EMUEDGE: A HYBRID EMULATOR FOR REPRODUCIBLE AND REALISTIC EDGE COMPUTING EXPERIMENTS

A Thesis

by

YUKUN ZENG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Radu Stoleru
Committee Members,	Dilma Da Silva
	I-Hong Hou
Head of Department,	Dilma Da Silva

December 2018

Major Subject: Computer Science

ABSTRACT

Numerous recent research efforts have been devoted to edge computing due to its key role in enabling emerging IoT applications. Prior to deploying edge technologies to real world environments, they need to be adequately tested, validated and tuned on a testing platform. However, to the best of our knowledge, a testing platform for edge computing that provides both networking and computing realism with low costs is still missing. In this thesis, we propose EmuEdge, a hybrid emulator based on Xen and Linux's netns for full-stack edge computing emulation. Supporting both containers and VMs, EmuEdge is the first that takes advantage of both OS-level and full system virtualization in edge computing emulation. The hybrid design of EmuEdge ensures on-demand isolations on both computation and networking while maintaining the flexibility of scaling with lightweight containers. Besides, our system supports real-world network replay and is fully configurable with EmuEdge APIs. Through extensive experiments, we prove that EmuEdge provides realistic computation isolation and network fidelity comparing to state-of-the-art emulators. We also demonstrate EmuEdge's compatibility with an actual edge computing platform and the emulation results are qualitatively similar to physical experiments.

DEDICATION

To my beloved families and Yang,

you are the ones who always understand.

ACKNOWLEDGMENTS

First and foremost, my sincere gratitude to the greatest advisor in the world, my dear fellow BMWer, Prof. Radu Stoleru for his advice and guidance throughout my Master's study. His patient instructions and tireless teachings lightened my way in the dark of research exploration.

Then it comes with my committee members, Prof. I-Hong Hou and Prof. Da Silva, this thesis wouldn't have been done so perfectly without their valuable insights and comments. With that being said I also wish I am not the worst student they've ever advised, which feels so true to myself.

Furthermore, I would like to thank all the CSE and OGAPS officers for their help all the way in the final examination process.

Thanks for Stoleru Group (LENSS), including Mengyuan, Wei, Chen, Ala, Akshay, Keishla, Liuyi, Mahima and others I may have missed. Their future feedbacks on my defense dry-run are greatly appreciated and I wish them good luck in finding the next ~~coffee guy~~ master student.

Special thanks to Mengyuan for his beautiful plots to appear in our paper, this thesis also owes him a huge debt. Also to Bingqian for her questionless contributions in the preliminary experiments, most of the experiments I insisted on running actually make ~~no~~ perfect sense.

Thanks also to the previous researchers in Mininet, Xen, Linux, Open vSwitch and many others, those are the giant shoulders that our work stands on.

Thanks to my past advisor Prof. Xuefeng Piao and my previous lab mates, their hardworking spirit still encourages me to work along the night today.

Thanks to my best friend in US: Lei Gao, Guanlun Zhao and the mysterious man who came from the same alma mater as Lei. I would miss so much fun if living a life without them.

And of course, many thanks to my families in China, who've always been my harbour in the storm no matter where I am. Lastly, I must thank one name in particular. Which is the best colleague, coauthor, friend and my second family, Yang Liu, whose unconditional support makes my MS journey possible.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Radu Stoleru and Professor Dilma Da Silva of the Department of Computer Science and Engineering and Professor I-Hong Hou of the Department of Electrical and Computer Engineering. Part of this thesis is rewritten from a paper manuscript (in submission) coauthored with Mengyuan Chao and Radu Stoleru. All other work conducted for the thesis was completed by the student independently.

Funding Sources

This work was performed under the following financial assistance award 70NANB17H190 from U.S. Department of Commerce, National Institute of Standards and Technology.

NOMENCLATURE

OGAPS	Office of Graduate and Professional Studies at Texas A&M University
TAMU	Texas A&M University
IoT	Internet of Things
AR	Augment Reality
VR	Virtual Reality
SDN	Software Defined Network
OS	Operating System
VM	Virtual Machine
NIC	Network Interface Controller
MStorm	Mobile Storm

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Motivation

According to the statistics last March, the amount of Apps on Android and iOS has exceeded 5M and keeps growing continuously. Among those applications, computationally intensive ones have begun to take lead. Typical future mobile Apps would be integrated with sensor-related functions like face recognition, heart rate detection as well as other new technologies such as Machine Learning, AR/VR, etc, enabling people to do more with their mobile devices with better user experience. However, the growing computational demands from applications also cast challenges to mobile device processing capabilities due to their inherent limitations on volume and power. With the same intention of traditional cloud computing systems [1, 2], mobile cloud computing [3, 4, 5] was then proposed to augment mobile devices with additional computational capability from the cloud by offloading computations to datacenters. Since most complicated computing tasks are offloaded to the cloud [6], mobile applications can provide indifferentiated services to devices with limited computing power. Recently, a new computing paradigm called edge computing, is proposed to further optimize cloud computing. As an extension of cloud computing, edge computing can greatly reduce backbone traffic by bringing the control of computing applications, data, and services away from the core to the edge of the Internet. Furthermore, exploiting computational sources physically nearby can reduce service latencies since network time is shortened. Though not until recent has edge computing been proposed and formalized, it is indeed a longheld idea to offload computations to nearby devices. Soon after cloud computing is introduced, a virtual cloud computing provider is proposed to enable file and computing resource sharing between vicinity mobile devices [7], however, the actual experiments conducted only consist of rough Hadoop setup on iPod Touches. Besides, dedicated mobile cloud platforms are also developed for different computational needs such as portable MapReduce in [8, 9, 10] for batch processing and data analysis workloads, Mobile Storm [11] for real-time stream processing. Such efforts in pushing intelli-

gence, data storage and processing capabilities closer to data originations (usually sensors, mobile devices and smart appliances in IoT literature) formed a new computing paradigm called edge computing, also referred to as edge-clouds, fog computing, cloudlets and etc. Edge computing can be regarded as the extension of utility computing concept proposed in cloud computing [12], and a key technology driven by market that various future applications would rely on [13]. Despite various advantages of edge computing, several major challenges have to be resolved before it can realize its full potential:

- **Privacy and Security:** edge computing is designed to fullfil the computational requirements of IoT and is involved with billions of householeld devices that generate sensitive information. Any communication on user privacy information should be well protected through encryption and data to be offloaded should be limited for privacy concerns.

- **Programmability:** datacenters in traditional clouds usually consist of homogeneous servers in terms of both hardware and OS, therefore software and programs can be shared easily due to the homogeneity. However, edge computing systems usually contains clusters of devices running on heterogeneous hardware and OSes which require platform-specific programming efforts. Therefore, a more general computing platform has became a necissity to resolve programmability issues in edge computing.

- **Offloading strategy:** in edge computing, low-level IoT devices such as sensors can choose to offload computations between edge, fog and cloud core. Intelligent strategies needed to be proposed and studied to satisfy the distinctive constrains (e.g., latency, energy, etc) of different edge computing applications.

Since no standard has been reached in this field yet, discussions on edge computing architectures are still on going, research efforts are devoted to various directions such as resource allocation [14, 15], computation offloading [16, 17, 18, 19] and security aspect [20, 21, 22]. However, before the actual deployment of the new research advances, or for example, a comprehensive edge computing platform, they must be well debugged, tuned and validated in terms of scalability, effi-

ciency, fault-tolerance and etc. Therefore, in this thesis, we are specially interested in the testing and validation of edge computing technologies, which is the key to facilitating the development of the whole edge computing ecosystem.

Hassles with Edge Computing Development: though edge computing helps in satisfying growing computing demands and reducing network traffic in the backbone, the testing and validation of such platforms are not easy if at all possible. Challenges in edge computing platform validation process include the following aspects:

- **Efficiency:** an edge computing platform always consists of a cluster of heterogeneous mobile devices and servers, operating on all of them manually in a test is inefficient, especially in the case that interactions with the mobile devices could rarely be automated.

- **Cost:** with on-demand public cloud services like AWS and Google cloud we no longer have to pay for large infrastructure bills for experiments on PC. However, this is not true with edge computing scenarios. A large-scale edge computing experiment would require a cluster of heterogeneous devices spanning from servers, PCs, mobile phones to IoT sensors, which could turn into a huge cost.

- **Heterogeneity:** edge computing platforms are typically composed of heterogeneous devices ranging from servers, network equipments, mobile devices to sensors, this means a high system complexity and heterogeneity. This adds up to costs due to larger varieties of devices to consider, and also reduces efficiency in requiring heterogeneous operations across platforms.

- **Network Complexity:** Different from previous cloud paradigms, edge computing naturally involves with a wider range of devices which results in a network complication. If combined with IoT infrastructures, an edge computing platform may be involved with hybrid networks including sensor networks, mobile networks and backbone networks. This implies mobility, different network protocols, complex topologies and traffic patterns as well as hybrid connectivities with wired/wireless medium, all of which contributed to difficulties in validating edge computing systems.

- **Reproducibility:** As motivated in Mininet [23], reproducibility is a major concern in networking researches. We argue that reproduction is the first and most significant step in identifying and understanding a problem for both academia and industry, across all fields of study. From the author's experience in the industry, companies are willing to invest huge manpower and funds purely for reproducing a customer bug. Reproducing a problem on edge computing platform can be extremely hard due to the high heterogeneity and network complexity.

Apparently, realistic testing of edge computing platforms will lead to tremendous monetary and time costs. Therefore, an easy-to-deploy test environment has became a necissity in the development and improvement of edge computing paradigms.

Despite numerous researches in edge computing field to improve the overall performance in various aspects, few focused on easing the difficulties in debugging, testing and validation process. iFogSim [24] is the pivoting test environment for edge computing, it models applications as DAGs and simplifies all elements in edge computing as sensor or actuator, which brings up many limitations and results in unrealism due to the nature of simulator. EmuFog [25] is a fog computing emulator developed based on Maxinet [26], which enables customization of fog computing infrastructure from scratch. However, EmuFog, Maxinet as well as their predecessors in [23, 27, 28] mostly focus on network emulation such as topology design, traffic control while the equally-important computation plane in edge computing are ignored. In this paper, we investigate the possibility of testing, prototyping and emulating actual edge computing platforms within lab settings in both computation and network perspective. After first identifying the differences between the testing of traditional network systems and edge computing platforms, we present a hybrid emulator called EmuEdge which extends from Mininet with support for distinctive edge computing platform characteristics. Similar to Mininet, we hope our work can bring insights leading to a new edge computing development paradigm, where large scale physical experiments in edge computing can be simplified to initing a topology definition file in a lab server.

1.2 Related Work and Background

In this section, we first present existing experimental tools for networking and edge computing and summarize their advantages and disadvantages. Then, we outline the design objectives of EmuEdge. Finally, we briefly introduce background on Xen [29], the full-system virtualization technology adopted by EmuEdge to supplement heterogeneity vacancy in previous emulators.

1.2.1 Edge Computing Experimental Tools

The experiments for edge computing mainly consists of two parts: networking - with communication delay, bandwidth, drop rates and jitter as its metrics; and computing - with computation delay and throughput as its metrics. Currently, most experimental tools for edge computing focus on networking, as they are evolved from the previous networking experimental tools. In this subsection, we introduce related work on experimental tools for both networking and edge computing by categorizing the existing work as follows:

Testbeds: The ideal way of testing a system is through reproducing the actual scenarios on a physical testbed. There are several networking testbeds that can be adapted for edge computing experiments, such as NCR [30], Emulab [31], Deterlab [32], PlanetLab [33], StarBED [34], GENI [35], etc. These testbeds make a large number of machines and network links available and use tools such as Dummynet [36] and NIST Net [37] to configure network link properties such as delays, drop rates, jitters [38]. There are also pure edge computing testbeds, such as Cumulus [39] and the SCC TestBed [40], which consist of a large spectrum of heterogeneous devices, OSes and network links. Although testbeds provide the most realistic experimental results, they are costly to build and maintain. Besides are limited by practical resource and replication limitations, and lack the flexibility to support experiments with custom topologies [38]. Additionally, the difficulty of reproducing problems and errors on physical testbeds has always been an unresolved issue.

Simulator: Discrete-Event simulations have been widely applied in network researches, renowned simulators such as Glomosim [41], NS-3 [42], OPNET [43] provide a cost-effective way for network prototyping. Their experiments are reproducible and convenient, but the models for their

hardwares, protocols and traffic generation patterns may raise fidelity concerns [38]. Besides, although these network prototyping tools are useful for edge computing simulation to some degree, they are limited to network simulation by nature, other factors such as computational realism are ignored. With the purpose of simplifying evaluation specifically for edge computing, iFogSim [24] and its extension [44] are proposed to model IoT and edge computing environments and measure the impact of resource management strategies. However, they simulate the network behavior through models based on assumptions and simplifications, which always leads to non-realistic results.

Emulators: Emulators are able to automatically configure and set up reproducible experiments emulating real world scenarios. Comparing to simulators, emulation usually incurs similar infrastructure cost while achieving better realism since it can run real code without changes. With recent advances in SDN such as Linux netem [45], Open vSwitch (OvS) [46] and OpenFlow [47], configuring network topologies and link properties in emulators has become possible. There are mainly two types of emulators. One supports *container-based emulation*, such as vEmulab [48], NetKit [49], Trellis [50], CORE [51], Mininet [23] and its descendants Mininet-HiFi [27], Mininet-WiFi [28] and Maxinet [26], which employ light-weight OS-level containers to achieve good scalability by sharing a single kernel. Based on Maxinet, an edge computing emulation framework named EmuFog [25] is also proposed to enable large-scale fog computing experiments by augmenting the preceding work with fog infrastructure design capabilities. Container-based emulators are cost-effective and promising in scalability, however they are based on *partial virtualization* and cannot emulate heterogeneous OSes. The other type of emulators support *full-system emulation*, such as ModelNet [52] and DieCast [53]. Such platforms use VMs as hosts to enable node heterogeneity and resource isolation. This is essential for edge computing emulation, as there are usually lots of heterogeneous devices in an actual edge computing scenario. Besides, full-system emulation supports live migration and cloning, which can be helpful in configuring and scaling process.

Different from previous work, EmuEdge is a ***hybrid*** emulator that aims to combine the features

Type	Simulator	Emulator-Con	Emulator-VM	Testbed	Hybrid
Example	*iFogSim*	*EmuFog*	*DieCast*	*Cumulus*	*EmuEdge*
Topo flexibility	✓	✓	✓		✓
Link realism				✓	✓
Traffic realism		✓	✓	✓	✓
Resource realism			✓	✓	✓
OS realism			✓	✓	✓
Functional realism		✓	✓	✓	✓
Easy replication	✓	✓	✓		✓
Low cost	✓	✓			✓
Good Scalability	✓	✓			✓

Table 1.1: Edge computing experimental platform features

of simulators, testbeds and emulators together, thereby providing varying degree of realisms for ***realistic and reproducible*** edge computing experiments.

1.2.2 EmuEdge Objectives

In order to create realistic and reproducible edge computing experiments, a platform needs to have the following characteristics:

Topology flexibility: The platform should be able to easily create experiments with different topologies or even dynamically changing topologies at runtime.

Traffic realism: The platform should be able to generate and receive real, interactive network traffic to and from the Internet/local network. The traffic between two hosts should go through network devices (switches or routers) the same way as in the real world.

Link realism: The platform should be capable of controlling the link quality of each link, such as delay, bandwidth, drop rate, etc., according to the real world link quality trace.

Resource realism: The platform should be able to emulate heterogeneous devices in the edge computing paradigm by allocating isolated computing resources to different hosts based on their actually available resources.

OS realism: The platform should be able to emulate devices with different OSes in the edge

computing paradigm by installing different hosts with different OSes.

Functional realism: The platform should be able to execute the same code as in the real devices.

