

**Department of Computer Science**

**Senior Year Project Report**

Cyclone: Deterministic Replay of Distributed Systems

Asad Alavi  
Hassan Shahid Khan  
Muzammil Abdul Rehman  
Zaid Saeed Khan

**Table of Contents:**

1. **Introduction to Cyclone 3**
2. **Contribution Overview 4**
3. **Research on Deterministic Replay 4**
4. **GUI 7**
5. **Overview of Timekeeper and LIME 9**
6. **Prototype 10**

# Introduction to Cyclone

Cyclone is a large-scale, long-term US Department of Defense project that aims to make the cloud more secure against adversaries. It leverages recent advancements in virtualization to encapsulate the cloud into “enclaves” and manipulate the environment in which the software executes. It will support dynamic capabilities such as controlling time (speeding up/slowing down/pausing processing), shape (mapping between physical & virtual cloud) and inputs (intercepting/restricting them) of the cloud. However, this requires the cloud to have functionalities such as cloning, rewinding, fast forwarding, replaying, breakpointing, pausing, stepping, isolation and migration, all of which are currently not supported. Virtual machines support some of these functions at single host level. The Cyclone project aims to find a way to extend these functionalities to bring the benefits of single-machine virtualization to entire clouds. Our group’s Senior Year Project contributes to the larger Cyclone project in two key ways: 1) determining a way to enable efficient deterministic replay for the cloud, which is an integral component of the Cyclone Project, and 2) building a viable proof-of-concept prototype of Cyclone by combining the different components of the stack. For the first part, we have looked into previous research in the domain and suggested combining two existing approaches to enable cloud-wide replay. We also explored a novel approach particularly suited to the DoD’s resource intensive setting. For the second part, we worked on integrating the different components of Cyclone by determining a multi-layer stack for the prototype.

# Contribution Overview

* Gained in depth context and background on nondeterminism in multiprocessor host setting and distributed system setting
* Conducted an extensive literature review
* Evaluated existing systems to determine which approach suits Cyclone best (concluded that speculative execution approaches provide minimal overheads)
* Proposed combining two existing approaches, DoublePlay and DEFINED, at the Host and Network level respectively, to enable deterministic execution for the cloud
* Proposed a theoretical alternate host level deterministic replay system (multiple clones)
* Built a comprehensive GUI for Cyclone
* Looked into ways of taking snapshots in Xen as a short term stand in for deterministic replay
* Gained context on other components of Cyclone: Live Migration and Time Dilation
* Currently building a prototype by piecing together LIME (Live Migration of Ensembles i.e. network of VMs) and Timekeeper (tool to slow down perceived time)

# Research on Deterministic Replay

Computer execution is inherently nondeterministic due to dependence on several random events like varying disk latency, device interrupts, lock acquisition order, system calls, data races etc, which cause different executions to diverge. Some of these events are intrinsic to computer execution due to their external (to the application) nature, while another type is due to multithreading and multiprocessor settings, whereby threads intertwine nondeterministically. On top of this, the network adds another layer of nondeterminism due to dropped/corrupted packets, out of order delivery, variable latency etc. Deterministic replay means ensuring that two different executions of a program take the exact same path (with respect to some granularity). Making deterministic replay possible for a whole cloud system includes doing so independently for both the hosts and the network. Previous research in this area has targeted only one of these at a time. There are three main approaches in the literature to ensuring deterministic replay at either level:

* Comprehensive Logging: Log all the information required to enable the execution to be replayed later (offline). But this has extremely high storage space requirements and usually imposes large time overhead (due to recording) as well.
* Deterministic Execution: force the host/network to execute deterministically relative to inputs by eliminating all sources of “internal” determinism (i.e. permitting only one serial path per input set) and logging all “external” nondeterministic input. However forcing deterministic execution, especially at the software level, slows down execution significantly.
* Speculative Execution: use some “guiding” principle to direct execution in a certain direction, rollback to last acceptable state when speculation fails (i.e. when execution takes a path other than the speculated/expected one), and replay from there. This helps reduce storage overhead by not requiring complete logging, while at the same time not imposing too much of a performance overhead by letting the system execute as normal (as opposed to restricting completely as in deterministic execution) and rolling back when required.

