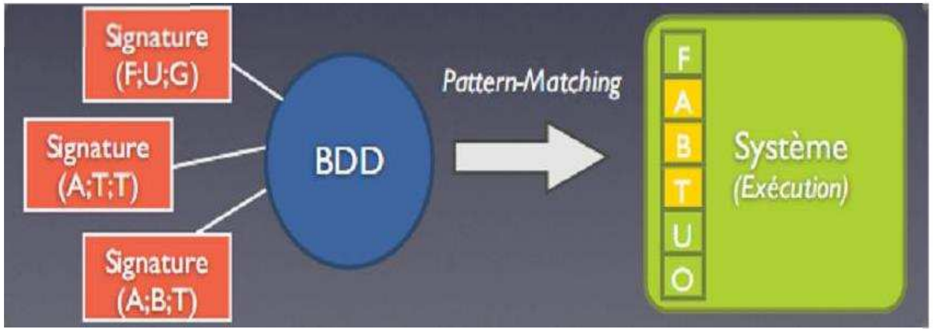
Source: Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., & Vigna, G. (2014, February). Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS).

Framework's Architecture



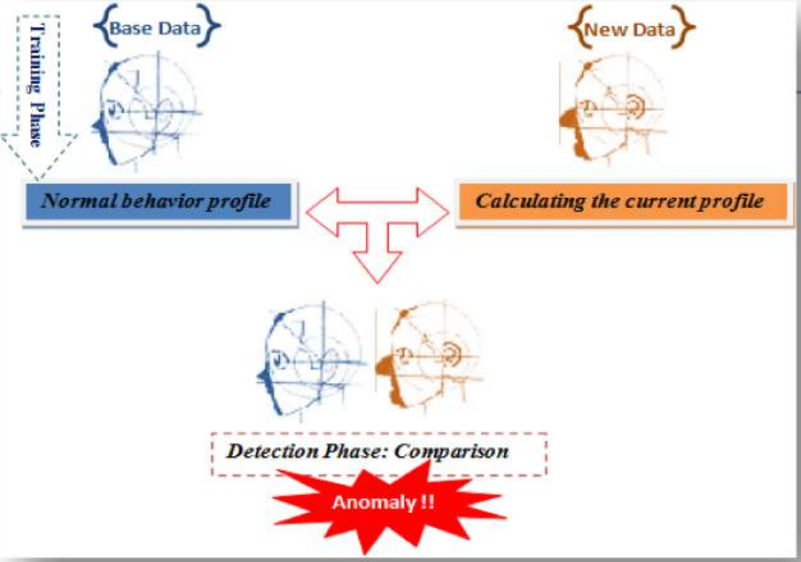
Intrusion Detection Techniques

	Signature-Based	Anomaly-Based
?	Looking for "known patterns" of specific malware activity (list of stored signature for each malware)	Learning phase: establishes a base of knowledge about "normal" behavior. Detection phase: once a behavior is too different from training data, it is considered abnormal.
+	Low false positive rate Very accurate and Fast	Can detect both known and unknown malwares Accuracy increases as increasing training data
-	Can only detect known intrusions Required memory budget : varying numbers of signatures. DB must be constantly updated	High false positive rate Slow

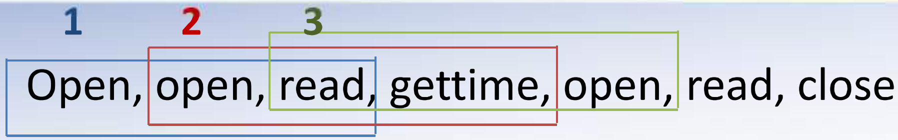


Fast evolution of signatures database

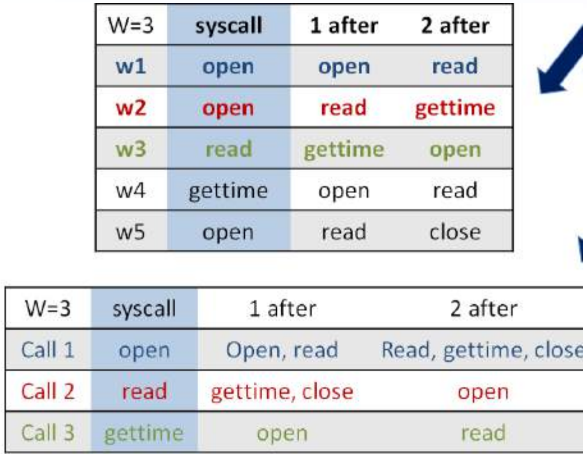
→ never feet memory of small-scale systems



Model Construction



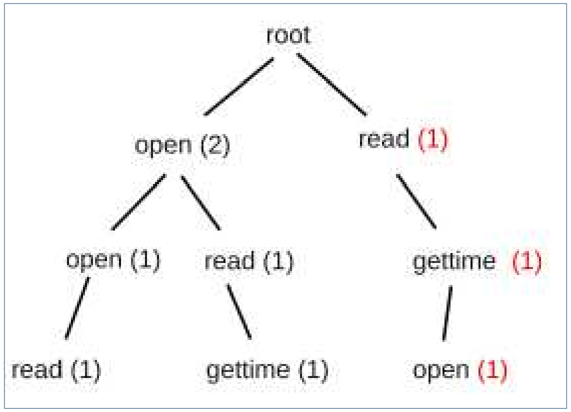
Lookahead pairs



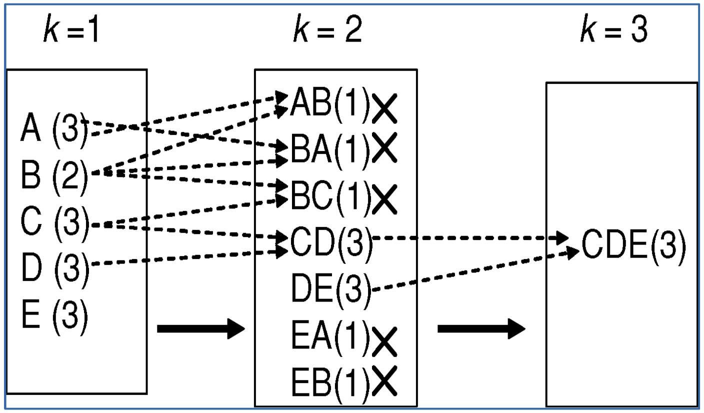
W=3	syscall	1 after	2 after
w1	open	open	read
w2	open	read	gettime
w3	read	gettime	open
w4	gettime	open	read
w5	open	read	close

W=3	syscall	1 after	2 after
Call 1	open	Open, read	Read, gettime, close
Call 2	read	gettime, close	open
Call 3	gettime	open	read

N-gram Tree

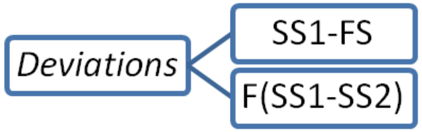


Varied-length N-grams



$$f(p_{k+1}) > \alpha \min(f(r_k), f(q_k))$$

Finite State Machines



$$\begin{aligned}
&\text{if } (Z_N(\boldsymbol{SS1}) < -2) &&\rightarrow P{=}0\\
&\text{if } (Z_{Ab}(\boldsymbol{SS2} \text{ or } \boldsymbol{FS}) < -2) &&\rightarrow P{=}1\\
&\text{if } (Z_N(\boldsymbol{SS2}) < -2) &&\rightarrow P{=}1
\end{aligned}$$

$$P(Anomaly)=\frac{(1 + SD_N(\boldsymbol{SS1})) * (\alpha * Z_N(\boldsymbol{SS1}) + 3)}{(Br(\boldsymbol{SS1})) * (Z_{Ab}(\boldsymbol{SS2} \text{ or } \boldsymbol{FS}) + 3) * (Z_N(\boldsymbol{SS2}) + 3)}$$

Experimental results

-Dataset-

- **Angry birds space**
  - Normal version: 1.1.0
  - Malicious version: 1.1.2
- **Candy Star**
  - Normal version: 1.0.3
  - Malicious version: 1.0.2
- **Ninja Chicken**
  - Normal version: 1.4.8
  - Malicious version: 1.4.5

Angry birds space

Loads additional code to locate the device, steal contacts and send text messages.

