
Online Malware Detection in Cloud Auto-Scaling Systems using Performance Metrics

by

Mahmoud Abdelsalam

Committee:

Prof. Ravi Sandhu: (Advisor)

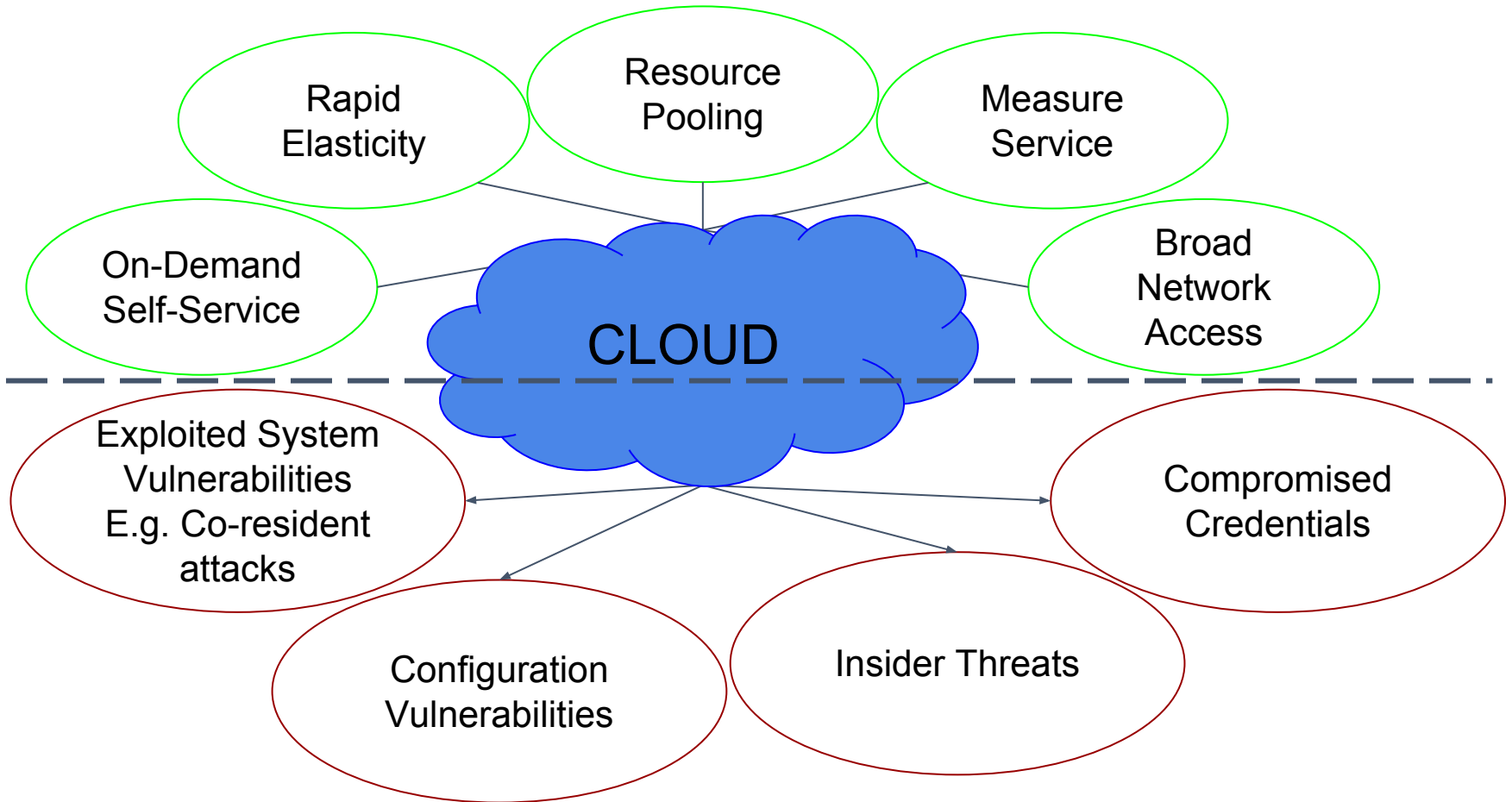
Dr. Ram Krishnan: (Co-advisor)

Dr. Matthew Gibson

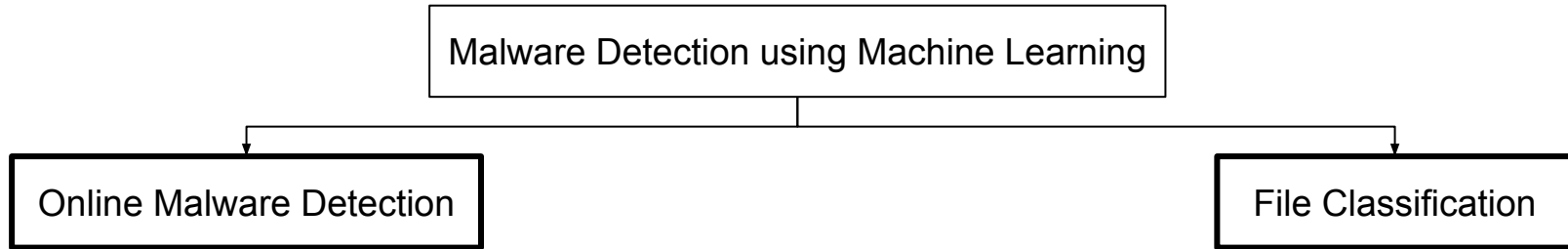
Dr. Murtuza Jadliwala

Dr. Gregory White

November 29, 2018



*Can we leverage cloud characteristics for online malware detection?
"Auto-Scaling"*

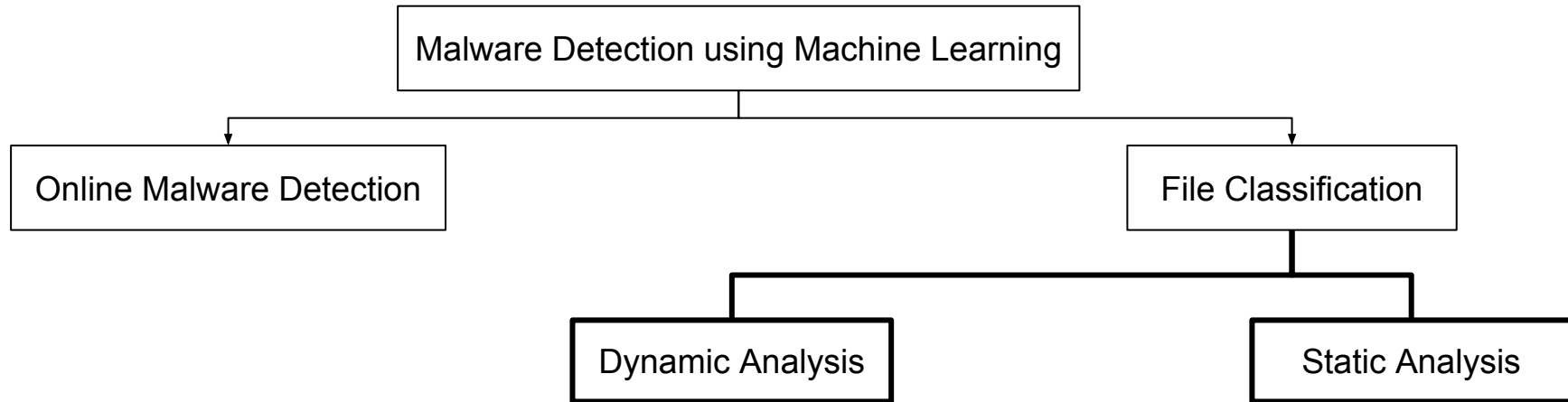


1. File classification:

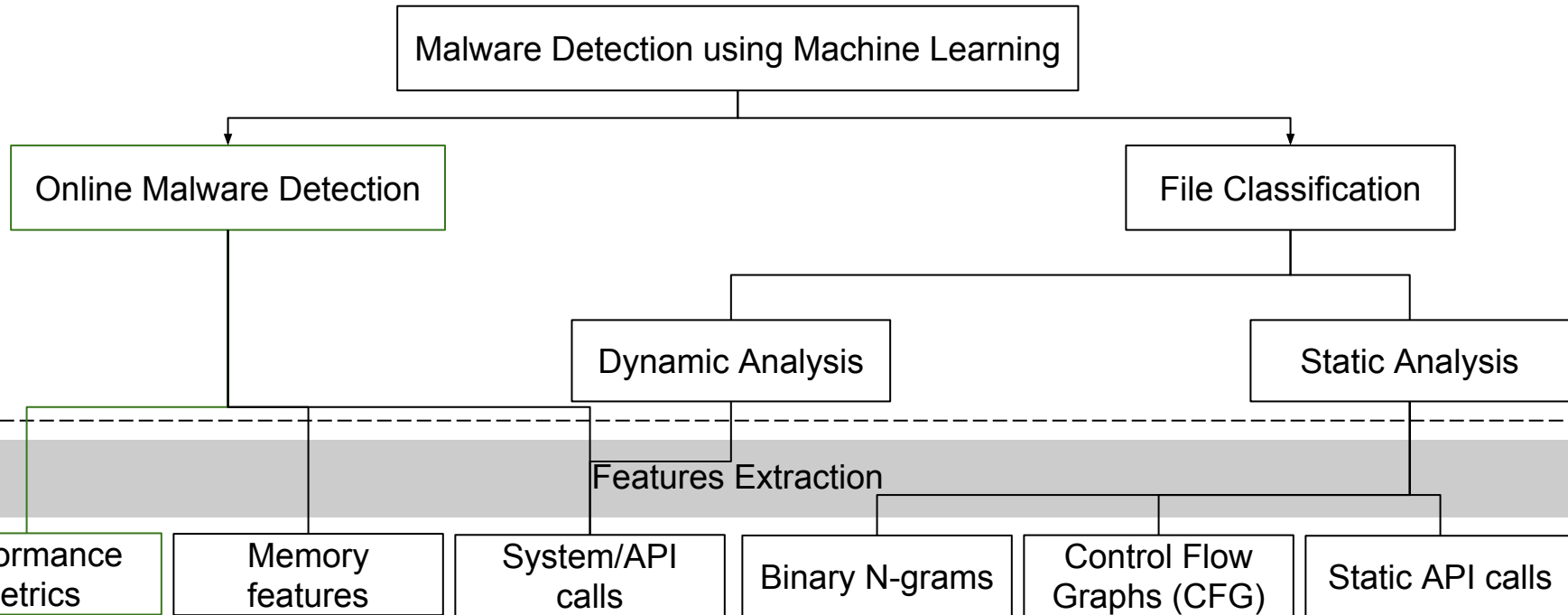
- Given a file/executable, classify if it's a malware or not by running it and observing its behavior.
- You have a file as a suspect.
- You don't keep monitoring them once they are clean.

2. Online malware detection:

- Assume that the malware got into the system and is executing.
- You keep monitoring the system's behavior for malware detection.
- You don't just focus on a given file, but the entire system (processes).



- **Static Analysis**
 - No malware execution takes place.
 - It is the process of analyzing executables by examining their code without actually executing them.
- **Dynamic Analysis**
 - Malware execute, typically, in an isolated environment and information about their behavior is logged/monitored.