Easy replication: It should be easy and fast to replicate an experimental setup and run an experiment.

Low cost: It should be inexpensive to set up different experiments in both money and time.

Good Scalability: The platform should incur little overhead, so that it can scale well when the required hosts increases.

Table 1.1 shows the comparison of above characteristics between exiting experimental tools and our EmuEdge. As we can see, simulators such as *iFogSim* can provide flexible topology, easy replication, good scalability with low cost. However, simulators are usually limited in fidelity due to their simplified models and unrealistic assumptions. Although the experimental results of testbeds such as *Cumulus* are convincing, they lack flexibility and are costly to setup and maintain. Container-based emulators such as *EmuFog* can ensure the link, traffic and functional realism with low costs and good scalability. However, they are typically based on OS-level virtualization, which cannot support OS heterogeneity and has no guarantees on resource realism. VM-based emulators such as DieCast [53] generally provide better resource and OS realism. However, they are usually considered to have inferior scalability and incur higher costs.

Different from all existing platforms, our goal is to design a new emulator platform that achieves all the above characteristics. With a VM-based emulator as its main component, EmuEdge also supports container-based hosts and allows real devices to be added to the emulation. Moreover, it enables configuring the network topology and link properties based on the synthetic traces generated by different simulators. To achieve this goal, EmuEdge employs Xen virtualization, which is briefly introduced in the following subsection.

1.2.3 Xen Architecture

First proposed in [29], Xen is now the state-of-the-art opensource virtualization platform that is adopted widely, majorly due to its scalability, OS neutrality, high performance and lightweight

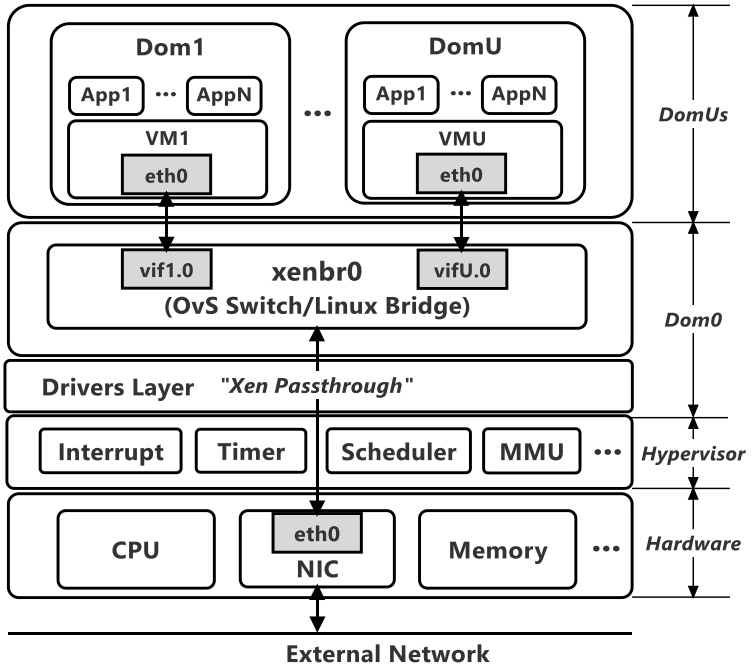


Figure 1.1: Xen internals and networking

features. The fact that Xen is adopted in various public clouds like AWS, GoGrid and Aliyun makes it possible to deploy and scale Xen-based platform leveraging pay-as-you-go services.

Xen Basics: Generally, a Xen instance can be divided into several levels as shown in Figure 1.1. The two distinctive components of Xen are domain and hypervisor. In Xen, a running VM instance is usually referred as a domain. Xen Hypervisor is a software layer that manages all hardware resources of the physical machine, except I/O devices. Xen allows domains to directly control physical devices such as NICs, disks, using PCI Passthrough. A special domain, namely Dom0, is designed as a control domain that contains drivers for all devices as well as a toolstack for managing DomUs (guest VMs).

Xen Networking: Xen is also equipped with powerful networking capabilities, Linux bridge is the standard networking mode adopted in Xen. On boot, Xen Dom0 creates a bridge, for example *xenbr0*, that connects to each physical NIC. Per startup of a new DomU, Xen generates a virtual interface (mostly referred to as *vif*) in Dom0 that links to the virtualized NIC in DomU and con-

nects it to a specified bridge created before, thereby all traffics from DomUs can be directed to the specified physical interfaces. With recent advances in SDN, another networking approach based on OvS [54] is also introduced in Xen, which differs from the standard mode in that the Linux bridge is replaced with a OvS switch to support more SDN features such as OpenFlow [47]. Bridges and OvS switches can also be created independent from physical interfaces, thereby enabling internal LANs between multi VMs. The case in Figure 1.1 is actually equivalent to Figure 1.2, through *xenbr0*, all Dom0 and DomUs are bridged to the same external subnet with physical NIC *eth0*.

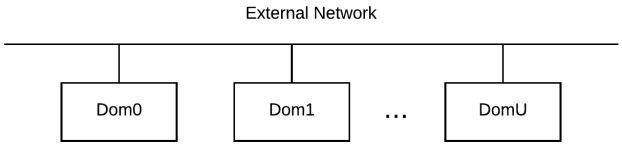


Figure 1.2: Physical equivalence to virtualized network in Figure 1.1

VM Snapshot/Clone[†]: one major advantage of VM comparing to other lightweight virtualization approaches is the live migration capability. Snapshot captures disk and memory state of the VM and has now become a standard in virtualization due to its importance in healthy state backup and restore. Interestingly, we found it equally useful in accelerating edge computing platform validation on EmuEdge. A snapshot of a well-configured VM instance (e.g., ready to run applications) can be scaled quickly to hundreds which could save us from manually repeating operations on physical devices. While several types of Snapshot are supported in Xen, we are mainly concerned about:

- **Disk-only snapshots:** as suggested by the name, disk-only snapshots only captures metadata and virtual disk storage for a VM, allowing exporting and restoring VM states for backup purpose. This type of snapshots require no support from VM itself and is crash-consistent.

[†]Snapshot is a feature of XenServer that Xen Project doesn't official support, however through LVM2 tools Xen can achieve the same functionalities as illustrated at https://wiki.xenproject.org/wiki/Xen_FAQ_High_Availability

- **Disk and memory snapshots:** besides VM configuration information and storage data, disk and memory snapshots captures the VM RAM state exactly by the time the snapshot is made. Therefore a VM instance is able to restore to a previous running state as is without rebooting. This kind of snapshot usually requires support from the host OS, but it is commonly provided in mainstream OSes.

In fact, snapshot is treated as a special VM that needs further provisioning for booting. XenAPI provides the same *clone* API for snapshot and vm. VM *clone* can be regarded as the process of snapshotting a VM and provisioning. Another Xen operation called *copy* may seem to be synonym of *clone* at the first glance, but they actually differ significantly. *clone* leverages Copy-on-Write to reduce operation time while *copy* incurs an immediate copy of the entire disk. In EmuEdge, VMs typically boot and scale through *cloning* a well-configured snapshot for its performance superiority.

1.3 Our Approach

At the first glance, both simulators and emulators can significantly reduce the cost of notoriously expensive edge computing testing thereby facilitating the development process. However, simulators usually require additional efforts to accomodate real systems and they are usually too simplified to capture real world details. Previous emulators such as Mininet can either be satisfactory for edge computing due to higher heterogeneity. Therefore, we propose EmuEdge, a new emulator designed for heterogeneous edge computing environments for better realism in both computation and network perspective. With EmuEdge, we promote the significance of on-demand realism in achieving faithful emulations with low cost.

Computation realism is the degree of computation isolation and heterogeneity. Higher degree of such realism can be applied when a computational intensive node is to be emulated such as a Hadoop slave node while a lower level may be sufficient for a network bounded device.

Network realism represents the degree to which EmuEdge links behavior like real world ones. EmuEdge provides several approaches for users to emulate real world link qualities, such as rate limiting, random losses and network replay from traces.

Hybrid integration is the ability of EmuEdge to combine virtual topologies with physical nodes. Physical nodes connecting to EmuEdge can interact with internal topologies through real links therefore specific hardware issues may be also discovered through EmuEdge experiments. External nodes integrated to EmuEdge are also considered the highest degree of realism.

1.4 Introduction

The major contributions of our work can be summarized as follows:

- We comprehensively studied the possibility of heterogeneous edge computing platform emulation through Xen in terms of scalability and realism. The experiment results demonstrated great gap on network performance between virtual and physical environment.

- Design and implementation of EmuEdge, a hybrid edge computing emulator which extends traditional emulators with heterogeneous system and hardware integration support for realistic edge computing experiments with low costs.

- Comprehensive approaches with link asymmetry and quality control, QoS tuning, network trace replay to address the network equivalence between EmuEdge and real-world.

- Two suites of APIs to easily define, interact and share edge computing prototypes with full details such as network topology, link qualities and VM configurations.

- Emulation fidelity validation of EmuEdge with a practical edge computing platform, in which our measured results reflected the real world performance with high fidelity.

The remainder of this paper is organized as follows: In Chapter 2, we present our preliminary thoughts on leveraging full-system virtualization for edge computing emulation as well as the challenges and limitations in our approach. Afterwards, we tackle these problems in Chapter 3 with a comprehensive design of a novel hybrid emulator called EmuEdge. Besides, we also showed the performance realism of EmuEdge in both network and computation perspective comparing with state-of-the-art emulators and real-world experiments. In Chapter 4, we demonstrate the compatibility of EmuEdge with actual edge computing applications. Finally we conclude our

work with a comprehensive discussion on both advantages and current limitations of EmuEdge in Chapter 5.

2. PRELIMINARY OBSERVATIONS ON VIRTUAL MOBILE STORM

Motivated by various difficulties of general testing and validation process in development of edge computing systems that rely on both networking and computation, we aim to develop a system based on Xen to extend the idea of SDN prototyping and emulation systems such as Mininet [23] to more realistic heterogeneous platforms.

In this chapter, we manage to emulate heterogeneous edge computing platforms with Xen, following the full-system virtualization choice of DieCast [53]. Then in our preliminary experiments, an actual edge computing application called MStorm [11] is successfully deployed on Xen. However, we also observed the network inequivalence between emulated system and real-world, which indicates the unrealisms to validate a practical edge computing system on Xen.

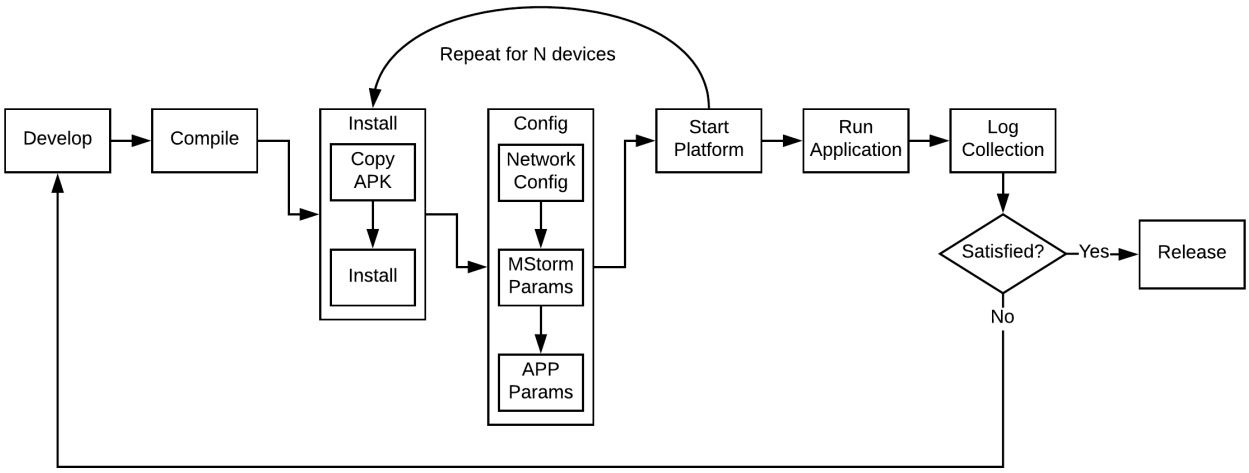


Figure 2.1: The development iteration of mobile storm

2.1 Case Study on Mobile Storm

Mobile Storm is a distributed real-time streaming processing platform on the edge of network [11]. Instead of offloading to the cloud, Mobile Storm is designed for mobile devices to offload

computational tasks to nearby computing resources, thereby greatly reducing backbone network traffics and improving real-time stream processing performance. In this section, we consider a software lifecycle of a typical edge computing platform Mobile Storm, where it's being iteratively developed, tested, debugged and finally released. The workflow described above can be shown specifically in Mobile Storm's case as in Figure 2.1.

Complexity of Interacting with Mobile Devices: in the validation process of each Mobile Storm development iteration, we have to repeatedly install, configure and start applications on multiple devices ranging from a master server to several mobile devices as shown in Figure 2.1. On the server side, the workflow described above can be somehow automated through scripts. However, with heterogeneous mobile devices, a perfect automation is not possible and the workflow turned into a complex set of hands-on interactions. With Mobile Storm, the average time needed for the whole deployment workflow is approximately 1 minute per device. This might seem minor at the first glance, however, in a scalability experiment where hundreds of devices are needed, hours will be taken to simply setting things up. The trial and error cost thus become tremendous for edge computing platforms such as Mobile Storm.

Reproducibility Concerns: like most edge computing platforms, the Mobile Storm system is fragile due to its mobility nature. This could be attributed primarily to two aspects:

- **Incompleteness of Mobile OS:** most of the OSs running on mobile devices, such as Android, have to trade off reliability and completeness for limited computing capabilities and energy constraints.

- **Unreliable Network:** network communications in the case where mobile devices are present rely heavily on unreliable wireless networks.

Therefore, any differences in OS configurations and network conditions might lead to software errors. Reproducing and correcting those platform-specific and network-specific problems requires at least similar OS and network conditions, if not exactly the same condition.

Cost Analysis: the Mobile Storm platform requires a server to run ZooKeeper and the master

server, this could be on any laptop or lab PC platform-independent thus we ignore the costs on the server side. However, the scalability and compatibility of Mobile Storm have to be tested with multiple heterogeneous mobile devices. Using a state-of-the-art mobile device such as Galaxy S8 would result in a cost of $599 per device, this can accumulate easily to tens of thousands when scaling over 16 devices.

System Heterogeneity: an ideal validation of a Mobile Storm system should include tests on heterogeneous devices. Besides, reproducing real-life software errors often requires us to go beyond the experiment setup and debug on specific device models, which is especially true in industry product development. The need for additional heterogeneous devices would further increase costs.

2.2 Mobile Storm Setup with VMs on Xen*

Observed the difficulties in validating and debugging Mobile Storm with physical devices, we seek to ease the process with virtualization. The most original ideas we have is limited to the scope of Mobile Storm virtualization by simply replacing physical devices with VMs to reduce the hardware costs, however this indeed inspired our following work on the design and implementation of EmuEdge. In the following sections, we will first walk through some internal designs of Xen which are fundamental to understanding our work. Then the Mobile Storm networking and deployment details on Xen will be discussed. Finally we compare between physical and virtual approachs for Mobile Storm deployment and summarize the limitations of the virtual approach.