We concluded that speculative execution is most suitable for Cyclone due to its hybrid nature of limiting both time and storage overheads. Existing speculative execution solutions are available separately for host level and network level replay, and although they have issues such as being designed for different purposes than that of Cyclone, they can possibly be used with small modifications. We proposed combining one existing method for each level to enable deterministic replay of the whole distributed cloud. The approaches are briefly described below:

**DoublePlay (Host Level):**

Runs a “uniparallel” second execution on spare cores in addition to the conventional execution. In this second execution, all the threads are timesliced onto single processor, with different ‘slices’ running concurrently on different processors. This makes replay of the multithreaded program similar to that of a conventional single thread by eliminating the need to log shared memory accesses, so that only the order of the timeslicing of threads needs to be logged. It is implemented as a modified linux kernel and glibc, but is not designed for distributed applications, and the source code is also no longer maintained.

**DEFINED (Network Level):**

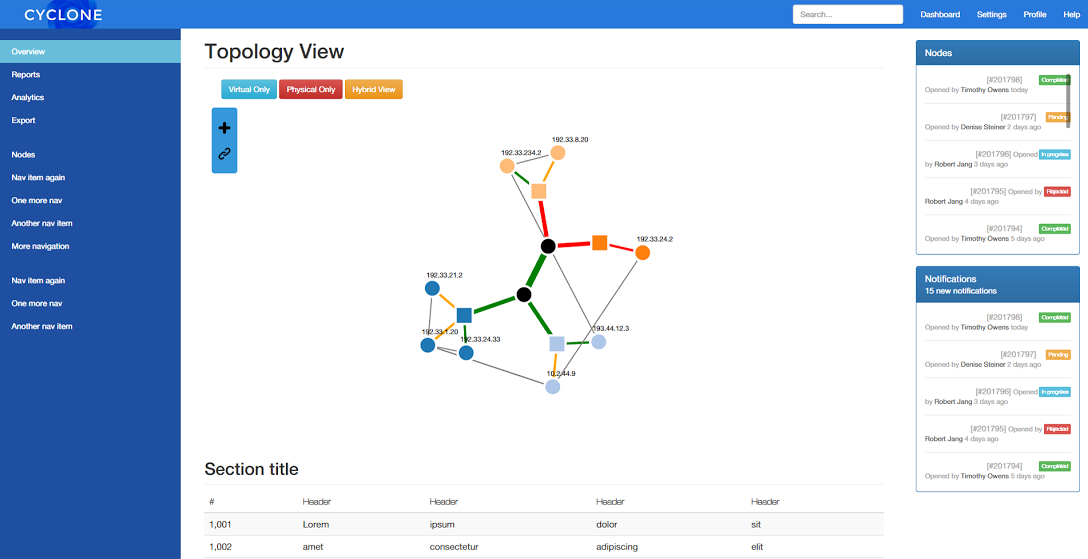
Provides deterministic execution of (just) the network for interactive control plane debugging. It is implemented as a library/shim layer below the control plane software. It uses an optimal ordering function to compute the expected order of arrival of packets across the network. It does not impose any restrictions on the packets at the time of sending, speculatively allowing them, and using rollback & replay when the expected order is violated. Its time overhead is minimal, but the tail is long when multiple rollbacks are required. Other concerns are that it hasn’t been cited much in literature (only 3 times), is meant for control (i.e. infrequent) traffic and is built on an outdated kernel.

Despite the drawbacks, our conclusion is that in the domain of existing approaches, these are the two most suitable to be combined for use in Cyclone. As an alternate to existing approaches for host level determinism, we proposed our “Multiple Clones” idea to leverage the abundance of computational resources available to the DoD. It was pursued only to an initial conceptual exploration phase before focus shifted to the prototype, and still has large conceptual gaps that need to be filled before implementation can be considered. The details are outlined below.

