AN MTD-BASED SELF-ADAPTIVE RESILIENCE APPROACH FOR CLOUD SYSTEMS

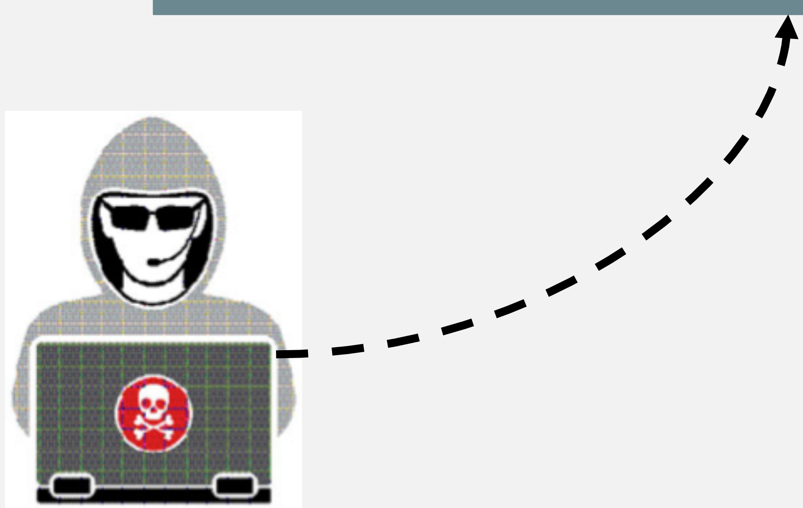
Miguel Villarreal-Vasquez[1], Bharat Bhargava[1], Pelin Angin[2], Noor Ahmed[3], Daniel Goodwin[4], Jason Kobes[4], Kory Brin[4]

[1] Department of Computer Science, Purdue University
[2] Department of Computer Engineering, Middle East Technical University
[3] Air Force Research Laboratory
[4] Northrop Grumman Corporation

Acknowledgments: This work was funded by the Northrop Grumman Cybersecurity Research Consortium. The prototype was implemented in collaboration with Northrop Grumman and internally presented to them in April 2017. The authors would also like to thank Dr. Leszek Lilien and Dr. Weichao Wang for their valuable comments.

MOTIVATION

MOTIVATION



- Replication approaches in cloud computing increase the attack surface
- We need resilient/self-healing systems that can accurately detect anomalies and dynamically adapt themselves to keep performing mission-critical functions even under attacks and failures.

RESEARCH QUESTION

- Is it possible to construct a generic attack-resilient framework for distributed cloud systems with a combination of dynamic network configuration and continuous replacement of virtual machines?

MOVING TARGET DEFENSE (MTD)

Attack Vectors

- Data
- Code
- Infrastructure
- Communications
- People

Resilient Approaches

- Moving Target Defense (MTD)
- Proactive Restore/C2
- Least Privilege Enforcement
- Trust Zone Segmentation
- Identity Attribution
- Encryption
- Root Trust

MOVING TARGET DEFENSE (MTD)

- The proposed **Moving Target Defense (MTD)** solution introduces resiliency and adaptability to the system through live monitoring, which transforms systems to be able to adapt and self-heal when ongoing attacks are detected

MOVING TARGET DEFENSE (MTD)

- **Adversaries have an asymmetric advantage**: They have the time to study a system, identify its vulnerabilities, and choose the time and place of attack to gain the maximum benefit
- **The idea of moving-target defense (MTD):** Imposing the same asymmetric disadvantage on attackers by making systems dynamic and therefore harder to explore and predict

Threat Avoidance Techniques!

STATE OF THE ART AND LIMITATIONS

REPLICATION/REDUNDANCY

Fault-Tolerance Systems

- **Solution**: Replication/ Redundancy:
- **Examples**: Quorum, Chain
- **Limitation**: Gives fault resiliency but increases attack surface at application level (common code base)

DIVERSIFICATION/RANDOMIZATION

Fault-Tolerance Systems

- **Solution**: MTD
- **Examples**: ASLR [9], NVersion [10] & IP-Hopping [11]
- **Limitation**: Do not protect the entire host

STATE OF THE ART AND LIMITATIONS

- The traditional defensive security strategy for distributed systems is to prevent attackers from gaining control of the system using well established techniques: Replication/Redundancy, Encryption, etc.
  - **Limitation**: Given sufficient time and resources, existing defensive methods can be defeated

STATE OF THE ART AND LIMITATIONS

- The state of the art of MTD solutions focus on randomization and diversification in particular layers of the system
  - **Limitation**: Do not protect the entire host

PROPOSED APPROACH

- **“Stay one-step ahead” of sophisticated attack**
  - Protect the entire stack through dynamic interval-based spatial randomization
  - Avoid threats in-time intervals rather than defending the entire runtime of systems through Mobility and Direction
  - System will start secure, stay secure and return secure
  - Increase agility, anti-fragility and adaptability of the system
  - Unified generic MTD framework that enables reasoning about behavior of deployed systems on cloud platforms

