EmuEdge: A Hybrid Emulator for Reproducible and Realistic Edge Computing Experiments

Yukun Zeng, Mengyuan Chao and Radu Stoleru
Department of Computer Science and Engineering, Texas A&M University
College Station, TX 77840
{yzeng,chaomengyuan,stoleru}@[cse.]tamu.edu

ABSTRACT

Numerous research efforts are devoted into edge computing due to its key role in enabling the 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 paper, we propose EmuEdge, a hybrid emulator based on Xen and Linux 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 definition language. Through extensive experiments, we proved 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.

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* [1], which is also referred to as fog computing [2], mist computing [3], edge-clouds [4] or cloudlets [5]. As one of the fastest-rising technologies to support the *resource-intensive* yet *delay-sensitive* IoT and AR/VR applications [6], edge computing has attracted the interests from both academia and industry. New proposals for edge computing architectures, middlewares, algorithms and applications [7–13] emerge constantly, which create great challenges for infrastructure providers to properly compare, choose and test suitable solutions.

Before the actual deployment of any 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, 4) interdependence of computation and network. Therefore, 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 the key to further edge computing development and improvement.

However, among the existing researches in the edge computing area, few focus on easing the difficulty of experimenting new architectures and applications while ensure the realism. iFogSim [14] 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 [15] and its descendants [16–18] can also be adapted to edge computing experiments. For example, EmuFog [19] is an emulator developed based on MaxiNet [18] 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 LTE, WiFi, 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 fail to support heterogeneous systems and computation realism. Cumulus [20] 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 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 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 levels of realism by supporting both physical and emulated nodes (container or VM),

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

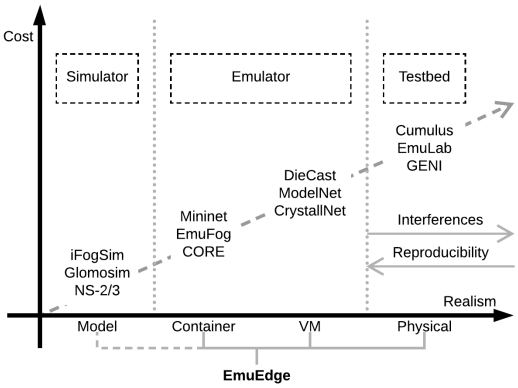


Figure 1: an overview on technology bound of existing testing platforms

which we consider as the key to achieving high fidelity emulation with low costs.

The major contributions of this paper include the following:

- 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 emulate real-world networks in EmuEdge.
- A suite of interactive APIs and the EmuEdge definition language as two approaches to define, interact and share edge computing prototypes with full details such as network topology, link qualities and VM configurations.
- We validated the realism of EmuEdge with a practical edge computing platform, in which the emulation results reflected the real world performance with high fidelity.

2 RELATED WORK AND BACKGROUND

In this section, we first present some existing experimental tools for networking and edge computing and summarize their advantages and disadvantages. Then, we propose the design objectives of EmuEdge. Finally, we introduce some background about Xen[21], the virtualization technology adopted by EmuEdge.

2.1 Edge Computing Experimental Tools

The experiments for edge computing mainly consists of two parts, one is networking, which concerns the communication delay, bandwidth, drop rates and jitter; the other is computing, which concerns the computation delay and throughput. Currently, most experimental tools for edge computing focus on networking, as they are evolved from the previous networking experimental tools. Therefore, in this subsection, we introduce related work about experimental tools for both networking and edge computing. We categorize the existing work as follows:

Testbed: The ideal way of testing a system can be done through reproducing the actual scenarios on a physical testbed. There are a bunch of networking testbeds that can be adapted for edge computing experiments, such as NCR [22], Emulab [23], Deterlab [24], PlanetLab [25], StarBED [26], GENI [27], etc. These testbeds make a large number of machines and network links available and use tools such as Dummynet [28] and NIST Net [29] to configure network link properties such as delays, drop rates, jitters [30]. There are also pure edge computing testbed, such as Cumulus [20] and SCC TestBed [31], which consist of a large spectrum of heterogeneous devices, OSs and network links. Although testbeds provide the most realistic experimental results, they cost money to build and keep running, have practical resource and replication limitations, and lack the flexibility to support experiments with custom topologies [30]. Besides, 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 [32], NS-3 [33], OPNET [34] 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 [30]. 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 [14] and its extension [35] 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.