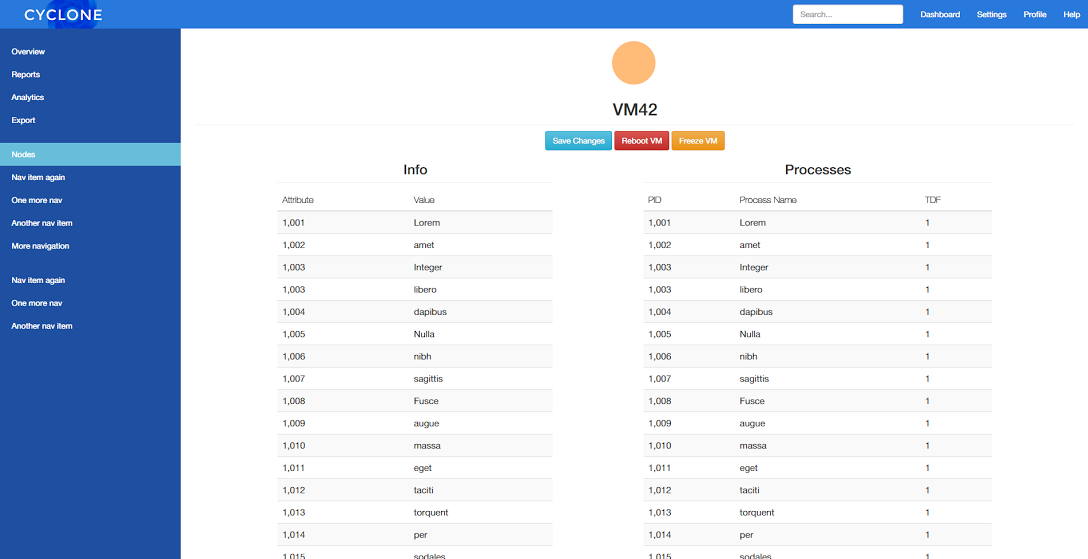
**Multiple Clones Proposal:**

The idea builds upon stable multithreading (from a previous paper called PARROT), which reduces the number of possible execution schedules in order to mitigate the state space explosion when computing the possible paths an execution can take. Our proposal was to log checkpoints of the program during normal execution and run multiple clones concurrently during replay, each of these running one of the limited number of permissible schedules. At each checkpoint, only those clones that match the state of the original would be maintained; all the others would be discarded. Multiple copies would be made of these remaining clones, and the process would be repeated for each interval. We also suggested using program specialization on inputs (from previous work by the NSG group on Occam) at each checkpoint to further reduce number of possible subsequent executions.