OBJECTIVES OF THE MTD SOLUTION

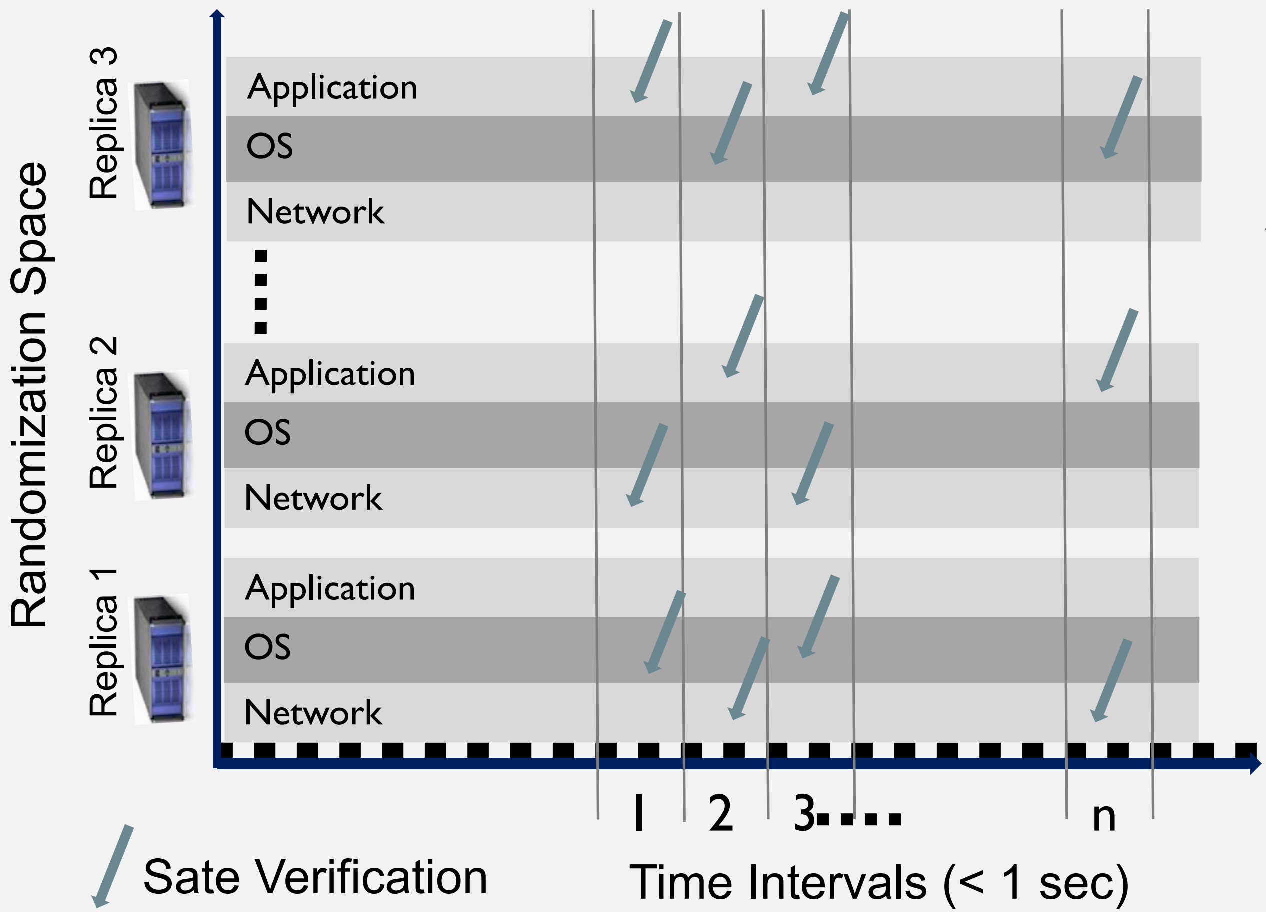
- Aims to **reduce** the need to continuously **fight against attacks** by decreasing the gain-loss balance perception of attackers.
- **Narrows the exposure window of a node to attacks**, which increases the cost of attacks on a system and lowers the likelihood of success and the perceived benefit of compromising it.

OBJECTIVES OF THE MTD SOLUTION

- The **reduction in the vulnerability window** of nodes is mainly achieved **through three steps**:
  - Partitioning the runtime execution of nodes in time intervals
  - Allowing nodes to run only with a predefined lifespan (as low as a minute) on heterogeneous platforms (i.e. different OSs)
  - Proactively monitoring their runtime below the OS

BENEFITS OF THE PROPOSED SOLUTION

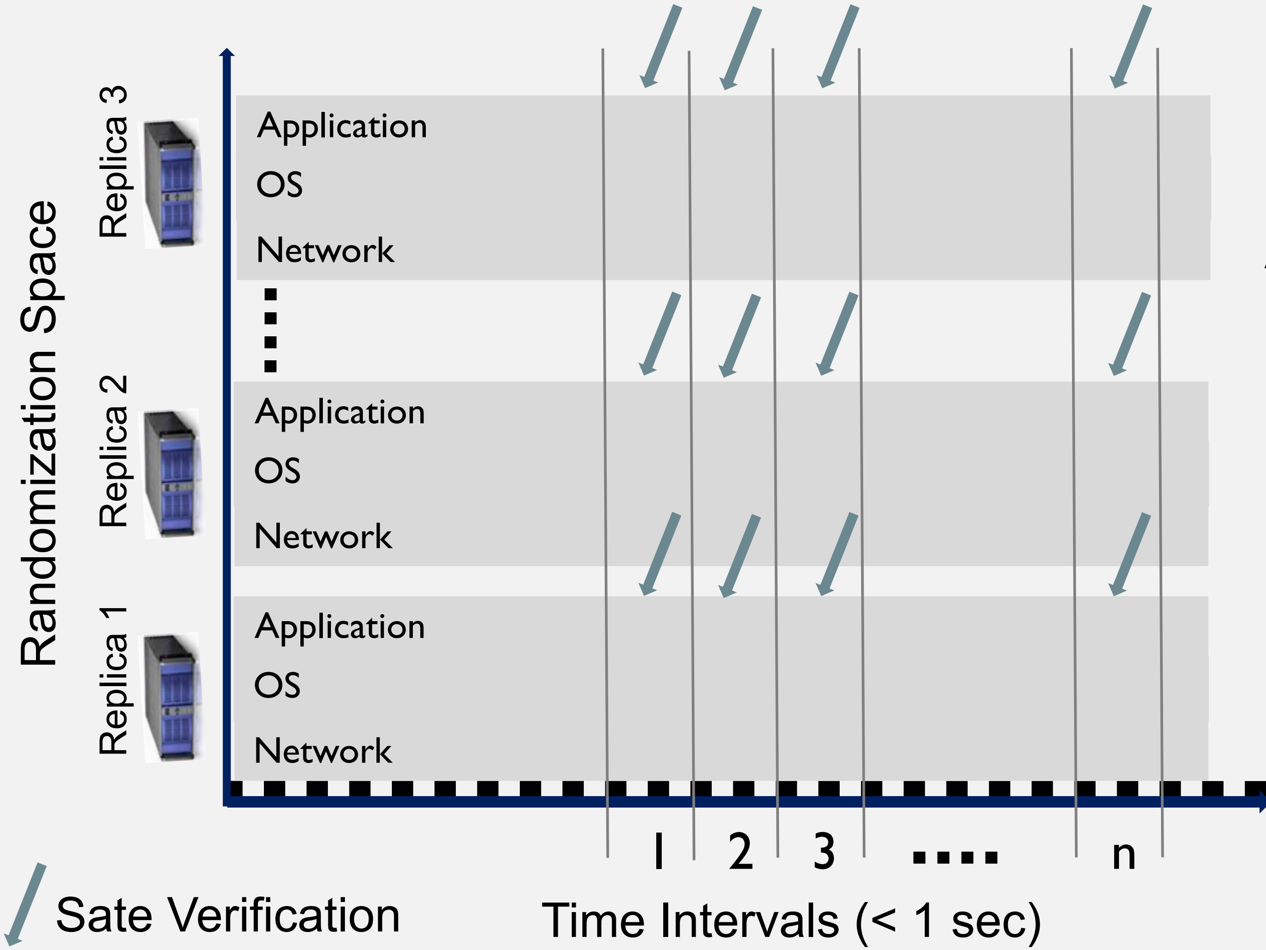
- **State of the Art System View:**



At a given time only **some** layers of the stack (Application, OS or Network) are checked/ protected