Emulator: Emulators are able to automatically configure and set up reproducible experiments emulating real world scenario. Comparing to simulators, emulation usually incurs similar infrastructure cost while achieves better realism since it can run real code without changes. With recent advances in SDN such as Linux netem [36], Open vSwitch (OvS) [37] and OpenFlow [38], 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 [39], NetKit [40], Trellis [41], CORE [42], Mininet [15] and its its descendants Mininet-HiFi [16], Mininet-WiFi [17] and Maxinet [18], 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 [19] 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 [43] and DieCast [44]. 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.

Different from all the above works, EmuEdge is a ***hybrid*** emulator that combines the features of simulators, testbeds and emulators together, which absorbs the advantages of different experimental methods for ***realistic and reproducible*** edge computing experiments.

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: edge computing experimental platform features

2.2 Objectives of EmuEdge

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 change the topology during the running time.

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 as in 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 OSs in the edge computing paradigm by installing different hosts with different OSs.

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 such that it can scale well when the required hosts increases.

Table 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 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, it's based on OS-level virtualization, which cannot support OS heterogeneity and has no guarantees on resource realism. VM-based emulators such as DieCast [44] generally provide better resource and OS realism. However, they are usually considered to have inferior scalability and incur higher costs.

Different from all the 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 add into 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 a virtualization technology called Xen, which is briefly introduced in the following subsection.

2.3 Xen

First proposed in [21], Xen is now state-of-the-art opensource virtualization platform that is adopted widely due to its scalability, OS neutrality, high performance and lightweight features. EmuEdge is built on Xen for full-system virtualization and networking, therefore we present a brief overview of Xen for better understanding.

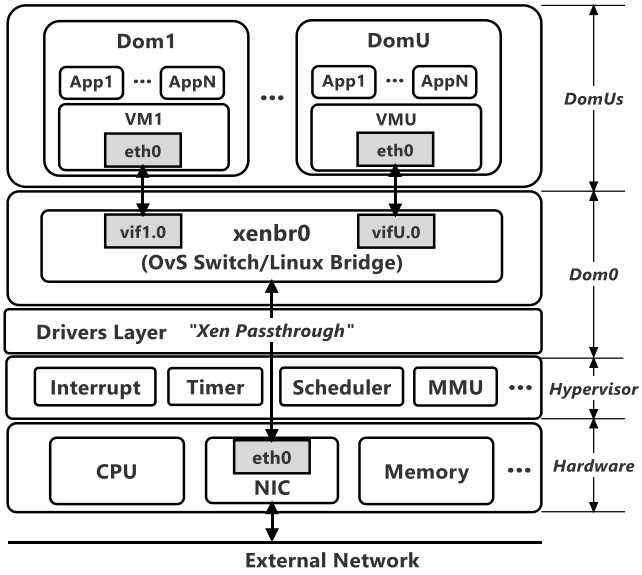


Figure 2: Xen internals and networking

Xen Basics: Generally, a Xen instance can be divided into several levels as shown in Figure 2. The two distinctive components of Xen are domain and hypervisor. In Xen, a running VM instance are 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 over 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: Linux bridge is the standard networking mode adopted in Xen. On boot, Xen Dom0 creates a bridge, e.g., *xenbr0*, for each physical NIC. When booting a DomU, Xen generates a virtual interface (mostly referred to as *vif*) in Dom0 that links to the virtual NIC in DomU and connects it to a specified bridge created before, thereby all traffics from DomUs can be directed to the physical interface. With recent advances in SDN, another networking approach based on OvS [45] 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 [38]. Bridges and OvS switches can also be created independent from physical interfaces, thereby enabling internal LANs between multi VMs.