2.2.1 Deploying Mobile Storm on Xen

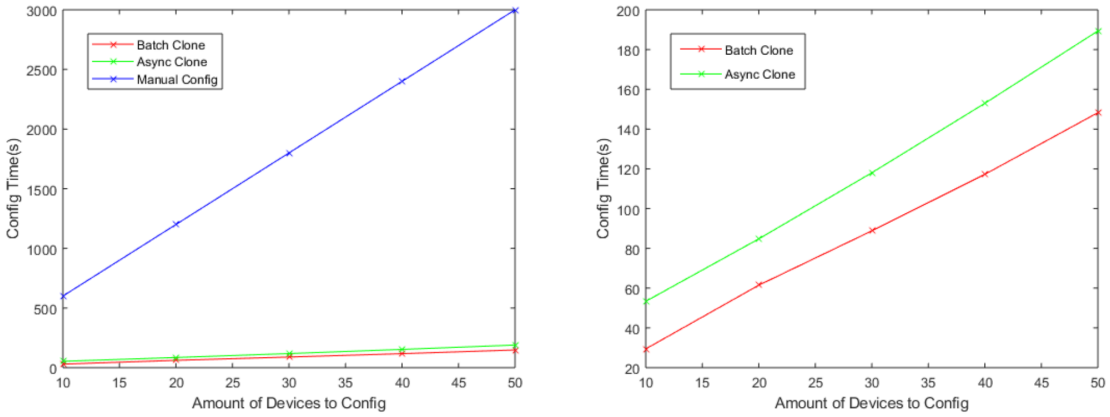
Similar to the development flow show in Figure 2.1, the Mobile Storm code will be compiled and packaged for installment on devices. However, with Xen Snapshot, the tedious repetitions on different mobile devices now turn into a simple clone process. The advantages of leveraging virtualization over physical devices in validation environment setup time are shown in 2.2. The

*The experiments in this paper are conducted on XenServer instead of the open source Xen Project. XenServer is a commercial distribution of Xen provided by Citrix Systems, Inc. We don't discriminate between them since most key concepts involved in the scope of this paper can apply to both.

cost of a typical small-scale virtualization setup is trivial, the Xen system demonstrates extraordinary compatibility and 2 old PCs from 2006 (Intel Core 2 and 4GB RAM) were able to run Xen and 5 Android VMs on top. In fact, any PC with virtualization support in CPU can be leveraged for hardware assisted virtualization. Even better, paravirtualized VMs can be ran on PCs without virtualization support, however OS kernel support will be needed in this case. With a larger experiment scale, a tremendous amount of time can be saved. Figure 2.2 shows the configuration time trend of 3 different setup approaches, with one physical and two different virtualization setup approaches. When scaling to a medium scale with 50 mobile devices, virtualization saved us ~47.5 min throughout the process. Furthermore, the difficulty in reproducing on specific hardware or software configurations can be solved to some extent. With Xen it's possible to instantly control over computing capabilities like memory and CPU to emulate a device with limited resources, a system specific errors might be reproduced by installing a new VM with target ROM (ROM refers to the firmware on a mobile device in our case). However, Xen is still limited in platform specific emulations. From our experiences, applications on Mobile Storm that requires Snapdragon API support simply crashes in VM due to incompatibility of CPU architecture. Though the batch cloning process reduces configuration time by 80% comparing to manual setup, we attempted to further improve efficiency by using XenServer Async API to issue tasks running in parallel. However, the Async API didn't perform as expected and demonstrates large fluctuations with higher time consumption on average. This probably can be attributed to the overhead of task scheduling and I/O bottleneck for storage migration. We believe the Async XenAPI would still be useful if heterogeneous workloads (such as combination of I/O and CPU bound tasks) are parallelized through it.

2.2.2 MStorm Networking on Xen

As discussed in Section 2.1, edge computing systems like Mobile Storm are renowned for high network complexity in multiple aspects. Both the testing, validation and possible reproduction environment require a realistic network setup. A typical Mobile Storm scenarios as shown in the left part of Figure 2.4 involves with several common network features with edge computing:



(a) Configuration time scales with amount of devices for different setup approach

(b) Zoomed in for cloning performance comparison between batch/async

Figure 2.2: Configuration time for setting up MStorm with different approaches

- **Hybrid Topology:** The Mobile Storm network consists of a wired network through which the router is connected to master server, and a wireless network that brings mobile devices together.
- **Mobility:** The computational contributions from devices are totally at will and targeted users are fully mobilized, which means they may leave and join at any time.
- **Unreliability:** Presence of interferences and noises in the wireless network can lead to unreliable wireless communications, such as delay, jitter, packet loss and corruption, intermittent transmission, etc.

With the physical Mobile Storm network being discussed, we summarize that problems might occur on a realistic Mobile Storm system due to the network complications. Therefore, an ideal edge computing validation platform should be able to preserve network conditions as is instead of purely connecting things together. From the topology perspective, virtualizing Mobile Storm doesn't change the network since all android VMs are running on the same subnet (10.0.0.0/24)

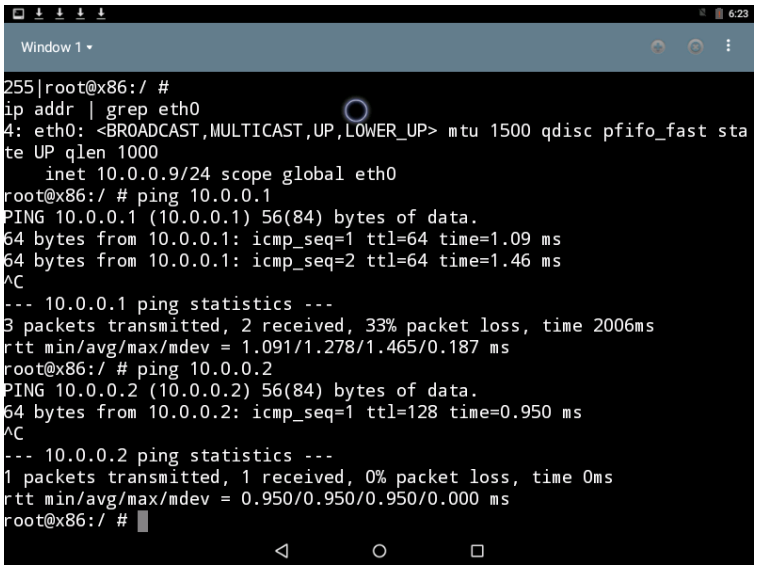


Figure 2.3: Network connectivity of a MStorm VM on Xen

with the router (10.0.0.1/24) and master server (10.0.0.2/24). Figure 2.3 shows the network status of a slave node, PING activities proved that interconnectivities are well preserved.

However, this might not be the case in terms of network quality. Two major differences in network that might greatly influence the realism of virtual Mobile Storm are:

Network Media: The right half of Figure 2.4 shows that in the virtualized system all Android VMs are connected through the default linux bridge *xenbr0* while in a physical cluster mobile devices are connected through wireless networks. The supposedly wireless connections between mobile devices become reliable in virtual Mobile Storm.

Centralized vs. Distributed Network Exit: All domains on *xenbr0* relies on the centralized physical interface *eth0* to travel beyond XenServer while in physical setup distributed wireless network interfaces are present on all mobile devices, this might bring unrealism on packet queuing.

2.2.3 Traffic Pattern Analysis in Mobile Storm

To better understand how network influences a edge computing platform like Mobile Storm, in this section we aim to identify the common traffic patterns in Mobile Storm and demonstrate how a network change can transform into a issue in a software's perspective.

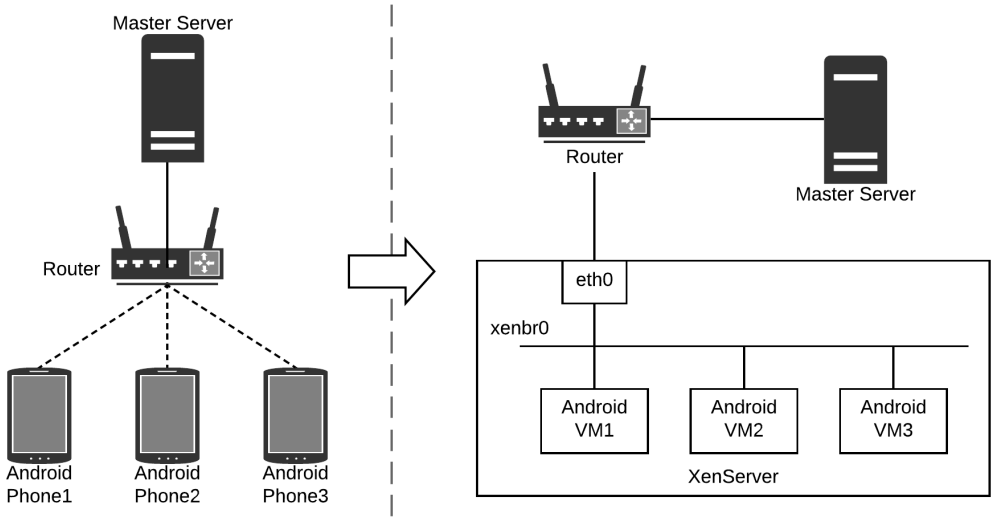


Figure 2.4: Physical(left) vs. virtual(right) MStorm cluster setup

Mobile Storm Components: typically a Mobile Storm system consists of a master server and a cluster of mobile devices referred to as slave nodes. The master server runs ZooKeeper for distributed coordination of slave nodes. Considered as a centralized task scheduler, the master server collects heartbeats from slave nodes to gain a global knowledge of the cluster status, based on which it can allocate and schedule tasks accordingly based on each slave node's workload.The network traffic patterns in actual Mobile Storm system can be categorized as follows:

- **Master-to-Slave:** a Master-to-Slave transmission happens when the master server transmits scheduling decisions and task input data to slaves through the wireless link. In such transmissions, though the reliability can be guaranteed by using TCP, the instability of wireless link might lead to high latencies in delivering scheduling decisions, low bandwidth would also result in higher response time as the input data cannot be transmitted timely.

- **Slave-to-Master:** typical Slave-to-Master transmission includes the regular heartbeat from slave to master server (sometimes referred to as status report), in some cases, the execution results might also be returned to master server through the wireless link. Late arrivals of heartbeat might lead to wrong scheduling decisions while delay of returning data lengthens

the average response time.

- **Slave-to-Slave:** one thing in common among various distributed and cloud computing platforms is the interaction between different slaves. The intercommunication between those slaves in the cluster can be transmissions for intermediate results, job/task migrations per scheduling requests, etc.

Apparently from the discussions above, the performance of Mobile Storm will be greatly influenced by the network. Under the wireless scenarios, duplication, corruption and loss of streaming data or management information, delays may influence scheduling decisions, all of which might result in performance degradation or even system failures. However, in the virtualized mobile cloud, either the highly reliable wired network or virtual I/O between VMs on the same server can correctly reflect the properties of a realistic wireless network. In the next section, we empirically prove this argument and show different characteristics between virtualized and actual wireless network in both 3 types of transmissions mentioned above.

2.3 Quantifying Network Inequivalence

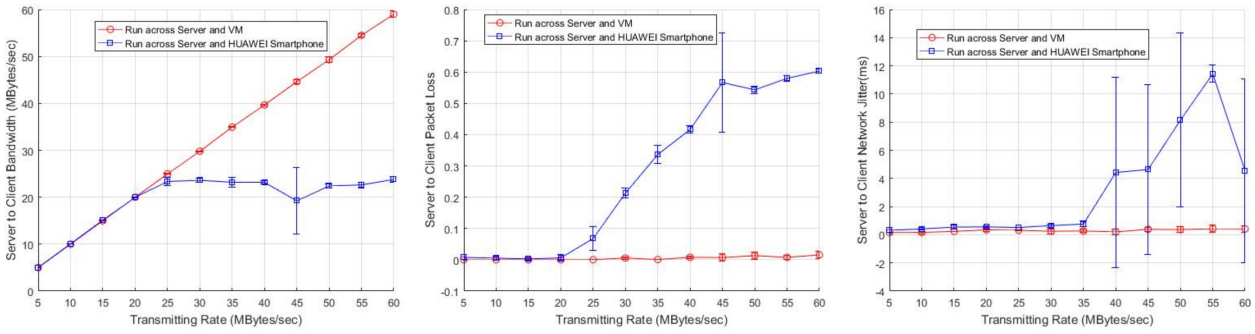
In this section, we demonstrate the network quality gap between physical and virtual Mobile Storm systems by measuring several common network metrics for each of the aforementioned traffic patterns.

Measurement Metrics: in our experiments, bandwidth is measured to show the long-term average performance and capacity while two other metrics packet loss and jitter are chosen primarily to reflect the reliability of the network.

Experiment Methodology: we regard each traffic pattern as a directional transmission to test independently. For each pattern, we setup a pair of iperf server/client at the sender and receiver respectively. Then we stress the network with different load (by controlling the sender rate from 5 MB/s to 60 MB/s) and measure the actual performance using metrics described above.

2.3.1 Master-to-Slave Network Characteristics

The metrics we measured under different traffic load for Master-to-Slave transmissions are shown in Figure 2.5. The figure shows a promising bandwidth around 25 MBps for physical Master-to-Slave network, this makes sense since the wireless media is preserved for the mobile phone exclusively. However, with the rise of transmitting rate at the sender, the packet loss rate and jitters also increase. In the meantime, variations in the wireless network become considerable after the 25 MBps threshold is exceeded while the virtualized network remains stable the whole time.



(a) Master-to-Slave Bandwidth (b) Master-to-Slave Packet Loss (c) Master-to-Slave Network Jitter

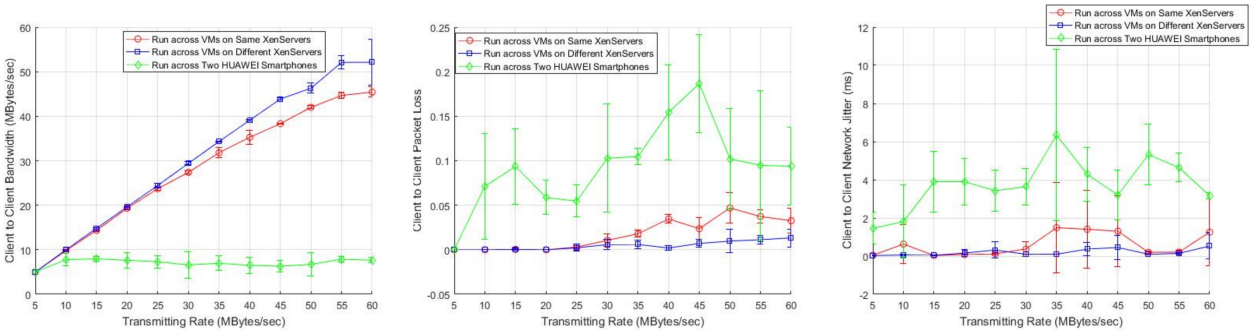
Figure 2.5: Master-to-Slave network characteristics

2.3.2 Slave-to-Slave Network Characteristics

The metrics we measured under different traffic load for Slave-to-Slave transmissions are shown in Figure 2.6. Different from the other two patterns, an additional case is considered for Slave-to-Slave traffic pattern. The underlying reason is that for Slave-to-Slave transmission in virtualized environment, the slave nodes can be sitting on the same physical server or two different servers.

Immediately after a short period of increasing bandwidth at the beginning in Figure 2.6, the wireless network is overloaded and the maximum bandwidth that it can achieve stays at around

8MBps. From the reliability perspective, considerable packet losses and jitters are observed from the start of wireless test. For the other two virtual network, they remained similar performance comparing to Master-to-Slave virtual network. As expected, the bandwidth of transmitting between VMs across two Xen servers (45 MBps) is lower than on one internal Xen network (52 MBps). The unreliability of wireless network is highlighted by consistently high packet loss and jitters in this case, which can be attributed to the share of the same wireless media by two mobile devices. For short, the wireless link in the actual scenario yields much lower bandwidth, produces a lot more jitters and packet losses, and is highly unstable.