Candy Star

Loads a shared library and DEX file

Read/modify/delete the contents of the SD card.

Ninja Chicken

Loads a shared library and DEX file

Read/modify/delete the contents of the SD card.

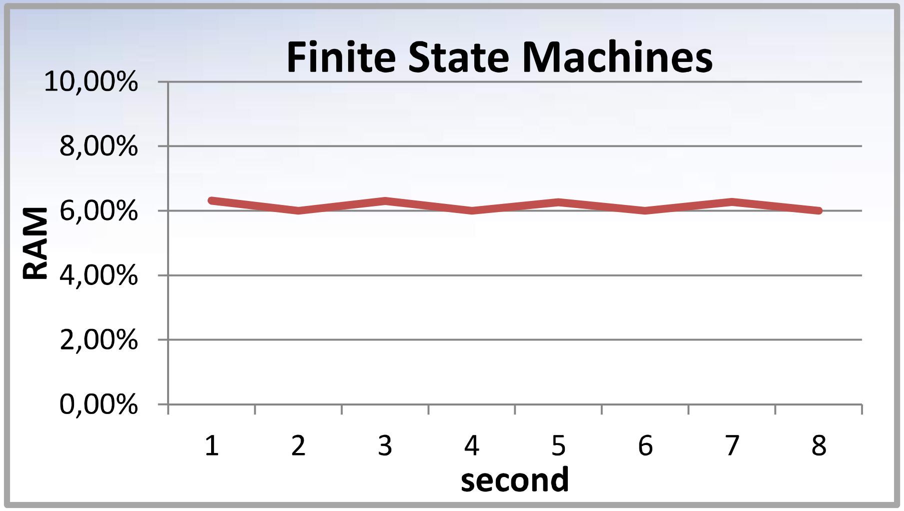
Read phone state + identify running applications.

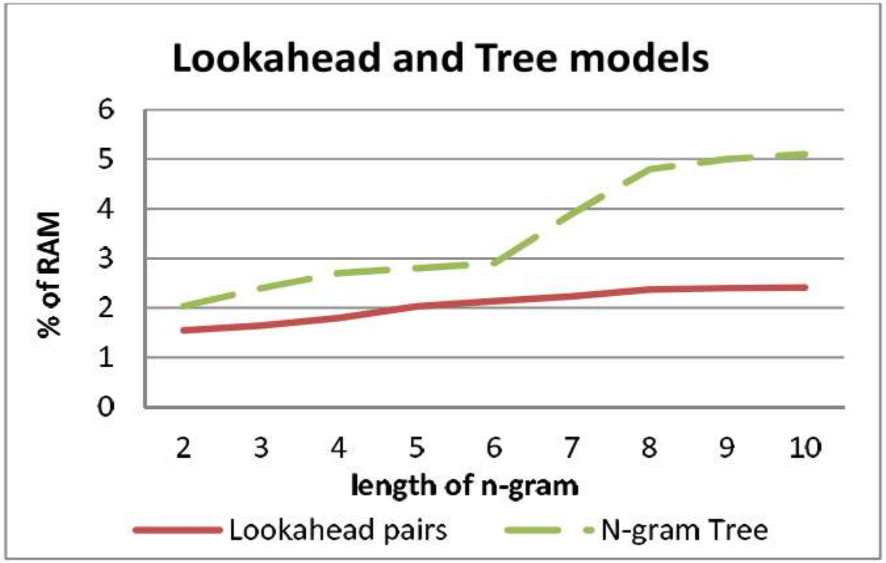
http://contagiominidump.blogspot.fr/
https://www.virustotal.com/intelligence/