VM Snapshot/Clone[2]**.** 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. 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. Several types of Snapshot are supported in Xen while we mainly concern about:

- **Disk-only snapshots:** disk-only snapshots only capture metadata and virtual disk storage for a VM, which allow exporting and restoring VM states for backup purpose.
- **Disk and memory snapshots:** besides VM configuration and storage data, disk and memory snapshots capture the VM RAM state, through which a VM instance can restore to a previous running state exactly.

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 while 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.

3 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 demonstrate the design of EmuEdge system as well as a framework of reproducing real-world experiments on EmuEdge.

3.1 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.

[2]Snapshot is a feature on 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

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 2. The hosts are independent from each other in terms of network, i.e., they hold different knowledges 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 [45], Linux Traffic Control [46] and Linux netem [36].

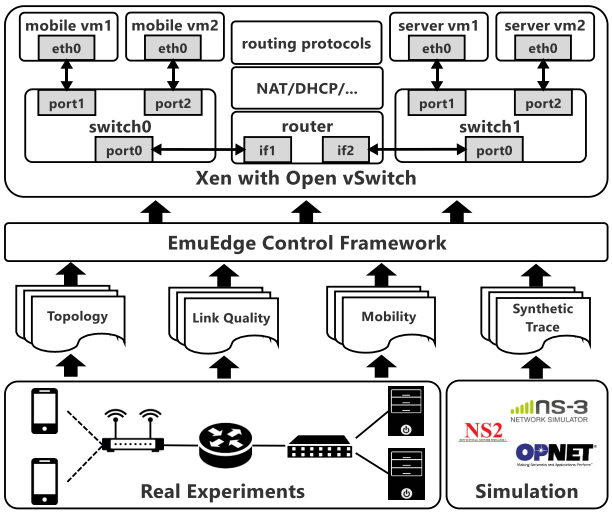


Figure 3: EmuEdge reality reproduction framework

Computation is an equally important part of typical edge computing platform, however it's rarely taken into account in previous emulators. EmuFog [19], a descendant of Mininet tailored for Fog Computing, are inherently limited to network bounded emulations. Computation bounded experiments such as Hadoop testing are considered out of the scope for Mininet systems according to [30]. 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 minimized through a hybrid combination of container and VM. Afterall, the cost of EmuEdge is still miniscale comparing to a physical edge computing deployment. For example, the venue fee of a field deployment for disaster response edge computing solution in Disaster City [47] 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, where physical nodes are mapping to virtual hosts. In EmuEdge, both wireless APs and Ethernet switches are abstracted as OvS switches, we differentiate the links primarily based on link quality emulation through QoS and network trace replay.

3.2 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 depicts the typical workflow of reproducing an field experiments 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 concern in EmuEdge includes:

- **Emulation Parameters:** primarily composed of hardware configurations such as allocation of CPU cores, memory, disk. In the cases where high-fidelity computation virtualization is needed, 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 adapt to 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.

4 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.

4.1 EmuEdge Components

To create a heterogeneous edge computing prototype, EmuEdge should be able to emulate both network and computation devices. The network infrastructure of edge computing systems 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, ends of Linux *veth* are the most common netifs in EmuEdge. EmuEdge designs the relationship between any nodes and network interfaces to be one-to-many so that any device might possibly be sitting in 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 a 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: Devices 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[1] 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 5, 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 knowledges. 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 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 a 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 it 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.

4.2 Network Realism

We consider two 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 applied 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 params 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,

[1] "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.

