2018 International Conference on Unmanned Aircraft Systems (ICUAS)
Dallas, TX, USA, June 12-15, 2018

Enabling High-Performance Onboard Computing with Virtualization for Unmanned Aerial Systems

Baoqian Wang and Junfei Xie
Department of Computing Sciences
Texas A&M University Corpus Christi
Corpus Christi, TX, 78412
Email:junfei.xie@tamucc.edu

Songwei Li and Yan Wan
Department of Electrical Engineering
University of Texas at Arlington
Arlington, TX, 76010
Email: yan.wan@uta.edu

Shengli Fu
Department of Electrical Engineering
University of North Texas
Denton, TX, 76203
Email: shengli.fu@unt.edu

Kejie Lu
Department of Computer Science and Engineering
University of Puerto Rico at Mayagüez
Mayagüez, Puerto Rico, 00681
Email: kejie.lu@upr.edu

***Abstract*—In recent years, unmanned aerial systems (UAS) have attracted significant attentions because of their broad civilian and commercial applications. Nevertheless, most existing UAS platforms only have limited computing capabilities to perform various delay-sensitive operations. To tackle this issue, in this paper, we develop a high-performance onboard UAS computing platform with the virtualization technique. Specifically, we first discuss the selection of microcomputers that are suitable for UAS onboard computing. We then investigate virtualization schemes that can effectively manage constrained resources in UAS, flexibly support UAS applications, and enable resource sharing among multiple UAS to achieve higher computing power. In our study, we compare the performance (such as computing, network, isolation, etc.) of two representative virtualization techniques including virtual machine (VM) and container, using KVM and Docker, respectively. Extensive experimental results demonstrate the performance trade-offs between VM and container, and validate the benefits of virtualization in supporting real UAS applications.**

***Keywords*—Unmanned Aerial Systems, Virtualization, Onboard Computing, Testbed Development, Performance Evaluation.**

I. Introduction

Over the past few years, unmanned aerial systems (UAS) have increasingly gained their popularity in broad commercial and civilian applications, such as forest-fire detection [1], reconnaissance [2], search and rescue [3], and 3-D mapping [4]. As existing UAS platforms have very limited onboard computing resources, computation-intensive tasks in these applications such as image processing are usually conducted at ground stations. However, due to the limited bandwidth resources of UAS-to-ground communication channels and the existence of unknown environmental disturbances that affect channel performance, deferring computing tasks to ground stations can lead to considerable delays, data losses or even transmission failures. With the growing complexity of UAS applications, real-time computing becomes increasingly critical for the success of many UAS tasks.

UAS are limited by the computing resources that they can carry. It is crucial to optimize the management of constrained UAS computing resources, and offload less time-critical tasks to other computing devices. This can be achieved by virtualization, which is known for its powerful resource management capabilities and live migration support that handles dynamic workloads [5]. Virtualization has many other attributes that can further enhance the capability of UAS. For instance, it can isolate unreliable and untrustworthy functionalities, and improve the resilience of UAS to malicious attacks [6]. It can also facilitate the concurrent execution of multiple applications with different operating system (OS) requirements on the same UAS. Furthermore, virtualization can also facilitate networked airborne computing operations on multiple connected UAS, with improved computational power for advanced UAS applications. As a step towards realizing this platform, our team has successfully developed a robust broadband UAS-carried communication platform that allows high-speed long-distance communication between two UAS [7]–[9]. In this paper, we investigate the capability of virtualization in enhancing the onboard computing capacity of UAS.

In the literature, virtualization for server-based devices has been widely studied over the past few decades [10]. Many works can be found that evaluate and compare the performance of different virtualization techniques for these devices. For instance, the performances of container- and hypervisor-based virtualization techniques are analyzed in [11], [12] and [13]. Paper [14] compares the two types of virtualization techniques. The use of virtualization in high performance computing and cloud environments has also been studied in e.g., [15], [16].

More relevant to this study, mobile virtualization has aroused increasing attentions with the wide-spreading use of mobile devices and fast evolution of ARM processors [10]. A container-based virtualization technique on Raspberry Pi 2 was investigated in [17]. Several hypervisor-based virtualization techniques on Cubieboard2 were compared in [18].

In [19], the performances of hypervisor- and container-based virtualization techniques on the Insignal Arndale board were compared. Since these studies were not directed to advanced UAS computing applications, they adopted microcomputers of very limited computing and storage resources, and their performance analysis was limited to CPU, Memory and Disk. The unique features of UAS, such as small payload, power constraint, and real-time computing need were not considered.

There are very limited studies on the direction of virtulization for UAS. The Kernel-based Virtual Machine (KVM), a hypervisor-based virtualization technique, was applied to enhance the resilience of UAS to malicious attacks, where KVM was implemented in Raspberry Pi 2 [6]. The Nutanix Acropolis [20], a UAS cloud platform, is another example that applies the virtualization technology. This platform uses the Acropolis Hypervisor to hold multiple virtual machines (VMs). However, a comprehensive investigation of mobile virtualization to enable high-performance onboard UAS computing is still lacking.