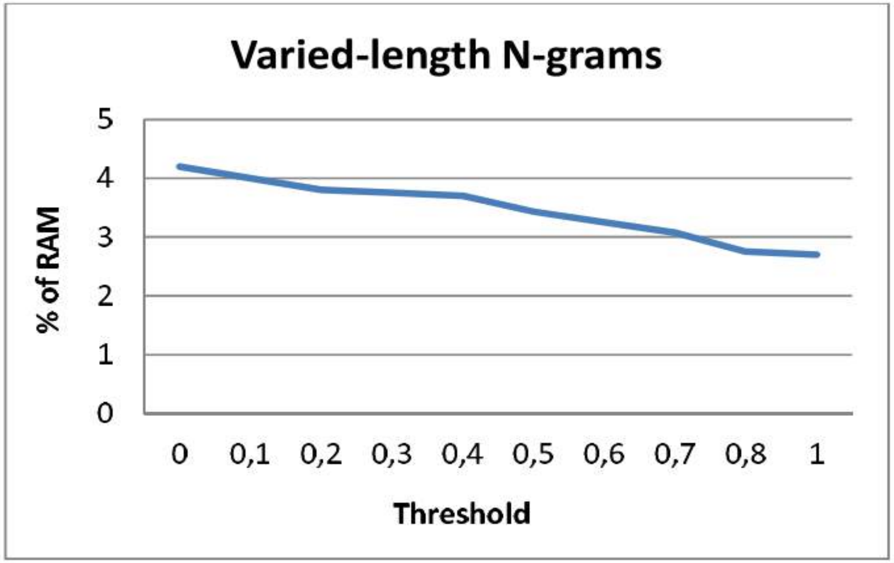
Experimental results

-Creating Normal profile-

- **RAM Overhead**



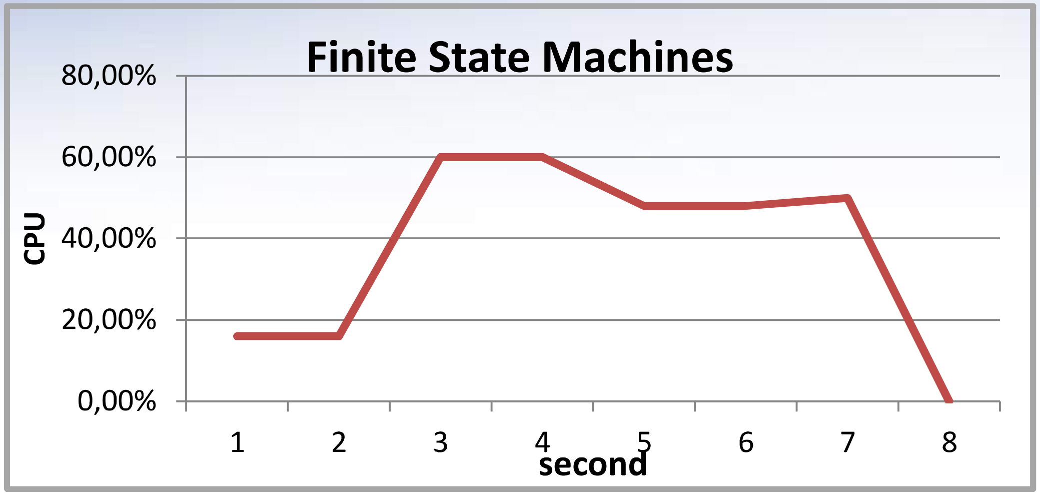


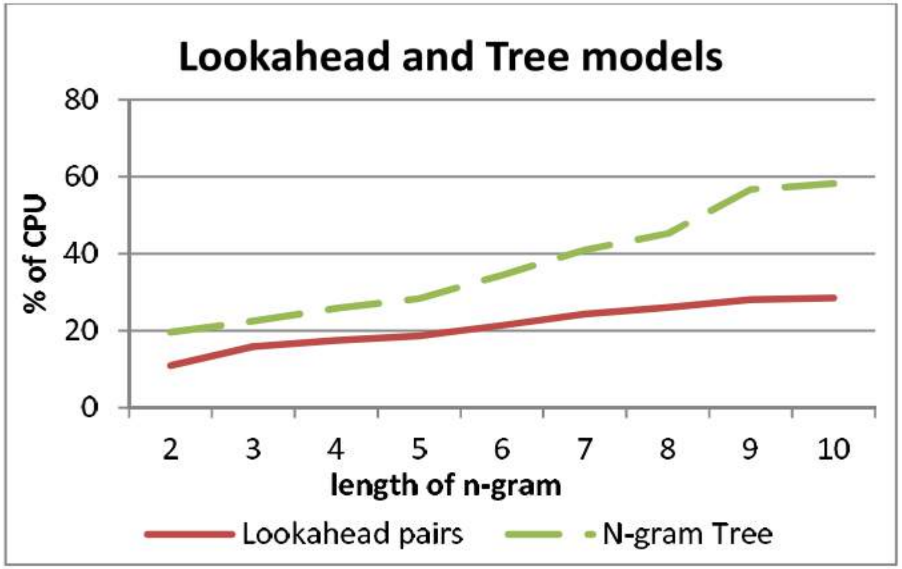


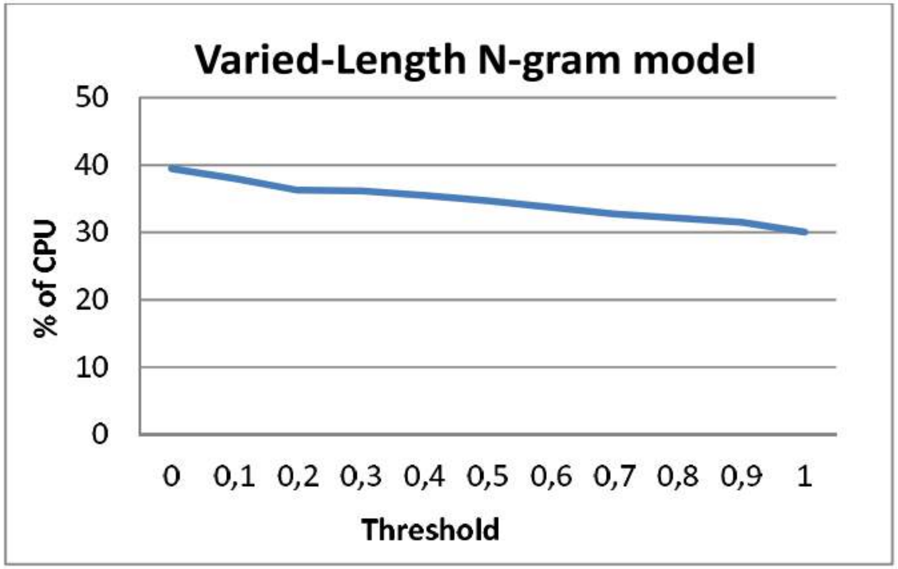
Experimental results

-Creating Normal profile-

- **CPU Overhead**



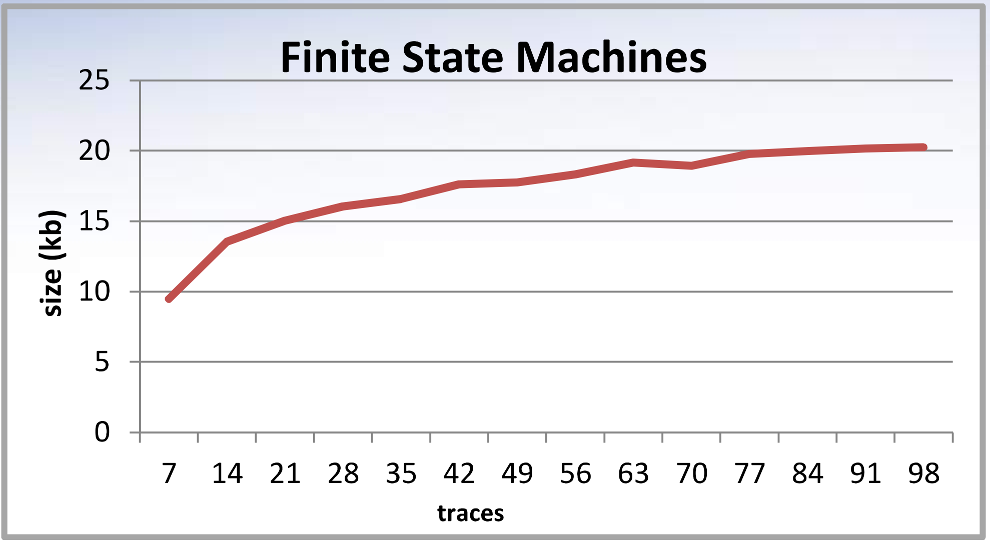


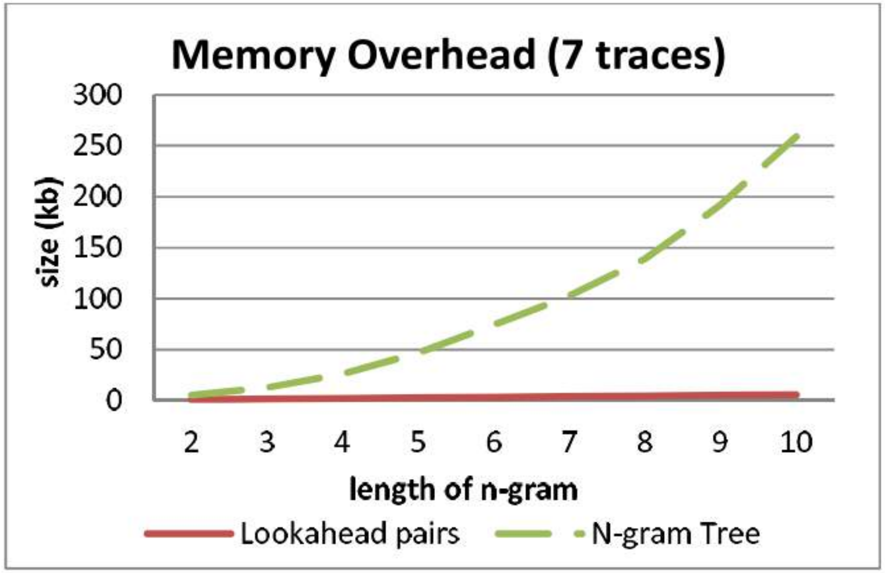


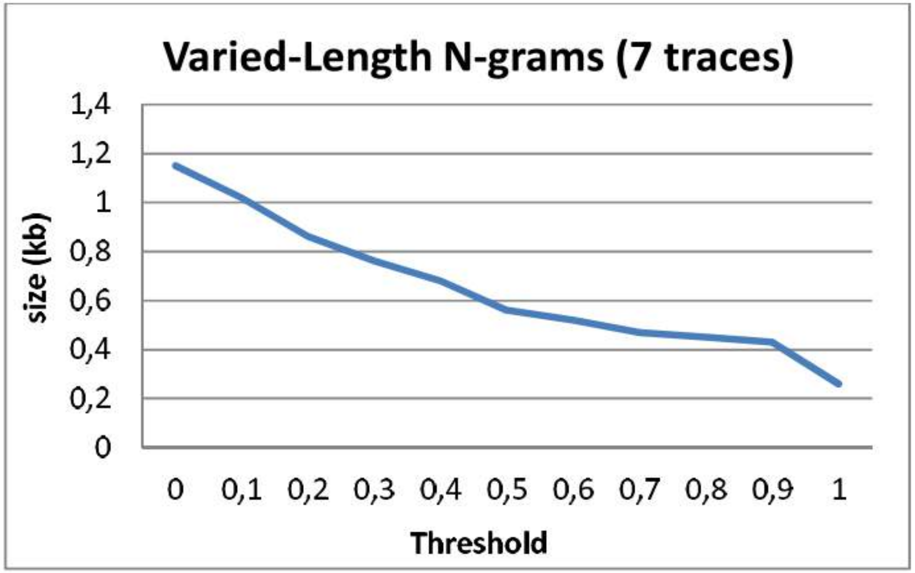
Experimental results

-Creating Normal profile-