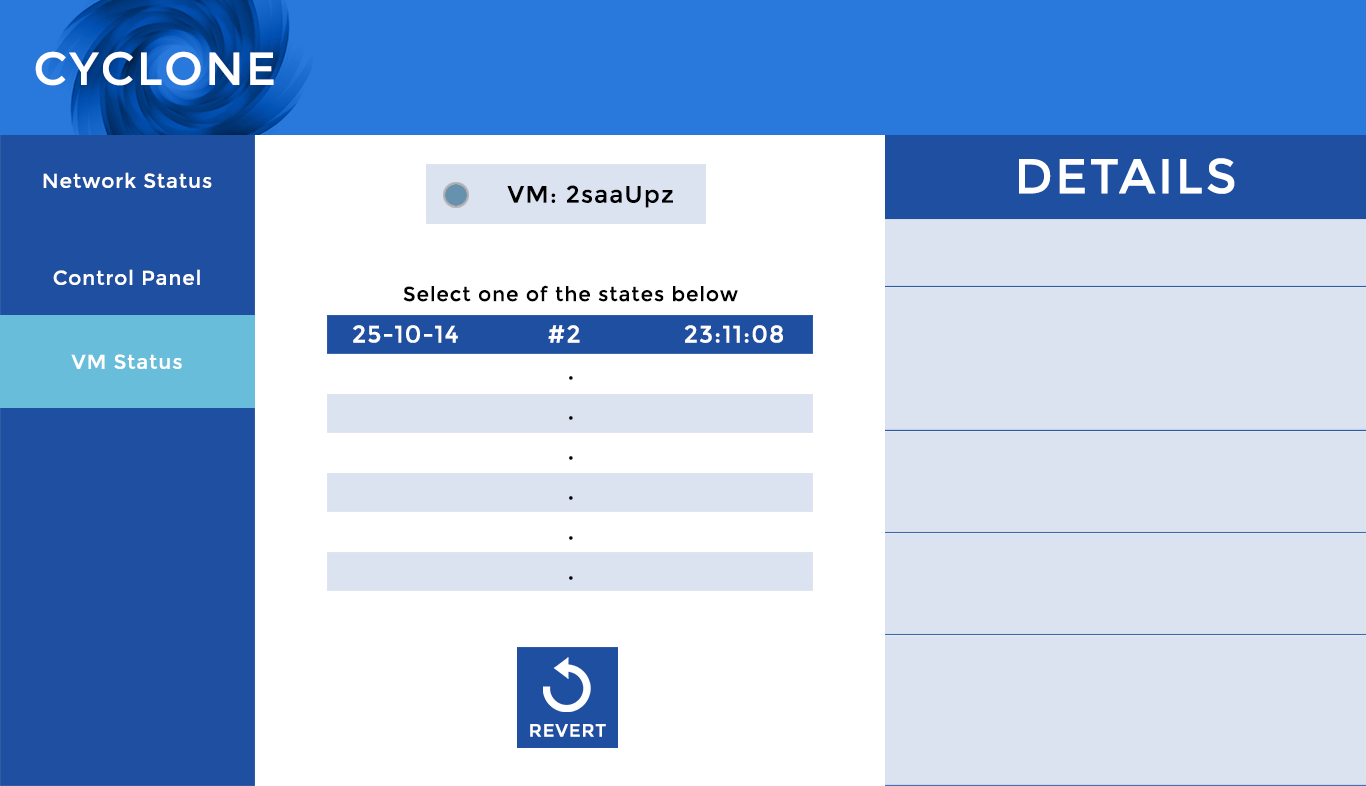
# GUI



The main page features the interactive topology in the center and with details and notifications about system activity on the right. At core of the User Interface for Cyclone is a visualization of the Network Topology. The visualization can be used to then access the various features of the Cyclone system such as Cloning, Migrating, Time-Dilation, etc. The square nodes represent physical nodes and the circular nodes attached to these are Virtual Machines. The black circular nodes represent switches. The topology can be filtered layer-wise (Virtual only, Physical Only) using the buttons on the top as well. The view also allows the user to add VMs and create links between Virtual Machines as well using the toolbar on the left.



When one of the nodes is clicked it redirects to that Node’s dedicated page. The page allows a user to do certain administrative tasks on the Virtual Machine such as Reboot, Remove and Pause and also access Cyclone-Specific functionality such as Time Dilation. The list of process allows a user to slow down individual processes using the Cyclone Kernel. The table on the left is populated with information and statistics about the VM.



Another page would allow the user to rewind a VM to past states in a manner similar to what is shown in the picture above.

The Final GUI will tie in with the system and allow users to easily operate the Cyclone system.

# Overview of Timekeeper and LIME

**LIME:**

Virtual machine migration is crucial to Cyclone as it allows seamlessly moving applications to a different set of physical resources (part of the “changing shape” design goal). This is important in case of an attack on the physical hosts, and must be done without taking the application offline as this would lead to packet loss and inconsistent network state. LIME (LIve Migration of Ensembles) allows migration of a complete network of VMs, along with their underlying network and management system. LIME works with SDN, Cloning the data-plane state to a new set of switches, then incrementally migrating VMs. It maintains synchronized state during migration, with both networks delivering traffic. This allows migration of a network of VMs without freezing the ensemble for significant time. The prototype is built on the Floodlight Openflow controller and has currently very limited functionality.

**Timekeeper:**

Slowing down a host, i.e. making it perceive the external network to be executing faster, allows simulation of “fast-forwarding” the cloud execution, which is required for the “changing time” design goal of Cyclone, more specifically fast forwarding to predict future host state in case of an attack. This requires embedding the VM in virtual time. Timekeeper is a lightweight set of modifications to the Linux kernel that gives VMs a notion of virtual time that can be slower or faster than real time. It modifies the gettimeofday() system call to return virtual time based on a time dilation factor, as well as some other system calls such as sleep. Timekeeper is built on the Linux 3.10 kernel which is outdated, causing significant problems with its integration with other components, which are often not compatible with the kernel.

# Prototype

The prototype for Cyclone will use simple VM snapshots (with rollback) as a very coarse stand-in for deterministic replay in the short term. Our goal was to integrate the two main components of Cyclone that have already been developed (Timekeeper and LIME) into a bare-minimum proof-of-concept model. The system stack is shown below in the figure.

Timekeeper is implemented in the kernel of the guest VMs (we are using KVMs, kernel virtual machines, as they are lightweight) on the Cyclone Network. Mininet, a network emulator, is used to give the sense of a real network since setting up a physical network would have been too time consuming and resource intensive. However, OpenVirtex, which is the network virtualization platform used by LIME, requires an underlying physical topology (it has not been successfully tested with an emulated network before). Attempting to get the mininet emulated physical topology integrated with OpenVirtex has proved extremely complicated. The KVMs must also be bridged to the emulated Mininet topology; this component is also in progress. For our prototype, due to lack of resources, we will use LIME to migrate the virtual network from and to the same emulated topology. For the purposes of the prototype, the GUI will not be included.