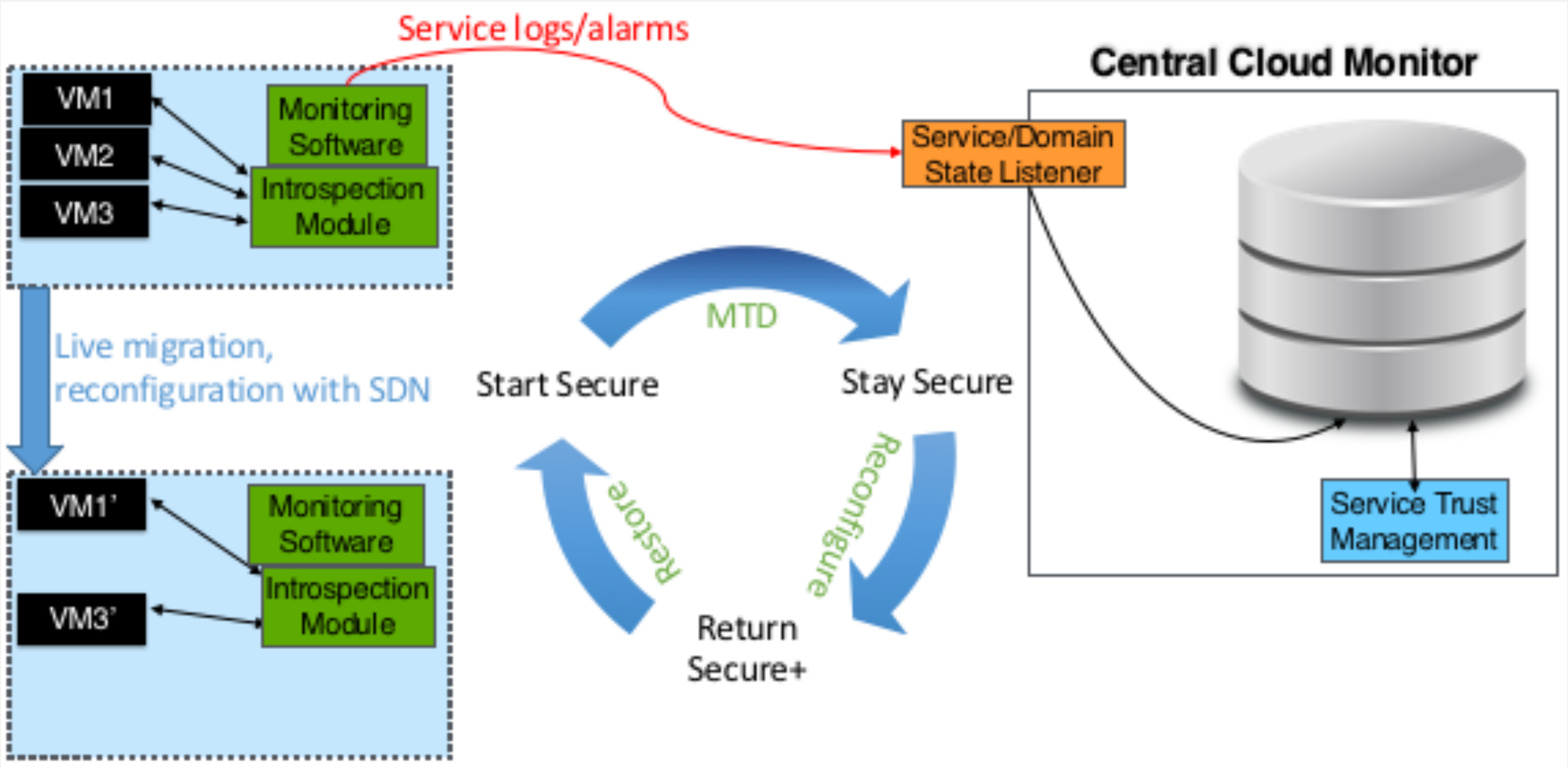
BENEFITS OF THE PROPOSED SOLUTION

- **Proposed Solution System View:**

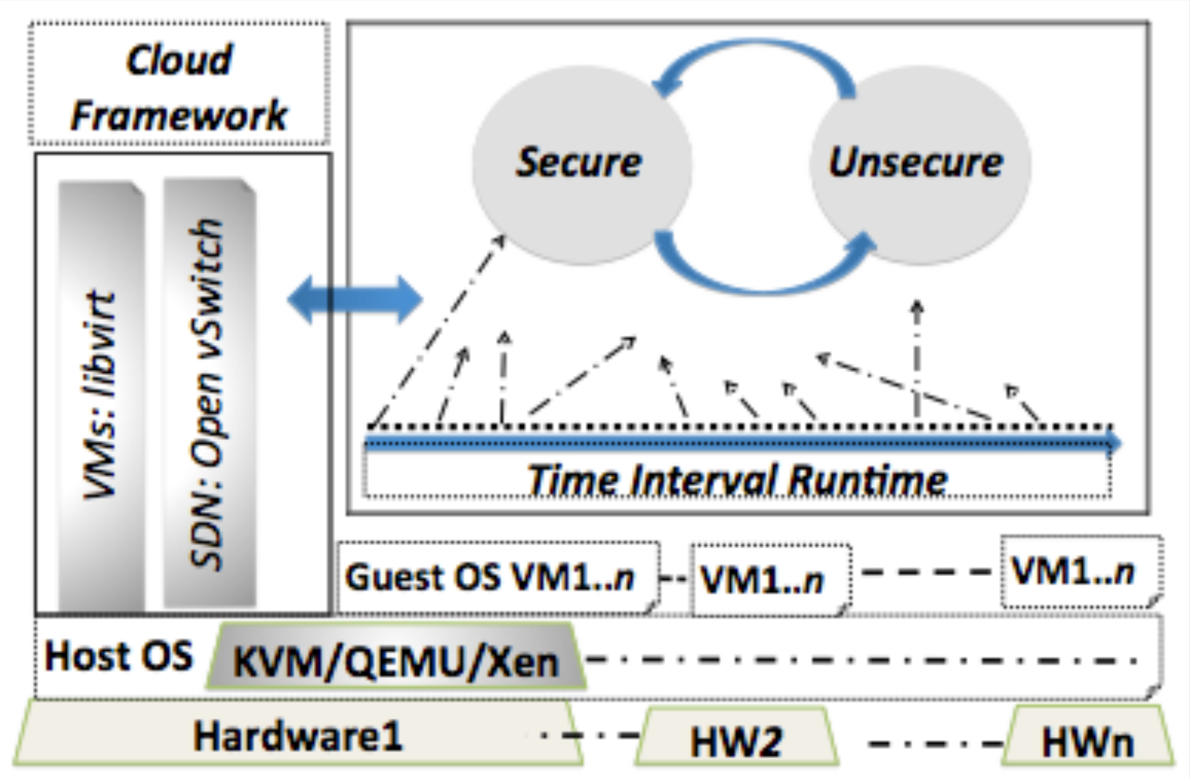


At a given time **all** layers of the stack (Application, OS or Network) are checked/protected.