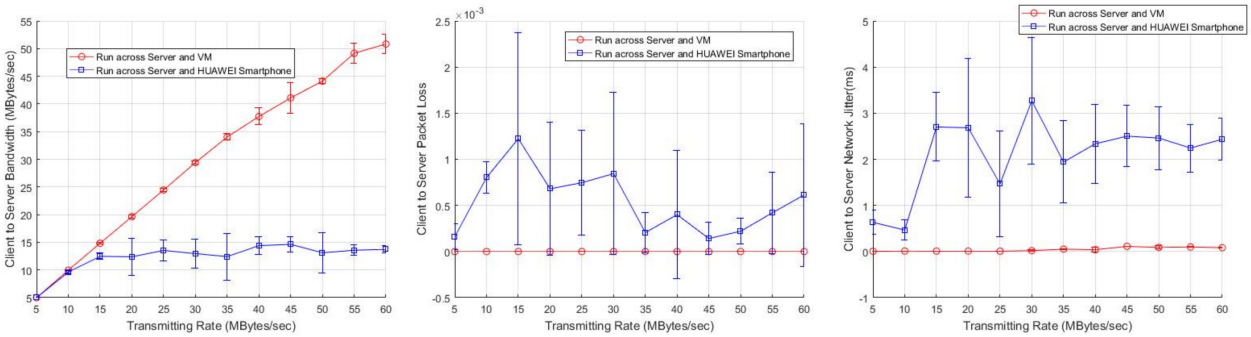


(a) Slave-to-Slave Bandwidth (b) Slave-to-Slave Packet Loss Rate (c) Slave-to-Slave Network Jitter

Figure 2.6: Slave-to-Slave network characteristics

2.3.3 Slave-to-Master Network Characteristics

The metrics we measured under different traffic load for Slave-to-Master transmissions are shown in Figure 2.7. Through the measurement figures, we can see that the results look better than the Slave-to-Slave case in all metrics, thanks to fewer interferences and media share. However, the bandwidth in this case is still not comparable to Master-to-Slave, for which we argue that it is a common thing to have a lower uplink bandwidth for wireless routers.



(a) Slave-to-Master Bandwidth (b) Slave-to-Master Packet Loss (c) Slave-to-Master Network Jitter

Figure 2.7: Slave-to-Master network characteristics

2.4 Conclusion

The takeaways from our preliminary attempts of migrating to a virtualized testing and validation environment can be summarized from two aspects:

Advantages: the Xen virtualization approach we proposed in this section has the following advantages:

- **Handle Heterogeneity:** different from the Mininet alike systems that leverages kernel-bounded partial virtualizations, we based on Xen to provide much better heterogeneous system support. Simply through installing target systems on VMs or configuring CPU/Memory on Xen we can provide a better approximation for example to reproduce a software error in specific case.

- **Live Migration:** the utilization of Xen Snapshot/Clone functionalities enable us to migrate or backup VMs as is, this can be useful in sharing hardware/software configurations and problem reproduction.

- **High Efficiency:** a Xen Snapshot can be also helpful in scaling an edge computing validation test, experiments showed in previous sections that the efficiency can be improved by 80% comparing the virtualization method to a manual operations.

Limitations: despite the advantages of our proposed solution, the current method does have

limitations that prevent it from emulating a realistic edge computing platform, in particular, the following problems must be addressed before applying our solution in production:

- **Scalability:** the Xen based approach is most efficient comparing to manual setup when a large scale of test is needed. However, a larger scale would require more computing capability especially when full-system virtualization is adopted. In Xen, memories allocated for each VM are preserved even when the VM is idle, therefore a minimum of around 32GB RAM would be necessary to emulate 30 VMs with 1GB RAM each (and around 2GB for management purpose in Xen Dom0).

- **Unrealistic Network:** in the simple Mobile Storm scenarios, it is shown that Xen is able to bring VMs and external nodes together in a network. However, network conditions in that case are much simplified in terms of network quality and is not a ideal reflection of real world. Furthermore, edge computing platforms can have much more complexer network topologies which cannot be defined easily on Xen.

In the following sections, a complete SDN based edge computing prototyping system called EmuEdge will be presented. Designs and approaches will be described to resolve both problems we mentioned above.

3. EMUEDGE: DEFINING NETWORKS ACROSS HETEROGENEOUS NODES

In this chapter, we present the design and implementation of EmuEdge, a hybrid emulator tailored specifically for edge computing prototyping from both computation and network perspective. The core idea of EmuEdge is to extend traditional Mininet alike system with compatibility for hybrid virtual and physical nodes to support heterogeneous computing platforms in one network. The remainder of this chapter will be organized as follows: First we motivate the need for a hybrid edge computing prototyping platform and review the literature for related work. Then we discuss insights and limitations of state-of-the-art solutions and set the goals for a more realistic emulation system aimed specifically for edge computing prototyping. After that, we will present the design of EmuEdge and demonstrate how it resolves the above-mentioned problems. Lastly we show some basic use cases of EmuEdge as well as how we adopted it in realistic edge computing prototyping.

3.1 Introduction

The recent efforts in pushing data processing and analysis to the edge of networks have formed a new computing paradigm called *edge computing* [55], which is also referred to as fog computing [56], mist computing [57], edge-clouds [58] or cloudlets [59]. As one of the fastest-rising technologies to support the *resource-intensive* yet *delay-sensitive* IoT and AR/VR applications [60], edge computing has attracted the interests from both academia and industry. New proposals for edge computing architectures, middlewares, algorithms and applications [14, 15, 16, 18, 19, 20, 21] emerge constantly, which create great challenges for infrastructure providers to properly compare, choose and test suitable solutions.

Before the actual deployment of new technologies, they should be sufficiently debugged, tuned, and validated in an experimental environment. This is challenging because an edge computing system usually involves: 1) much larger scale and geographic complexity; 2) hybrid network infrastructures (e.g., LTE, WiFi and Ethernet); 3) heterogeneous nodes ranging from sensors and smartphones on the edge to rack servers in the backbone; and 4) interdependence of computation

and network. Therefore, a realistic edge computing platform validation is considered to be expensive and time-consuming, if at all possible. An easy-to-use test environment that is realistic in both computation and network plane has become key to further edge computing development and improvement.

However, among the existing research in the edge computing area, few focus on easing the difficulty of experimenting new architectures and applications while ensuring the realism. iFogSim [24] is a pivoting simulator for edge computing. It simplifies all elements in edge computing as sensor or actuator and models applications as DAGs, which brings up many limitations in realism. Network emulators such as Mininet [23] and its descendants [38, 28, 26] can also be adapted to edge computing experiments. For example, EmuFog [25] is an emulator developed based on MaxiNet [26] for edge computing emulation, which enables customization of edge computing infrastructure from scratch. However, it mainly improves Mininet on network plane such as topology design, traffic control but overlooks the computation plane. Besides, edge computing systems are usually composed of unreliable and high latency networks such as WiFi and LTE which cannot be emulated faithfully in Mininet. Mininet-WiFi supplements Mininet with basic wireless network and hardware integration support. However, it's still limited by Mininet container hosts and fails to support heterogeneous systems and computation realism. Cumulus [39] is a distributed and flexible computing testbed prototype for edge cloud computational offloading, which leverages a large spectrum of heterogeneous devices, communication methods, and OSs. Despite the realism of using real infrastructures, the cost of building and running such a testbed can be huge and their poor flexibility to change the topology is also a major concern. *To the best of our knowledge, there is no existing testing platform for edge computing that takes both the networking and computing realism, as well as heterogeneity, into account, while incurring low costs for setting and running up the experiments.*

In this paper, we propose EmuEdge, the first hybrid emulator that combines full system virtualization, container and physical infrastructures together to reproduce real world edge computing platforms with high fidelity. Different from Mininet alike systems, we define reproducibility as

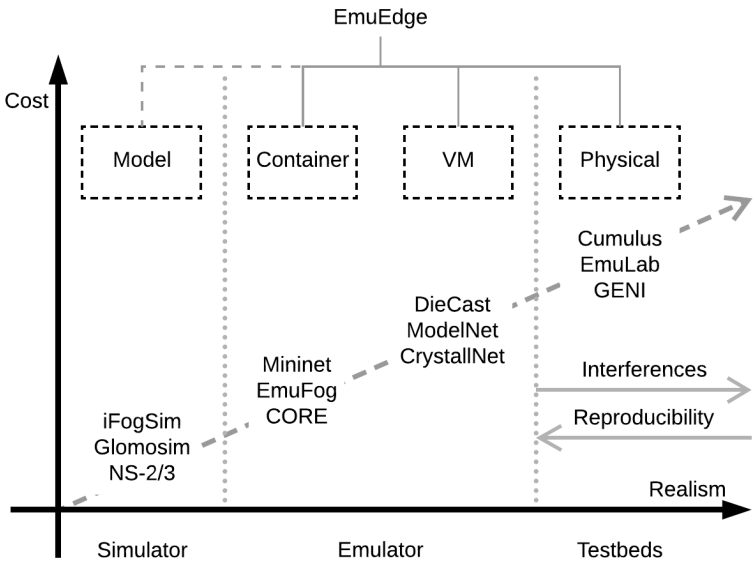


Figure 3.1: An overview of technology bound of existing testing platforms

the ability to replay a real world scenario in emulations with high fidelity, where fidelity is the degree to which EmuEdge emulations match real world experiments. To that end, EmuEdge augments previous emulation solutions with better realism in both computation and network plane. Through network replay and full system virtualization, unreliable wireless networks and heterogeneous computations in real world can be emulated in EmuEdge. As shown in Figure 3.1, previous testing platforms are usually bounded to specific technologies with deliberate tradeoffs between realism and costs. Different from other works, the hybrid design of EmuEdge enables on-demand degrees of realism by supporting both physical and emulated nodes (container or VM), which we consider as key to achieving high fidelity emulation with low costs.

3.2 EmuEdge Architecture

EmuEdge is an efficient and reproducible emulator designed specifically for hybrid edge computing systems from both computation and network perspectives. EmuEdge augments Mininet alike systems with better isolation and heterogeneity support, which allows it to emulate hybrid edge computing platforms possibly composed of heterogeneous nodes with low costs. In this section, we present the design of EmuEdge system as well as a framework of reproducing real-world

experiments on EmuEdge.

3.2.1 Design Overview

Similar to Mininet-HiFi, we adopted *netns* for network-bounded node virtualization. This enables EmuEdge to have comparable scalability with Mininet alike systems. EmuEdge also provides heterogeneous OS-level virtualization through Xen to combine the emulation of both computation and network plane in an edge computing platform. Beyond that, EmuEdge supports physical interfaces for external access. Thus, new components adding to existing infrastructures can be integrated virtually and tested before actual deployment. With our extensible design, it is even possible to extend EmuEdge with more heterogeneous nodes such as docker.

In our current implementation, the virtual hosts in EmuEdge can be both VMs and Mininet alike containers. Each of the virtual hosts can be regarded as a blackbox with one or more exposed virtual interfaces in the control domain (Dom0), such as *vif1.0* in Figure 1.1. The hosts are independent from each other in terms of network, i.e., they hold different information about the network such as routing tables, ARP caches. The only exposure of a host is its external virtual interfaces, which are managed by EmuEdge for network definition purposes. Network topologies, per-node network capabilities and link qualities are then defined with state-of-the-art SDN (Software Defined Networking) solutions such as OvS [54], Linux Traffic Control [61] and Linux netem [45].

Computation is an equally important part of a typical edge computing platform, however it's rarely taken into account in previous emulators. EmuFog [25], a descendant of Mininet tailored for fog computing, is inherently limited to network bounded emulations. Computation bounded experiments such as Hadoop testing are considered out of the scope for Mininet systems according to [38]. To address this gap, we design EmuEdge to enable realistic computational emulation through on-demand full system virtualization with Xen. Though it is usually considered that full system virtualization is heavyweight, we argue that this tradeoff is sometimes necessary to support heterogeneity and the cost can be reduced through a hybrid combination of container and VM. Afterall, the cost of EmuEdge is still miniscale comparing to a physical edge computing deployment.

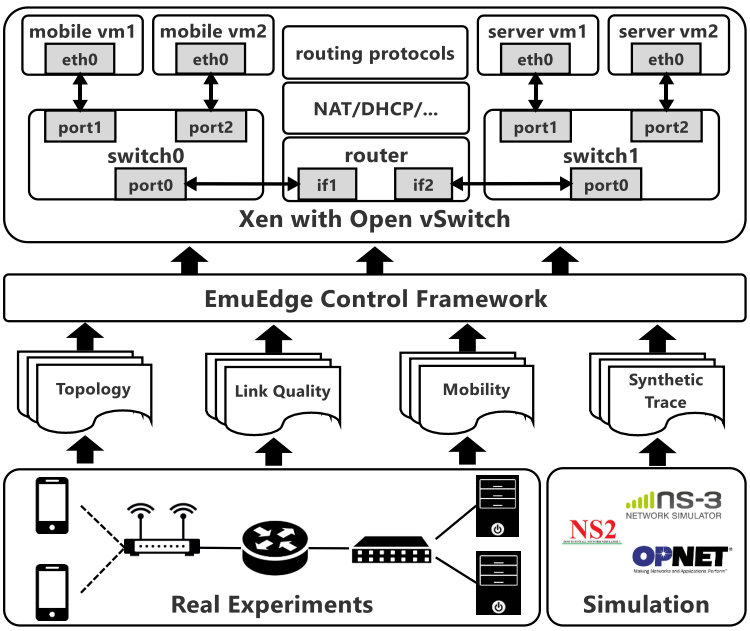


Figure 3.2: EmuEdge reality reproduction framework

For example, the actual dollar cost for a field deployment for disaster response edge computing solution in Disaster City [62] is USD 5,000 (excluding labor and hardware costs). However, with EmuEdge, we are able to replay the captured network and hardware configurations with no additional costs besides several lab PCs. A reflection of real world experiment on EmuEdge is shown in Figure 3.2, where physical nodes are mapped to virtual hosts. In EmuEdge, both wireless APs and Ethernet switches are abstracted as OvS switches, and we differentiate the links primarily based on link quality emulation through QoS and network trace replay.

3.2.2 EmuEdge Reproduction Framework

Besides the computational nodes in edge computing, we also stress the importance of networking performance in edge computing. EmuEdge leverages OvS and Linux virtual interfaces such as *veth* in order to customize network topologies in the virtual edge computing system. In addition to that, both network shape and computation isolation of an edge computing system can be fully controlled in EmuEdge. Figure 3.2 depicts the typical workflow of reproducing a field experiment

on EmuEdge. We envision the input of EmuEdge including both data collected from experiments or synthetic trace generated by network simulators. Major input parameters that we handle in EmuEdge include:

- **Emulation Parameters:** primarily composed of hardware configurations such as allocation of CPU cores, memory, disk. In cases where high-fidelity computation virtualization is needed, these parameters can be tailored more specifically, for example by configuring CPU priority, cap and even affinity.

- **Network Topologies:** in EmuEdge, most common network components are virtualized, such as router, switch, node and link. Therefore, a real world network topologies can be defined as is, on EmuEdge.

- **Network Traces:** the most realistic traces that EmuEdge takes are experiment logs, through which EmuEdge restores the traffic shape, link quality and mobility patterns of an actual network scenario, thereby enabling high-fidelity and reproducible emulations.

- **Synthetic Traces:** due to many limitations in experiments EmuEdge can also work with synthetic traces generated by simulators such as ns-3, which could further fulfill EmuEdge with the capability of emulating corner cases that cannot be covered by actual experiments.

3.3 EmuEdge Implementation

Though the support for hybrid edge computing sounds tempting, implementing EmuEdge is a tedious cove due to the heterogeneity of nodes in the system. In the following sections, we will discuss more details on EmuEdge designs and show the typical workflows to interact with EmuEdge.

3.3.1 EmuEdge Components

To create a heterogeneous edge computing prototype, EmuEdge should emulate both network and computation devices. The virtual network infrastructures in an EmuEdge emulation follow

similar compositions with physical edge computing networks. We summarize the primary components currently in EmuEdge as follows:

Network Interfaces: Network interfaces in EmuEdge are virtual Linux netifs that usually belong to certain VMs or *netns*. Such a netif can be any virtual interfaces supported by Linux. However, interface pairs that belong to Linux *veth* links are the most common netifs in EmuEdge. EmuEdge designs the relationship between any nodes and network interfaces to be one-to-many so that a device can be connected to different subnets.