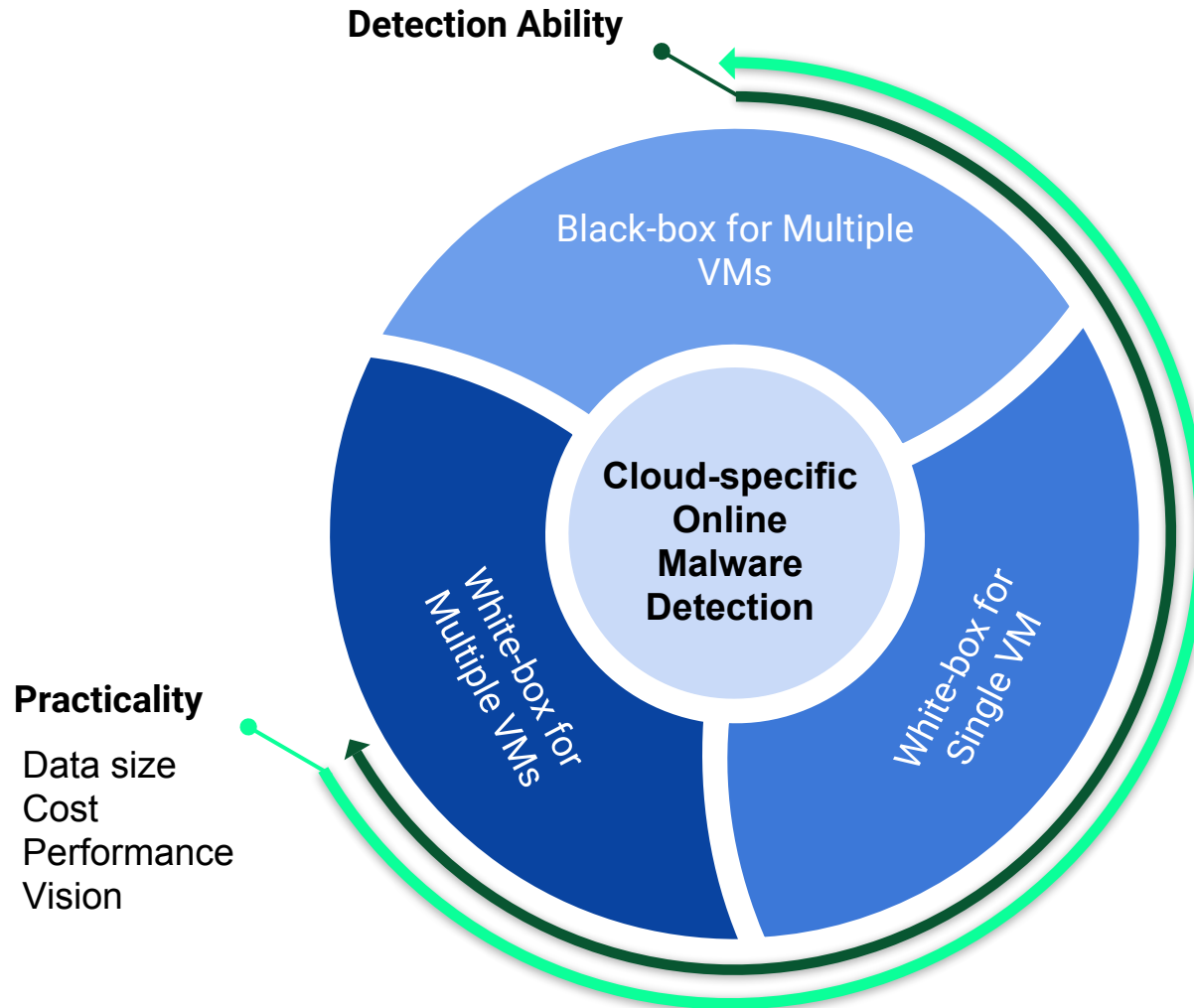


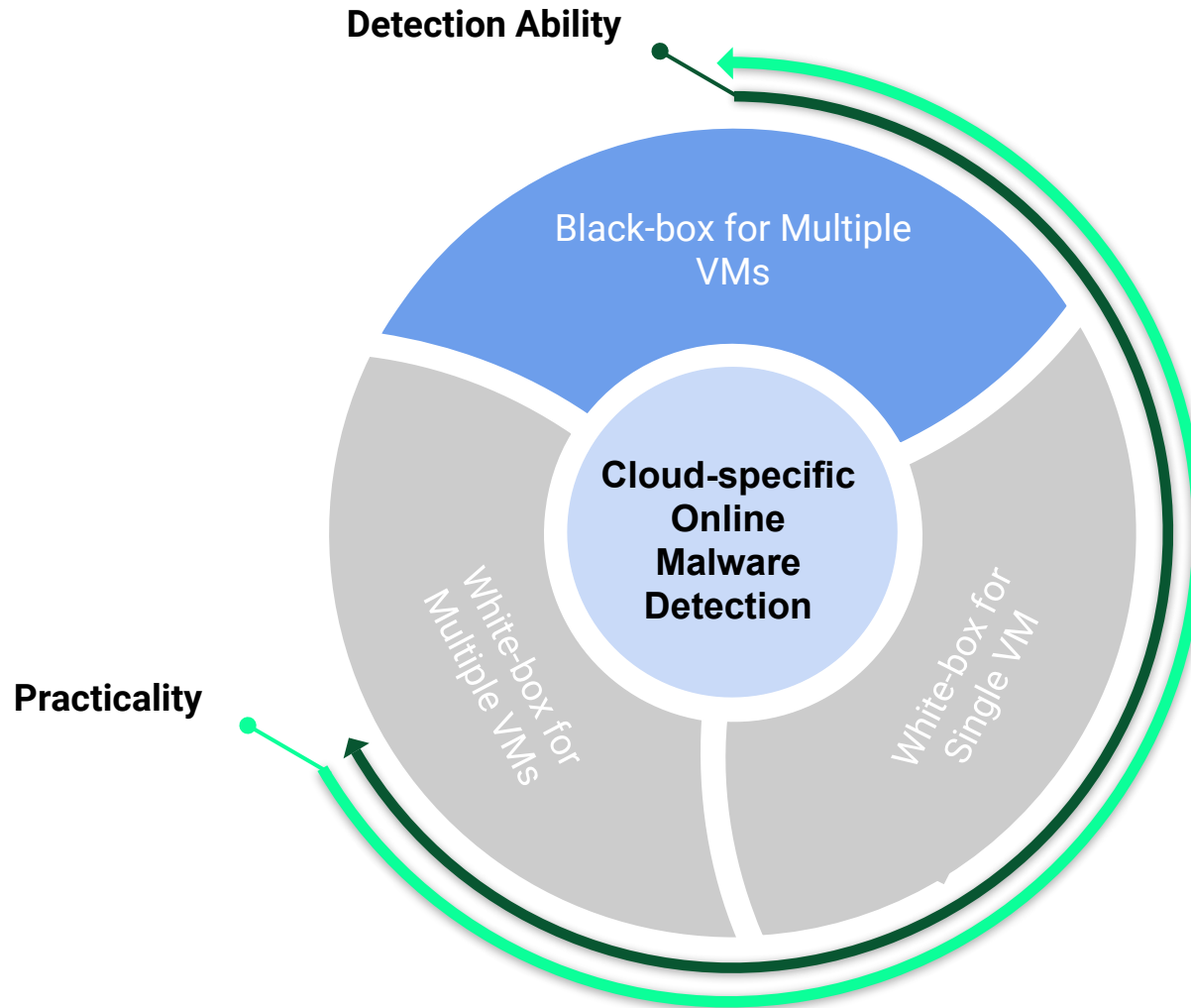
Cloud-specific falls under the “online malware detection” category.

Most, if not all, **cloud-specific** research:

- ✓ Restrict the selection of features to those that can only be fetched through the hypervisor.
- ✗ Leverage cloud characteristics for online malware detection.

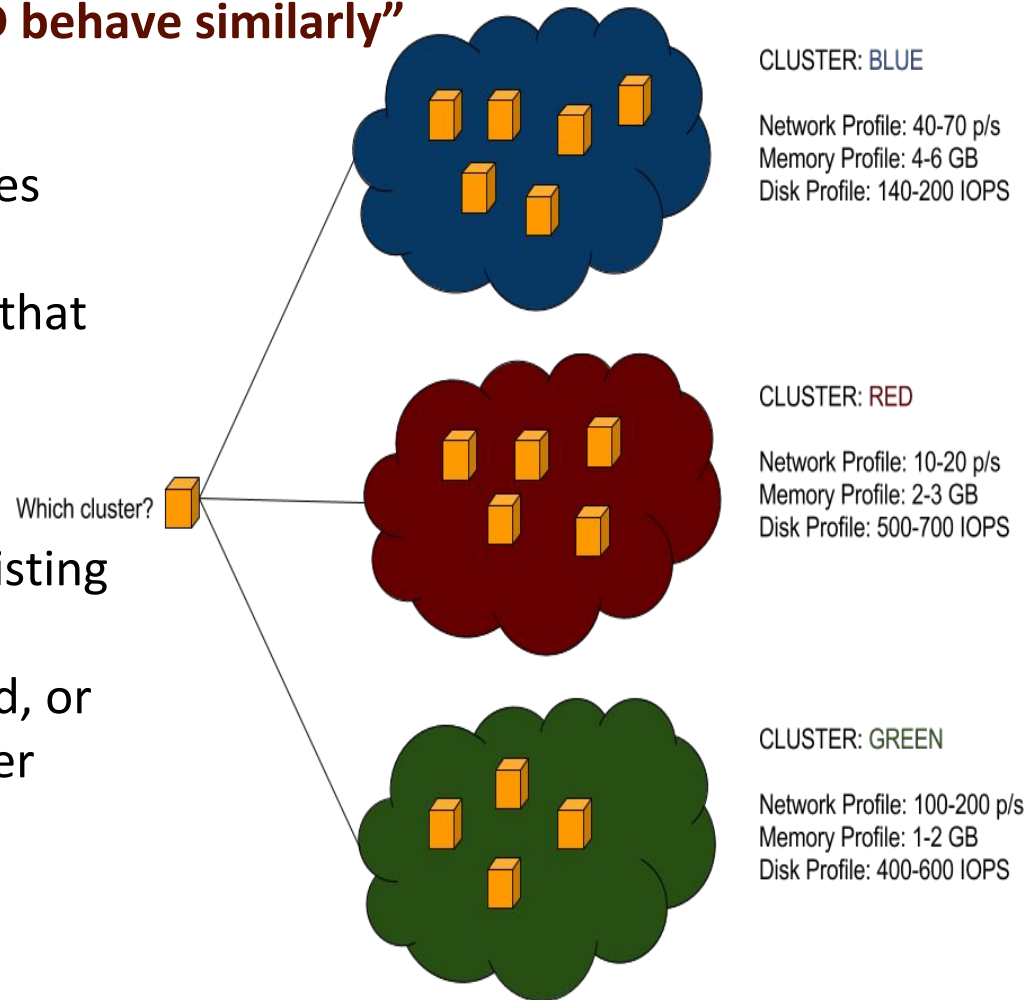
- **Problem statement:**
 - Malware will always find a way-in to infect cloud infrastructures.
 - Presence of a gap between malware prevention and malware detection methods.
 - Lack of *cloud-specific* online malware detection methods.
- **Thesis statement** - Cloud unique characteristic “auto-scaling” can effectively be utilized for online malware detection within a single-tenant’s virtual resources, in black-box and white-box granularity using performance metrics.





“VMs doing the same function SHOULD behave similarly”

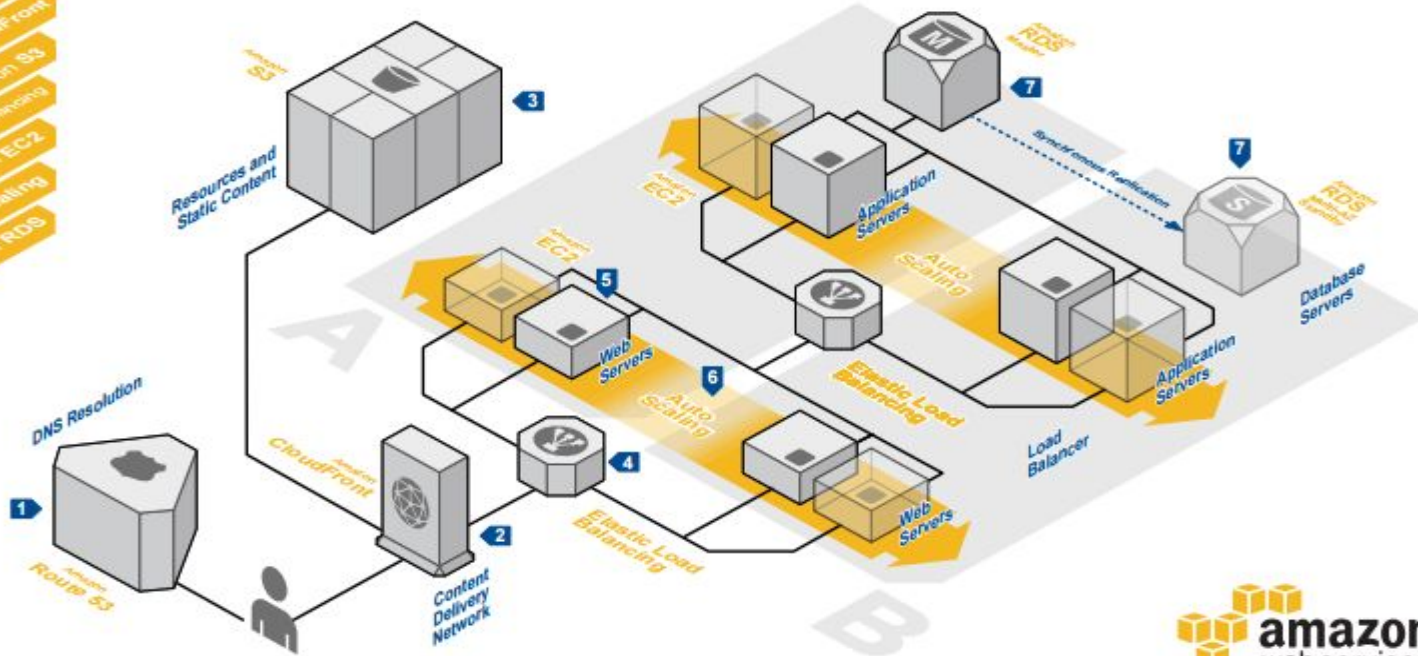
- Cluster VMs based on their attributes
- When a new VM gets created, “fit” that VM with existing cluster
 - If successful: good
 - If not: report anomaly
 - Admin force fits into an existing category -> consequently cluster profiles are updated, or
 - Admin creates a new cluster for this VM



- AWS Reference Architectures**
- Amazon Route 53
 - Amazon CloudFront
 - Amazon S3
 - Elastic Load Balancing
 - Amazon EC2
 - Auto Scaling
 - Amazon RDS

WEB APPLICATION HOSTING

Highly available and scalable web hosting can be complex and expensive. Dense peak periods and wild swings in traffic patterns result in low utilization of expensive hardware. Amazon Web Services provides the reliable, scalable, secure, and high-performance infrastructure required for web applications while enabling an elastic, scale-out and scale-down infrastructure to match IT costs in real time as customer traffic fluctuates.