APPROACH OVERVIEW



MTD ARCHITECUTRE



Components:

(1) Virtual Reincarnation (ViRA)

(2) Proactive Monitoring

(3) SDN Network Dynamics

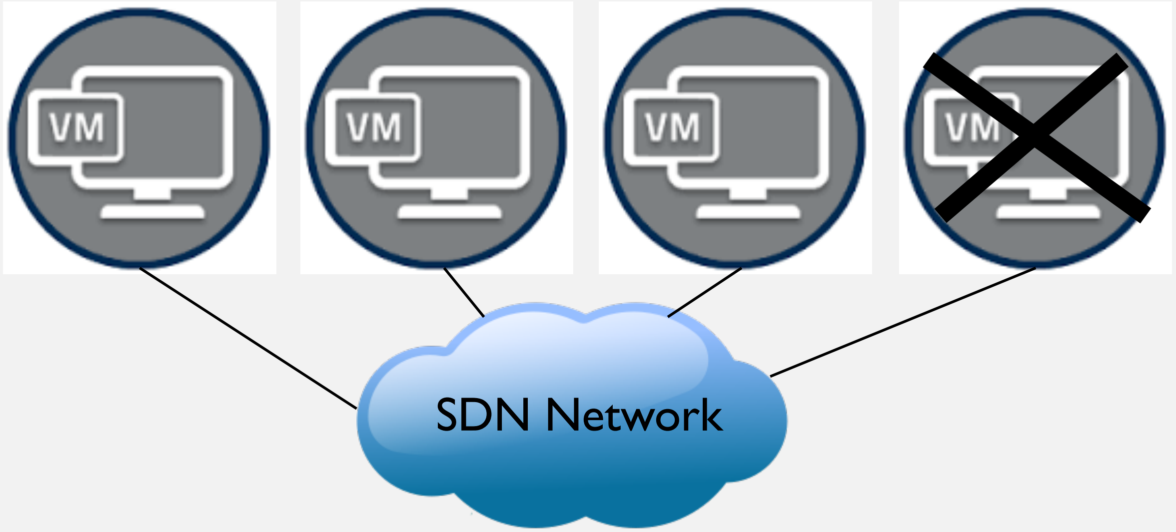
(4) Systems States and Application Runtime

MTD ARCHITECUTRE

- The MTD framework consists of the following four components:
  - Virtual Machine Reincarnation (ViRA)
  - Proactive Monitoring
  - SDN Network Dynamics
  - Systems States and Application Runtime
- The framework will protect the whole stack; not only particular layers

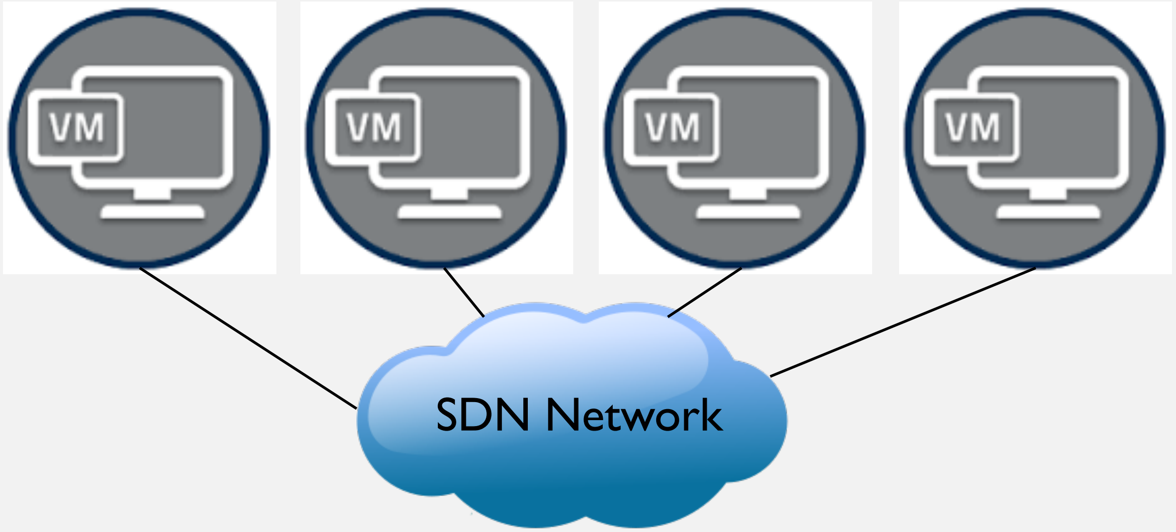
APPROACH DETAILS

- Nodes run a distributed application on a given platform for a controlled period of time
- The running time is chosen in a way that successful ongoing attacks become ineffective
- The new fresh machine will integrate to the system and continue running the application after its data is updated



APPROACH DETAILS

- Nodes run a distributed application on a given platform for a controlled period of time
- The running time is chosen in a way that successful ongoing attacks become ineffective
- The new fresh machine will integrate to the system and continue running the application after its data is updated



VIRTUAL REINCARNATION

- Randomization and diversification technique where nodes (virtual machines) running a distributed application vanish and reappear on a different virtual state with different guest OS, Host OS, hypervisor, and hardware .