Links: In the most common cases a link is a Linux *veth* pair that connects across different *netns*, with each end of it being an independent Linux netif that can be attached to virtual switches. *veth* link is an abstraction of wired link from real world with each end being the physical interface. Not all peers on *veth* links can be controlled, for example, Xen VMs only expose one end of their *veth* link to Dom0 while the other end is managed by the host OS.

Devices: A device is the abstraction of nodes that do not usually support networking functionalities. Though, it is possible that a device can act like a router in real world, EmuEdge also preserves such possibilities. Currently EmuEdge supported devices include VM and container. The introduction of "heavy-weight" VM* in EmuEdge supplements container in supporting heterogeneity of edge computing systems and providing better realism in computation plane. In fact, EmuEdge encourages container host usages as possible to improve scalability. However VM is inevitable in most edge computing cases, such as for emulating an Android mobile device. As already shown in Figure 3.4, the design of EmuEdge is flexible and can be extended with other hybrid devices such as Docker.

Routers: Similar to container host, routers in EmuEdge are actually *netns* with private network knowledge. However, routers usually have multi interfaces in different networks and support various network functionalities such as DHCP, NAT, routing and forwarding, which are all supported in EmuEdge. For a normal router on EmuEdge, any connection from other nodes to router requires an opening of new Linux netif, which is tedious and counterintuitive. Therefore, a new

*"Heavy-weight" comparing to *netns*, Xen VMs are hardware-assisted and further optimized in multi aspects, practically an Android VM starts in around 10s while a whole-system fast-clone takes less than 3s.

type of router called XenRouter is introduced which comes with an attached OvS switch. In this way, nodes can join the router by connecting their netif with the switch.

Switches: OvS switch is the default software switch with Xen, it can provide the same networking semantics of an L2 hardware switch to bring virtual devices and other nodes together.

Physical: EmuEdge supports hybrid emulation. For example, we may connect EmuEdge VMs to external routers by bridging them to arbitrary physical interfaces. EmuEdge does not discriminate between physical and virtual interfaces, one can even integrate a physical wireless NIC in the emulation. Multi-server emulation are also possible as long as they are interconnected.

3.3.2 Network Realism

We consider two types of network realism in EmuEdge. Topology realism reflects the network architecture while traffic realism is primarily designed to match link qualities in real world. In EmuEdge, the components in a network are categorized into nodes and links. Through OvS switch, EmuEdge is able to define the network topologies as-is based on real world setup. Furthermore, EmuEdge employs Linux *tc* and *netem* for link-based bidirectional traffic shaping. Replaying a real-world scenario can be done by simply setting up an adjacency list topology and tuning network quality parameters. The network quality parameters can also be a distribution pattern like normal distribution or arbitrary distributions captured from reality.

Network Traffic Shaping: Testbeds and emulators are usually considered to be realistic prototyping and experimenting platforms since they run real code in continuous time. However, in this paper, we argue that they are not satisfactory for testing a complex real-world edge computing system. The lab settings in both testbeds and emulators are too perfect to validate such systems especially when we consider the fault tolerance capabilities. For example, unreliability and mobility in a real-world wireless network can barely be replayed on testbeds and emulators. In our developing experiences with DistressNet-NG [63], a mobile edge computing system based on resilient broadband communications, an application that works well in testbeds might simply crash due to intermittent or high-latency wireless transmissions in field deployment. Therefore, a fully controllable network environment is necessary to better approximate real-world scenarios, in addition to

	Base	*Variation*	*Correlation*	*Distribution*	*Replay*
Delay	✓	✓	✓	✓	✓
Packet Loss	✓		✓		
Packet Duplication	✓		✓		
Packet Corruption	✓		✓		
Packet Reordering	✓		✓		

Table 3.1: Control function support for different link metrics

reproducibility and isolation. EmuEdge exposes functions for defining per-link network metrics through traffic shaping as shown in Table 3.1. Every metric can be configured based and correlation parameters. The base parameter is a fixed time for delay (also known as roundtrip time), or a fixed random ratio for packet losses and others. Correlation control aims to emulate consistency in real-world networks, for example, one packet loss implies the network is more congested thus the probability of losing following packets would raise as a consequence. Both random variations and variations following certain distribution can be added to a base delay for a link. EmuEdge relies on *netem* for configuring above mentioned metrics therefore we refer readers to NISTNet [37] for more details on traffic shaping internals. Moreover, EmuEdge provides a module for replaying a delay trace to approximate real-world scenarios. Unfortunately EmuEdge is still limited in replaying other metrics, however we argue that those are usually invisible from the application layer if reliable protocols like TCP are applied therefore we focus on delay and mobility instead in this paper.

Real-world Network Replay: Besides traffic shaping with approximate parameters and classic distributions[†]. EmuEdge also provides a tool suite that analyzes and summaries the delay traces from real world for future replay in lab settings. Based on a trace file, e.g., PING logs, EmuEdge calculates a distribution table, which is essentially a scaled and translated inverse to the trace data cdf (cumulative distribution function) [37]. By combining the distribution table with statisti-

[†]*netem* supports normal, pareto and pareto normal distribution by default

cal metrics (e.g., mean, variation and correlation) learned from the trace, EmuEdge can replay it through traffic shaping with high fidelity.

Rate Limiting for Network Tuning: An EmuEdge link without QoS control can easily transmit data at 30Gbps while ordinary network cables are usually limited to around 100Mbps. To deal with the inconsistency, EmuEdge supports rate limiting using *tbf* (Token Bucket Filter), with which we can set bandwidth limits on links to approximate actual link performance. Interestingly, besides pure rate limiting purpose, we discovered in practice that more accurate bandwidth shaping can be achieved with careful *tbf* parameter tuning, which will be discussed more in the experimental sections.

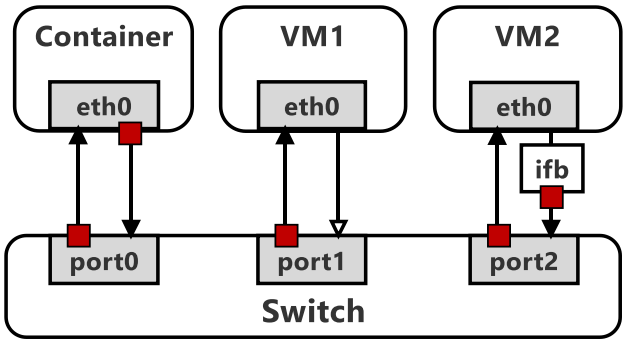


Figure 3.3: Bidirectional QoS approaches (red boxes represents tc egress control)

Dealing with Link Asymmetry: Link asymmetry is common in real life scenarios, for example, ISPs always set tighter limits on upstream bandwidth for users. In EmuEdge, asymmetric links between nodes are emulated through bidirectional traffic shaping, i.e., applying different QoS rules on two directions of a virtual link. Integrating *tc* and *netem* in EmuEdge to shape bidirectional link qualities is mostly trivial. Linux *veth* always comes in pair, therefore, bidirectional shaping between *netns*s and OvS switches can be simply achieved by applying egress shaping [‡] on both ends

[‡]Bidirectional shaping cannot be achieved with control on only one end of *veth* since Linux lacks the support of ingress shaping.

of *veth* link as shown in the c1-switch (container to switch) connection of Figure 3.3. However, VM nodes in EmuEdge are much isolated than *netns*, Xen Dom0 cannot control over *veth* ends in DomUs. Therefore, we propose an intermediate shaping method to enforce link quality on VM egress traffic as shown in vm1-switch connection of Figure 3.3. EmuEdge creates intermediate *ifb* interfaces between *eth0* in VM and its corresponding *veth* end *p0* (also referred to as a port on *switch*) in Dom0. By redirecting ingress traffic from *ifb* to *p0*, we can now control VM egress through applying egress shaping on *ifb* instead of *eth0*. For fully controllable *veth* pairs such as link between containers and switches, EmuEdge follows the tradition to avoid unnecessary *ifb* overheads. Other possible approaches to achieve bidirectional shaping on VM related links might also be available, such as associating each VM with a dedicated bridge or OvS switch, which might result in even larger overhead. We consider the investigation of other such approaches and their overheads out of scope in this paper.

3.3.3 Computation Realism

EmuEdge focuses majorly on two types of computational realism, which includes computation heterogeneity and isolation. Multiple degrees of realism are supported in EmuEdge to achieve proper computation realism with minimal costs. With physical integration undoubtedly being the ultimate degree of realism, in this section we discuss the degree of realism brought by EmuEdge container and VM, respectively.

Container: EmuEdge containers are essentially *netns* similar to Mininet hosts. Likewise, EmuEdge containers provide exclusive virtual interfaces, ports and unique network knowledges to processes. An EmuEdge container can be viewed as an independent host sharing the same kernel with EmuEdge server. Partial computation isolation can be achieved by limiting CPU time on EmuEdge containers through *cgroup* (linux control groups), which allows a group of processes running, e.g., processes running in the same container, to be scheduled and managed as a whole from the host system's perspective. The lightweight OS-level virtualization nature of *netns* enable us to scale large and fast within a PC while on the other hand limited us from heterogeneous system support.

VM: Xen VMs are more isolated and realistic hosts available on EmuEdge due to its ability to run heterogeneous systems on single machine. EmuEdge VM supports improved isolation in the following aspects:

- **CPU Cap:** CPU cap is the *cgroup* counterpart in EmuEdge to limit maximum CPU time that can be allocated to a certain VM. Careful CPU cap manipulation or other alternatives (e.g., CPU priority) should be enforced to avoid possible starvations.

- **CPU Masking:** In addition to CPU time allocation, EmuEdge supports CPU masking which provides better isolation by bounding dedicated physical cores with VMs. Proper masking: 1) provides more computation isolation among VMs; and 2)improves emulation efficiency by reducing CPU resource contentions and context switches, especially when system overloading.

- **Memory Allocation:** Both dynamic and static memory can be defined for VMs on EmuEdge. However, we usually allocate memories to VMs statically to provide a better isolated system, which cannot be done on typical container based emulators.

Besides, VMs run on independent file systems naturally although the I/O throughput is shared among them. We observed fair I/O behaviors among different running VMs with negligible variations.

3.3.4 Scalability and Extensibility

Easy Reproduction and Scaling: Reproducing problems in edge computing platforms are costly and time consuming if at all possible due to their scale and complexity. Traditional network emulators partially solved the reproducibility issue by simplifying the setup, scaling process and providing a controllable environment. However, we consider those as stateless solutions. For example, Mininet doesn't support live migration of containers, which means we may have to reconfigure things to the previous state for repeatedly reproducing a scenario. EmuEdge takes advantages of Snapshot/Clone functionalities in Xen to help us capture a complete target status of a

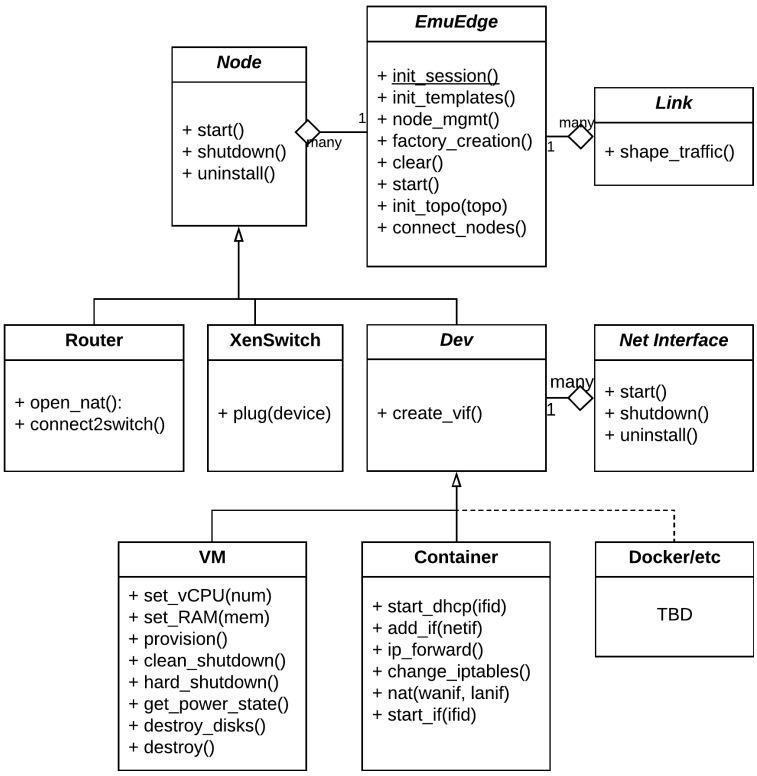


Figure 3.4: Extensible design of EmuEdge to support heterogeneous emulation

VM for reproduction use. Additionally, a snapshot of a well configured machine can be fast cloned and scaled. In the experimental section, we present how we ported realistic edge computing applications on EmuEdge. Through snapshotting and cloning, we observed a 80% of deployment time reduction comparing to manual setup. A pure VM setup in EmuEdge does not scale as well as container based solutions do, primarily due to the static memory allocation in EmuEdge[§]. For example, the maximum number of 2GB RAM VMs that can be supported by a 32GB PC is 14 (with partial resources reserved for Xen Dom0). However, we emphasize EmuEdge is an on-demand system with the flexibility to virtualize most hosts as lightweight containers.

Extensibility: EmuEdge is designed with flexible architectures, and the relations between different EmuEdge components are shown in Figure 3.4. Through proper abstraction, EmuEdge system enables standardized behaviors of network components with different implementation details.

[§]CPU is not the scalability bottleneck since it's possible to overallocate vCPUs than available physical cores on Xen.

So far, EmuEdge supports VM and container as virtual devices and five other types of Linux/Xen network interfaces (netif). Adapting EmuEdge with additional virtualization platforms such as docker would require trivial efforts.

3.4 EmuEdge User Interface

EmuEdge provides two easy-to-use approaches for fast heterogeneous edge computing prototyping: 1) an API suite including management functionalities for both *netns*, Linux netif and Xen VMs; and 2) configuring edge computing prototype through JSON, with adjacency list based network graph description. The JSON definition method can be easily adopted as interfaces to other software such as a GUI for network topology definition, which we consider out of scope in this paper.

3.4.1 Create Edge Network with EmuEdge Python API

Creating a network with EmuEdge API is easy and intuitive. With EmuEdge imported in an interactive Python command line, we can create a simple Android VM plugged into a XenRouter, get all elements ran and then cleared by:

```
xnet=xnet_interactive()
d1=xnet.create_new_dev("tandroid", "d1",
            True, vcpu=2, mem=2048, vif_prefix="tap")
d2=xnet.create_new_dev("tcentos", "d2",
            True, vcpu=2, mem=2048)
r1=xnet.create_new_xrouter("r1", "10.0.0.1/24")
xnet.connect(d1, r1)
r1.plug(xnet.session, d2)
xnet.start_all()
xnet.clear()
```

The *create_new_dev* API creates a vm using “android” snapshot as template and override it with a new configuration of 2 fixed vCPUs and 2048 MB static memory. For XenRouter, the initialization

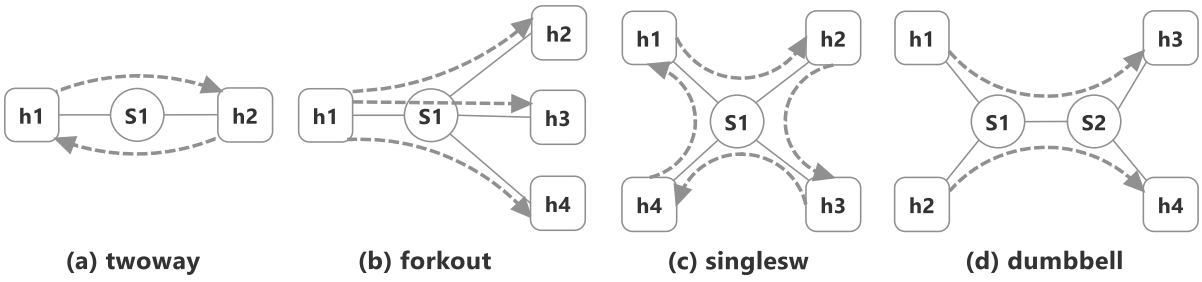


Figure 3.5: Network fidelity validation topologies

method assigned the ip "10.0.0.1/24" to the initial interface it has after creation.