In this paper, we investigate two representative virtualization techniques, VM using KVM [21] and container using Docker [22], with the purpose of enhancing UAS' onboard computing capability for more advanced UAS applications. The main contributions are summarized as follows:

1) *Recommendation of suitable microcomputers for high-performance onboard UAS computing:* Through a comprehensive analysis of the desired features of UAS onboard computing, we provide recommendations of suitable microcomputers.
2) *Implementation of KVM and Docker on Jetson TX2:* The newly released microcomputer, Jetson TX2, has rarely been studied in supporting virtualization functionalities. We provide instructions to implement KVM and Docker on Jetson TX2 selected in this study.
3) *Comprehensive performance evaluation of KVM and Docker in supporting UAS applications:* Extensive experiments are conducted to measure the performances of KVM and Docker from the aspects critical to UAS applications, including computing, network, and isolation. The insights obtained from the experimental results provide guidelines for the selection of appropriate virtualization techniques for UAS applications.
4) *Application of virtualization to real UAS applications:* Experiments are also conducted to illustrate the benefits of KVM and Docker in facilitating resource management and improving computing performance for real UAS applications.

The rest of this paper is organized as follows. Section II reviews KVM and Docker. Section III discusses the selection of appropriate onboard microcomputers for UAS, and the implementation of KVM and Docker on Jetson TX2. The performances of KVM and Docker are evaluated comprehensively in Section IV. Section V investigates the performances of KVM and Docker in supporting real UAS applications. Section VI concludes the paper with a brief summary and future works.

II. Review of KVM and Docker

Virtualization refers to the process of creating virtual (software-based) representations of physical hardware resources, which allows multiple OSs to coexist and share resources on the same hardware platform. Virtualization can significantly improve resource utilization and reduce hardware costs. In addition, virtualization can also enhance security and fault tolerance through the isolation technique.

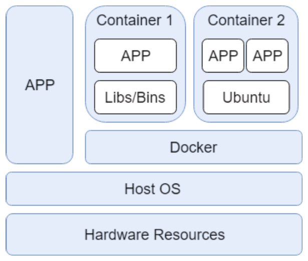
Based on the abstraction level, virtualization can be categorized into two types [23]: *hypervisor-based* virtualization and *container-based* virtualization. The hypervisor-based virtualization is performed directly on top of the hardware, while the container-based virtualization abstracts at the OS level. In this section, we briefly overview KVM [21] and Docker [24], which are representative hypervisor- and container-based virtualization techniques, respectively.

A. KVM

KVM (see Figure 1(a) for the system architecture) uses the hypervisor to abstract hardware resources and create VMs, which are simulated operating environments and access physical hardware resources through the hypervisor [25]. We call the hardware that runs the hypervisor as the *host* and emulated machines on top of the host (i.e., VMs) as the *guest*. Multiple applications (APPs) can run inside the VMs.

Unlike many hypervisor-based virtualization techniques that rely completely on the hypervisor to emulate and manage hardware resources, the KVM hypervisor is integrated into the Linux mainline kernel, and thus can leverage the functionalities of the Linux systems such as scheduling, memory management and timer handling for resource management. Moreover, in KVM, device emulation is offloaded from the hypervisor to a user-space software, usually quick emulator (QEMU) [26], which simplifies the virtualization process. This feature also makes KVM an efficient hypervisor-based virtualization technique that can be quickly deployed. A more detailed description of the hypervisor and QEMU, which are required to run VMs, is provided as follows.

1) *Hypervisor:* Hypervisor [21], also called VM monitor, is a software that acts as a control and translation system between VMs and the host. It requires the hardware virtualization support to achieve core virtualization features such as CPU and memory virtualization. In particular, the CPU virtualization requires the hypervisor to run in a privileged CPU mode higher than the guest OSs, and the memory virtualization requires the second level address translation [27] for VMs. The hypervisor retains the control of hardware resources and provides good isolation for the guest and host systems through trap-and-emulate. In particular, when a VM executes a sensitive instruction which may conflict the host system or other VMs, it



(a) (b)

Fig. 1. System architectures of a) KVM and b) Docker.

traps to the hypervisor which then executes this instruction or emulates the effect of this instruction without execution.

2) *QEMU:* KVM mainly focuses on the CPU and memory virtualization and cannot create VMs by itself. It requires an emulator, usually QEMU, to emulate peripheral devices such as disk, network, and USB controller and thus achieve the I/O virtualization.

B. Docker

Docker (see Figure 1(b) for the system architecture) is a popular container-based virtualization technique. It has attracted significant attentions because of its features including rapid application deployment, portability across machines, simplicity, and faster configurations [24]. Different from hypervisor-based solutions that require virtualization of hardware resources which generates significant performance overhead, Docker provides a lightweight virtual environment called container by abstracting at the OS level. The container shares the kernel with the host OS, and hence its creation only requires a root file system (e.g., Ubuntu). A single APP can also be packaged as a container with provided libraries (libs) and bins. These features make containers much smaller and more lightweight than VMs, consume fewer resources and can be quickly deployed with lower performance overhead. Due to the OS-level virtualization, containers do not require hardware virtualization support.

Docker relies on the following Linux system utilities to run containers:

1) *Linux namespaces* which realize the isolation among containers, by restricting processes in a namespace and only using resources associated with that namespace. Six namespaces, including Process ID (PID), Networking (NET), InterProcess Communication (IPC), Managing Filesystem Mount Points (MNT), Unix Timesharing System (UTS) and User [28], achieve the isolation for different resources. For instance, PID namespaces isolate process ID number space, thus allowing processes with the same PID to run in different PID namespaces. The NET namespaces isolate resources related to networking, such as network devices.
2) *Control groups (cgroups)* which realize the resource allocation and isolation for containers. They allow Docker to assign available resources to containers with optional limits and constraints. Note that the Linux namespaces determine what resources a process can access, while the cgroups determine how many resources a process can use.
3) *Advanced Multi-Layered Unification Filesystem (AuFS)* which is used by Docker to manage files. AuFS is a layered filesystem that can merge multiple directories into a single representation of the directory. AuFS also allows different containers to be built by simply adding new layers into the base OS images. As only one copy of the base images is required, storage saving and faster deployment of containers can be achieved.