System Overview

- The user's DNS requests are served by Amazon Route 53, a highly available Domain Name System (DNS) service. Network traffic is routed to infrastructure running in Amazon Web Services.
- Static, streaming, and dynamic content is delivered by Amazon CloudFront, a global network of edge locations. Requests are automatically routed to the nearest edge location, so content is delivered with the best possible performance.
- Resources and static content used by the web application are stored on Amazon Simple Storage Service (S3), a highly durable storage infrastructure designed for mission-critical and primary data storage.
- HTTP requests are first handled by Elastic Load Balancing, which automatically distributes incoming application traffic among multiple Amazon Elastic Compute Cloud (EC2) instances across Availability Zones (AZs). It enables even greater fault tolerance in your applications, seamlessly providing the amount of load balancing capacity needed in response to incoming application traffic.
- Web servers and application servers are deployed on Amazon EC2 instances. Most organizations will select an Amazon Machine Image (AMI) and then customize it to their needs. This custom AMI will then become the starting point for future web development.
- Web servers and application servers are deployed in an Auto Scaling group. Auto Scaling automatically adjusts your capacity up or down according to conditions you define. With Auto Scaling, you can ensure that the number of Amazon EC2 instances you're using increases seamlessly during demand spikes to maintain performance and decreases automatically during demand to minimize costs.
- To provide high availability, the relational database that contains application's data is hosted redundantly on a multi-AZ (multiple Availability Zones—zones A and B here) deployment of Amazon Relational Database Service (Amazon RDS).

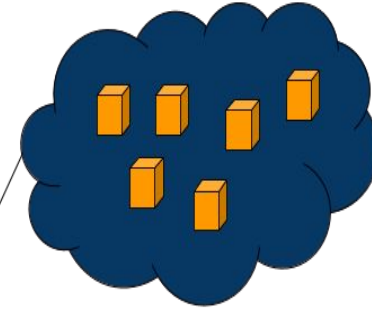
Metric	Description	Unit
CPU utilization	Average CPU utilization	%
Memory usage	Volume of RAM used by the VM from the amount of its allocated memory	MB
Memory resident	Volume of RAM used by the VM on the physical machine	MB
Disk read requests	Rate of disk read requests	rate/s
Disk write requests	Rate of disk write requests	rate/s
Disk read bytes	Rate of disk read bytes	rate/s
Disk write bytes	Rate of disk write bytes	rate/s
Network outgoing bytes	Rate of network outgoing bytes	rate/s
Network incoming bytes	Rate of network incoming bytes	rate/s

```

make initial guesses for means (centroids)
   $m_1, m_2, \dots, m_k$ 
set the counters  $n_1, n_2, \dots, n_k$  to zero
counter  $j$  //Number of current VMs
set stabilizingTime[1.. $j$ ] to  $y$  minutes
set assigningTime[1.. $j$ ] to  $z$  minutes
def VMClusters[1.. $k$ ] //VM  $i$  belongs to cluster
  [1.. $k$ ]
set VMClusterCounts[1.. $j$ ][1.. $k$ ] to zero
until interrupted
  get the next sample  $x$ 
  get VM  $v$  // $x$  sample belongs to VM  $v$ 
  if stabilizingTime[ $v$ ] > 0
    decrease stabilizingTime[ $v$ ]
    continue //get next sample
  end_if
  if  $v$  is assigned to a cluster
     $c = \text{VMClusters}[v]$ 
    if  $x$  is not anomaly
      increment  $n_c$ 
      replace  $m_c$  with  $m_c + (1/n_c) * (x - m_c)$ 
    else
      //Anomaly code here
      Report anomaly
    end_if
  end_if
  if  $v$  is not assigned to any cluster
    if  $m_i$  is closest to  $x$ 
      increment  $n_i$ 
      replace  $m_i$  with  $m_i + (1/n_i) * (x - m_i)$ 
    end_if
    //If time end assign it to a cluster
    if assigningTime[ $v$ ] <= 0
      set VMClusters[ $v$ ] to index of
         $\max(\text{VMClusterCounts}[v])$ 
    else
      if  $m_i$  is closest to  $x$ 
        increment VMClusterCounts[ $v$ ][ $i$ ]
      end_if
      decrease assigningTime[ $v$ ]
    end_if
  end_if
end_until

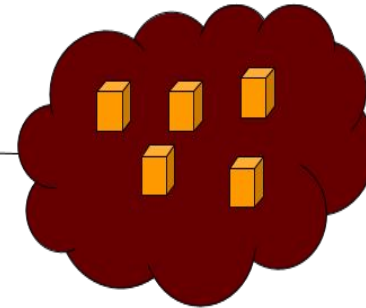
```

Which cluster?



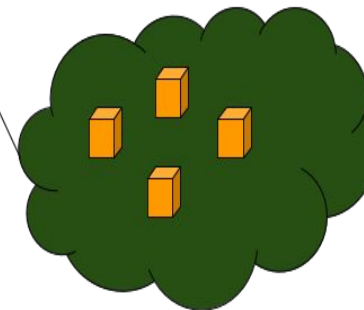
CLUSTER: BLUE

Network Profile: 40-70 p/s
Memory Profile: 4-6 GB
Disk Profile: 140-200 IOPS



CLUSTER: RED

Network Profile: 10-20 p/s
Memory Profile: 2-3 GB
Disk Profile: 500-700 IOPS



CLUSTER: GREEN

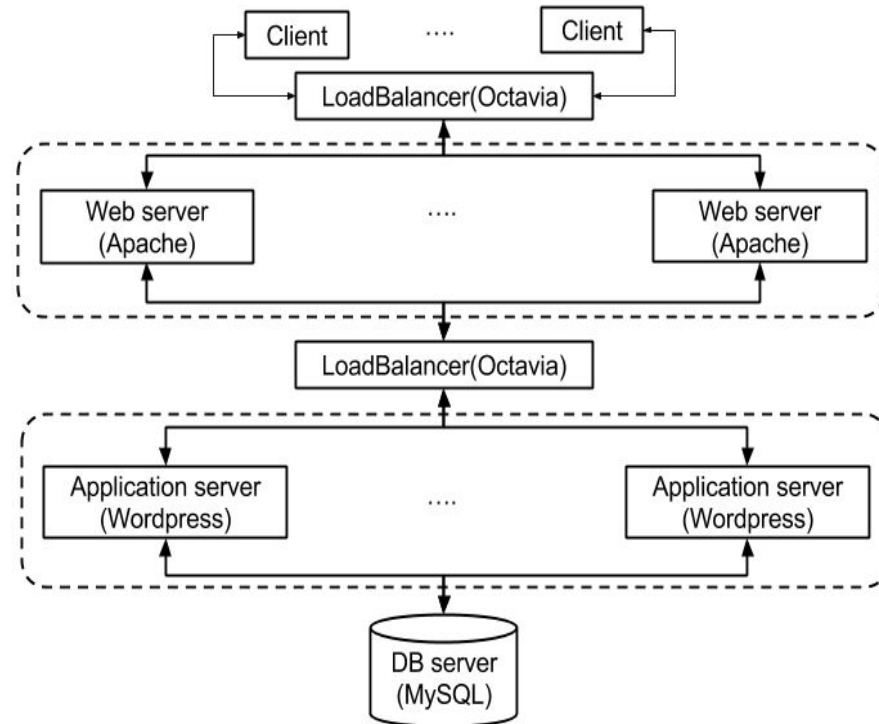
Network Profile: 100-200 p/s
Memory Profile: 1-2 GB
Disk Profile: 400-600 IOPS

Simple & realistic traffic generation

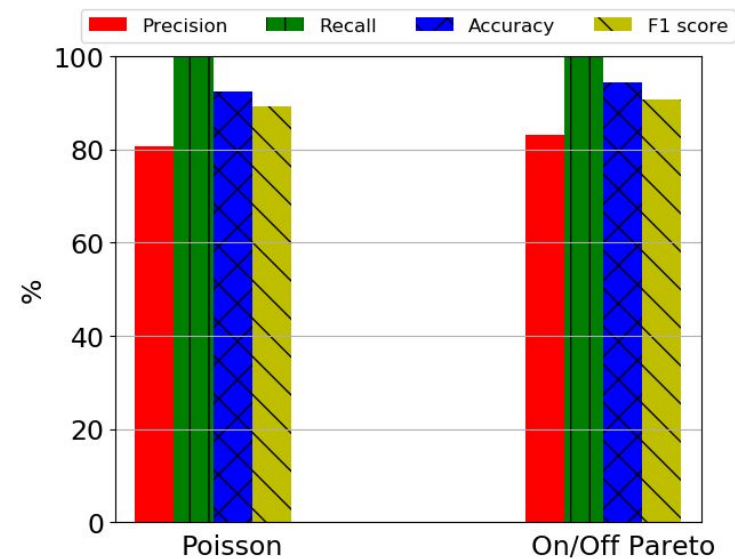
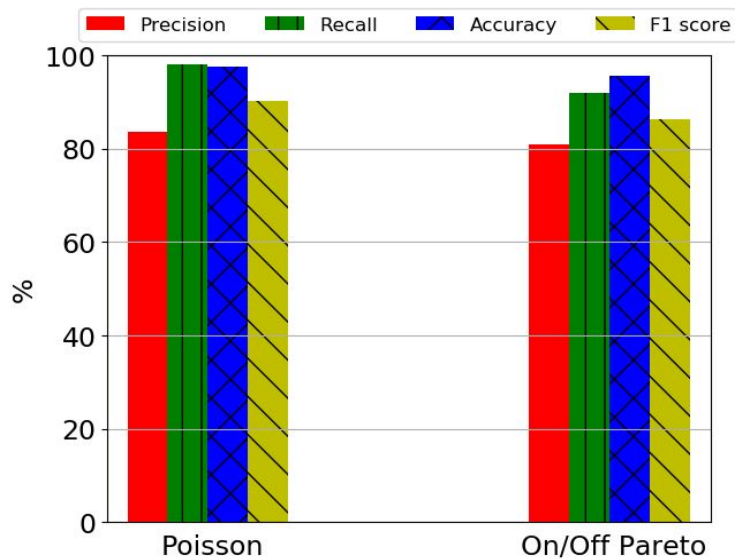
- Poisson
 - Used in many cases due to its simplicity.
- On/Off Pareto
 - Internet traffic is proved to be of self-similar nature.