3.4.2 EmuEdge JSON API

Besides the interactive Python API approach, a more intuitive way of predefining an EmuEdge topology for batch prototyping and emulation is to use JSON object with adjacency list as the topology graph. Comparing to the previous Python API, the JSON method is more concise and straightforward. A front-end UI can be made with trivial efforts based on JSON API for even more intuitive definition process. For different type of nodes, we have different properties in configurations as shown in Figure 3.6.

Through EmuEdge JSON API, both the interfaces and their corresponding IPs can be defined. Besides that, NAT and DHCP server can be configured at any certain interface on the router. In EmuEdge, NAT function on a certain interface would open it as a WAN exit so that packets transmitting over the router may be forwarded to external network through the NAT interface. As discussed before, Xen Dom0 is unable to control the *veth* ends in DomUs which means we cannot assign IPs to VMs directly. Therefore, DHCP server can be configured on routers in EmuEdge to avoid manual efforts in setting IPs on every VM. The “neighbors” array is in fact an adjacency list that defines network topologies. Each element of it is a directed link through a certain interface to other nodes, possibly with link quality definitions. With *tc* (Linux Traffic Control) and netem, EmuEdge supports bidirectional link QoS with a wide range of parameters, including delay, packet loss, duplication, reorder, corruption and bandwidth. Furthermore, statistical correlations

Router Def with DHCP/NAT

```
{
   "id":1,
   "name":"prouter0",
   "type":3,
   "nat":{
      "is_open":false
   },
   "dhcp":{
      "is_open":true,
      "if":0,
      "range_low":"128.194.142.50",
      "range_high":"128.194.142.254"
   },
   "nat":{
      "is_open":true,
      "nat_ifs":[
         0
      ],
      "lan_if":2
   },
   "ifs":[
      {
         "id":0,
         "ip":"128.194.142.1/24"
      }
   ],
   "neighbors":[
      {
         "if":0,
         "id":0
      }
   ]
},
```

VM Def with Link Control

```
{
   "id":7,
   "name":"android2",
   "type":1,
   "image":"android",
   "vcpus":4,
   "mem":2048,
   "override":true,
   "vif_prefix":"tap",
   "neighbors":[
      {
        "id":3,
        "link_control":{
          "netem":{
            "delay":{
              "base":"50ms",
              "variation":"10ms",
              "correlation":null,
              "distribution":"normal"
            },
            "duplicate":{
              "base":"0.3%",
              "correlation":"25%"
            },
            "reorder":{
              "base":"0.3%",
              "correlation":"25%"
            }
          }
        }
      }
   ]
}
```

Figure 3.6: Edge computing topology definition with EmuEdge JSON API

and distributions can also be set to better emulate a network. A real-world network trace can be easily captured and translated into distributions for repeated replays on EmuEdge through our trace analysis module.

3.5 Experimental Evaluations

In this section, we validate the effectiveness of EmuEdge from both computation and network perspective. For network fidelity, we stress the importance of approximating real-world performance and compare EmuEdge to state-of-the-art emulators with classic bandwidth experiments. We later show EmuEdge's configurability by tuning parameters to emulate more stabilized networks. Additionally, we replay a physical wireless link on EmuEdge to demonstrate its capability of emulating real-world as is. Lastly we share some experiences on achieving computation isolation and show that EmuEdge provides near-perfect isolation comparing to container based emulators.

3.5.1 Network Fidelity Validation

To demonstrate the effectiveness of EmuEdge in ensuring network fidelity, we adopted validation tests as proposed in Mininet benchmark [38], where Mininet-HiFi was shown to have less variations and higher reproducibility than testbeds in these experiments. Differently, EmuEdge emphasizes the importance of emulation realism, and we define fidelity as the degree to which our emulation environment matches real-world. Though it's impossible to replay an experiment exactly, even with testbeds, we argue that a qualitatively similar testing environment is sufficient to discover most problems in reality.

In our experiments, the four network test topologies emphasizing on different network properties as shown in Figure 3.5 are applied in both physical, Mininet and EmuEdge setup. EmuEdge is currently limited in connecting VMs directly, due to the fact that Xen VMs are bounded to OvS switch/Linux bridge by default. Therefore for twoway test, we used a slightly different topology from [38] where hosts connect to each other through a switch instead of direct link. The experiment setup for our three comparison scenarios are as follows:

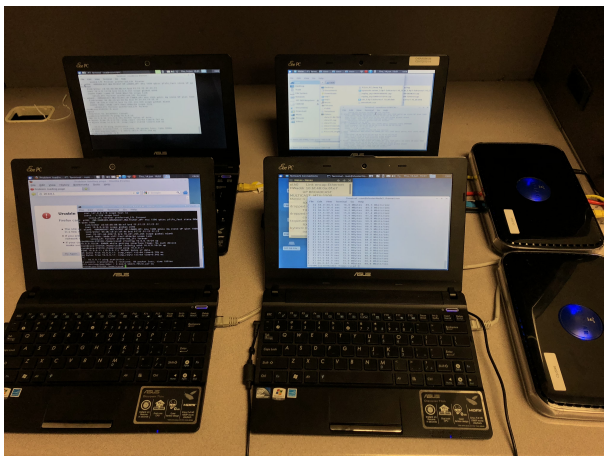


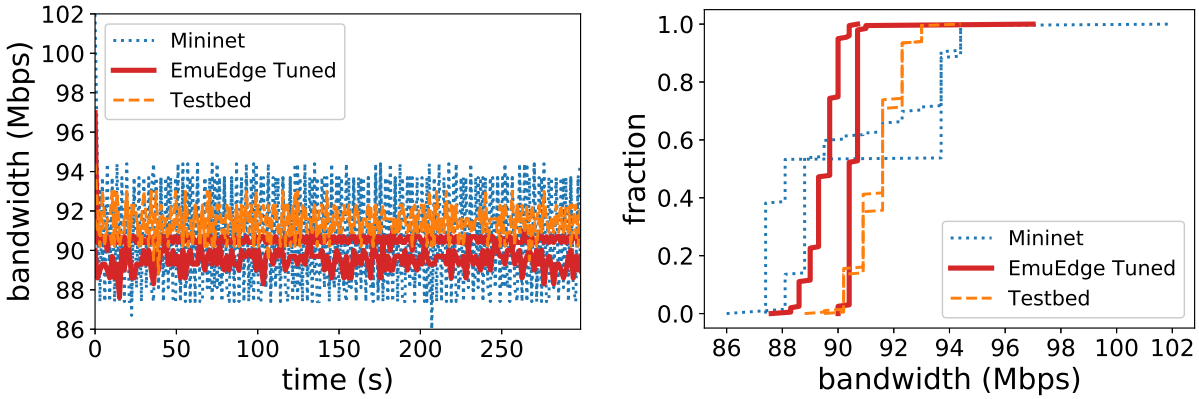
Figure 3.7: Experiment testbeds

- **Testbed:** As shown in Figure 3.7, for physical experiments, we used 4 ASUS Eee PC as the

hosts and two NetGear routers purely acting as switches. The connections between hosts and routers are changed accordingly to experiment topologies. According to our measurements, physical links maintain a consistent bandwidth from 90 to 95 Mbps.

- **Mininet:** we used Mininet version 2.2.1 with resource provisioning and link rate limiting, Mininet hosts and switches are created accordingly based on topologies. Bandwidth limits on links are set to 95 Mbps through *TCLink*.

- **EmuEdge:** since the EmuEdge containers are implemented similarly with Mininet, we omit experiments for them and focus on EmuEdge VM hosts. We choose CentOS 7 as host OS and applied the same bandwidth limits for EmuEdge links.[¶]



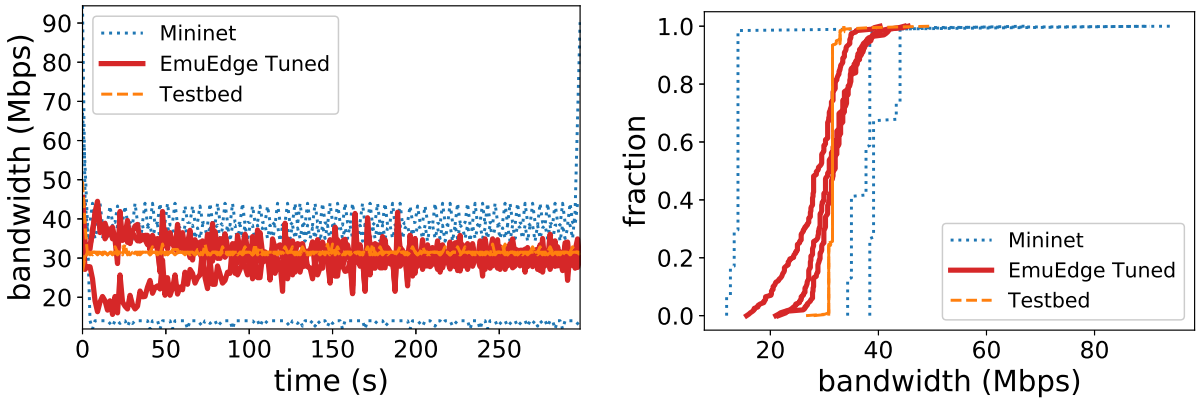
(a) Twoway bandwidth varying by time

(b) Twoway bandwidth cdf

Figure 3.8: Twoway bandwidth performance comparisons

For each topology, we run *iperf* on all hosts for 300 seconds, at a 0.75Hz bandwidth logging rate. We aim to show the long-term performance since we observed a converging process for both Mininet and EmuEdge. Both the bandwidth measured over time at each *iperf* client and their cumulative distributed functions (cdf) are shown in Figure 3.8–3.11. Different from Mininet

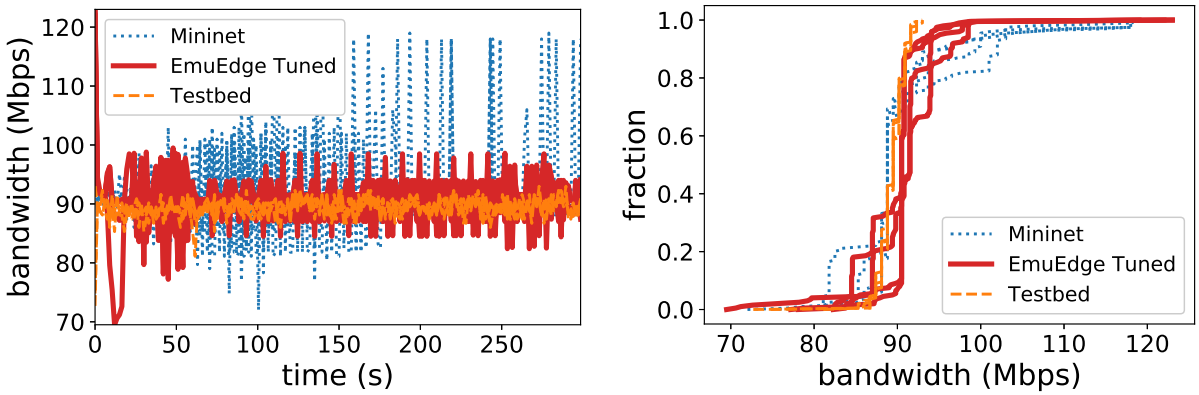
[¶]Both Mininet scripts and EmuEdge topologies for experiments are available at https://github.com/ykzeng/emuedge/tree/master/topo/exps/.



(a) Forkout bandwidth varying by time

(b) Forkout bandwidth cdf

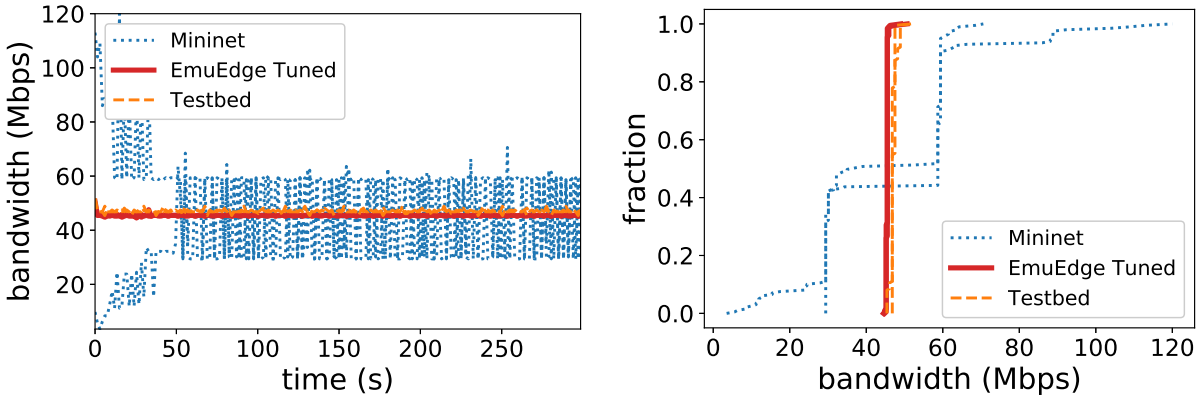
Figure 3.9: Forkout bandwidth performance comparisons



(a) Singlesw bandwidth varying by time

(b) Singlesw bandwidth cdf

Figure 3.10: Singlesw bandwidth performance comparisons



(a) Dumbbell bandwidth varying by time

(b) Dumbbell bandwidth cdf

Figure 3.11: Dumbbell bandwidth performance comparisons

benchmark, in almost all our experiments, the testbeds network demonstrated extraordinary stability and fairness with trivial fluctuations. In the *twoway* test, Mininet demonstrated good fairness and rate limiting capabilities apart from the beginning drop. However, the fluctuations of Mininet differ a lot from reality and are much more substantial comparing to EmuEdge VMs. Then in the *forkout test*, we noticed a fairness problem in Mininet, two links on the right of Figure 3.5b seem to have shared a large portion of the *h1-s1* link bandwidth unevenly. With EmuEdge, the bandwidth splitted unfairly among links in the beginning, however it converges overtime and approaches the testbeds performance closely in the last 100 seconds. In the case of *singlesw* test, both Mininet and EmuEdge are performing badly with many sudden fluctuations though the cdf looks similar to reality. Lastly in *dumbbell* test, EmuEdge demonstrates extraordinary fidelity and realism comparing to Mininet, where both fluctuation and cdf match well with physical links. Similar to *twoway* test, Mininet limits rate very well but with regular fluctuations, after the converging process. Overall, we consider EmuEdge significantly outperforms Mininet in terms of experiment realism while maintaining similar fairness among shared links.

EmuEdge Network Tuning: Though EmuEdge yields good fairness and realism in above experiments overall, we observed from Figure 3.10 that sometimes EmuEdge fluctuates too much and does not converge well. This didn't match well with real world and can lead to unconvincing experiment results. Therefore, we modified our EmuEdge setup with well-tuned rate limiting parameters, which can be easily done through EmuEdge JSON API. Besides bandwidth, burst and limit are the other two major parameters to specify in EmuEdge for rate limiting purposes. Based on our experience, we summarize that a lower burst size will set tighter upper limits on instantaneous bandwidth thereby reducing fluctuations. Besides, limit is the length of packet queue on outgoing queue, which also adds uncertainties into transmissions. By carefully tuning those two parameters, we managed to stabilize EmuEdge *singlesw* bandwidth overtime as shown in Figure 3.12. The tuned links behave much more similar to testbed, in fact the stability of it even outperforms testbed. This inspires us that network bandwidth realism can be emulated for different target environments by tuning EmuEdge links, however a more comprehensive study is needed to

investigate specific tuning methods.