Improve Resiliency

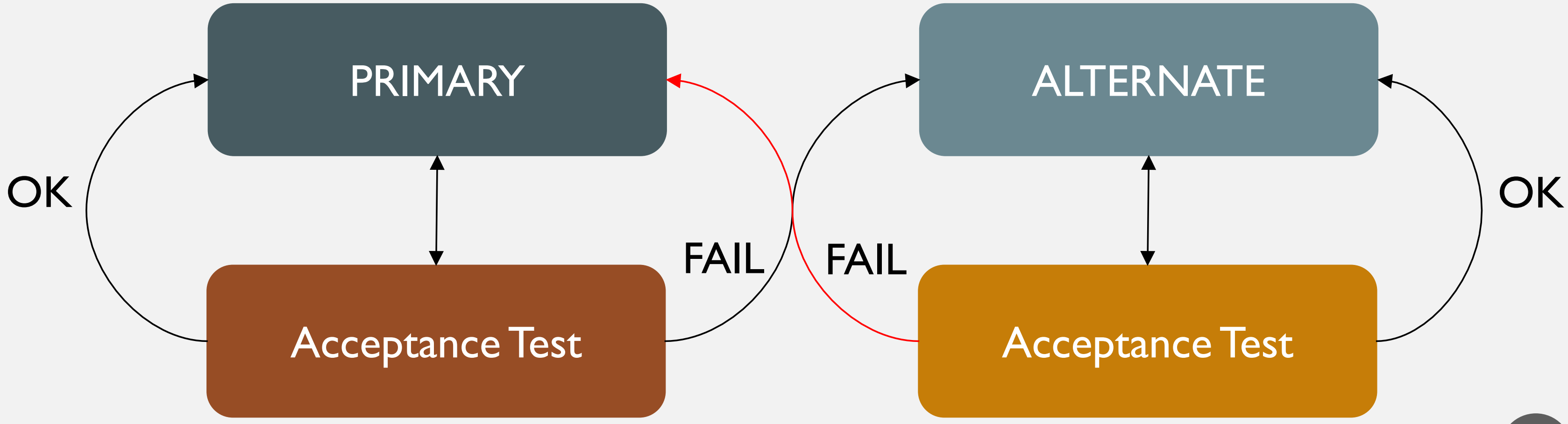
Improve Anti-Fragility

Virtualized Environment

CREATION OF REPLICAS

How do we create replicas?

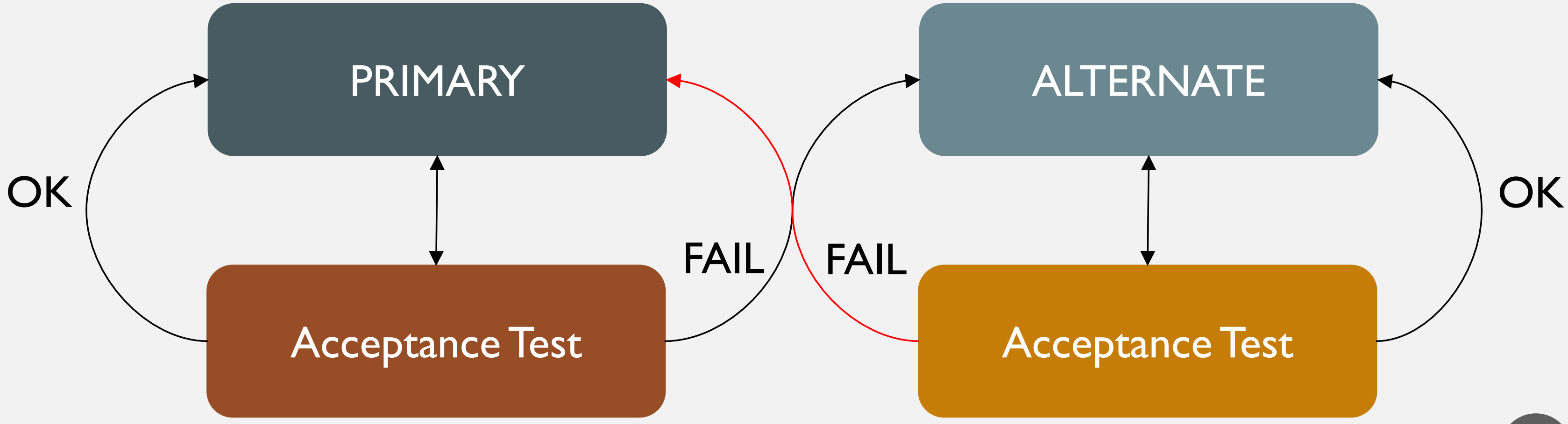
- Primary VM runs as no failures are detected.
- Alternate VM takes place when a failure occurs
- Acceptance tests are adjusted independently to guarantee system operation
- Alternate learn from Primary and become more robust to failures/attacks experimented by primary



CREATION OF REPLICAS

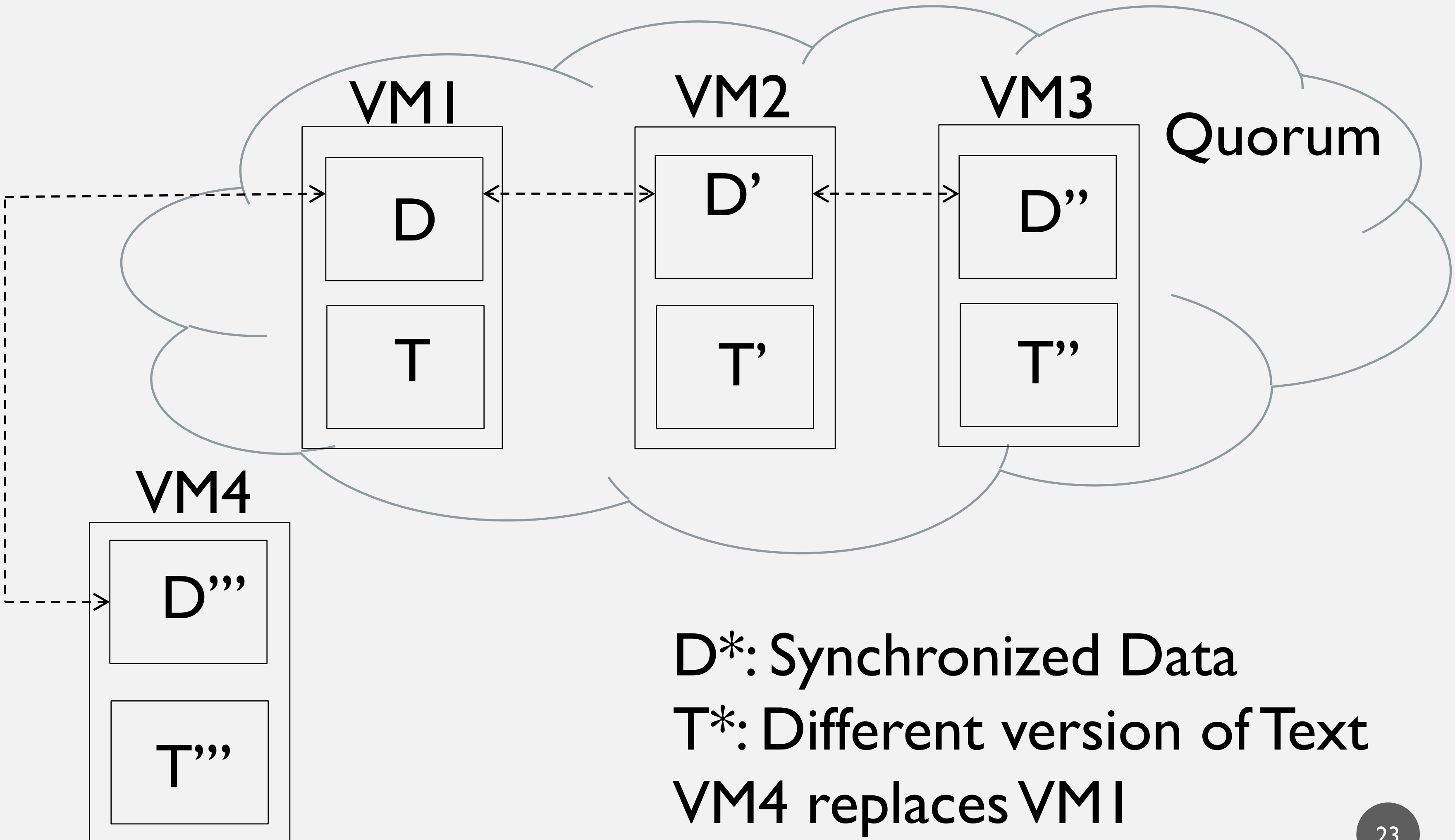
Challenges:

- Reduce downtime when Primary is replaced by Alternate and vice versa
- Keep the state of the machine (either Primary or Alternate) after the replacement to achieve uninterrupted operation
- Keeping the state (stateful reincarnation) allows the system to be application-agnostic



CREATION OF REPLICAS

Stateful Reincarnation Ideas:



D*: Synchronized Data

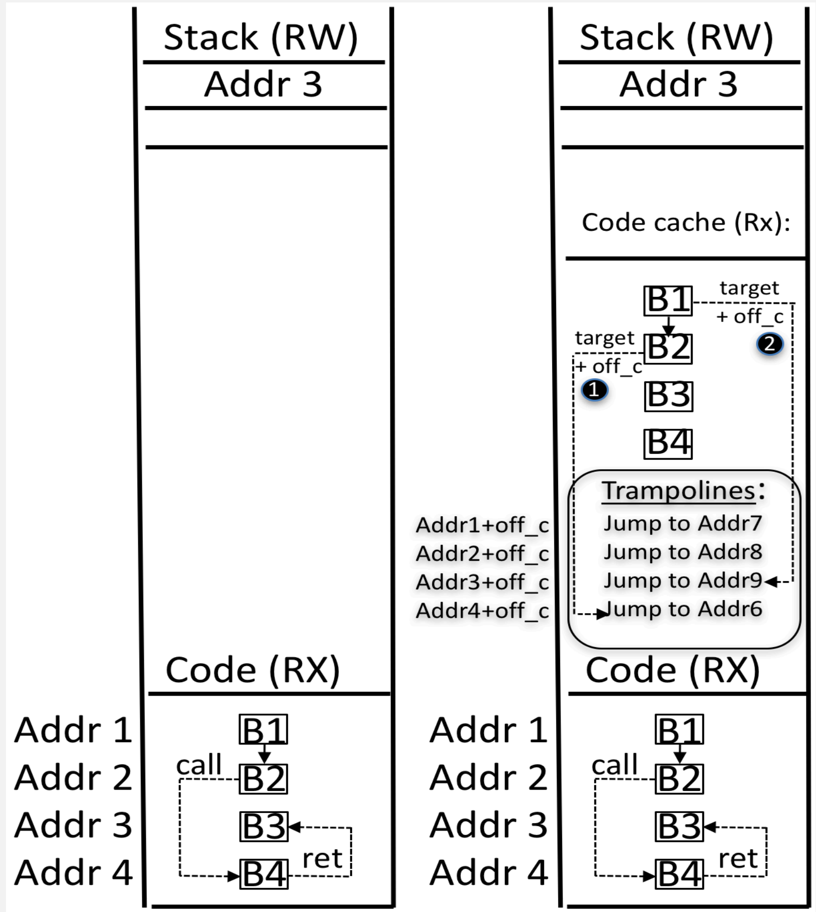
T*: Different version of Text

VM4 replaces VM1

CREATION OF REPLICAS

Stateful Reincarnation Ideas:

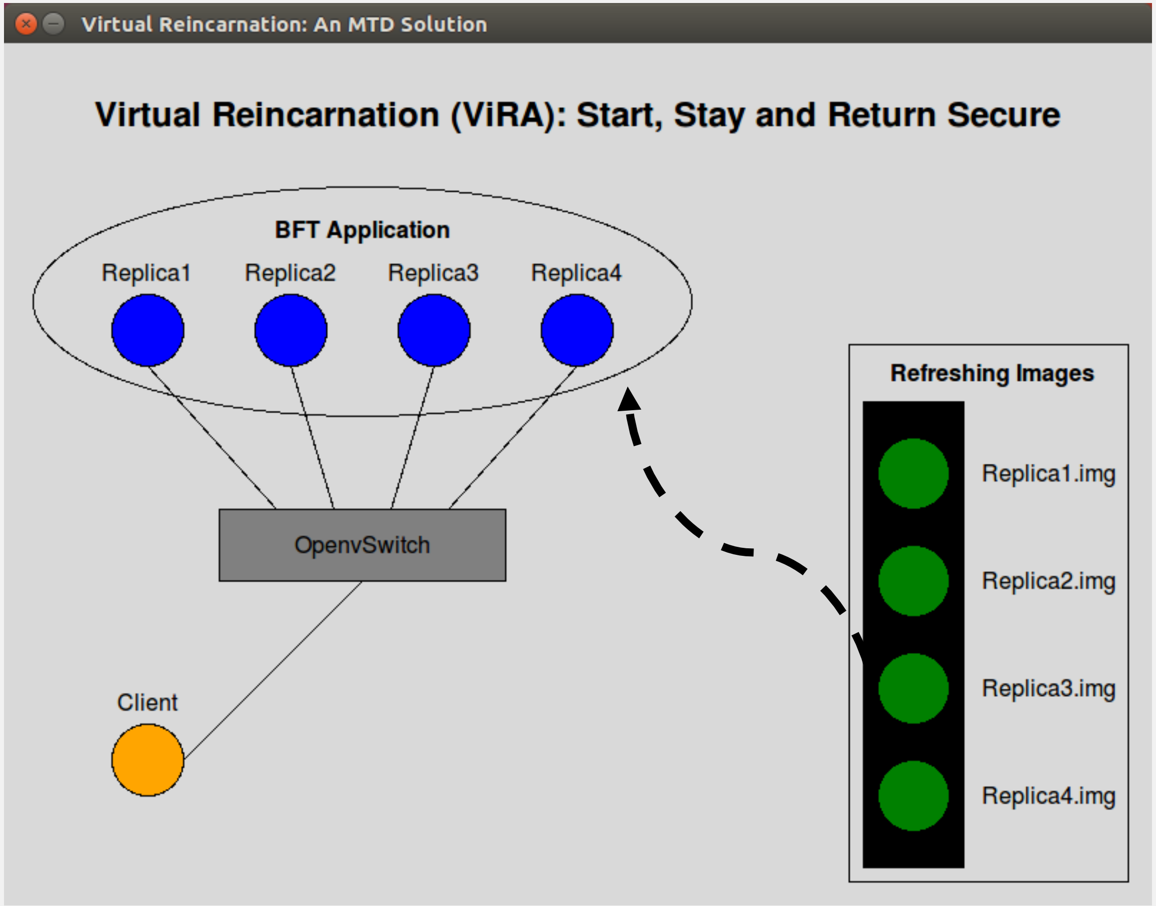
- Create different versions of binaries
- The original code is kept and set with read-only permission so that it can be used as part of the reference to the new locations of the blocks in the re-randomized version.
- We avoid identifying and updating code position pointers in each randomization process by keeping a table of trampolines as shown in (b). Each block is located at a fixed offset (i.e., **off_c**) with respect to the trampoline table.
- The pointers (in the original code space) are dynamically redirected to its respective address in the code variant when it is de-referenced