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

Table 2: control function support for different link metrics

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 [48], 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 reproducibility and isolation. EmuEdge exposes functions for defining per-link network metrics through traffic shaping as shown in Table 2. Every metric can be configured base and correlation parameters. The base parameter is a fixed time for delay (also known as round trip 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 abovementioned metrics therefore we refer readers to NISTNet [29] 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[2], 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) [29]. By combining the distribution table with statistical metrics (e.g., mean, variation and correlation) learnt 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 experiment sections.

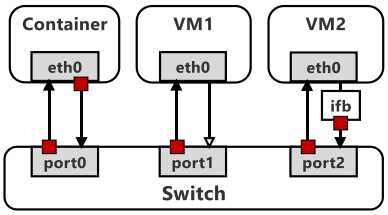


Figure 4: bidirectional QoS approaches (red boxes represents tc egress control)

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 [3] on both ends of *veth* link as shown in the c1-switch (container to switch) connection of Figure 4. 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 4. 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 overheads. We consider the investigation of other such approaches and their overheads out of scope in this paper.

4.3 Computation Realism

EmuEdge focuses majorly on two types of computational realism, which includes computation heterogeneity and isolation. Multiple levels of realism are supported in EmuEdge to achieve proper computation realism with minimal costs. With physical integration undoubtedly being the ultimate level in fidelity, 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

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

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

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 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 allocated 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.

4.4 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 VM for reproduction use. Additionally, a snapshot of a well configured machine can be fastly cloned and scaled. In the experiments section, 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[4]. 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.

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

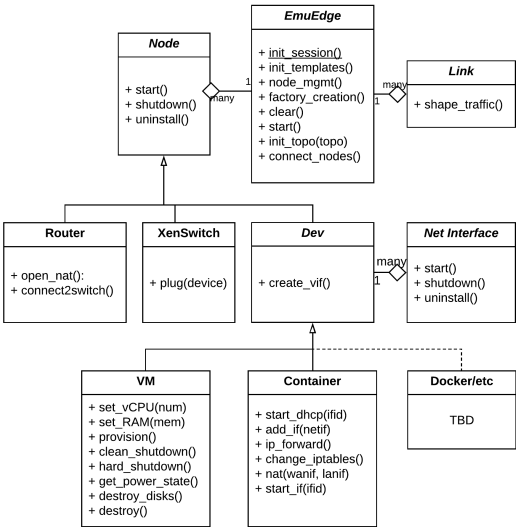


Figure 5: extensible eesign of EmuEdge to support heterogeneous emulation

Extensibility: EmuEdge is designed with flexible architectures, relations between part of EmuEdge components are shown in Figure 5. Through proper abstraction, EmuEdge system enables standardized behaviors of network components with different implementation details. So far, EmuEdge supports VM and container as virtual devices and another five types of Linux/Xen network interfaces (netif). Adapting EmuEdge with additional virtualization platforms such as docker would require trivial efforts.

5 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, 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.

5.1 Create Edge Network with EmuEdge 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:

```
import xen
xnet=xen.xnet_interactive()
d1=xnet.create_new_dev("android", "d1",
    True, vcpu=2, mem=2048)
r1=xnet.create_new_xrouter("r1", "10.0.0.1/24")
r1.plug(d1)
xnet.start_all()
xnet.clear()
```

The *create_new_dev* API creates an vm based on "android" snapshot and override the new VM configurations with 2 fixed vCPUs and

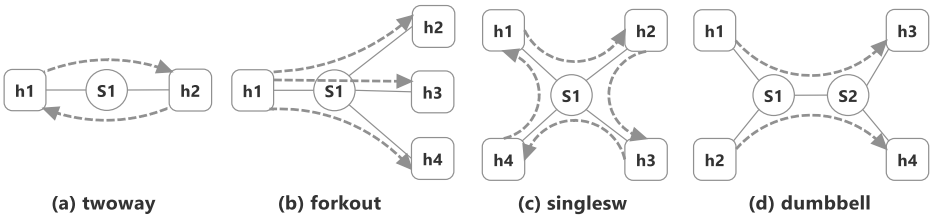


Figure 6: network fidelity validation topologies

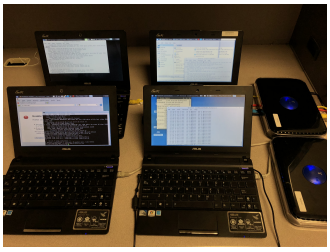


Figure 7: experiment testbeds

2048 MB static memory. For XenRouter, the initialization method assigned the ip "10.0.0.1/24" to the initial interface it has after creation.