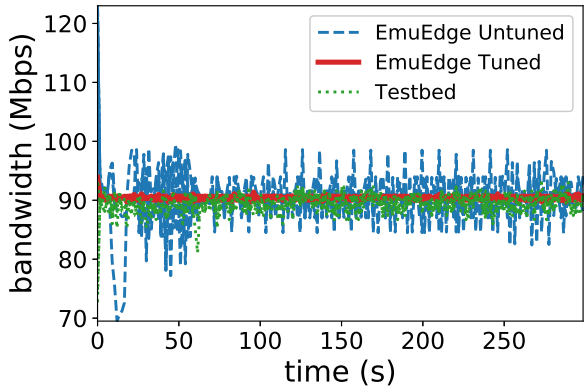


Figure 3.12: Tuned singlesw bandwidth in EmuEdge

3.5.2 Replaying Wireless Network

Wireless networks add uncertainties into edge computing systems with delays, jitters and possibly intermittent transmissions due to host mobility. EmuEdge supports network trace replay to faithfully recreate real-world networks within a server. Primarily two approaches are available for such purposes in EmuEdge:

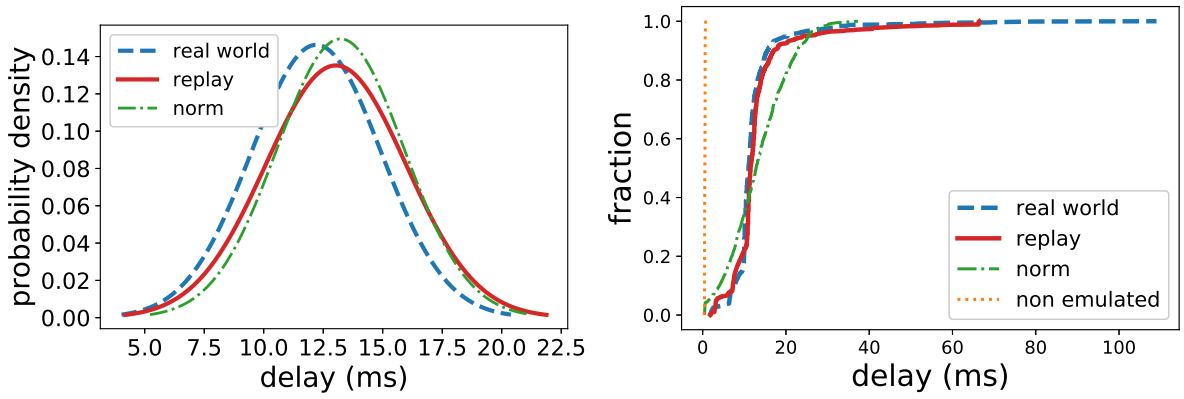
Normal approximation: EmuEdge is able to approximate normal distributed delays among links. One can easily generate necessary stats information (e.g., mean and standard deviation) that captures the normal distribution approximation based on a real-world network trace for normally approximated EmuEdge replay.

Customized replay: Besides the default distributions provided, we can also define our own distribution based on network traces through the EmuEdge trace analysis module. A customized distribution table and statistical parameters will be generated and stored in EmuEdge *dist_db*. After that, we can emulate the same link anytime by simply specifying the customized distribution to emulate through EmuEdge JSON API.

As a proof-of-concept experiment, we demonstrate the above methods on EmuEdge to replay the delays in a wireless link between a LinkSys AP and an Android mobile phone. For trace collecting, we captured 1000 PING rtts in log file called *wifi* and then interact with the EmuEdge trace analysis module by:

```
trace/rtt_log2dist.sh wifi
```

this analyzes the logs, generates distribution table and saves information in a distribution database, which contains distribution tables and stats information such as mean, standard deviation. The stats information are then used as parameters for normally approximated replay. For customized distribution emulation, we simply set link distribution param *distribution* to be *wifi* in our target EmuEdge topology. Then we start two Android VMs linked to virtual router with normal approximation and customized distribution respectively. Lastly, we run 1000 PINGs on both VMs and physical Android phones to their corresponding virtual/physical routers. The rtt results are presented in forms of probability density function and cdf in Figure 3.13.



(a) Probability density function

(b) Cumulative distributed function

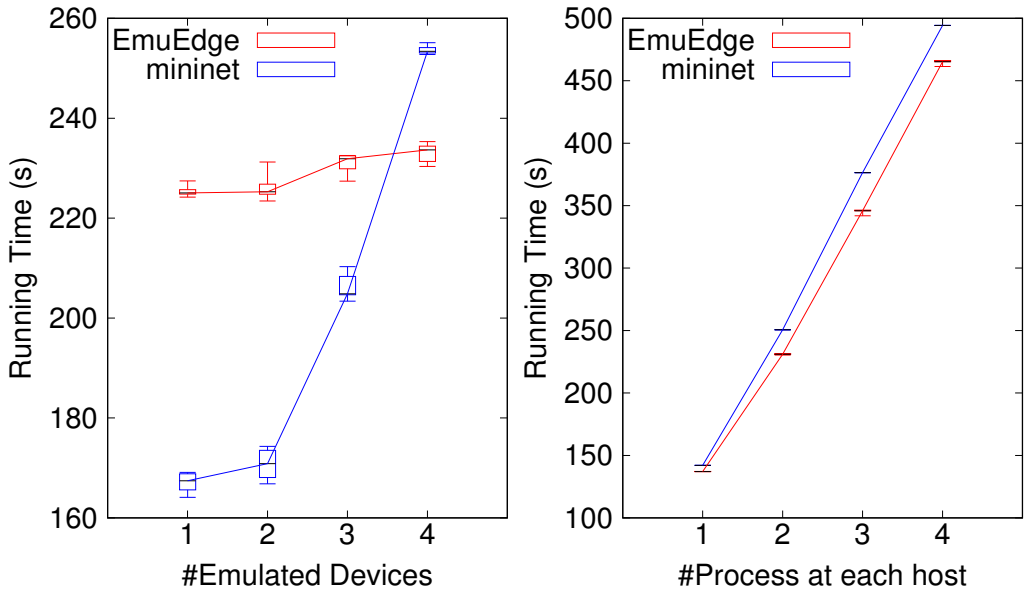
Figure 3.13: Comparison between delays in EmuEdge non-emulated, norm approx, replay and real-world wireless link

The non emulated case can be regarded as EmuEdge baseline, i.e., pure EmuEdge link without any alteration, we omitted it in probability density function due to its great gap from others. From

Figure 3.13a, it's apparent that both normal approximation and customized replay can closely mimic reality. The normal distribution approach fits even better than replay to some extent since it's purely approximated based on the exact (σ, τ) captured from reality. However, replay reflects the actual link near-perfectly in terms of cdf as shown in Figure 3.13b while normal method is statistically too ideal.

3.5.3 Computational Realism Validation

As discussed before, a realistic edge computing testing platform should be able to emulate both computation and network to be practically useful. In this section, we validate the computation realism of EmuEdge comparing to Mininet containers. We consider our experiments reflect full-system vs. OS-level virtualization comparison since Mininet container adopts the same isolation and resource management approach with other OS-level virtualizations such as LXC [64] and Docker [65].



(a) Computation isolation (b) Computation efficiency

Figure 3.14: Computational realism comparison between EmuEdge VM and Mininet containers

Computation Isolation: Isolation has always been a key factor to consider in virtualization approach evaluation. Emulators that fail to provide adequate isolation for computational nodes would yield unrealistic results. Ideally in an edge computing system, each physical node is computationally independent, i.e., workloads on different nodes do not influence each other. Therefore, we consider the case when multiple such nodes are emulated in a single PC. In our experiments, two PCs with exactly same hardware (Quad core Intel i7 CPU and 32GB RAM) are used for Mininet and EmuEdge emulation respectively. We argue EmuEdge containers would have similar performance with Mininet since they are implemented similarly, therefore we focus on comparing EmuEdge VM with Mininet container. For Mininet, *CPULimitedHost* are used to limit the host within CPU time of one physical core, while EmuEdge VMs are configured with one dedicated physical CPU core and 2GB memory. For each PC, we run multiple emulated nodes that are fully occupied by CPU-intensive MapReduce [66] workloads. We then measure workloads execution time on each node while increasing total number of node running. As shown in Figure 3.14a, both EmuEdge and Mininet can guarantee resource fairness among different hosts, the execution time variations between hosts in each run are negligible. However, with increasing number of emulated computation nodes, the average execution time in Mininet containers demonstrate large fluctuations while EmuEdge hosts run stably and independently. We argue the trivial execution time increase in EmuEdge is due to system overloading, since the total CPU utilization of the system exceeds 75% with 3 or more emulated devices.

Computation Efficiency: Interestingly, we also observed from Figure 3.14a that when both PCs are stressed over 75%, the performance degradation of Mininet containers are significant that it even exceeds the execution time in EmuEdge VMs utilization hits 100% with four emulated nodes. This is against the fact that containers are much more lightweight than VMs and hence should yiled less overhead and better performance. We are then inspired to further investigate the performance between container and VM. Instead of adding computational nodes, we fixed the number of nodes to 4 and attempt to stress them with more workloads in each run. The results in Figure 3.14b show that our observation is no coincidence and containers tend to run slightly

slower than VMs under high utilization. The results seem to be contradictory with [67]. However, we argue that VM vs. container performance results are bounded to specific cases. Moreover, recent advances in virtualization have demonstrated through distributed operations on Xen, VM can be actually lighter and safer than containers [68]. Particularly in our experiments, Mininet containers are less efficient while overloaded since they suffer from higher centralized scheduling overhead and context switches while EmuEdge VMs have better processor affinity.

Lesson Learned: Actually, the advantages of VM shown in computational realism doesn't diminish containers' significance in emulation. In fact, we consider containers to be cost-effective since it enforces considerable isolation with less overhead. Therefore, we again emphasize that proper decisions in choosing emulation nodes are key to improving realism and reducing costs. Generally, containers are sufficient for emulation of nodes that are network-bounded or Lo-Fi computation-bounded. Meantime, applications requiring isolated resources or heterogeneous OSes can be run with VMs. For example, a container might be enough for emulating a functional Apache web server while several VMs with fixed CPUS and RAM is needed for emulating an edge cloud to evaluate the worst case performance. Additionally, the hybrid nature of EmuEdge also enables us to integrate existing infrastructures into our emulation, such as a remote AWS instance.

Through previous experiments, we demonstrated advantages of EmuEdge in performance fidelity comparing to Mininet and testbeds from both computation and network perspectives. However, we consider the experiment setups much more simplified comparing to actual edge computing systems. Therefore, in this section, we deploy an actual edge computing platforms on EmuEdge with hybrid infrastructures and real-world network interactions to further demonstrate the compatibility and realism of EmuEdge. Figure 4.1 depicts the physical and hybrid setup for following experiments, the hybrid setup can be fully virtualized with EmuEdge by using a Master Server container.

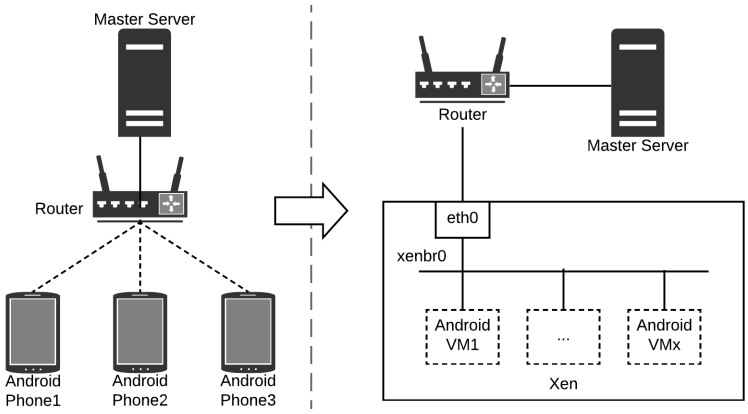
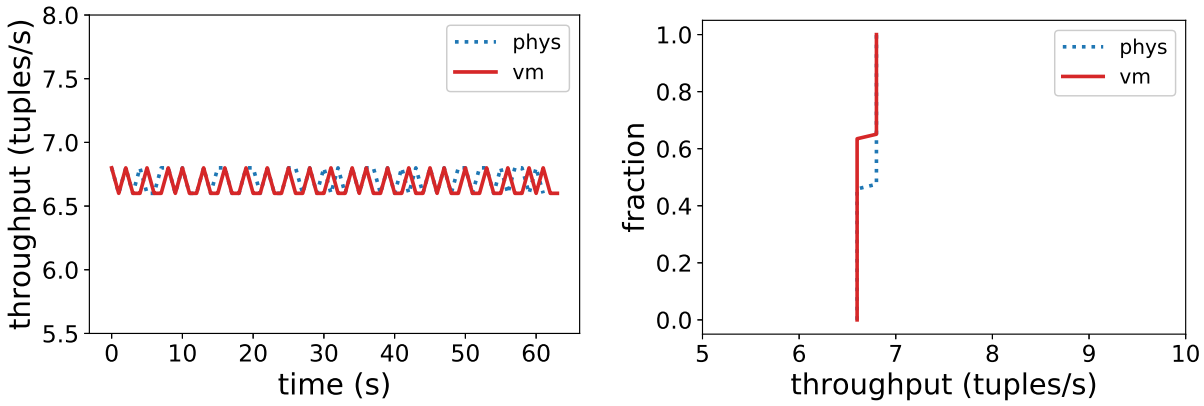


Figure 4.1: Physical and hybrid (EmuEdge) setup of actual edge computing platforms in expriments

Mobile Storm (MStorm) [11] is an online distributed stream processing system on Android. Different from the datacenter counterpart [69], MStorm is designed for critical scenarios such as military operations and disaster response, where networks are limited. The soldiers or responders operate in teams and connect with each other over a manpack LTE or Wi-Fi access point. Due to mission criticality MStorm need to be well tuned and validated before deployment. However,

field deployment of infrastructures and applications for MStorm is onerous and costly. To ease the testing process of MStorm, We seek to replay real world MStorm setup on EmuEdge, therefore we first consider the realism of MStorm emulation on EmuEdge.

Performance Realism: To validate the performance realism of virtual MStorm (vMStorm) on EmuEdge, we run a benchmark application called RandomSentenceStats. In the application, multi Android phones form a cluster and a source node will generate random sentences for statistical processing by downstream nodes. At this moment, we limit the experiment scope to a single processing node, thus the data generation and processing are done on the same device. To emulate computational performance under different workloads, we generate the stream with inter-arrival time (IAT) following different distributions and monitor overall system throughput. The experiment results in Figure 4.2–4.5 show that the performance vMStorm (vm) perfectly matches the reality (phys) under all scenarios. Also, we observed trivial throughput improvement on EmuEdge, apparently due to more advanced hardware.