III. Microcomputer Selection and Virtualization Implementation

In this section, we first discuss the selection of microcomputers for UAS applications that require high performance onboard computing capability. The implementation of KVM and Docker on the selected microcomputer is then discussed.

A. Microcomputer Selection

The selection of the microcomputers for high-performance computing onboard UAS should consider computing, system control, communication, network, and virtualization characteristics. For instance, the microcomputer should have a powerful CPU (e.g., at least 2 cores), sufficient memory (e.g., 8GB) and storage (e.g., at least 32GB) to support most computing needs. Based on these criteria, an initial search leads to four candidates: Jetson TX2 [29], UDOO X86 ULTRA [30], UP Squared [31] and LattePanda Alpha [32]. Next, we conduct a detailed comparison among these microcomputers from various aspects including the processing capability, connectivity, dimension, power consumption, OS, weight, virtualization support, and storage.

Table I summarizes the comparison results. In particular, Jetson TX2 has the highest computing power, which contains a hex-core CPU complex (combining a dual-core Denver 2 and a quad-core ARM A57). The more number of cores allows Jetson TX2 to create more well-isolated VMs or containers that each occupies a core exclusively. In addition, Jetson TX2 has a very powerful GPU, which permits real-time image or video processing and deep learning capabilities. It also provides an out-of-the-box high-throughput wireless local area network (WLAN) interface. From the dimension aspect, Jetson TX2 is smaller than the other three microcomputers, but it needs to run on a carrier-board [29]. The original carrier-board for Jetson TX2 has a large size (17cm $\times$ 17cm), which limits its use for UAS applications. UDOO X86 ULTRA outperforms the other two in power consumption and OS support. UP Squared has a largest storage. All four microcomputers support virtualization.

The above analysis provides us with guidelines to select proper microcomputers for UAS. A trade-off should be

TABLE I. COMPARISON OF DIFFERENT MICROCOMPUTERS

	Jetson TX2	UDOO X86 ULTRA	UP Squared	LattePanda Alpha
CPU	Denver 2 (2 cores) 2MB Cache, 2GHz + ARM® A57 (4 cores) 2MB Cache, 2GHz	Intel® Pentium N3710 (4 cores) 2MB Cache, 2.56GHz	Intel® Apollo Lake (2-4 cores)	Intel® 7th Gen M3-7Y30 (2 cores) 4 MB Cache, 2.60GHz
GPU	NVIDIA Pascal^{TM} 256 CUDA cores, 1.12GHz	Intel® HD Graphics 16 units, 405-700MHz	Intel® Gen 9 HD with 12 (Celeron) or 18 (Pentium) Execution Units	Intel® HD Graphics 615 300-900MHz
Memory	8GB LPDDR4	8GB DDR3L	up to 8GB LPDDR4	8GB LPDDR3
Connectivity	1 Gigabit Ethernet 802.11ac WLAN, Bluetooth	1 Gigabit Ethernet M.2 Key E slot for optional Wireless modules	2 Gigabit Ethernet	1 Gigabit Ethernet 802.11ac WLAN, Bluetooth
Dimension	50×87 mm	120×85 mm	85.6×90 mm	$113 \times 80 \times 13.5$ mm
Power consumption (under load)	7.5 watt	6 watt	Not available	Not available
OS	Linux	Windows 10, 8.1, 7, Linux, Android	Windows 10, Linux, Android	Windows, Linux
Weight	85g	117g	Not available	104g
Virtualization support	Yes	Yes	Yes	Yes
Storage	32GB eMMC	32GB eMMC	up to 128GB eMMC	64GB eMMC

achieved among multiple performance aspects based on the needs of specific UAS applications. For instance, if flight time is more critical than real-time processing, UDOO X86 ULTRA that consumes less power may be selected. In this study, we explore Jetson TX2 of high computing capacity to enable high-performance onboard UAS computing. To resolve the dimension issue caused by the carrier-board, we have developed a small, lightweight and battery-powered carrier-board for Jetson TX2.

B. Implementation of KVM and Docker on Jetson TX2

In this section, we provide instructions on the implementation of KVM and Docker in Jetson TX2 to achieve isolation of CPU, memory, and I/O. The GPU virtualization for Jetson TX2, which is more complicated, is left for furture research.

1) Implementation of KVM: As Jetson TX2 supports virtualization, we implement KVM on this platform. In particular, to achieve CPU virtualization, KVM hypervisor needs to run in a mode with a privilege higher than the guest OSs. This can be achieved by the *Hyp* mode in Jetson TX2 [33], which is also the default CPU mode.

To run KVM hypervisor, we need to enable KVM configurations available in the Linux kernel of Jetson TX2 when we compile the kernel, which are disabled by default. Note that the KVM functionality has been merged to the Linux kernel for ARM devices since version 3.9 [21]. Another modification to make is to enable the virtual generic interrupt controller (vGIC), which is required by KVM to manage virtual interrupts generated by VMs [21]. This is achieved by modifying the device tree of Jetson TX2.

With the KVM hypervisor built successfully, the next step is to create VMs, which requires the installation of several software, including QEMU for I/O virtualization, AAVMF [34] for guest OS installation, and *libvirt* [35] for VM management. To improve the usability of VMs, additional software can be installed, such as *brctl* [36] that helps the management of VM networks and *virt-manager* [37] that provides a graphical user interface to facilitate VM management.