- **Storage Overhead**





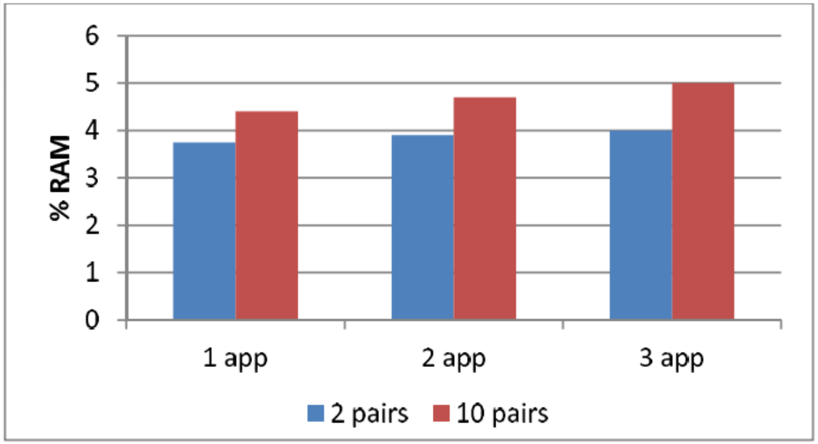


Experimental results

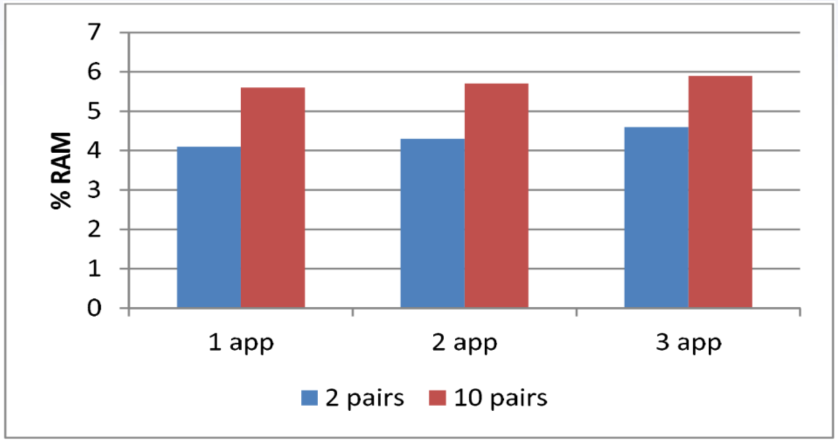
-Scanning 1, 2 and 3 applications in parallel-

- **RAM Overhead**

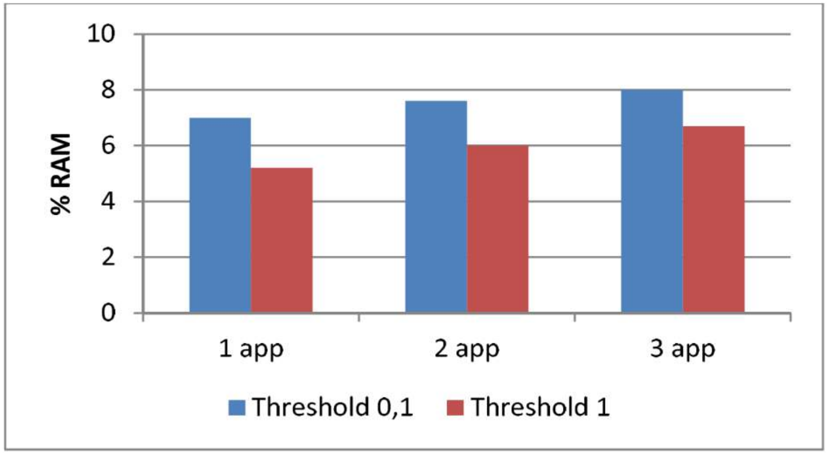
Lookahead



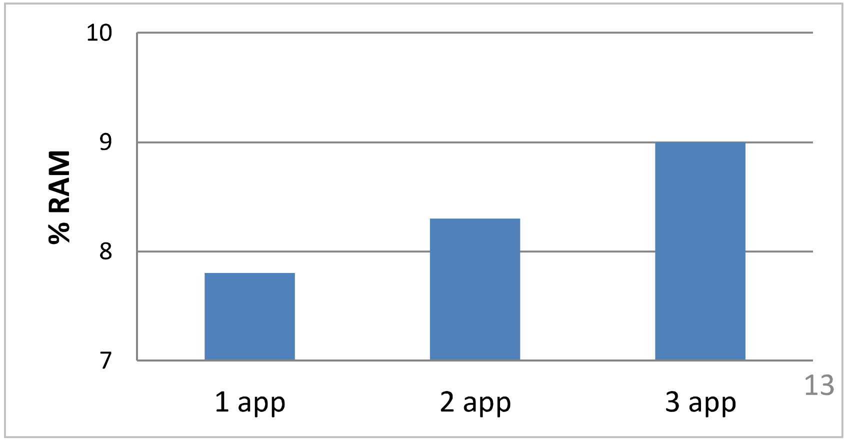
N-Gram Tree



VL N-gram



Finite State Machines

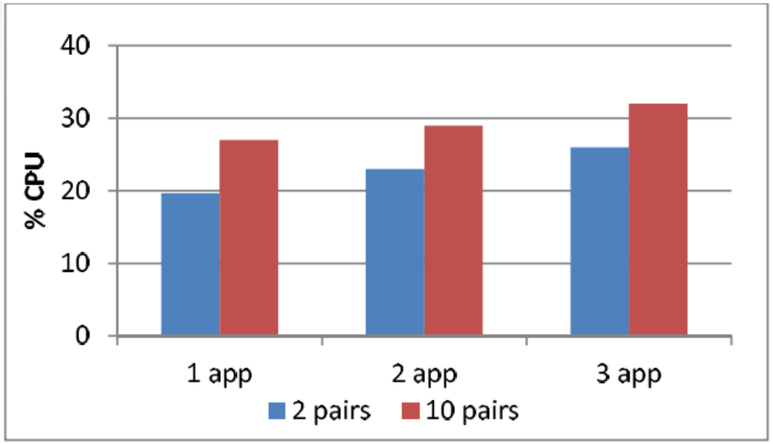


Experimental results

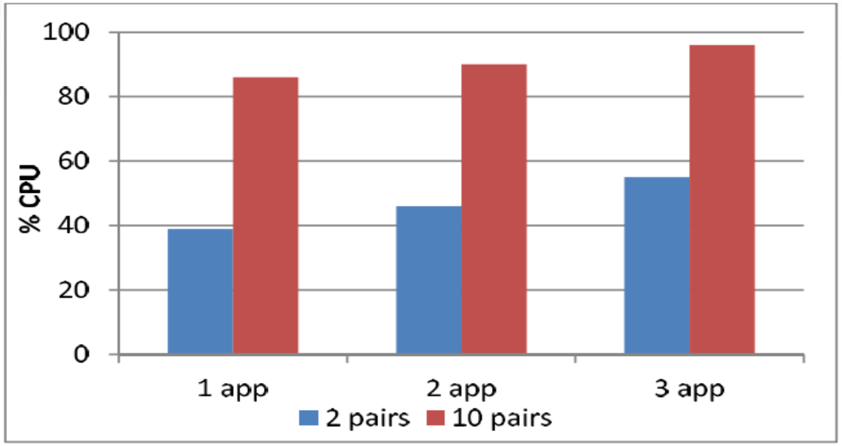
-Scanning 1, 2 and 3 applications in parallel-