(a) (b)

Z. Wang, C. Wu, J. Li, Y. Lai, X. Zhang, W. Hsu and Y. Cheng. ReRANZ: A Ligh-Weight Virtual Machine to Mitigate Memory Disclosure Attacks. To appear in VEE2017.

VIRTUAL REINCARNATION



"Active machines are replaced by new ones with a totally new image"

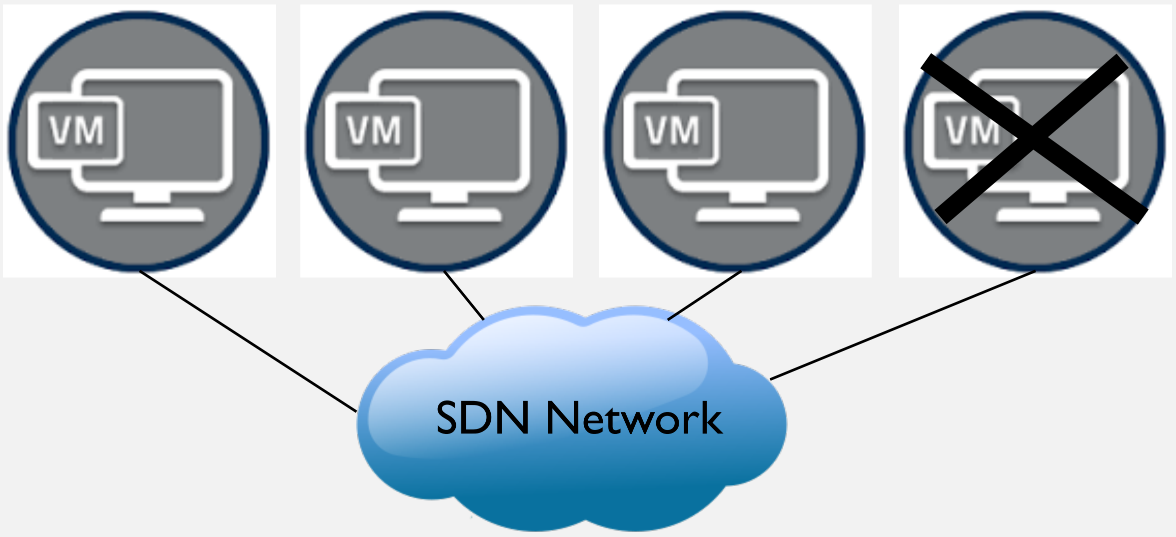
https://www.dropbox.com/s/fqjh75su0p908ic/NGCRC-2017-Bhargava-DEMO2.mp4?dl=0

PROACTIVE MONITORING

- Operates at the hypervisor level
- Helps for performing node reincarnation effectively rather than blindly
- *Based on Virtual Machine Introspection* (VMI)
- Proactively gathers live memory data (at host OS) in intervals and reacts if anomalous behavior is detected
- Use libvmi library for introspection with negligible performance overhead
  - When application is hijacked, address offsets show new entries for injected code
  - When application is terminated and a new malicious one created, it could end up with a different process ID or memory address offset