2) Implementation of Docker: As Docker performs at the OS level, it relies on the functionalities of the Linux system to achieve virtualization. Usually, to run Docker containers in a Linux system, kernel configurations such as cgroups and namespaces need to be enabled. Recently, a new kernel for Jetson TX2, i.e., L4T 28.2 RC [38], was released, which officially supports Docker and does not require any additional configurations.

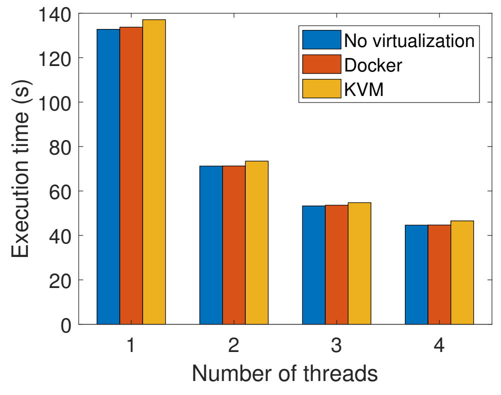
IV. COMPARATIVE PERFORMANCE EVALUATION

Virtualization significantly extends the functionality of UAS, but also introduces performance overhead due to resource partition, isolation and emulation. In this section, we investigate the performance of KVM and Docker on Jetson TX2 in supporting UAS applications.

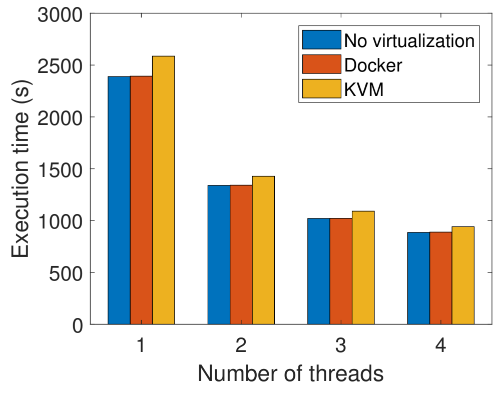
A. Computing Performance

Computing performance of the onboard microcomputer is critical for the success of real-time UAS tasks. Performance degradation caused by virtualization needs to be minimized. In this section, we evaluate the impact of KVM and Docker on the computing performance of Jetson TX2.

1) Experimental Setup and Benchmark: The experimental setup consists of three Jetson TX2, where KVM and Docker are implemented separately on two microcomputers, and no virtualization is applied on the third one. A single VM (or container) is then created by KVM (or Docker). The same OS, i.e., Ubuntu 16.04 LTS for ARM with Linux kernel version 4.4, is implemented on both host and guest systems.

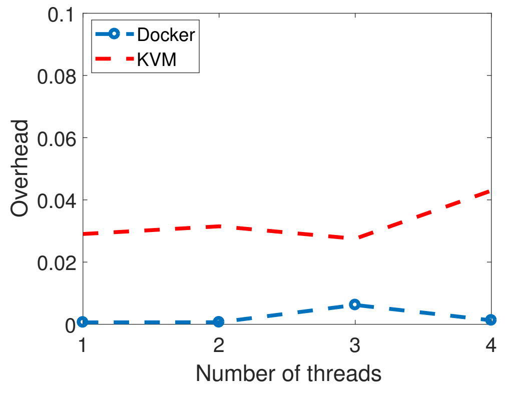


(a)

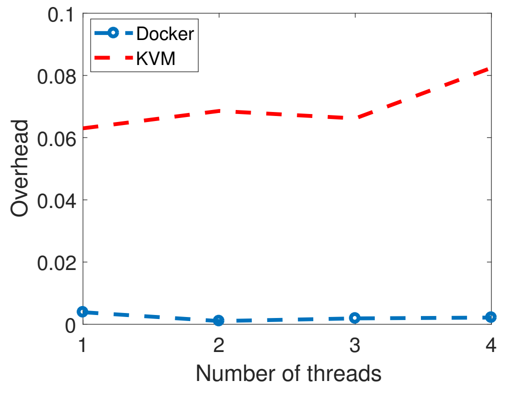


(b)

Fig. 2. Execution time of the LU application running in Jetson TX2 of different virtualization setups and with increasing number of threads introduced. The cubic sizes in the LU application are set to a) $64 \times 64 \times 64$ and b) $102 \times 102 \times 102$.



(a)



(b)

Fig. 3. Computing performance overhead generated by KVM and Docker with increasing number of threads introduced. The cubic sizes in the LU application are set to a) $64 \times 64 \times 64$ and b) $102 \times 102 \times 102$.

To evaluate the computing performance of the three microcomputers, we adopt the NAS Parallel Benchmark [39], which contains a set of computation-intensive applications. The execution times of these applications are used to estimate the computing performance. This benchmark also supports multi-threading, and thus allows the applications to be executed by multiple cores in parallel to test the parallel computing performance. In this experiment, we run one of the applications called Lower-Upper Gauss-Seidel solver (LU), which performs a synthetic computational fluid dynamics calculation for a cubic region of configurable size [39]. A larger cubic results in more computational costs. We implement this benchmark in the VM (or container) if available, or otherwise directly in the host system.