- **CPU Overhead**

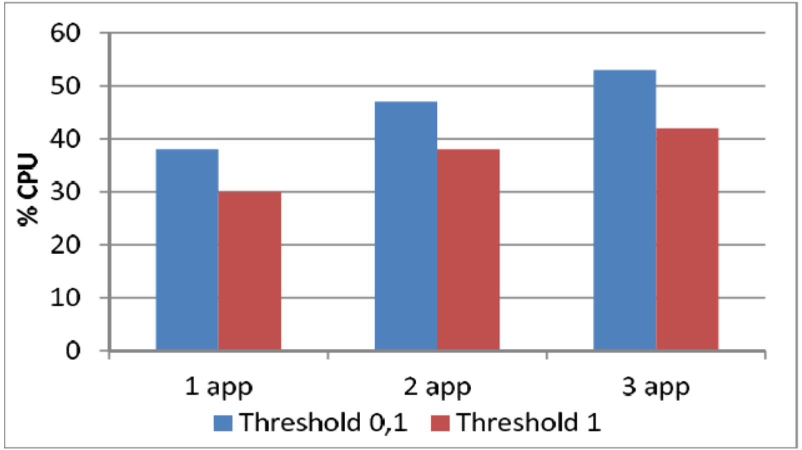
Lookahead



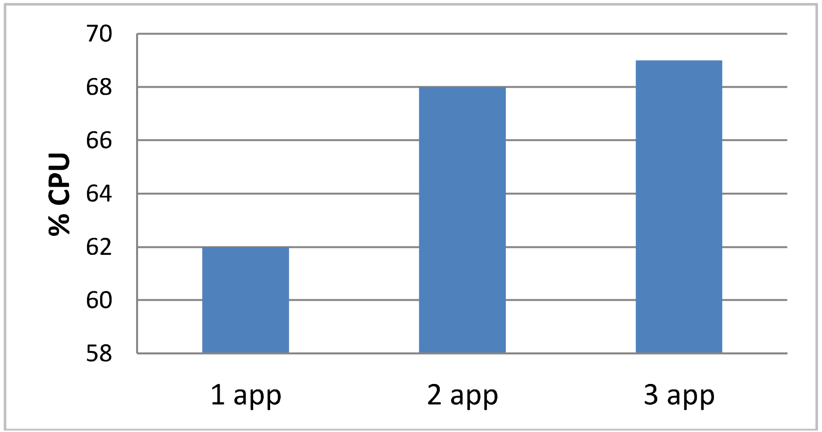
N-Gram Tree



VL N-gram



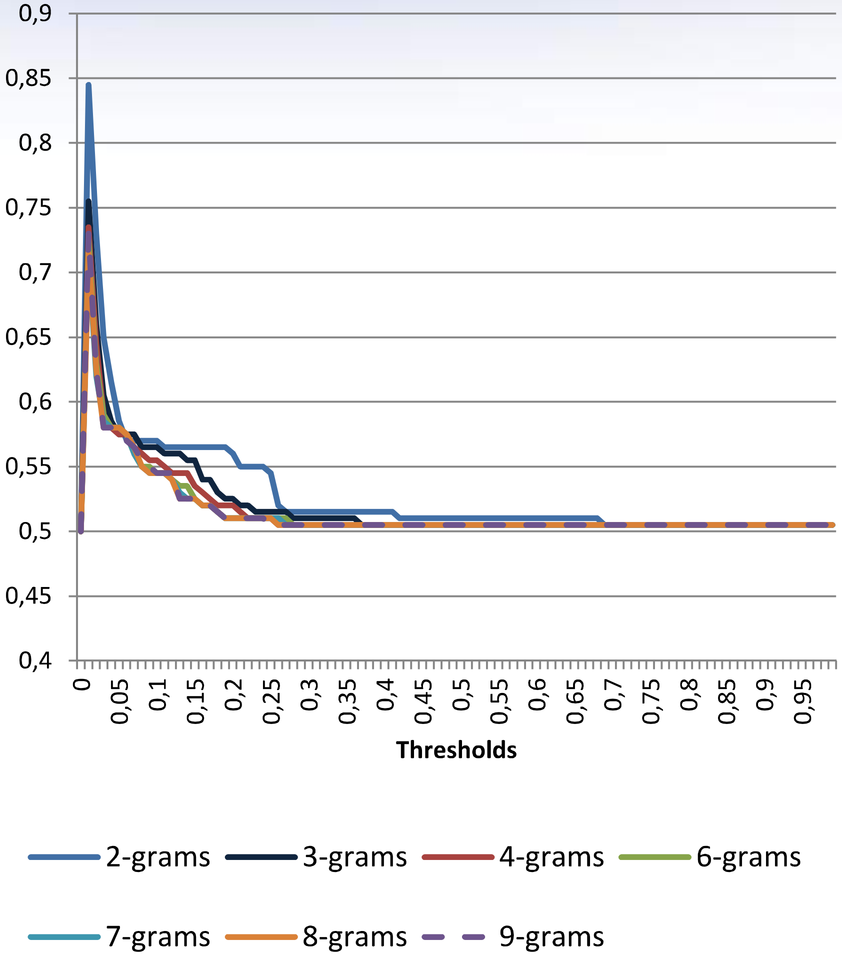
Finite State Machines