5.2 EmuEdge Definition Language

Besides the interactive API approach, a better 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 API based approach, the JSON method is more concise and straightforward. Like we have discussed before, a front-end UI can be made with trivial efforts for even more intuitive definition process. For different type of nodes, we have different properties in configurations as shown in Figure 8.

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 8: EmuEdge topology definition language in JSON

Through our definition language, 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. Configuring NAT at a certain interface would open it as a WAN exit so that any packets flowing through the router may be forwarded through it. 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 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.

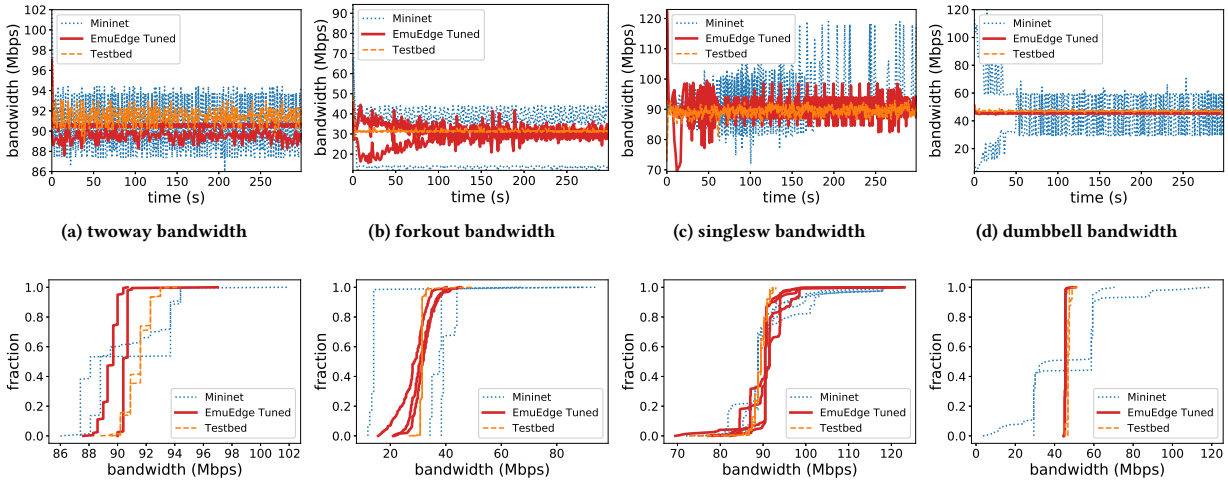
6 EXPERIMENTAL EVALUATIONS

In this section, we validate the effectiveness of EmuEdge from both the computational 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.

6.1 Network Fidelity Validation

To demonstrate the effectiveness of EmuEdge in ensuring network fidelity, we adopted validation tests as proposed in Mininet benchmark [49]. Mininet-HiFi was shown to have less variations and higher reproducibility than testbeds in these experiments[30]. 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 **??** 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 [49] where hosts connect to each other through a switch instead of direct link.