The simulation parameters are as follows:

- Generator: On/Off Pareto, Poisson
- Number of concurrent clients: 50
- Requests arrival rate/hour: 3600
- Type of requests: GET and POST (randomly generated)

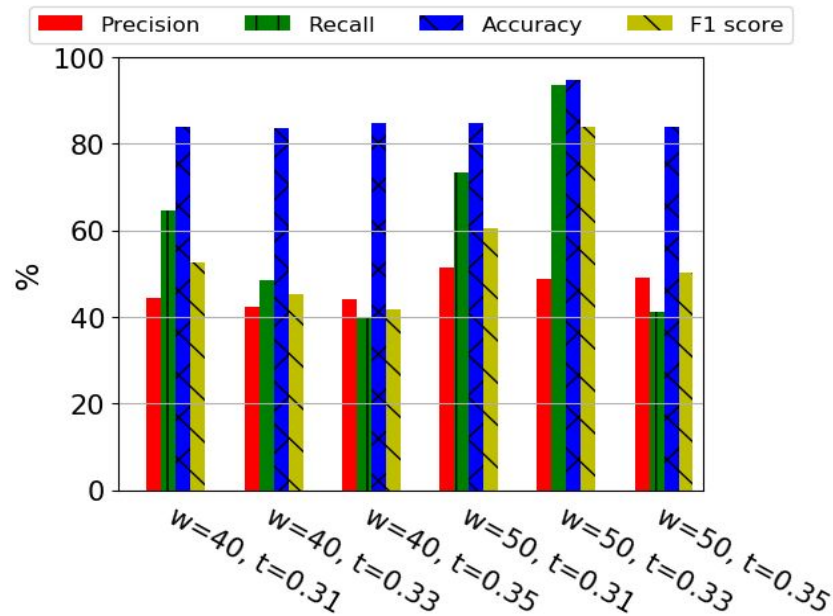


- Injected Anomalies:
 - cpu, memory and disk intensive
- EDoS: One form of EDoS is to create some VMs while remaining dormant and idle

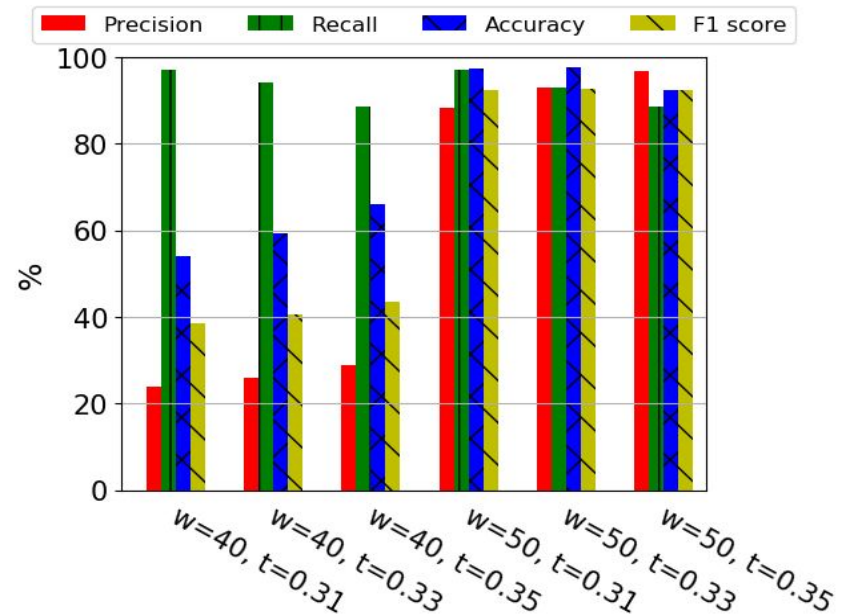


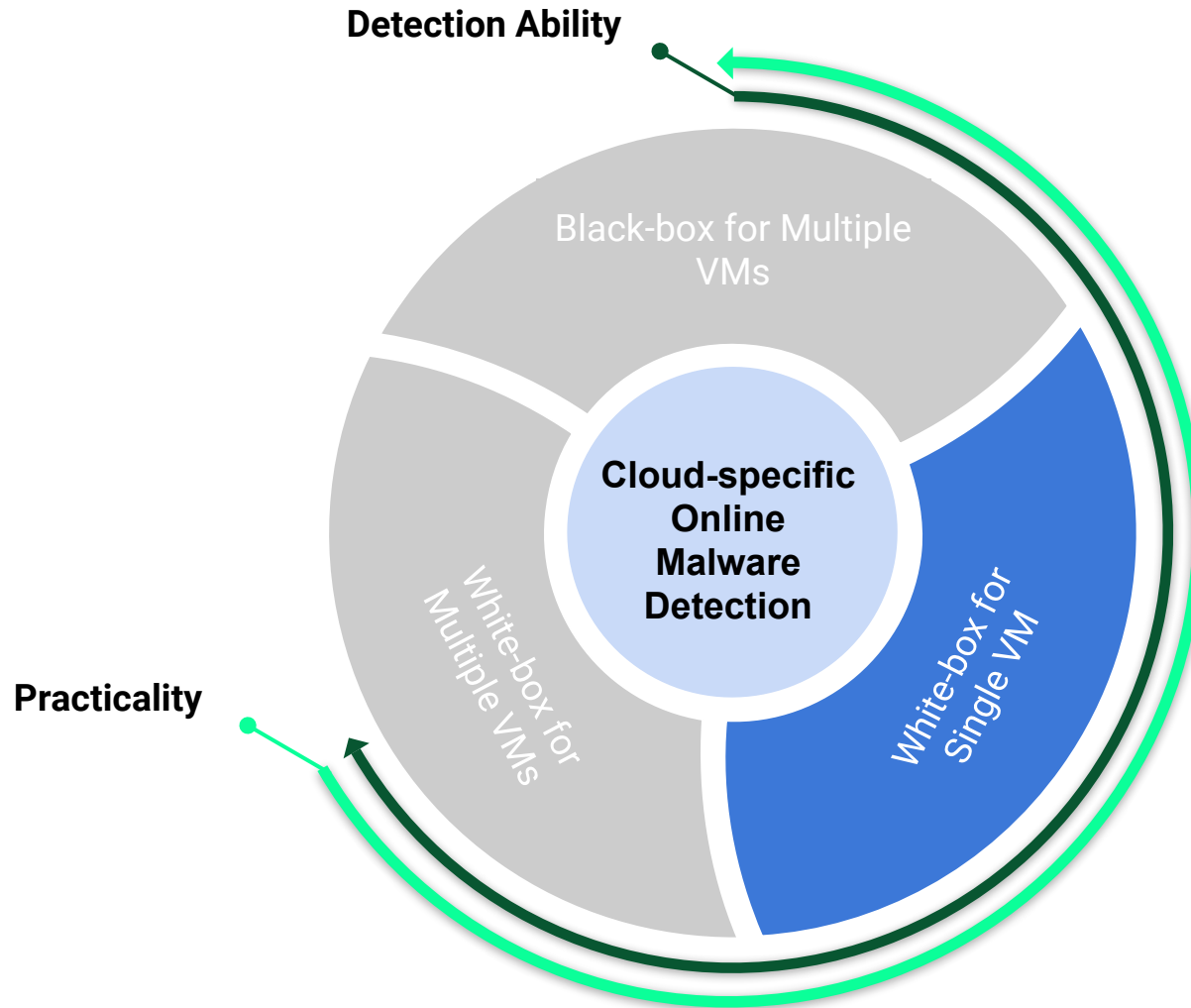
- Ransomware is a very critical threat to cloud.
- Netskope's quarterly cloud report states that 43.7% of the cloud malware types detected in cloud apps are common ransomware delivery vehicles.

Ransomware (KillDisk) - Poisson



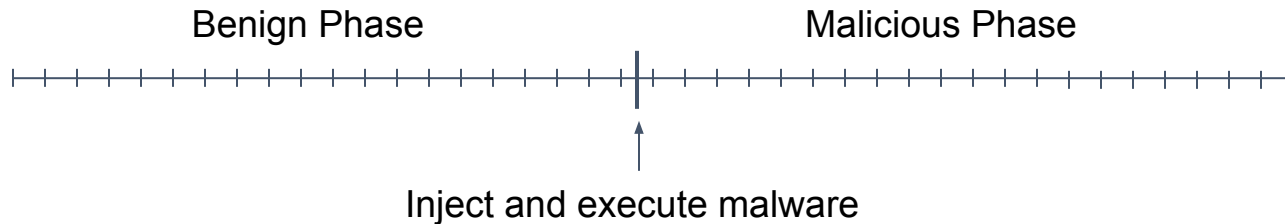
Ransomware (KillDisk) - On/Off Pareto





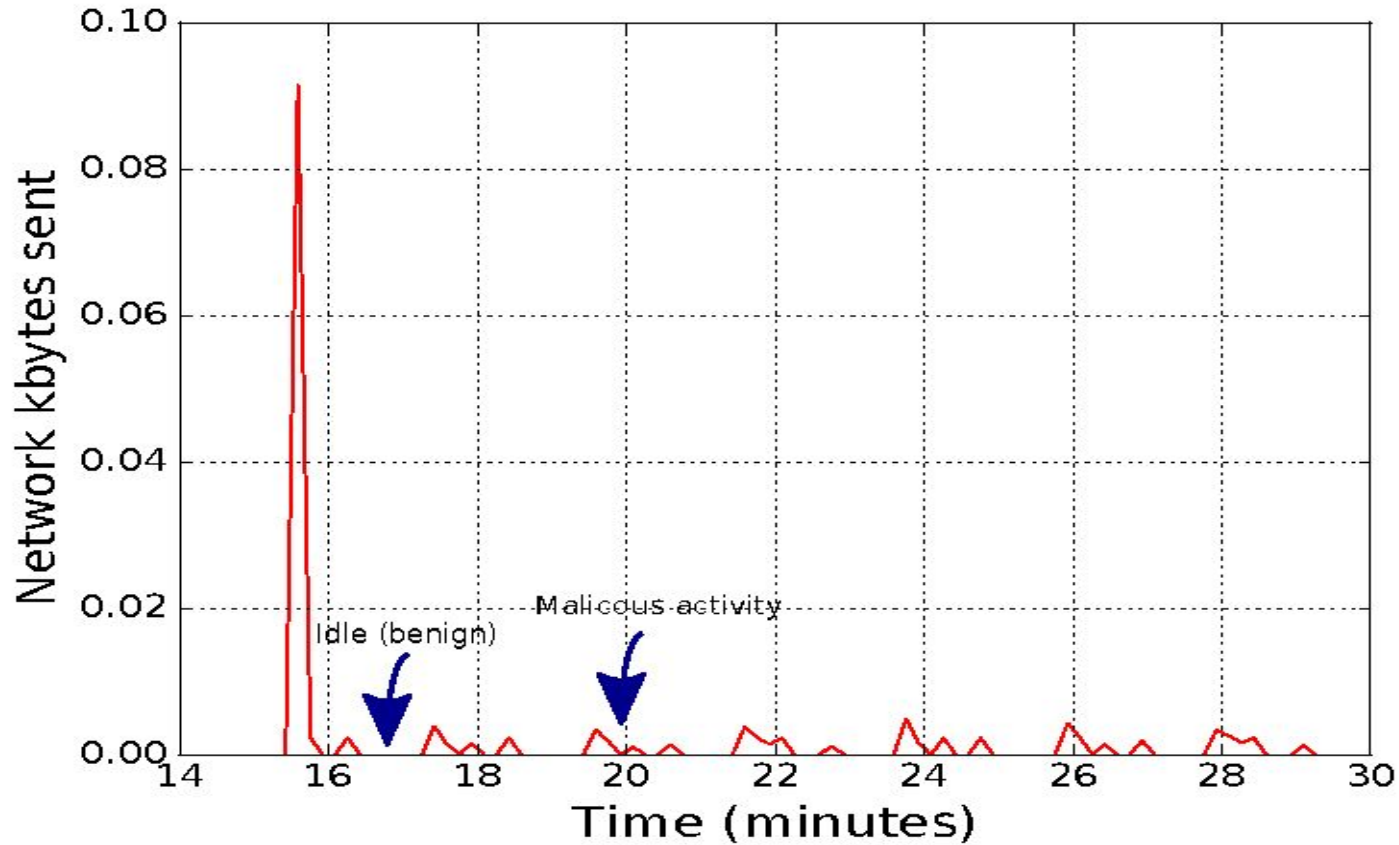
Goals:

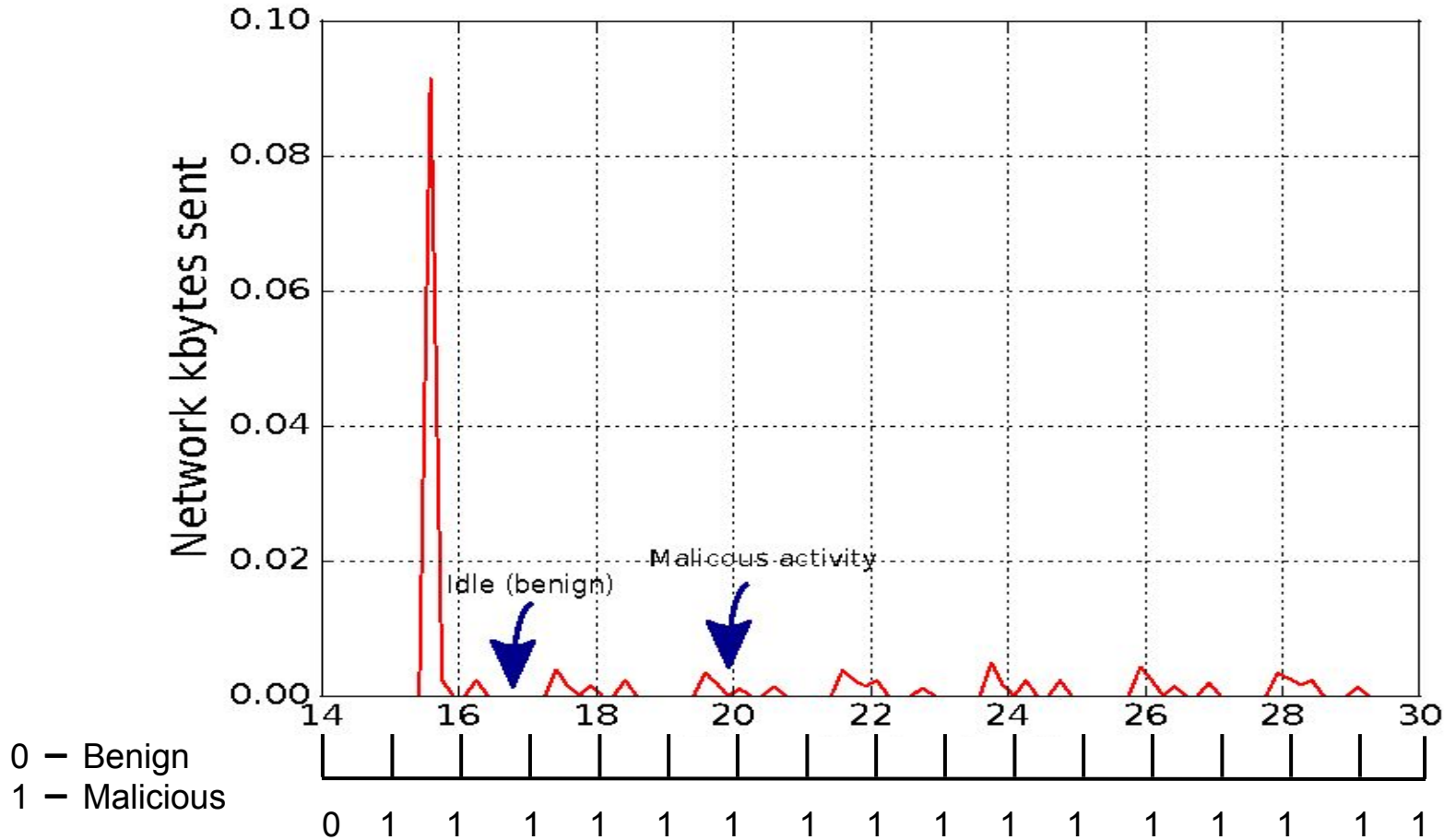
- The feasibility of applying CNN to VMs online malware detection using fine-grained process performance metrics.
- Tackling the mislabeling problem by using 3d CNNs.

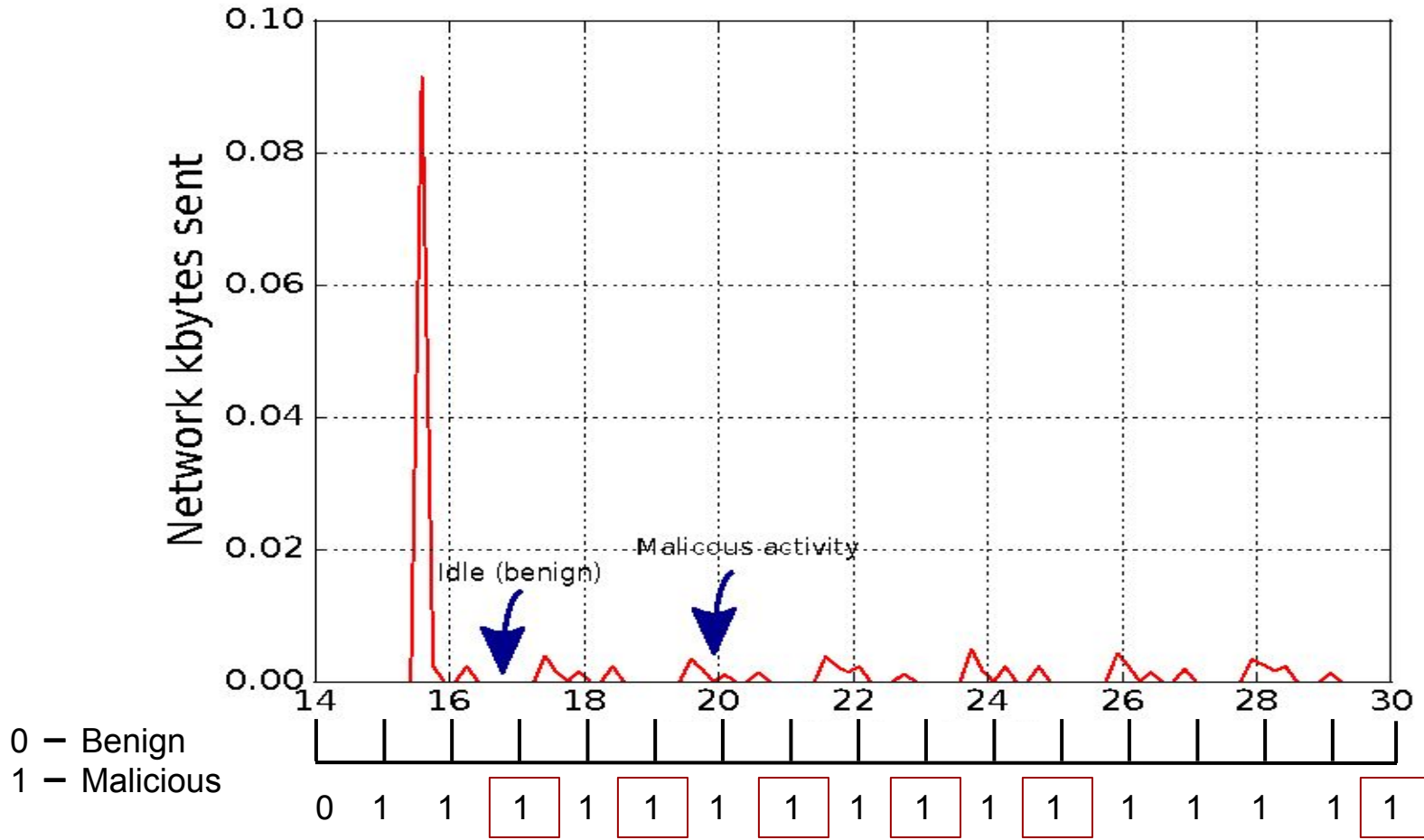


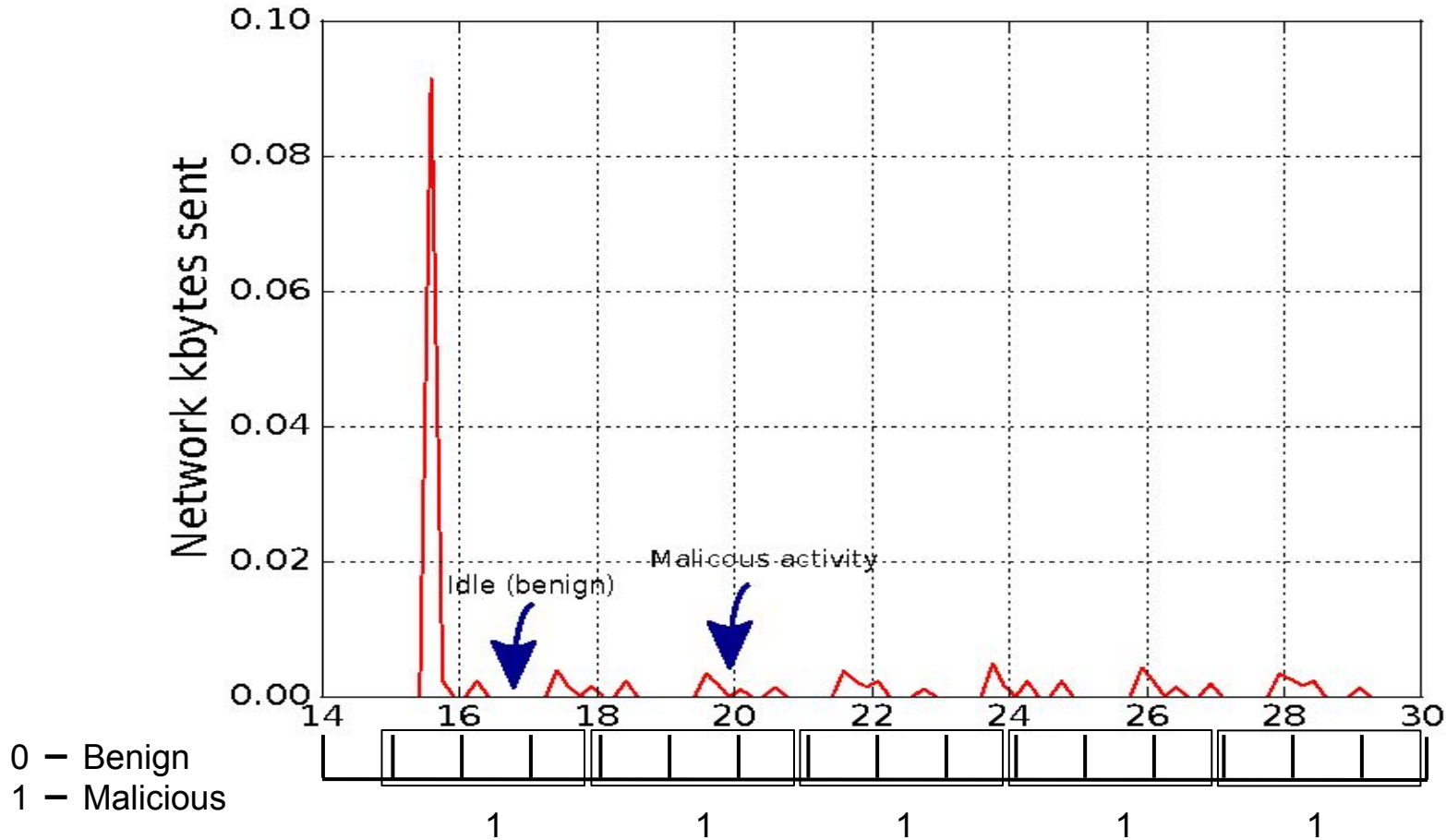
During the training phase, there is no guarantee that a malware exhibited malicious behavior.

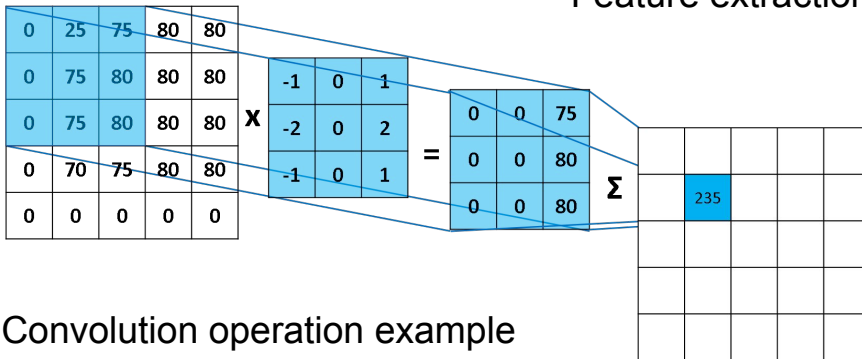
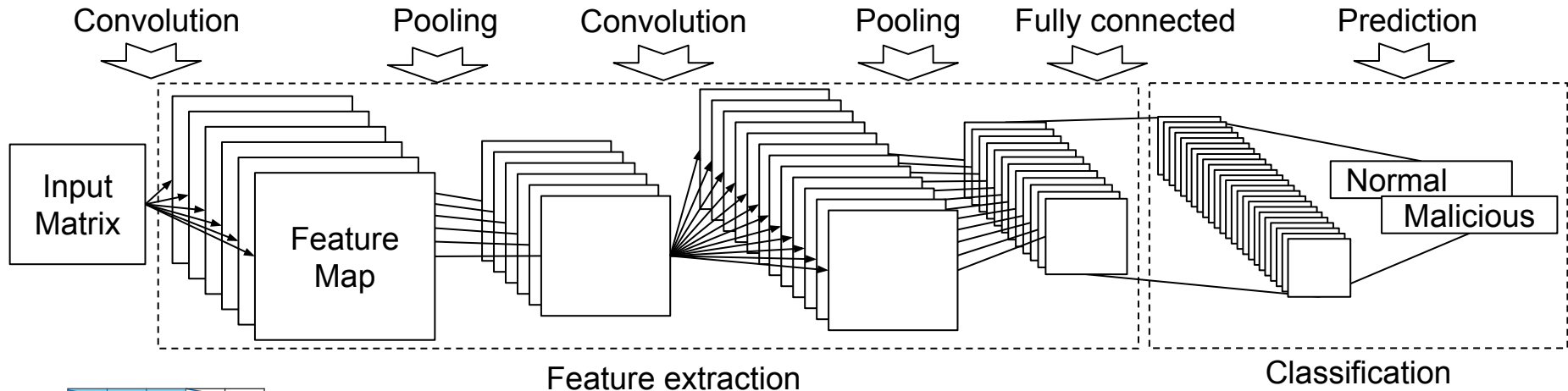
- A malware may never show a malicious activity during the training phase at all.
- + More common scenario is when a malware periodically (e.g., every 1 minute) performs malicious activities such as stealing and sending some information to its Command and Control servers (C&Cs).