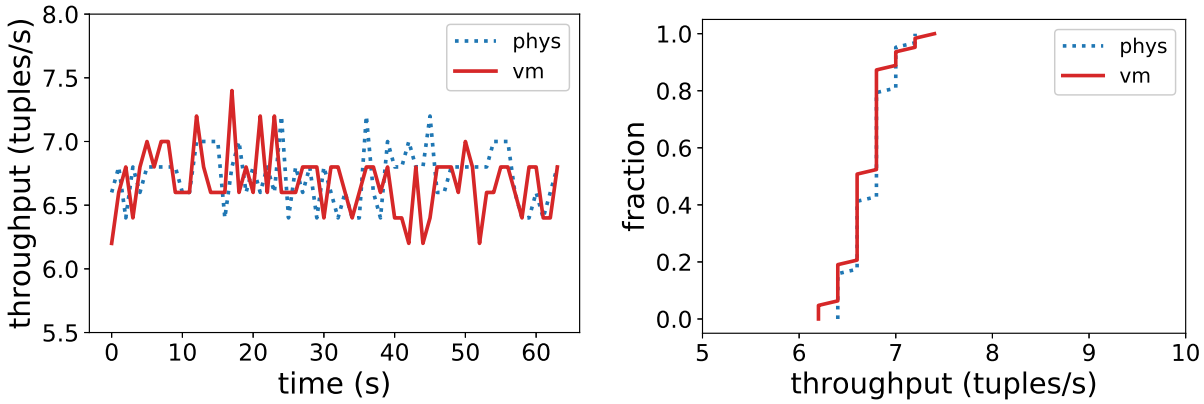


(a) Throughput of constant IAT

(b) Throughput CDF of constant IAT

Figure 4.2: Throughput of physical and VM (EmuEdge) MStorm with workloads following constant IAT pattern

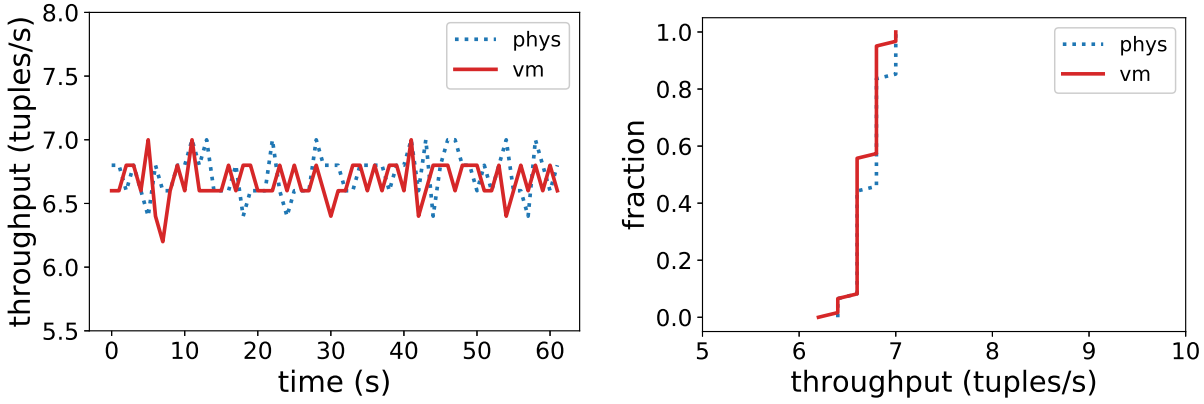
Scalability Realism: Besides single node performance, we investigate on how EmuEdge reflects system performance improvement when scaling and compare the results with an identical



(a) Throughput of UR IAT

(b) Throughput CDF of UR IAT

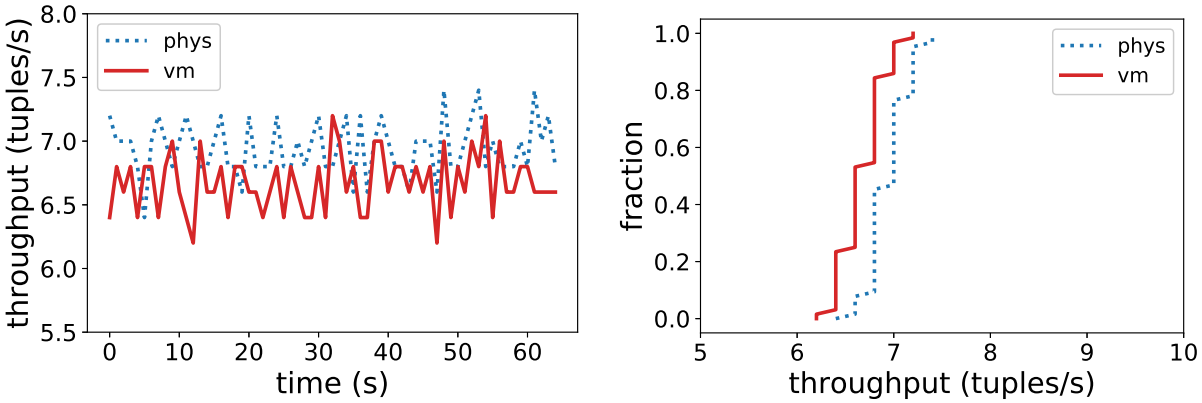
Figure 4.3: Throughput of physical and VM (EmuEdge) MStorm with workloads following UR IAT pattern



(a) Throughput of Gaussian IAT

(b) Throughput CDF of Gaussian IAT

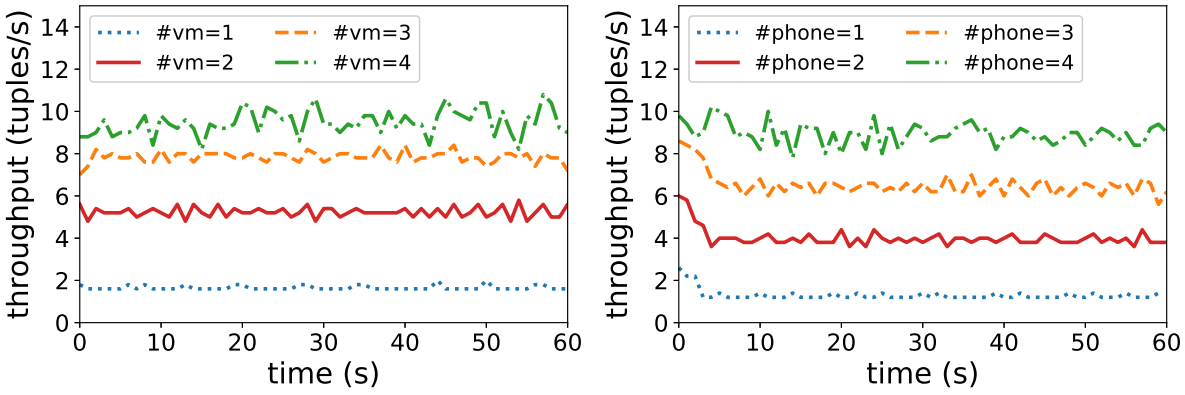
Figure 4.4: Throughput of physical and VM (EmuEdge) MStorm with workloads following Gaussian IAT pattern



(a) Throughput of Pareto IAT

(b) Throughput CDF of Pareto IAT

Figure 4.5: Throughput of physical and VM (EmuEdge) MStorm with workloads following Pareto IAT pattern



(a) vMStorm Scalability

(b) MStorm scalability

Figure 4.6: Realistic scalability experiments on EmuEdge

physical cluster. In this experiment, the stream generation rate is fixed at 10 tuples/s following uniform IAT. We apply higher computational complexity that a single node cannot handle and then scale both clusters from 1–4 nodes. Figure 4.6 depicts the overall throughput of both systems. Apparently due to heavier workload, the throughput of both single nodes are limited under 2 tuple/s. With additional nodes, vMStorm matches physical performance with similar increase trend. Apart from that, the fluctuations in throughput are also reflected in EmuEdge. Both clusters demonstrate larger performance fluctuations with more nodes.

5. CONCLUSIONS

Despite the proposals of numerous prototypes and architectures, the tremendous costs of testing heterogeneous edge computing systems have prevented it from realizing its value in the IoT era. Built upon container based emulators, EmuEdge unsets the OS-level virtualization bound and extend the emulation to hybrid setups supporting different degrees of realism. As shown in our experiments, the introduction of VM enables better computation realism in terms of heterogeneity support and computation isolation. On the network perspective, we aim differently than previous emulators by reproducing networks close to reality through tuning and replaying network traces. However, EmuEdge is still limited in several aspects:

Background Workload Realism: In edge computing, mobile nodes can handle both local applications and offloaded computations at the same time. Therefore, the overall performance of an edge computing platform can be greatly influenced by background workloads on edge nodes. However, EmuEdge computational nodes are dedicated, which leads to the lack of background workload realism.

Edge Nodes Compatibility: Full system virtualization supports a wide range of common OSes running simultaneously within single machine, which enables much better heterogeneity on EmuEdge. However, besides computation nodes running mainstream OSes such as Linux and Android, edge computing involves other data collecting nodes such as sensors that cannot be emulated virtually. Though, we argue this is a shortcoming of all emulators and a worthy tradeoff for better realism comparing to simulators. Besides, it's still possible to integrate those nodes through hybrid EmuEdge setup.

Network Dynamics and Mobility: Real-world wireless networks, such as Wi-Fi and LTE, usually change dynamically due to user motions and noises. For examples, signal strength at a mobile device might change dramatically when the user moves between rooms and buildings. Currently EmuEdge replays wireless network assuming that variations in the network are consistent in the long term therefore cannot emulate device mobilities perfectly. We are currently pursuing

other methods in tracing and replaying network dynamics to further improve EmuEdge realism.

Despite the current limitations, we envision the on-demand degrees of realisms on EmuEdge is a key step to reproducible edge computing experiments. With EmuEdge, emulating an edge computing system with heterogeneous OSes and close-to-reality network can be done in lab settings with minimal costs. We hope this advancement could greatly facilitate the debugging and testing process of edge computing platforms. Besides that, EmuEdge can be also regarded as a hybrid extension of Mininet that fills the gap on computation plane. Therefore, it is possible to adapt EmuEdge for general experiments that are bounded by both network and computation.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," *National institute of standards and technology*, vol. 53, no. 6, p. 50, 2009.

[3] Y. Wang, R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: perspectives and challenges," *Wireless Personal Communications*, vol. 80, no. 4, pp. 1607–1623, 2015.

[4] H. Qi and A. Gani, "Research on mobile cloud computing: Review, trend and perspectives," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pp. 195–202, ieee, 2012.

[5] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[7] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, p. 6, ACM, 2010.

[8] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, 2009.

[9] T. Kakantousis, I. Boutsis, V. Kalogeraki, D. Gunopulos, G. Gasparis, and A. Dou, "Misco: A system for data analysis applications on networks of smartphones using mapreduce," in

Mobile Data Management (MDM), 2012 IEEE 13th International Conference on, pp. 356–359, IEEE, 2012.

[10] J. George, C.-A. Chen, R. Stoleru, and G. Xie, "Hadoop mapreduce for mobile clouds," *IEEE Transactions on Cloud Computing*, 2016.

[11] Q. Ning, C.-A. Chen, R. Stoleru, and C. Chen, "Mobile storm: Distributed real-time stream processing for mobile clouds," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pp. 139–145, IEEE, 2015.

[12] Armbrust, M. and Fox, A. Griffith, R. Joseph, A.D. Katz, R. Konwinski, A. Lee, G. Patterson, D. Rabkin, A. Stoica, I. et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[13] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[14] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[15] J. Plachy, Z. Becvar, and E. C. Strinati, "Dynamic resource allocation exploiting mobility prediction in mobile edge computing," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pp. 1–6, IEEE, 2016.

[16] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[17] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[19] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[20] S. Ramgovind, M. M. Eloff, and E. Smith, "The management of security in cloud computing," in *Information Security for South Africa (ISSA), 2010*, pp. 1–7, IEEE, 2010.

[21] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[22] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*, pp. 117–122, IEEE, 2014.

[23] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, ACM, 2010.

[24] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[25] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," *arXiv preprint arXiv:1709.07563*, 2017.

[26] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Networking Conference, 2014 IFIP*, pp. 1–9, IEEE, 2014.

[27] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 253–264, ACM, 2012.

[28] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 384–389, IEEE, 2015.

[29] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS operating systems review*, vol. 37, pp. 164–177, ACM, 2003.

[30] L. Pridmore, P. Lardieri, and R. Hollister, "National cyber range (ncr) automated test tools: Implications and application to network-centric support tools," in *AUTOTESTCON, 2010 IEEE*, pp. 1–4, IEEE, 2010.

[31] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 255–270, 2002.

[32] J. Wroclawski, T. Benzel, J. Blythe, T. Faber, A. Hussain, J. Mirkovic, and S. Schwab, "Deterlab and the deter project," in *The GENI Book*, pp. 35–62, Springer, 2016.

[33] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[34] T. Miyachi, K.-i. Chinen, and Y. Shinoda, "Starbed and springos: Large-scale general purpose network testbed and supporting software," in *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, p. 30, ACM, 2006.

[35] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.

[36] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[37] M. Carson and D. Santay, "Nist net: a linux-based network emulation tool," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[38] B. Heller, *Reproducible network research with high-fidelity emulation*. PhD thesis, Stanford University, Stanford, CA, 2013.

[39] H. Gedawy, S. Tariq, A. Mtibaa, and K. Harras, "Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading," in *Cloudification of the Internet of Things (CIoT)*, pp. 1–6, IEEE, 2016.

[40] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *Standards for Communications and Networking (CSCN), 2016 IEEE Conference on*, pp. 1–7, IEEE, 2016.

[41] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," in *ACM SIGSIM Simulation Digest*, vol. 28, pp. 154–161, IEEE Computer Society, 1998.

[42] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*, pp. 15–34, Springer, 2010.

[43] X. Chang, "Network simulations with opnet," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pp. 307–314, ACM, 1999.

[44] M. I. Naas, J. Boukhobza, P. R. Parvedy, and L. Lemarchand, "An extension to ifogsim to enable the design of data placement strategies," in *Fog and Edge Computing (ICFEC), 2018 IEEE 2nd International Conference on*, pp. 1–8, IEEE, 2018.

[45] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, pp. 18–23, 2005.

[46] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer.," in *Proceedings of the 8th ACM Workshop on Hot Topics in Networks*, pp. 1–6, 2009.

[47] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[48] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the emulab network testbed.," in *USENIX Annual Technical Conference*, pp. 113–128, 2008.

[49] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, p. 7, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[50] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, p. 72, ACM, 2008.

[51] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pp. 1–7, IEEE, 2008.

[52] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 271–284, 2002.

[53] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, "Diecast: Testing distributed systems with an accurate scale model," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 2, p. 4, 2011.

[54] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch.," in *NSDI*, pp. 117–130, 2015.

[55] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[56] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[57] R. K. Barik, A. Tripathi, H. Dubey, R. K. Lenka, T. Pratik, S. Sharma, K. Mankodiya, V. Kumar, and H. Das, "Mistgis: Optimizing geospatial data analysis using mist computing," in *Progress in Computing, Analytics and Networking*, pp. 733–742, Springer, 2018.

[58] I. Hou, T. Zhao, S. Wang, K. Chan, *et al.*, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 291–300, ACM, 2016.

[59] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.

[60] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "Its hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," tech. rep., Technical Report, December 2017.[Online]. Available: https://1drv. ms/b/s, 2018.

[61] W. Almesberger, "Linux traffic control-next generation," in *Proceedings of the 9th International Linux System Technology Conference (Linux-Kongress 2002)*, pp. 95–103, sn, 2002.

[62] T. A. E. E. Service, "TEEX Disaster City." `https://teex.org/Pages/about-us/disaster-city.aspx`, 2018. [Online; accessed 25-July-2018].

[63] E. Nunez, "DistressNet-NG: Resilient Mobile Broadband Communication and Edge Computing." `https://www.nist.gov/ctl/pscr/`, 2017. [Online; accessed 25-July-2018].

[64] M. Helsley, "Lxc: Linux container tools," *IBM devloperWorks Technical Library*, vol. 11, 2009.

[65] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[66] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[67] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pp. 233–240, IEEE, 2013.

[68] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My vm is lighter (and safer) than your container," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 218–233, ACM, 2017.

[69] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 147–156, ACM, 2014.