(a) twoway bandwidth (b) forkout bandwidth (c) singlesw bandwidth (d) dumbbell bandwidth

(e) twoway bandwidth cdf (f) forkout bandwidth cdf (g) singlesw bandwidth cdf (h) dumbbell bandwidth cdf

Figure 9: bandwidth of different network topologies

The experiment setup for our three comparison scenarios are as follows:

- **Testbed:** As shown in Figure 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.[5]

For each of the 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 9. Different from Mininet 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 7 (b) 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 9 (c) that sometimes EmuEdge fluctuates too much and does not converge well. This didn't match well with real world and can lead to untrusted experiment results. Therefore, we modified our EmuEdge setup with well-tuned rate limiting parameters, which can be easily done through EmuEdge definition language. 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 **??**. 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.

6.2 Replaying Wireless Network

Wireless networks add uncertainties into edge computing systems with delays, jitters and possibly intermittent transmissions due to

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

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:

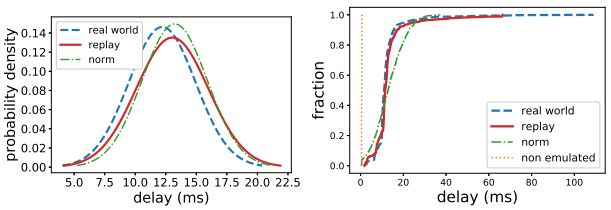
Normal approximation: EmuEdge is able to approximate normal distributed delays among links. One can easily generate necessary stats information (e.g., mean and standard deviation) that captures the normal distribution approximation based on a real-world network trace for normally approximated EmuEdge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

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 10.



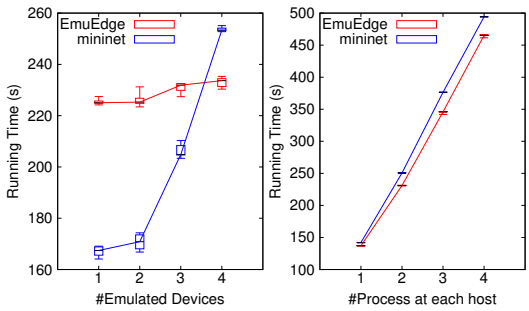
(a) probability density function **(b) cumulative distributed function**

Figure 10: 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 10a, 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 10b while normal method is statistically too ideal.

6.3 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 [50] and Docker [51].

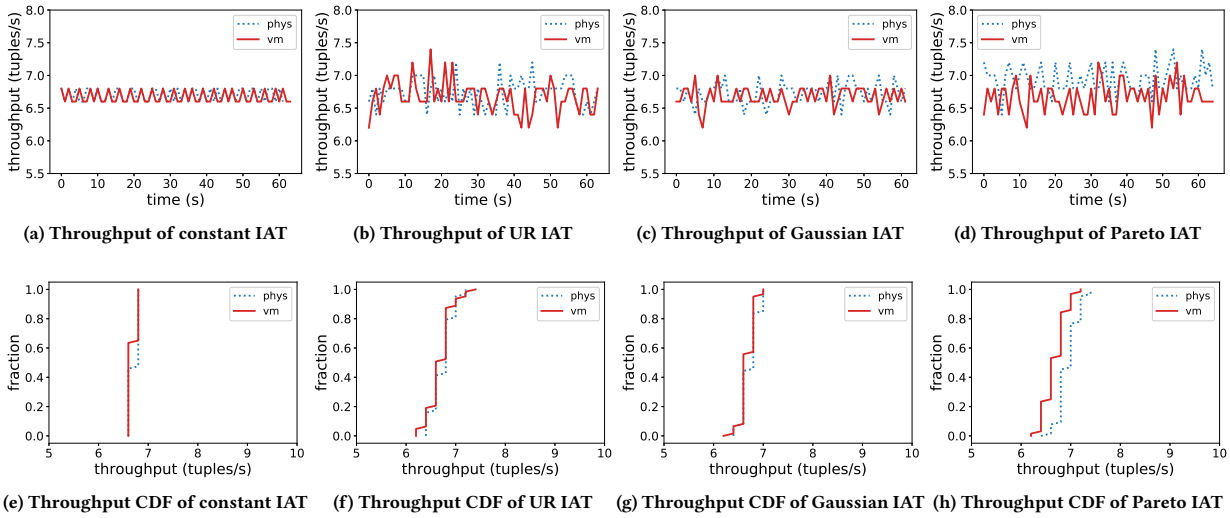


(a) computation isolation **(b) computation efficiency**

Figure 11: 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 [52] workloads. We then measure workloads execution time on each node while increasing total number of node running. As shown in Figure 11a, 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 11a that when both PCs are stressed over 75%, the performance degradation of Mininet containers are significant that it even