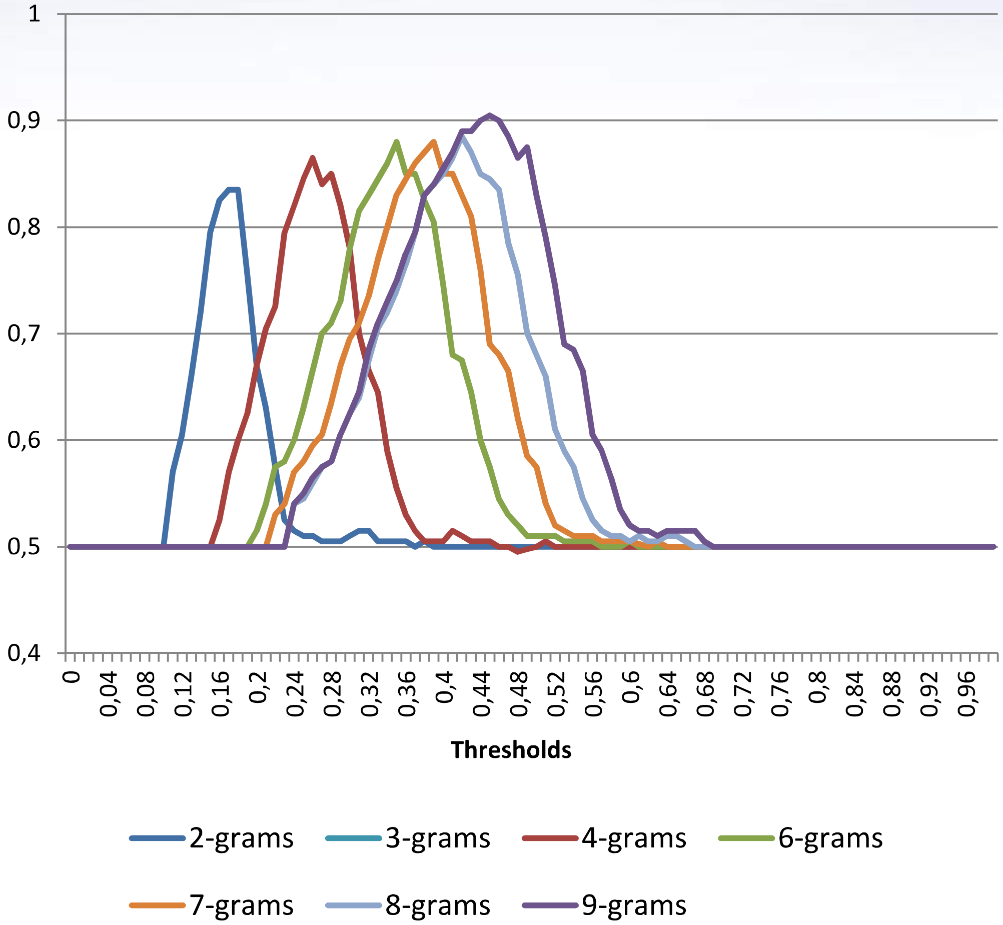
Experimental results

-Accuracy= (TP+TN)/(TP+TN+FP+FN)-

Lookahead

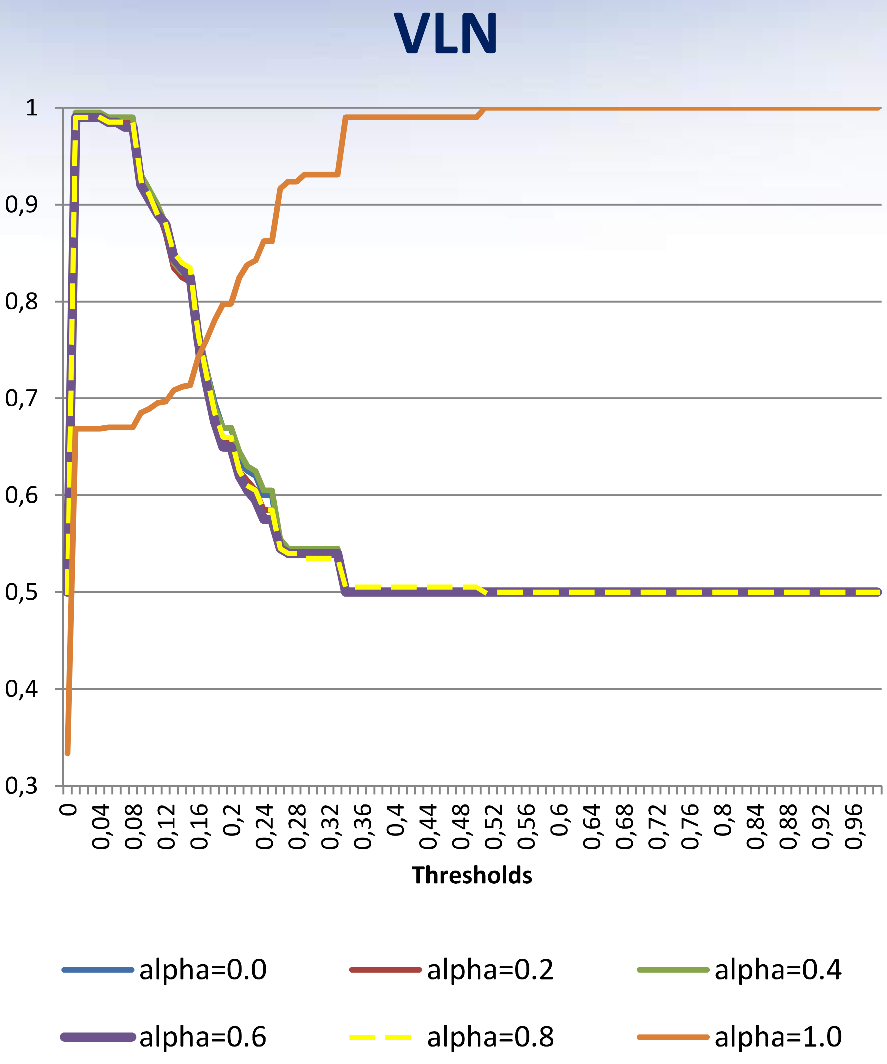


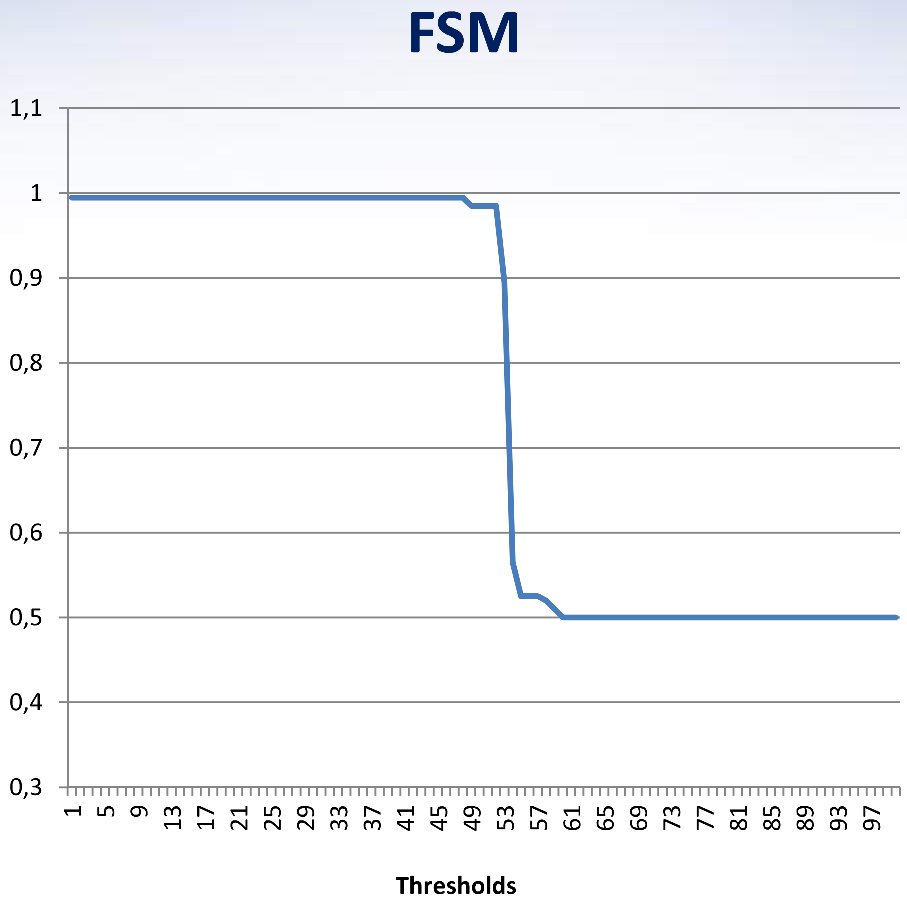
Tree



Experimental results

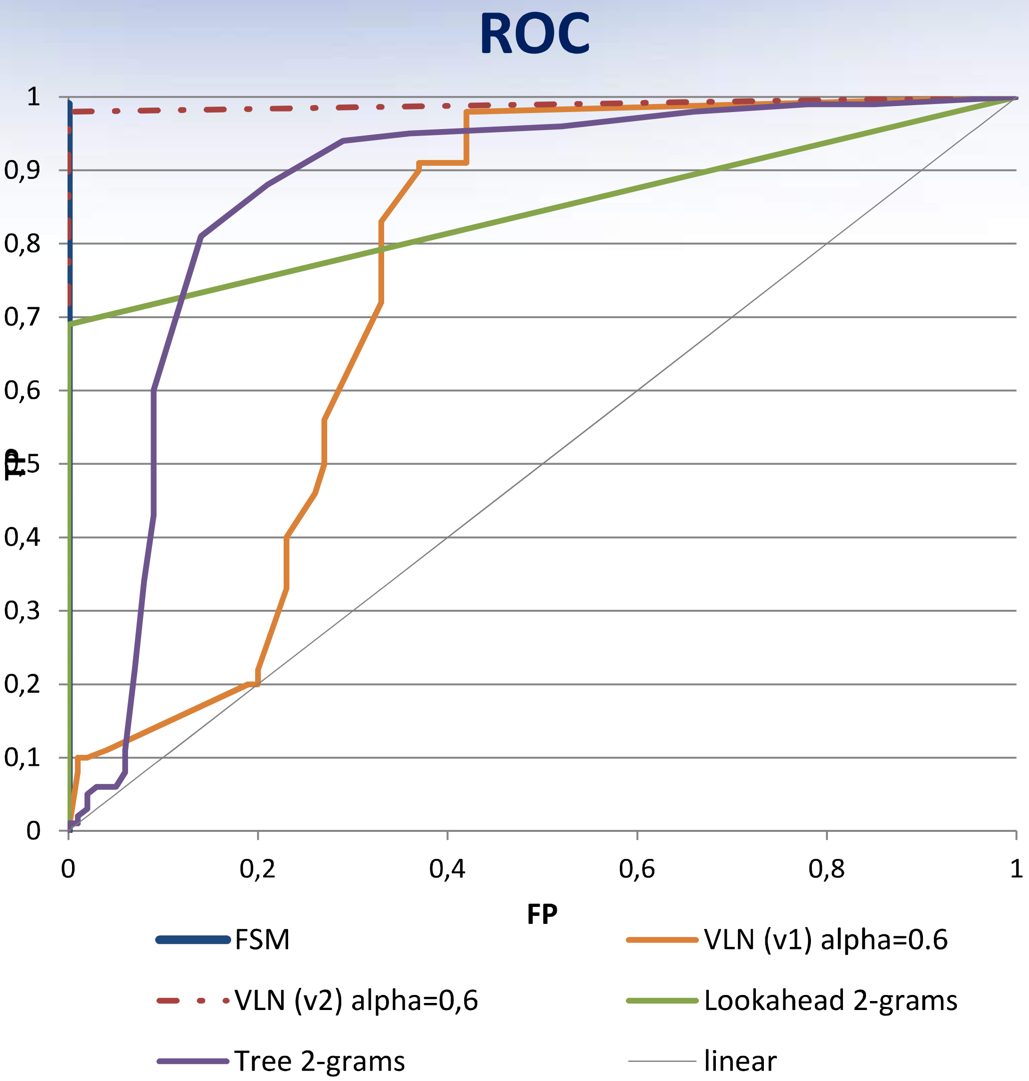