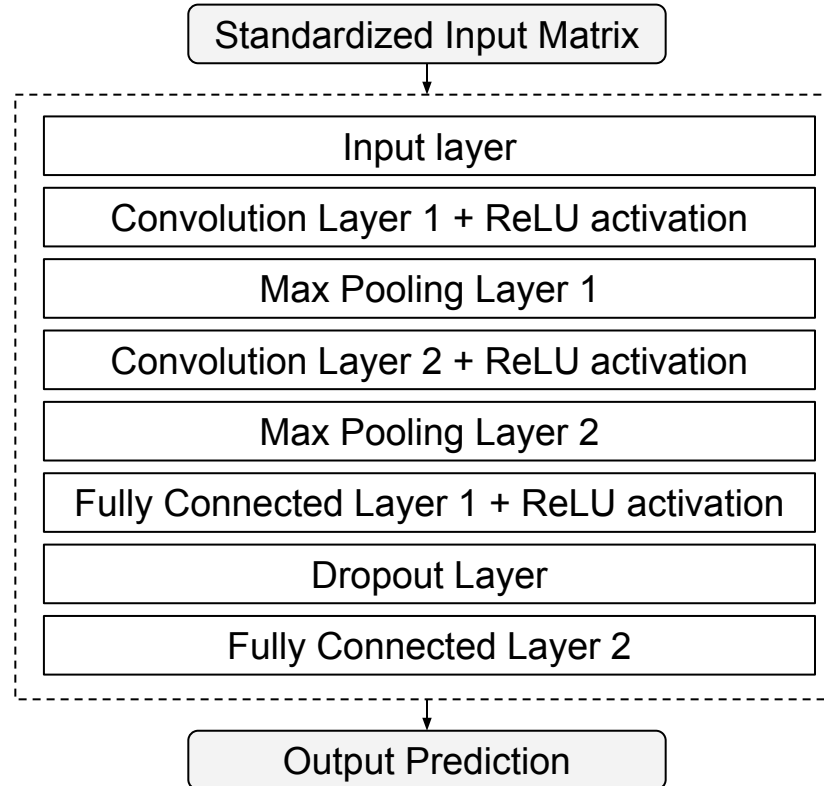




Convolution operation example
Ref: blog.csdn.net

- We use performance metrics as a way of defining a process behavior.
- 28 process-level performance metrics.
- These metrics can easily be fetched through the hypervisor.

Metric Category	Description
Status	Process status
CPU information	CPU usage percent, CPU times in user space, CPU times in system/kernel space, CPU times of children processes in user space, CPU times of children processes in system space.
Context switches	Number of context switches voluntary, Number of context switches involuntary
IO counters	Number of read requests, Number of write requests, Number of read bytes, Number of written bytes, Number of read chars, Number of written chars
Memory information	Amount of memory swapped out to disk, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Number of dirty pages, Amount of physical memory, text resident set (TRS), Memory used by shared libraries, memory that with other processes
Threads	Number of used threads
File descriptors	Number of opened file descriptors
Network information	Number of received bytes, Number of sent bytes

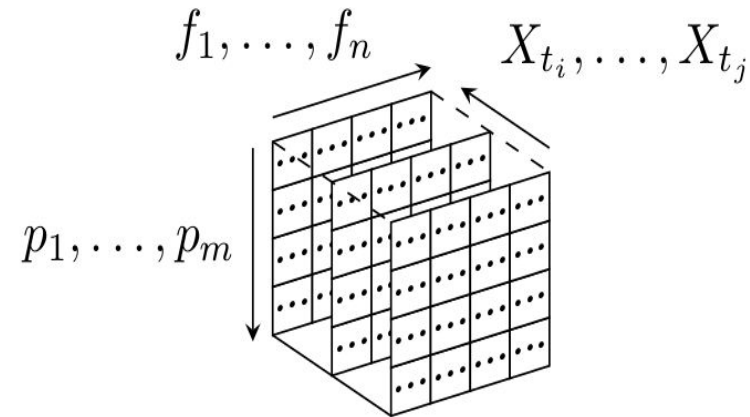


We represent each sample as an image (2d matrix) which will be the input to the CNN.

Consider a sample x_t at a particular time t , that records n features (performance metrics) per process for m processes in a VM:

$$\mathbf{X}_t = \begin{bmatrix} & f_1 & f_2 & \dots & f_n \\ p_1 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_m & \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

The 3d CNN model input includes multiple samples over a time window. The input matrix is:



- CNN requires the same process to remain in the same row in each sample.
- The CNN in computer vision takes fixed-size images as inputs, so the number of features and processes must be predetermined.

Use the **max** process identification number (PID) which is set by the OS?

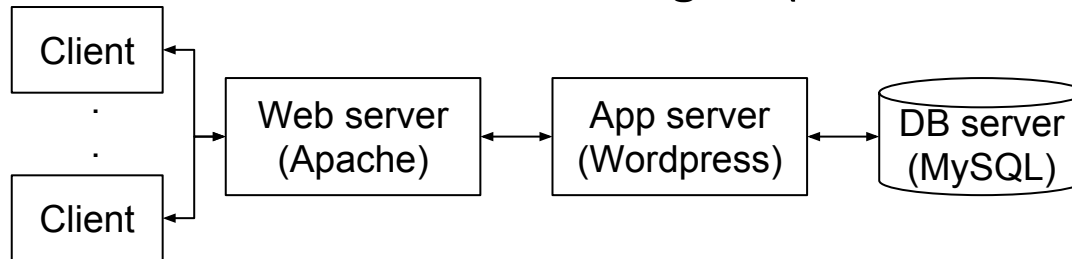
- The limit (max number of PIDs) is defined in `/proc/sys/kernel/pid_max` which is usually 32k.
- Huge input matrix!
- Change the max PID number defined?
 - Kernel confusion if wrap around happened too often.
- there is no guarantee that, for instance, a process with a PID 1000 at a particular time is going to be the same process at a later time.

- We define a process, referred to as unique process, by a 3-tuple:
 - process name
 - command line used to run process
 - hash of the process binary file (if applicable)
- We set the maximum number of unique processes to 120 to accommodate for newly created unique processes.

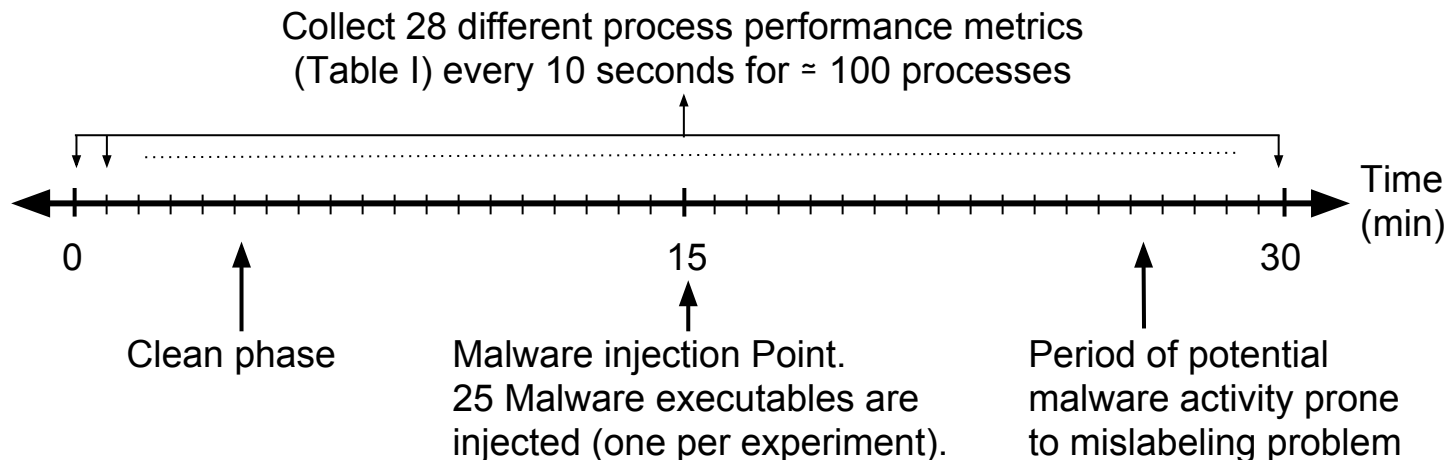
pid	name	cmd	hash	kb_sent	cpu_user	sample_time
1241	php-fpm7.0	php-fpm: pool www	7eb8522425...	33.61710	0.03000	2018-06-15 11:19:04
1240	php-fpm7.0	php-fpm: pool www	7eb8522425...	38.79308	0.00000	2018-06-15 11:19:04
1221	php-fpm7.0	php-fpm: master process (/etc/php/7.0/...	7eb8522425...	0.00000	0.02000	2018-06-15 11:19:04
1287	python	python	23eeeb4347...	0.00000	0.15000	2018-06-15 11:19:04

Unique Process						
name	cmd	hash	AVG(kb_sent)	AVG(cpu_user)	sample_time	
php-fpm7.0	php-fpm: pool www	7eb8522425...	36.2051	0.0150	2018-06-15 11:19:04	
php-fpm7.0	php-fpm: master process (/etc/php/7.0/...	7eb8522425...	0.00000	0.0200	2018-06-15 11:19:04	
python	python	23eeeb4347...	0.00000	0.1500	2018-06-15 11:19:04	

- Our experiments were conducted on Openstack.
- To simulate a real world scenario, we used a 3-tier web architecture and a self-similar traffic gen. (on/off Pareto) is used.