2) Experimental Results: In this study, we run the LU application of different problem sizes on the three microcomputers. Each experiment is repeated for 10 times and the average results are presented to reduce the experimental uncertainty. Note that this procedure is also applied for each experiment conducted in the following studies. The average execution time required by the three microcomputers to run the LU application with increasing number of threads is shown in Figure 2. As expected, the Jetson TX2 without virtualization has the best performance. Docker outperforms KVM, and has a very minor impact on the computing performance. The superiority of Docker is more evident when running the LU application of larger problem size that requires more memory resources, as shown in Figure 2(b). This can be explained by the complexity of KVM in allocating memory resources. In particular, KVM requires second level address translation to allocate virtual memory for VMs, while Docker directly utilizes the Linux system utility, i.e., cgroups, to allocate memory for containers without additional translation.

For better illustration, we further evaluate the overhead introduced by KVM (or Docker) using the following equation:

$$\text{Overhead} = \frac{T_v - T_h}{T_h} \qquad (1)$$

where T_v and T_h respectively represent the execution time of running the LU application in the VM created by KVM (or container created by Docker) and directly in the microcomputer without virtualization. As we can see from Figure 3, KVM creates more overhead than Docker, and the overhead grows with the increase of problem size. This observation matches the results shown in Figure 2. Another interesting observation we can obtain from Figure 3 is that the overhead created by KVM tends to increase with more threads introduced, while the performance of Docker is relatively stable. This is because KVM needs to virtualize physical CPU cores for VMs to execute multi-threading tasks, while Docker does not require CPU virtualization to achieve this.

B. Network Performance

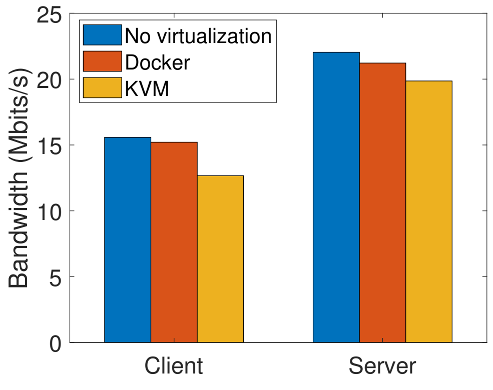
Information sharing among UAS as well as between UAS and ground stations requires the onboard UAS computing platforms to have a good network performance. In this section, we evaluate the network performance of KVM and Docker on Jetson TX2.

1) Experimental Setup and Benchmark: In multi-UAS applications, two types of communication links exist, ground-to-UAS and UAS-to-UAS. We here design two experiments to test both scenarios. Specifically, in the first experiment, we link a Jetson TX2 and a ThinkPad E540 laptop to simulate the communication between a UAS and a ground station. KVM or Docker is implemented on Jetson TX2. In the second experiment, we link two Jetson TX2 to simulate the communication between two UAS, and the same virtualization technique is implemented in both platforms. In both experiments, the communication link is established through a Technicolor TC8715D Wi-Fi router with a bandwidth of 40 Mbps.

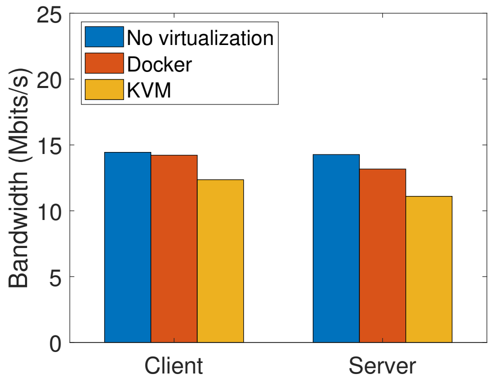
To measure the network performance, we adopt the Iperf benchmark [40], which is implemented on both Jetson TX2 and the ThinkPad laptop. This benchmark works by sending data streams from the client side to the server side and then measuring the throughput. In this study, Jetson TX2 can either be a client or server, and both cases are tested. Packets are transferred based on TCP/IP with the size set to 128KB.

2) Experimental Results: The network performances of Jetson TX2 virtualized by KVM or Docker under both network settings (i.e., client and server) in the two experiments are shown in Figure 4. The performance of Jetson TX2 without virtualization is also evaluated for the comparison purpose. As shown in the figure, Docker achieves better network performance than KVM in both experiments. This is because Docker adopts a simpler network virtualization mechanism. In particular, containers directly use the Linux system utilities, such as network namespace and veth pair [41], to communicate with the outside world without emulating hardware devices. However, KVM introduces additional software, e.g., QEMU, to emulate network devices for VMs, which results in additional network overhead.

Now let us analyze the results from each experiment. In the first experiment that simulates the ground-to-UAS communication (see Figure 4(a)), higher bandwidth is achieved when Jetson TX2 acts as a client. This may be caused by the different network devices used by Jetson TX2 and the ThinkPad laptop. Of interest, in the second experiment (see Figure 4(b)), when two identical Jetson TX2 with virtualization communicate with each other, the bandwidth measured at the server side is slightly smaller than that measured at the client side. This is because messages from the outside world cannot directly access VMs (or containers) due to security



(a)



(b)

Fig. 4. Bandwidth of the communication link a) between Jetson TX2 and ThinkPad laptop, and b) between two Jetson TX2. The bandwidth is tested at Jetson TX2 when it acts as a client or server.

issues. To receive packets, port forwarding is required [42], [43], which may introduce additional overhead.

C. Isolation Performance

In UAS applications, different programs need to be isolated for security purposes. There are also cases when multiple programs with varying system requirements need to be implemented in the same microcomputer. These scenarios can be realized by running these programs in different VMs (or containers), where the isolation performance is of importance.