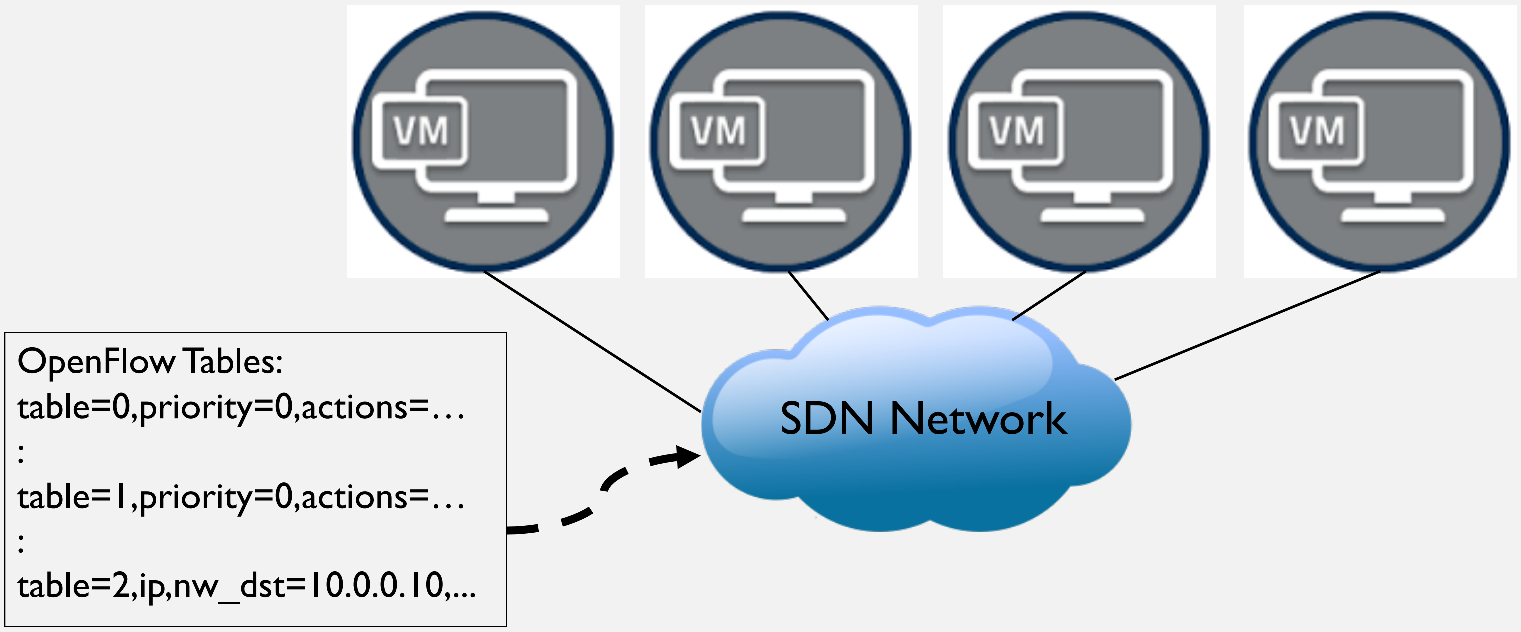
SDN NETWORK DYNAMICS

- Network devices are reconfigured via OpenFlow on-the-fly
- New added flows redirect traffic intended for the old machine to the new machine



SDN NETWORK DYNAMICS

- Network devices are reconfigured via OpenFlow on-the-fly
- New added flows redirect traffic intended for the old machine to the new machine

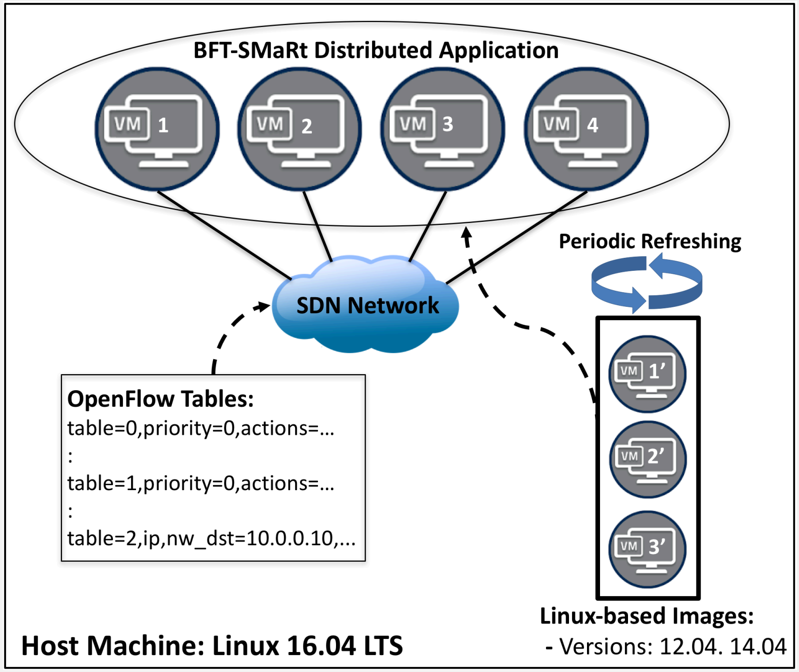


SDN NETWORK DYNAMICS

- New machines can be integrated to the system with their own IP addresses
- No waiting for the IP address of the old machine
- Downtime is reduced

MEASUREMENTS

- A Byzantine fault tolerant (BFT-SMaRt) distributed application was run on a set of Ubuntu (either 12.04 or 14.04 randomly selected).
- VMs run in a private cloud, and are connected with an SDN network using Open vSwitch
- The reincarnation is stateless, i.e. the new node (e.g. VM1') does not inherit the state of the replaced node (e.g. VM1).
- The set of new VMs are periodically refreshed to start clean and the network is reconfigured using OpenFlow when a VM is reincarnated to provide continued access to the application.



MEASUREMENTS

1. **VM restart time**: Time it takes the machine to respond to be full operational since it is started.
2. **Virtual creation time**: Time to create the new image of the VM.
3. **Open vSwitch flow injection time**: Time it takes to inject new flows to Open vSwitch

Measurements	Times
VM restart time	$\sim 7s$
VM creation time	$\sim 11s$
Open vSwitch flow injection time	$\sim 250ms$

Note: that the important factor for system downtime here is the Open vSwitch flow injection time, as VM creation and restart take place before the reincarnation process

MEASUREMENTS

- Aim to estimate the time it takes the new machine to be full operational.
- VM creation and restart take place before the reincarnation process
- The important factor for system downtime here is the Open vSwitch flow injection time

FUTURE WORK

- Enhanced live monitoring techniques
- Instrumentation to measure overhead more accurately
- Test other stateless applications on the MTD framework
  - E.g.: Upright (Public and Subscribe System)

FUTURE WORK

- Stateful Virtual Reincarnation Support:
  - Can we preserve the state of the virtual machine during the reincarnation process to make the solution application-agnostic?
  - Test the framework with Secure SOA Services (stateful reincarnation)

PRESENTATION AND PUBLICATIONS

1. NGC Cyber Resilient Systems IRAD (http://www.northropgrumman.com)
2. Enterprise Resiliency IRAD (http://www.northropgrumman.com)
3. Ahmed, N., and Bhargava, B. Towards Targeted Intrusion Detection Deployments in Cloud Computing. In the Int. Journal of Next-Generation Computing Vol. 6, No 2, IJNGC - JULY 2015.
4. N. Ahmed. Design, Implementation, and Experiments for Moving Target Defense. PhD Thesis, Purdue University, 2016.
5. N. Ahmed and B. Bhargava. From Byzantine Fault-Tolerance to Fault-Avoidance: An Architectural Transformation to Attack and Failure Resilience. To Appear in IEEE Transactions on Cloud Computing, TCC 2016.
6. N. Ahmed and B. Bhargava. Disruption-Resilient Publish/Subscribe: A Moving Target Defense Approach. The 6th International Conference on Cloud Computing and Services Science, CLOSER 2016.
7. N. Ahmed and B. Bhargava. Mayflies: A Moving Target Defense Framework for Distributed Systems. 3rd ACM workshop on MTD in conjunction with ACM Conference on Computer and Communications Security (CCS), Vienna, 2016.
8. R. Ranchal, D. Ulybyshev, P. Angin, and B. Bhargava. Policy-based Distributed Data Dissemination. *CERIAS Security Symposium, April 2015* ***(Best poster award)***
9. V. Pappas, M. Polychronakis and A. Keromytis. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization." In IEEE Security and Privacy (SP), 2012.
10. L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing. 1978.
11. M. Carvalho and R. Ford. Moving-target defenses for computer networks. IEEE Security & Privacy 12.2 (2014).