-Accuracy= (TP+TN)/(TP+TN+FP+FN)-

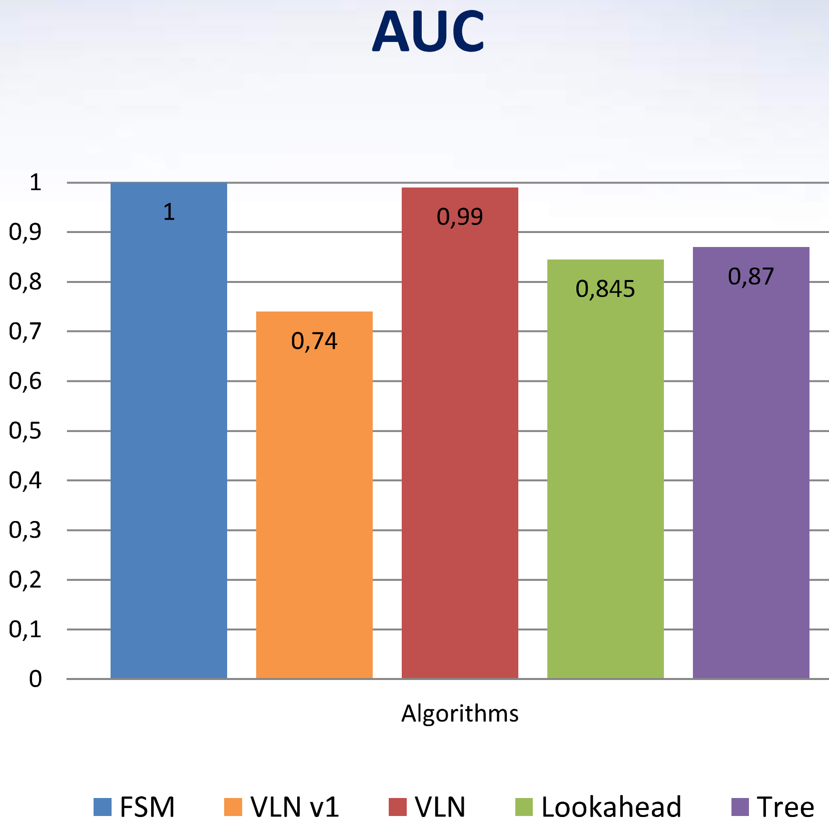




Experimental results

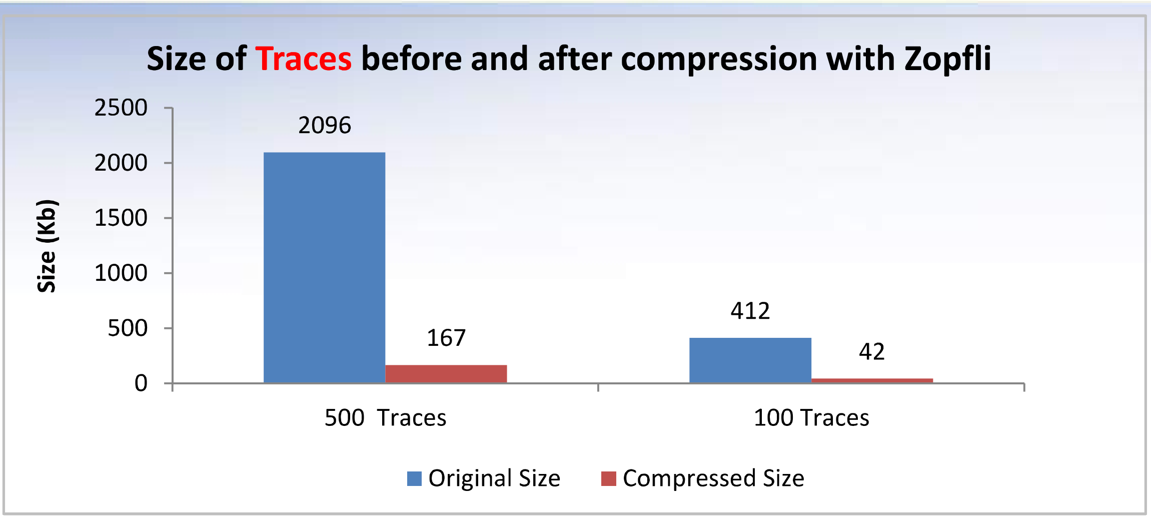
-ROC curves-

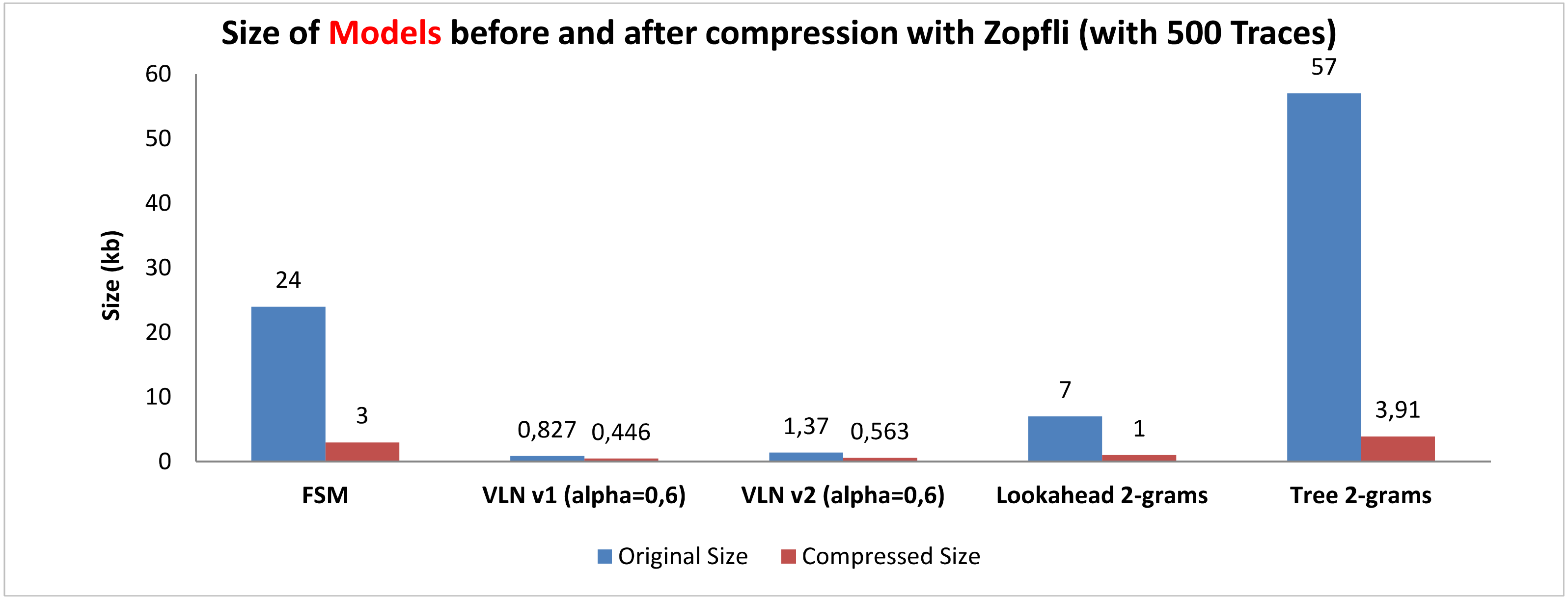




Storage:

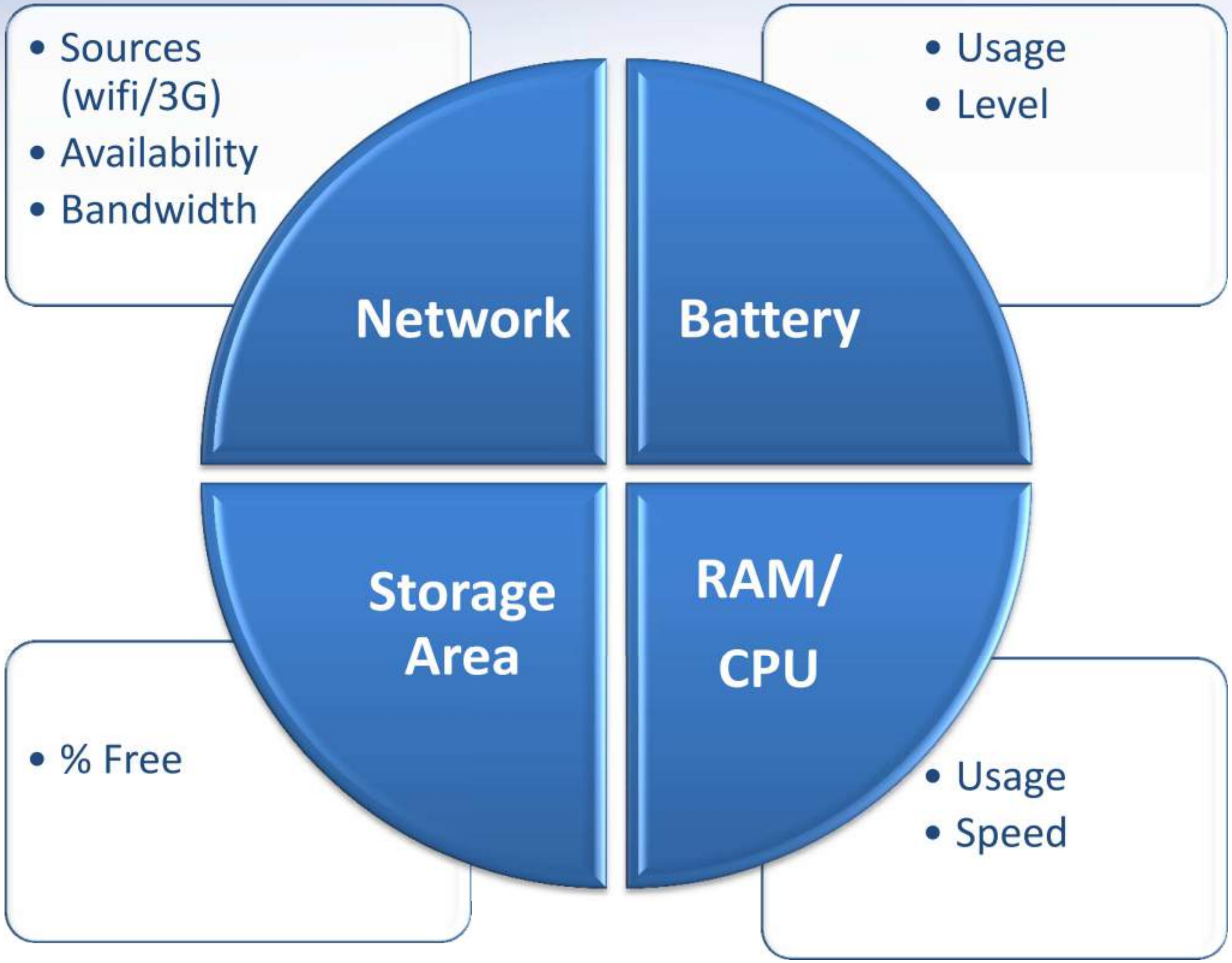
-Zopfli compression algorithm-





Profiling:

-Profiling parameters-



Network
- Sources (wifi/3G)
- Availability
- Bandwidth

Battery
- Usage
- Level

Storage Area
- % Free

RAM/ CPU
- Usage
- Speed

Profiling:

-Trace management-

network interface	Free memory space	Decisions
$B_{max} > \alpha_B$	----------------	- Send current and compressed traces to the server - Update model
$B_{max} < \alpha_B$	$S_{free} < \alpha_S$	- Increase the threshold of the model “Varied-length N-grams” in order to reduce the size of the model to be saved. - Decrease the size of n-grams (window size) for lookahead and tree models.
$B_{max} < \alpha_B$	$S_{free} > \alpha_S$	- Save traces in the device - Compress the traces when they reach a certain number (the compression is slow but it saves more space and reduce the cost of data transfer and battery use)

Profiling:

-Model and Scan management-

Battery	RAM	CPU	Decisions
$B > \alpha_{Batt}$	$R < \alpha_{RAM}$	$C < \alpha_{CPU}$	**Scan using more than one model.** **Maximize accuracy :** Increase the size of n-grams (window) for Tree model, Decrease the threshold of the model "Varied-length N-grams."
$B > \alpha_{Batt}$	$R > \alpha_{RAM}$	$C < \alpha_{CPU}$	**Scan using just one model.** **Minimize the amount of data being processed:** Decrease the size of n-grams (window) for lookahead and tree models. Increase the threshold of the model "Varied-length N-grams." **Decrease depth analysis with the model tree:** During scanning with tree model, handles only a part of the tree n-gram (a sub-tree)
$B > \alpha_{Batt}$	-------------	$C > \alpha_{CPU}$	**Scan using just one model.** **Minimize the amount of data being processed:** Decrease the size of n-grams (window) for lookahead and tree models. Increase the threshold of the model "Varied-length N-grams." **Decrease depth analysis with the model tree:** During scanning with tree model, handles only a part of the tree n-gram (a sub-tree). **Minimize the number of treatments** Do not send traces to the server Do not compress the traces
$B < \alpha_{Batt}$	---------------	--------------	**Scan only** **Decrease depth analysis with the model tree:** During scanning with tree model, handles only a part of the tree n-gram (a sub-tree).

Project Summary

Designing a Trade-Off Between Usability and Security:

- Platform: Android
- Security module:
  - □ Data Collection → system calls
  - □ Data Processing
  - □ Scan/Model Management
    - Signature based detection VS Anomaly based detection
    - Anomaly based algorithms : Lookahead, Tree, Varied-length N-grams, FSM
- Storage module:
  - □ Zopfli compression algorithm
- Profiling module:
  - □ Profiling parameters : Network status , Battery, RAM/CPU, Storage

What’s Next ?

Designing a Trade-Off Between Usability and Security:

- Platform: Android, **Linux**
- Security module:
  - Data Collection → system calls, **LTTng**
  - Data Processing
  - Scan/Model Management
    - Signature based detection VS Anomaly based detection
    - Anomaly based algorithms : Lookahead, Tree, Varied-length N-grams, FSM, **other algorithms**
- Storage module:
  - Zopfli compression algorithm
- Profiling module:
  - Profiling parameters : Network status , Battery, RAM/CPU, Storage
  - **Dynamic decision maker**
    - Monitoring system behavior and selecting the best anomaly detection Algorithm.

Any
Questions