1) Experimental Setup and Benchmark: To test the isolation performance of KVM and Docker, we conduct a set of experiments similar to those described in [15]. In particular, two Jetson TX2 are used in these experiments, where each runs two guests (VMs or containers). Each guest is assigned with half of the total CPU resources. The resource partition and allocation are achieved by the hypervisor in KVM, and cgroups in Docker.

The Isolation Benchmark Suite (IBS) [44] is used to measure the isolation performance. The key idea is to run stress tests that consume intensive resources in one guest, and measure the impact of these tests in the well-behaved guest. The two guests are well-isolated if the performance degradation of the well-behaved guest is small. IBS consists of six stress tests, including CPU intensive, memory intensive, disk I/O intensive, fork bomb, network transmit intensive, and network receive intensive. In this study, we run each of these stress tests in one guest, and run the LU application with cubic size of $64 \times 64 \times 64$ in the other guest. We then measure and compare the execution time of the LU application under stress tests, denoted as T_s, with its execution time evaluated when no stress tests are performed, denoted as T_n. The performance degradation, which reflects the isolation performance, is then measured by the following equation:

$$\text{Performance Degradation} = \frac{T_s - T_n}{T_n} \times 100\% \qquad (2)$$

2) Experimental Results: As shown in Table II, both Docker and KVM perform well in the CPU intensive test, as different CPU cores are assigned to different VMs (or containers). However, the impact of other stress tests on the performance of the well-behaved guest is noticeable, indicating the relative weakness of KVM and Docker in isolating memory, Disk I/O and network resources. Overall, Docker outperforms KVM in most of these tests due to the use of Linux namespaces, which provide good isolation properties. However, KVM relies on the hypervisor's trap-and-emulate mechanism to achieve isolation, which hangs the rest of the VMs when one traps to the hypervisor.

An exception is the poor performance of Docker in the fork bomb test, which quickly creates a large number of processes that overwhelm the OS until the process table is full. This can be caused by the sharing of the process table among containers.

TABLE II. PERFORMANCE DEGRADATION OF THE WELL-BEHAVED GUEST IN DIFFERENT STRESS TESTS

	Docker	**KVM**
CPU	0.36%	0.41%
Memory	5.03%	6.0%
Disk I/O	2.56%	2.9%
Fork bomb	6.24%	1.28%
Network receiver	2.25%	4.68%
Network sender	1.73%	2.53%

D. Discussion

The above comparative studies show the promising performance of Docker compared with KVM in computing, network and isolation of most hardware resources. However, Docker is less resilient to malicious attacks as the containers share the same OS kernel. Both KVM and Docker support live migration on server-based devices, but the realization of live migration for microcomputers is still in its infancy.

V. VIRTUALIZATION FOR REAL UAS APPLICATIONS

In this section, we further investigate the capability of KVM and Docker in facilitating real UAS applications. In particular, the computing performance of KVM and Docker in running UAS image processing applications is first evaluated. The benefits of KVM and Docker in improving resource utilization is then demonstrated through experiments.

A. Computing Performance

Image processing is a typical application performed onboard of UAS. In this study, we use the OpenDroneMap [45], an open-source software for UAS imagery processing, to measure the computing performance of KVM and Docker in supporting real UAS applications. Two OpenDroneMap functions of different complexity are selected. The experimental setting is similar as that described in Section IV-A1. In particular, KVM and Docker are implemented separately on two Jetson TX2 and another Jetson TX2 without virtualization for the comparison purpose. OpenDroneMap functions are then implemented in the guest (VM or container) or the host if no virtualization is applied. To evaluate the computing performance of KVM and Docker, 41 UAS images downloaded from the website [46] are used. The size of each image is 3.9MB.

In the first experiment, we evaluate the computing performance of KVM and Docker in performing image resizing, which requires small amount of computing resources. The average execution time required by each Jetson TX2 to resize a UAS image with compression ratio of 48.7% is shown in Figure 5(a). The results match the ones obtained in Section IV-A1.

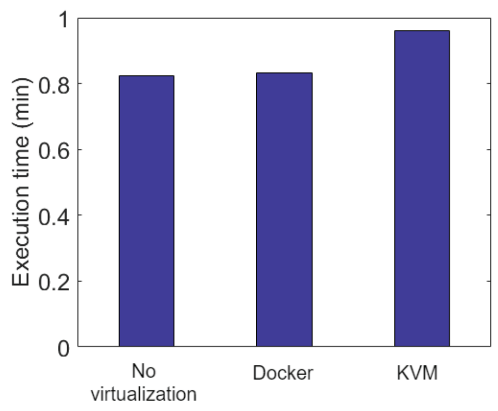
To test the performance of KVM and Docker in supporting computationally intensive UAS tasks, we choose the 3-D model reconstruction function in OpenDroneMap. This function involves a series of complicated operations including image resizing, extracting point clouds and reconstructing 3-D geographical models. The average execution time of the three microcomputers to process a UAS image is shown in Figure 5(b). The comparison of Figures 5(a) and 5(b) indicates the distinct advantage of Docker over KVM in computing complicated UAS tasks. The 3-D geographical model reconstructed from the 41 UAS images is shown in Figure 6. In the future, we will build a UAS testbed to measure the performance of KVM and Docker in facilitating UAS operations that require real-time operations such as navigation and object tracking.

B. Resource Management

Virtualization is well known for its capability in resource management. In this study, we investigate the benefits of KVM and Docker in improving resource utilization and computing performance for UAS.

Generally, in a computing platform without virtualization, applications running simultaneously can influence each other, due to resource sharing and context switches among the processes of these applications. To illustrate this fact,

(a)



(b)

Fig. 5. Execution time of processing a UAS image using the a) image resizing and b) 3-D model reconstruction functions in OpenDroneMap, which are evaluated on three Jetson TX2 of different virtualization setups.