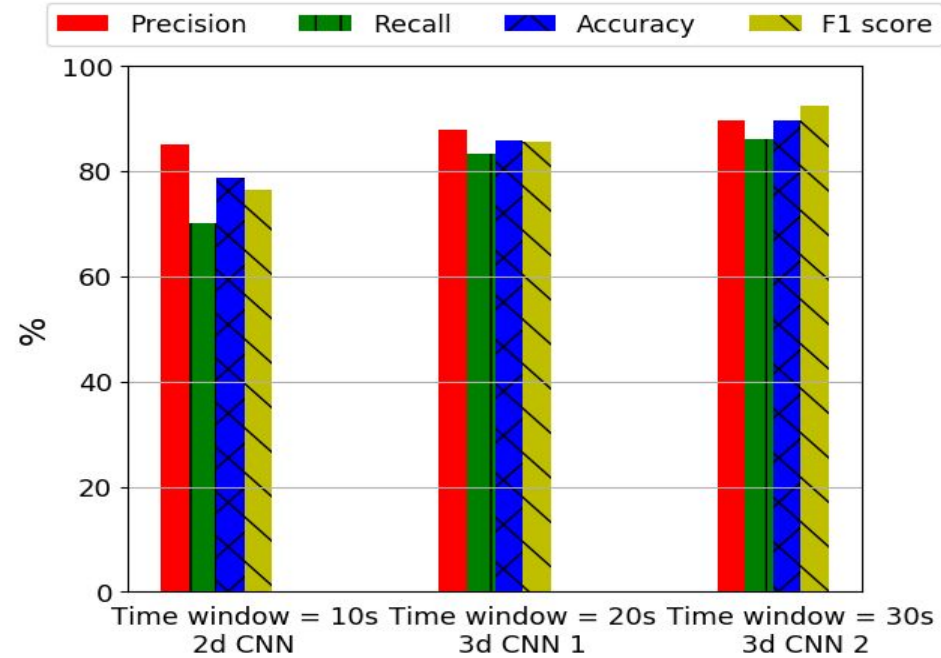


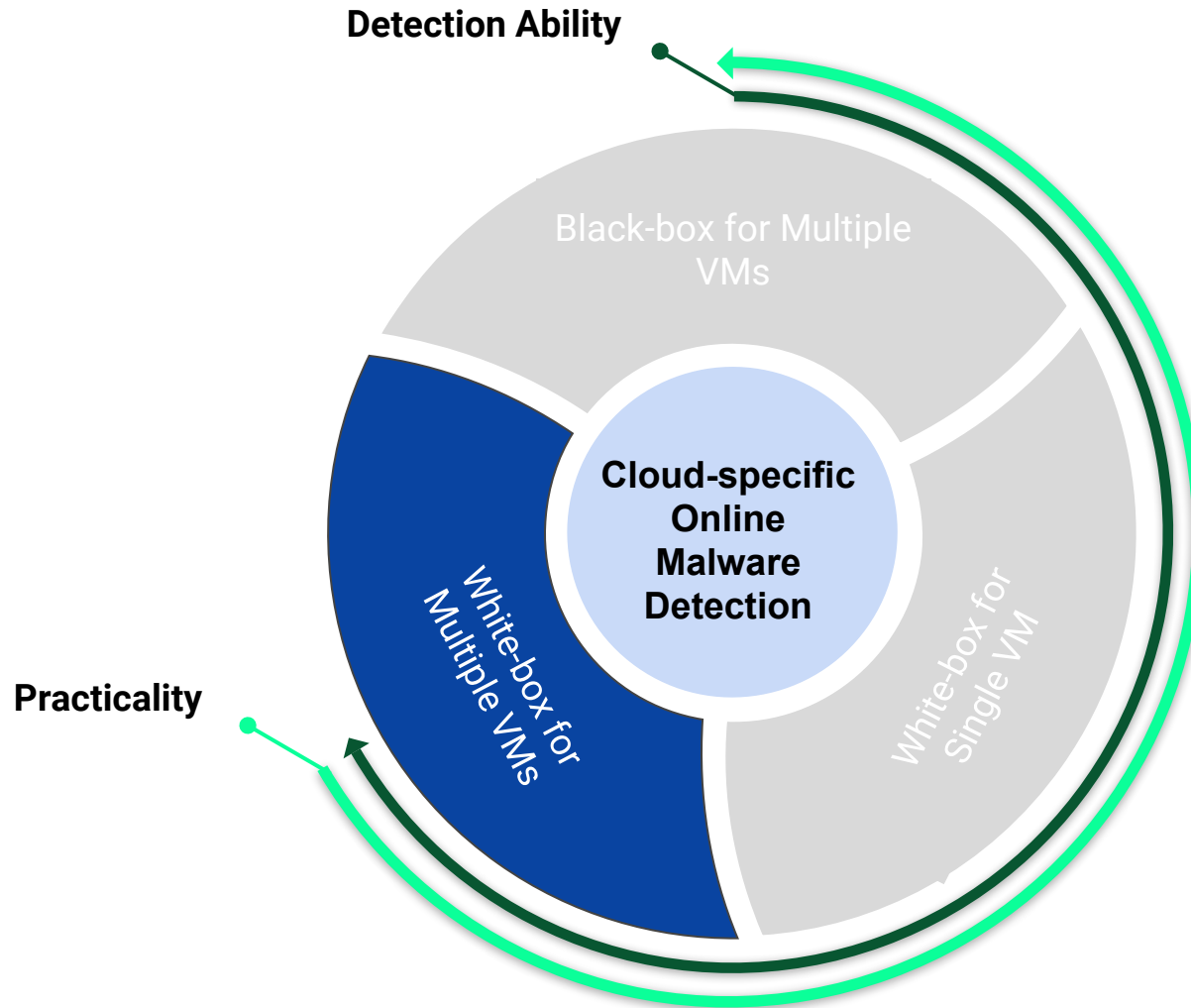
- Data collection:



2d CNN

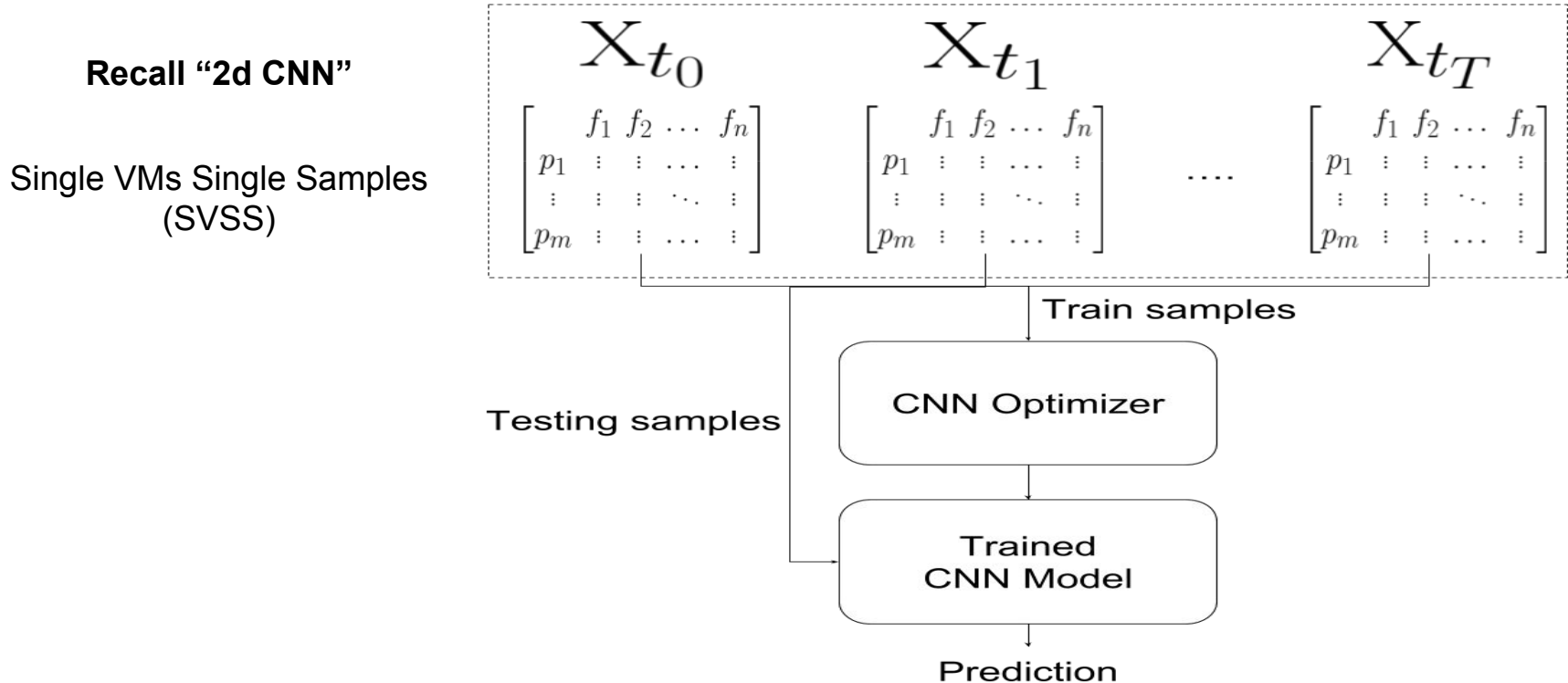
3d CNN

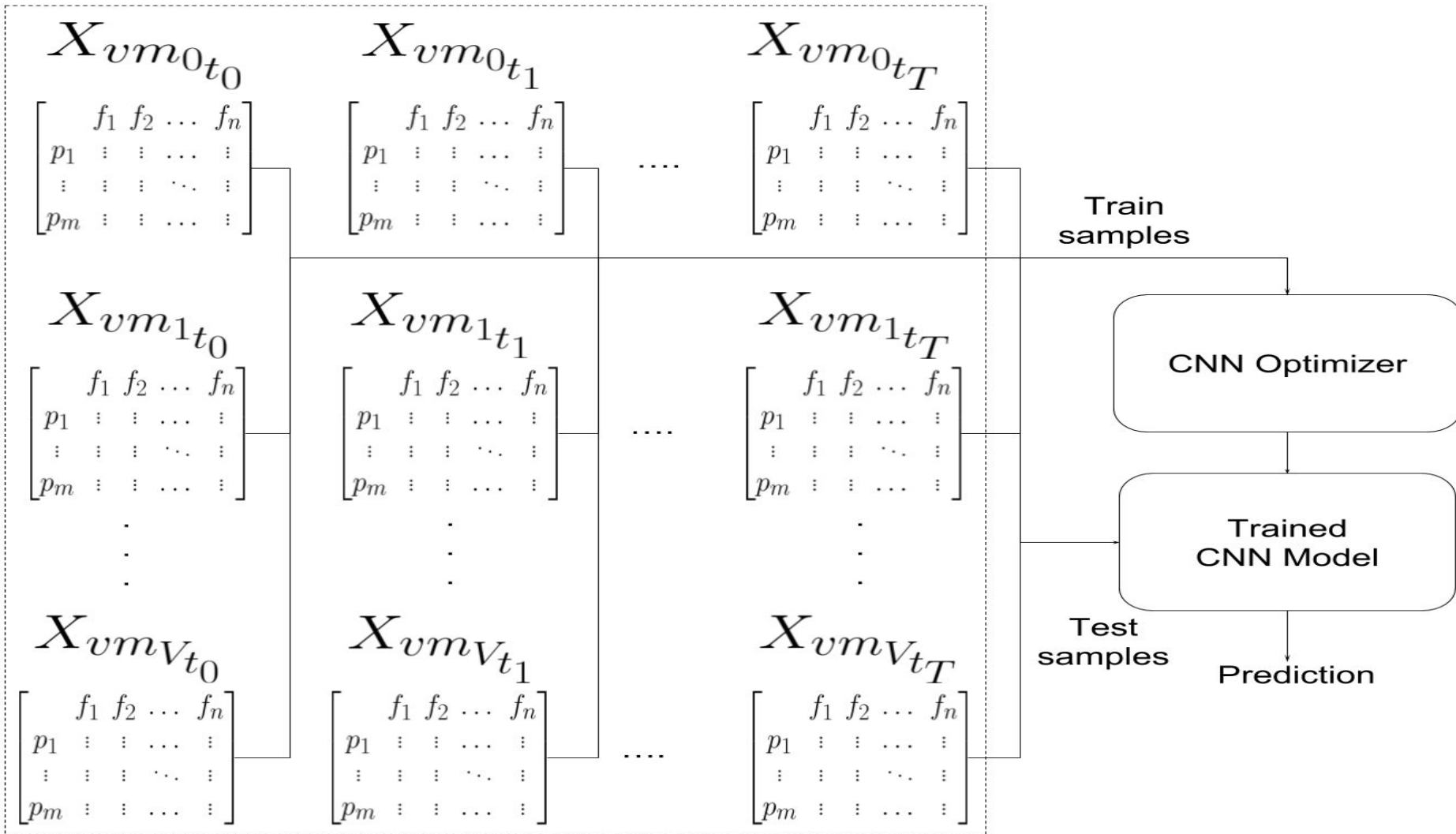




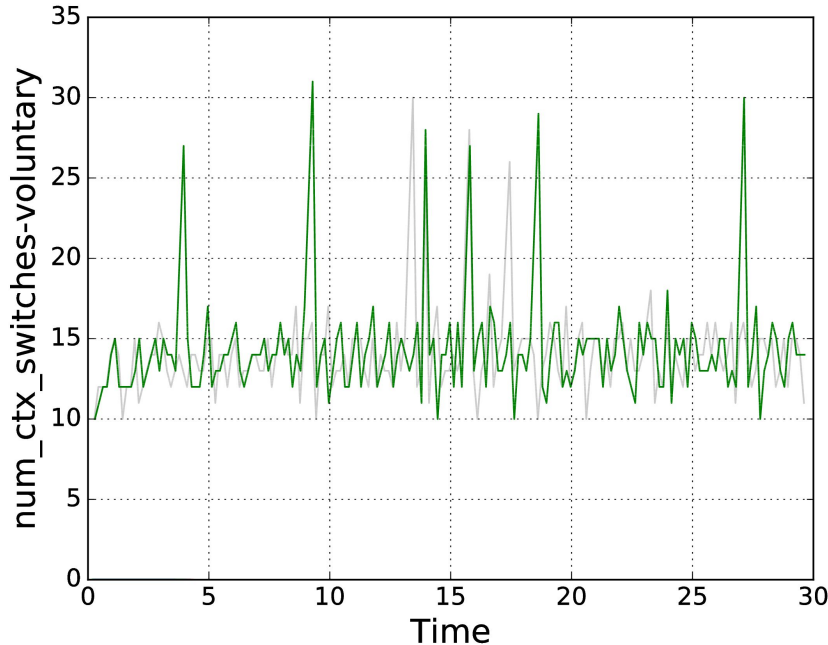
Goal: **Leverage auto-scaling for whitebox online malware detection** by

1. Using 2d CNN for multiple VMs.
2. Introducing a novel approach of pairing samples to accommodate for correlations between VMs.