(a) Throughput of constant IAT (b) Throughput of UR IAT (c) Throughput of Gaussian IAT (d) Throughput of Pareto IAT

(e) Throughput CDF of constant IAT (f) Throughput CDF of UR IAT (g) Throughput CDF of Gaussian IAT (h) Throughput CDF of Pareto IAT

Figure 12: performance comparison between vMStorm and physical MStorm

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 11b 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 [53].

However, we argue that VM vs. container performance results are bounded to specific cases. In our settings, Mininet containers possibly suffer from higher centralized scheduling overhead 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.

7 PRACTICAL EXPERIENCES

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 13 depicts the physical and hybrid setup for following experiments, the hybrid setup can be fully virtualized with EmuEdge by using a Master Server container.

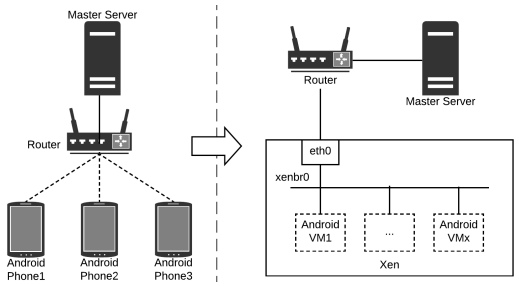
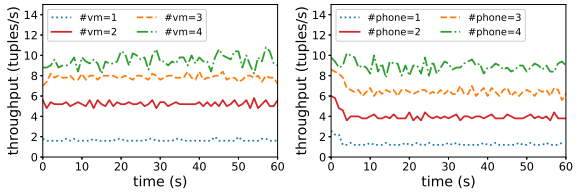


Figure 13: physical and hybrid (EmuEdge) setup of actual edge computing platforms in expriments

Mobile Storm (MStorm) [54] is an online distributed stream processing system on Android. Different from the datacenter counterpart [55], 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 12 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.

Scalability Realism: Besides single node performance, we investigate on how EmuEdge reflects system performance improvement when scaling and compare the results with an identical 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 14 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.



(a) vMStorm Scalability **(b) MStorm scalability**

Figure 14: realistic scalability experiments on EmuEdge

8 DISCUSSION

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 levels 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 are still limited in several aspects:

Background Workload Realism: In edge computing, mobile nodes can be handling 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 lost 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 realism of 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] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh 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*, pages 13–16. ACM, 2012.

[3] Rabindra K Barik, Ankita Tripathi, Harishchandra Dubey, Rakesh K Lenka, Tanjappa Pratik, Suraj Sharma, Kunal Mankodiya, Vinay Kumar, and Himansu Das. Mistgis: Optimizing geospatial data analysis using mist computing. In *Progress in Computing, Analytics and Networking*, pages 733–742. Springer, 2018.

[4] I Hou, Tao Zhao, Shiqiang Wang, Kevin 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*, pages 291–300. ACM, 2016.

[5] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[6] Ting He, Hana Khamfroush, Shiqiang Wang, Tom La Porta, and Sebastian Stein. ItâĂŹs hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. Technical report, Technical Report, December 2017.[Online]. Available: https://1drv. ms/b/s, 2018.

[7] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[8] Jan Plachy, Zdenek Becvar, and Emilio Calvanese 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*, pages 1–6. IEEE, 2016.

[9] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2016.