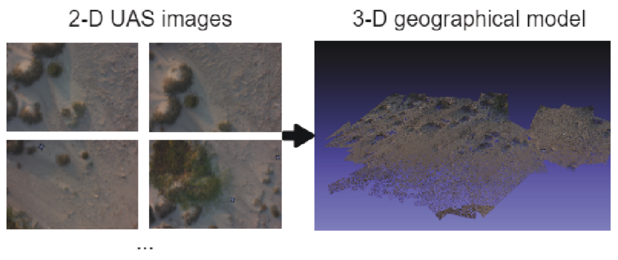
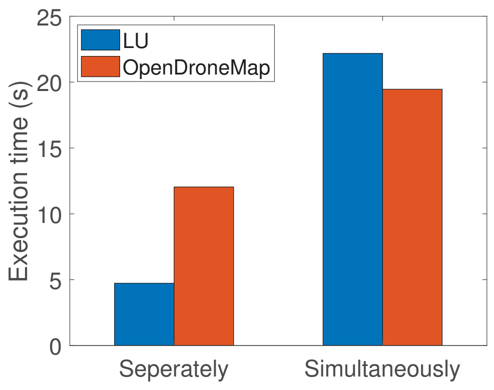
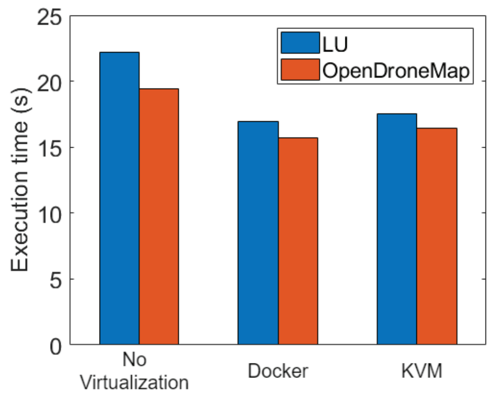


Fig. 6. The 3-D geographical model generated from the 41 UAS images using the 3-D model reconstruction function in OpenDroneMap.

we conduct a simple experiment that compares the execution times of two applications when running separately and when running simultaneously in Jetson TX2. The two applications used in this experiment are the LU application with cubic size set to $36 \times 36 \times 36$ and the image resizing function in OpenDroneMap that processes 41 UAS images. As shown in Figure 7(a), when we run two applications simultaneously, the execution time increases significantly by around 367% for the LU application and 61% for OpenDroneMap, and the LU application consumes even more time than the



(a)



(b)

Fig. 7. a) Execution time of two applications when running separately and when running simultaneously in Jetson TX2 without virtualization. b) Execution time of two applications running simultaneously in three Jetson TX2 of different virtualization setups.

OpenDroneMap.

To reduce the performance degradation observed in the previous experiment, we apply the virtualization technology to achieve resource allocation and isolation. For instance, in this experiment, we create two guests (VMs or containers) in Jetson TX2, each of which runs a different application. To achieve a better overall performance, we further assign the guest that runs OpenDroneMap, which consumes more computing resources with three CPU cores. The other guest is assigned one CPU core. As shown in Figure 7(b), running applications in different guests assigned with properly allocated resources significantly improves the overall computing performance. In addition, Docker also demonstrates better performance than KVM.

VI. Conclusion

In this paper, we studied the virtualization on UAS platforms to enable high-performance onboard computing. We first developed guidelines to select the microcomputers that are suitable to perform UAS onboard computing tasks. Two representative virtualization techniques, i.e., VM using KVM and container using Docker, in supporting UAS applications were then investigated comprehensively through experiments. The results show that Docker outperforms KVM in most performance aspects, including computing, network, and isolation of most hardware resources. However, Docker containers are less secure than VMs. The benefits of KVM and Docker in supporting real UAS applications were also demonstrated. In the future, we will build a Jetson TX2-based UAS testbed to measure the performance of virtualization in supporting real-time UAS operations. We will also investigate live migration using KVM and Docker to enable networked UAS computing.

Acknowledgement

We would like to thank National Science Foundation under Grants CI-1730589/1730675/1730570/1730325, CNS-1522458, and CAREER-1714519, for the support of this work.

References

[1] D. W. Casbeer, R. W. Beard, T. W. McLain, S.-M. Li, and R. K. Mehra, "Forest fire monitoring with multiple small uavs," in *Proceedings of the 2005 American Control Conference*. Hilton Portland Portland, Oregon: IEEE, June 2005.

[2] E. Kuiper and S. Nadjm-Tehrani, "Mobility models for uav group reconnaissance applications," in *Proceedings of the 2006 International Conference on Wireless and Mobile Communications*. Vancouver, Canada: IEEE, July 2006.

[3] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[4] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[5] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, 2015.

[6] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*. Pittsburgh, PA: ACM, April 2017.

[7] J. Chen, J. Xie, Y. Gu, S. Li, S. Fu, Y. Wan, and K. Lu, "Long-range and broadband aerial communication using directional antennas (acda): design and implementation," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 793–10 805, 2017.

[8] J. Xie, F. AI-Emrani, Y. Gu, Y. Wan, and S. Fu, "Uav-carried long-distance wi-fi communication infrastructure," in *Proceedings of the AIAA Infotech@ Aerospace*, San Diego, California, January 2016.