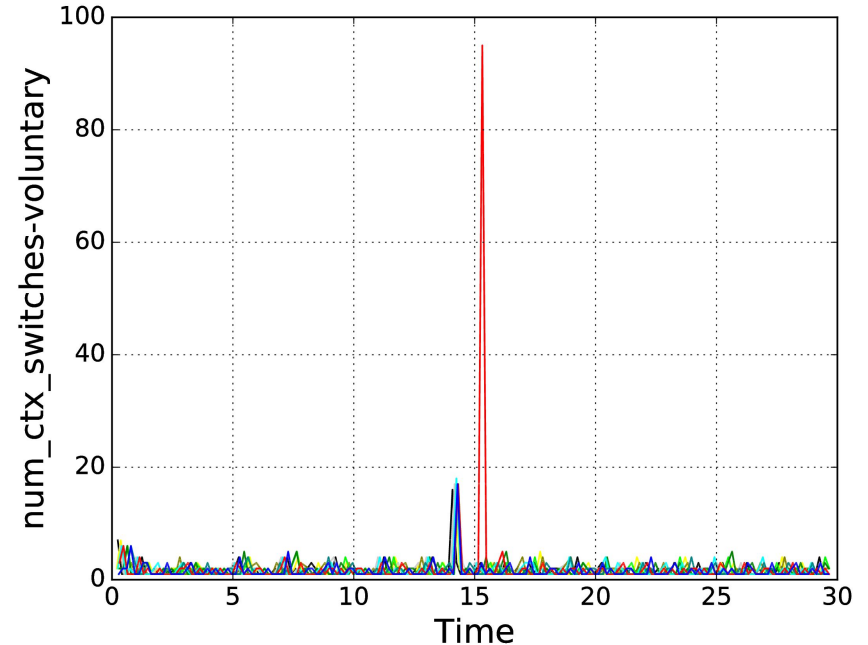




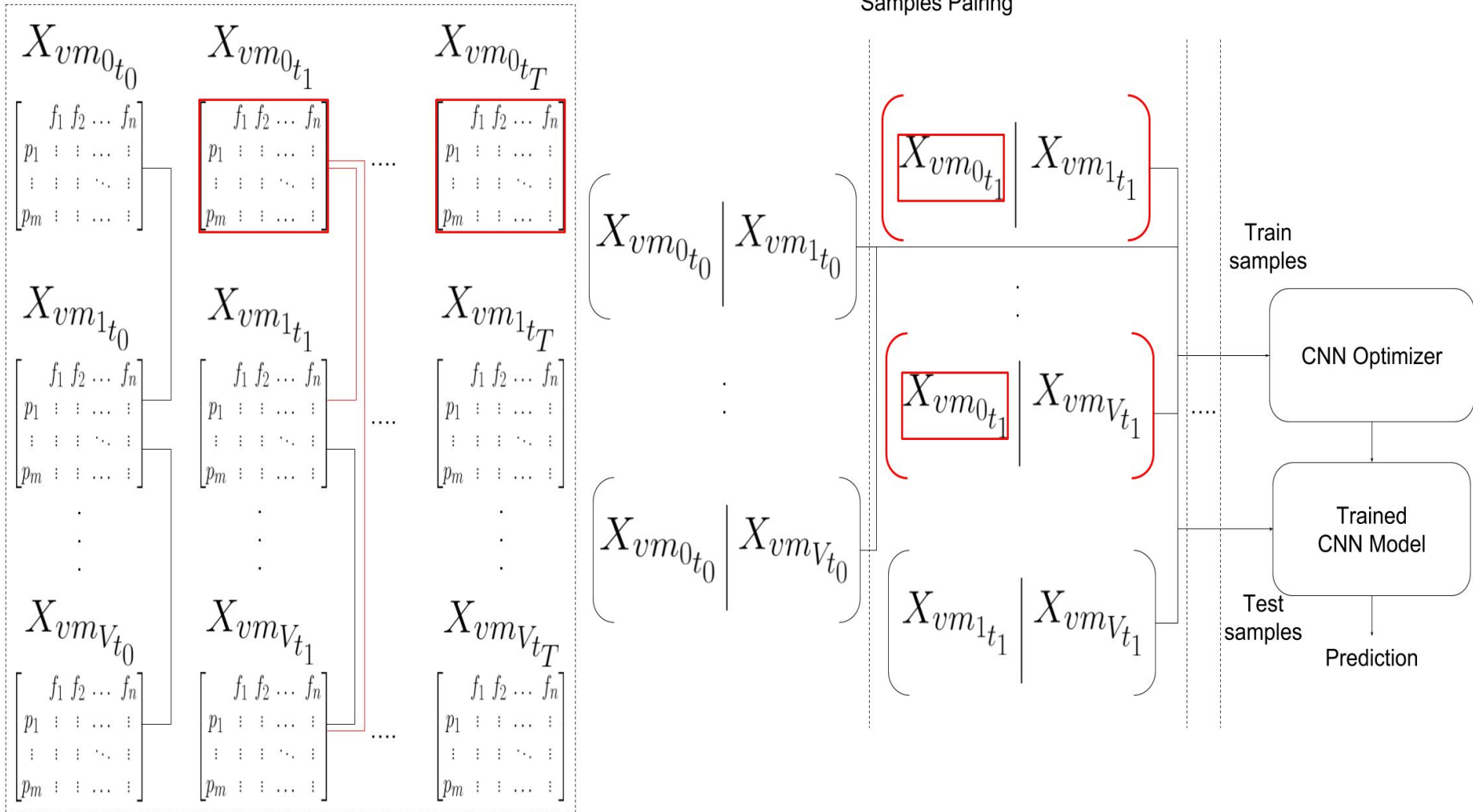
What do we gain from having multiple VMs in an auto-scaling scenario?
“Correlation between VMs”



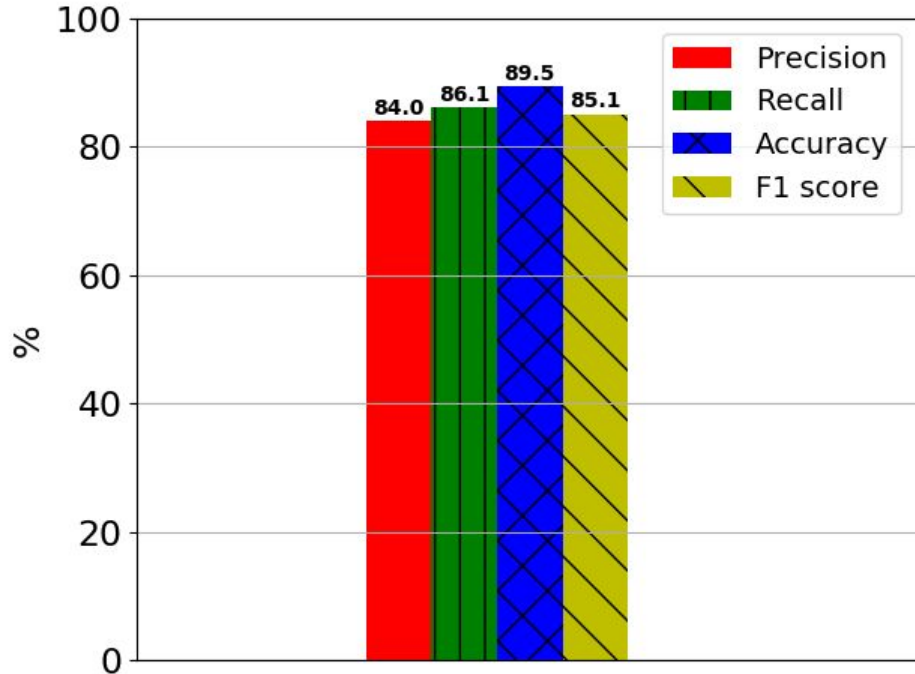
Number of used voluntary context switches over 30 minutes for two different runs of the same unique process



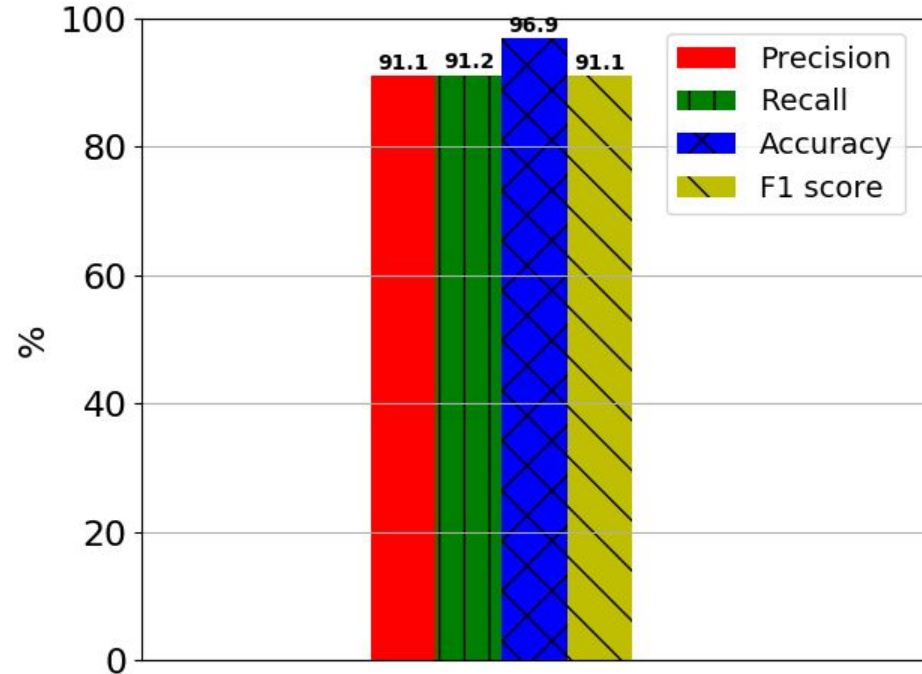
Number of used voluntary context switches over 30 minutes for one run of 10 VMs in an auto-scaling scenario.



MVSS



MVPS



The goal of this thesis was to provide a develop cloud-specific online malware detection methods by leveraging cloud characteristics (i.e., auto-scaling).

In satisfying dissertation objectives:

1. We developed an online anomaly detection system for cloud IaaS that targeted highly-active malware in an auto-scaling scenario.
2. We developed an effective approach for detecting malware using process-level features for low-level malware in a single VM scenario.
3. We developed a pairing samples approach for detecting malware using process-level features that targeted low-level malware in an auto-scaling scenario.

Future Work:

- Applying and testing multiple architectures (e.g., hadoop systems or containers)
- Investigating and leveraging more cloud characteristics for security.

- **Mahmoud Abdelsalam**, Ram Krishnan, and Ravi Sandhu “*Clustering-Based IaaS Cloud Monitoring*”, In Proceedings 10th IEEE International Conference on Cloud Computing (CLOUD), Honolulu, Hawaii, June 25-30, 2017, 8 pages.
- **Mahmoud Abdelsalam**, Ram Krishnan, Yufei Huang and Ravi Sandhu “*Malware Detection in Cloud Infrastructures using Convolutional Neural Networks*”, In Proceedings 11th IEEE International Conference on Cloud Computing (CLOUD), San Francisco, CA, July 2-7, 2018, 8 pages.
- **Mahmoud Abdelsalam**, Ram Krishnan, and Ravi Sandhu “*Online Malware Detection in Cloud Auto-Scaling Systems Using Shallow Convolutional Neural Networks*”, submitted for peer review in IEEE ICC CISS 2019.

Questions/Comments