[10] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[11] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[12] Sumant Ramgovind, Mariki M Eloff, and Elme Smith. The management of security in cloud computing. In *Information Security for South Africa (ISSA), 2010*, pages 1–7. IEEE, 2010.

[13] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[14] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar 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*, 47(9):1275–1296, 2017.

[15] Bob Lantz, Brandon Heller, and Nick 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*, page 19. ACM, 2010.

[16] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264. ACM, 2012.

[17] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 384–389. IEEE, 2015.

[18] Philip Wette, Martin Draxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[19] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. *arXiv preprint arXiv:1709.07563*, 2017.

[20] Hend Gedawy, Sannan Tariq, Abderrahmen Mtibaa, and Khaled Harras. Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading. In *Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE, 2016.

[21] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[22] Lori Pridmore, Patrick Lardieri, and Robert Hollister. National cyber range (ncr) automated test tools: Implications and application to network-centric support tools. In *AUTOTESTCON, 2010 IEEE*, pages 1–4. IEEE, 2010.

[23] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[24] John Wroclawski, Terry Benzel, Jim Blythe, Ted Faber, Alefiya Hussain, Jelena Mirkovic, and Stephen Schwab. Deterlab and the deter project. In *The GENI Book*, pages 35–62. Springer, 2016.

[25] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[26] Toshiyuki Miyachi, Ken-ichi Chinen, and Yoichi 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*, page 30. ACM, 2006.

[27] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[28] Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.

[29] Mark Carson and Darrin Santay. Nist net: a linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[30] Brandon Heller. *Reproducible network research with high-fidelity emulation*. PhD thesis, Stanford University, 2013.

[31] Jakub Dolezal, Zdenek Becvar, and Tomas 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*, pages 1–7. IEEE, 2016.

[32] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *ACM SIGSIM Simulation Digest*, volume 28, pages 154–161. IEEE Computer Society, 1998.

[33] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[34] Xinjie Chang. Network simulations with opnet. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 307–314. ACM, 1999.

[35] Mohammed Islam Naas, Jalil Boukhobza, Philippe Raipin Parvedy, and Laurent 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*, pages 1–8. IEEE, 2018.

[36] Stephen Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.

[37] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[38] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[39] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference*, pages 113–128, 2008.

[40] Maurizio Pizzonia and Massimo 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*, page 7. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[41] Sapan Bhatia, Murtaza Motiwala, Wolfgang Muhlbauer, Yogesh Mundada, Vytautas Valancius, Andy Bavier, Nick Feamster, Larry Peterson, and Jennifer Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 72. ACM, 2008.

[42] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. Core: A real-time network emulator. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE, 2008.

[43] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Operating Systems Review*, 36(SI):271–284, 2002.

[44] Diwaker Gupta, Kashi Venkatesh Vishwanath, Marvin McNett, Amin Vahdat, Ken Yocum, Alex Snoeren, and Geoffrey M Voelker. Diecast: Testing distributed systems with an accurate scale model. *ACM Transactions on Computer Systems (TOCS)*, 29(2):4, 2011.

[45] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *NSDI*, pages 117–130, 2015.

[46] Werner Almesberger. Linux traffic control-implementation overview. Technical report, 1998.

[47] Texas A&M University System. Teex | disaster cityÂő, 2018.

[48] Emily Nunez. Distressnet-ng: Resilient mobile broadband communication and edge computing. 2017.
[49] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Mininet performance fidelity benchmarks. *Tech. Rep.*, 2012.
[50] Matt Helsley. Lxc: Linux container tools. *IBM devloperWorks Technical Library*, 11, 2009.
[51] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.
[52] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[53] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange, and Cesar AF 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*, pages 233–240. IEEE, 2013.
[54] Qian Ning, Chien-An Chen, Radu Stoleru, and Congcong Chen. Mobile storm: Distributed real-time stream processing for mobile clouds. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 139–145. IEEE, 2015.
[55] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.