[9] Y. Gu, M. Zhou, S. Fu, and Y. Wan, "Airborne wifi networks through directional antennae: An experimental study," in *Proceedings of the 2015 IEEE Wireless Communications and Networking Conference (WCNC)*. New Orleans, LA: IEEE, March 2015.

[10] J. Shuja, A. Gani, K. Bilal, A. U. R. Khan, S. A. Madani, S. U. Khan, and A. Y. Zomaya, "A survey of mobile device virtualization: Taxonomy and state of the art," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1–36, 2016.

[11] E. Casalicchio and V. Perciballi, "Measuring docker performance: What a mess!!!" in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. L'Aquila, Italy: ACM, April 2017.

[12] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," vol. 41, no. 3, pp. 275–287, 2007.

[13] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative comparison of xen and kvm," in *Proceedings of Xen Summit*, Boston, MA, June 2008.

[14] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Proceedings of the 2015 IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS)*. Philadelphia, PA: IEEE, March 2015.

[15] 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 *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Belfast, UK: IEEE, February 2013.

[16] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," *Advanced Science and Technology Letters*, vol. 66, no. 2, pp. 105–111, 2014.

[17] R. Morabito, "A performance evaluation of container technologies on internet of things devices," in *Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. San Francisco, CA: IEEE, April 2016.

[18] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi, "Embedded hypervisor xvisor: A comparative analysis," in *Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Turku, Finland: IEEE, March 2015.

[19] M. Raho, A. Spyridakis, M. Paolino, and D. Raho, "Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing," in *Proceedings of the 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. Riga, Latvia: IEEE, November 2015.

[20] "Edge Computing for UAVs, UASs, and Drones," Accessed: April 30, 2018. [Online]. Available: https://www.nutanix.com/go/edge-computing-for-drones.html

[21] C. Dall and J. Nieh, "Kvm/arm: the design and implementation of the linux arm hypervisor," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 333–348, 2014.

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

[23] M. Eder, "Hypervisor-vs. container-based virtualization," in *Proceedings of the seminars "Future Internet" (FI) and "Innovative Internet Technologies and Mobile Communications" (IITM)*, Mnchen, Germany, July 2016.

[24] "Docker," Accessed: April 30, 2018. [Online]. Available: https://www.docker.com/

[25] "Kvm," Accessed: April 30, 2018. [Online]. Available: https://www.linux-kvm.org/page/Main_Page

[26] F. Bellard, "Qemu, a fast and portable dynamic translator." in *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, vol. 41, Anaheim, CA, April 2005.

[27] R. A. Vega, "Method and system for a second level address translation in a virtual machine environment," Patent US7 428 626B2, September, 2008.

[28] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[29] "Jetson TX2 module," Accessed: April 30, 2018. [Online]. Available: https://elinux.org/Jetson_TX2

[30] "UDOO X86," Accessed: April 30, 2018. [Online]. Available: https://www.udoo.org/udoo-x86/

[31] "UP Squared," Accessed: April 30, 2018. [Online]. Available: http://www.up-board.org/upsquared/specifications-up2/

[32] "LattePanda Alpha," Accessed: April 30, 2018. [Online]. Available: https://www.kickstarter.com/projects/139108638/lattepanda-alpha-soul-of-a-macbook-in-a-pocket-siz

[33] P. Varanasi and G. Heiser, "Hardware-supported virtualization on

arm," in *Proceedings of the Second Asia-Pacific Workshop on Systems*. Shanghai, China: ACM, July 2011.

[34] F. Scheffler, "Porting of μ cernvm to aarch64," Tech. Rep., 2016.

[35] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," in *Proceedings of the Conference on Design, Automation and Test in Europe*. Dresden, Germany: European Design and Automation Association, March 2010.

[36] J. Robinson, "Kernel korner: Linux as an ethernet bridge," *Linux Journal*, vol. 2005, no. 135, p. 11, 2005.

[37] M. J. Hammel, "Managing kvm deployments with virt-manager," *Linux Journal*, vol. 2011, no. 201, p. 7, 2011.

[38] "Jetpack," Accessed: April 30, 2018. [Online]. Available: https://developer.nvidia.com/embedded/jetpack-notes

[39] D. H. Bailey, *NAS Parallel Benchmarks*. Boston, MA: Springer US, 2011, pp. 1254–1259.

[40] A. Tirumala, T. Dunigan, and L. Cottrell, "Measuring end-to-end bandwidth with iperf using web100," in *Passive and Active Monitoring Workshop*, no. SLAC-PUB-9733, San Diego, CA, April 2003.

[41] V. Marmol, R. Jnagal, and T. Hockin, "Networking in containers and container clusters," in *Proceedings of netdev 0.1*, Ottawa, Canada, February 2015.

[42] Z. Wei, G. Xiaolin, H. R. Wei, and Y. Si, "Tcp ddos attack detection on the host in the kvm virtual machine environment," in *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science (ICIS)*. Shanghai, China: IEEE, May 2012.

[43] R. Mabry, J. Ardonne, J. N. Weaver, D. Lucas, and M. J. Bays, "Maritime autonomy in a box: Building a quickly-deployable autonomy solution using the docker container environment," in *Proceedings of OCEANS 2016 MTS/IEEE Monterey*. Monterey, CA: IEEE, September 2016.

[44] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*. San Diego, CA: ACM, June 2007.

[45] "OpenDroneMap," Accessed: April 30, 2018. [Online]. Available: http://opendronemap.org/

[46] "Sample Image set," Accessed: April 30, 2018. [Online]. Available: https://github.com/OpenDroneMap/odm_data_copr.git