Information-Acquisition-as-a-Service for Cyber-Physical Cloud Computing*

Silviu S. Craciunas Andreas Haas Christoph M. Kirsch Hannes Payer
Harald Röck Andreas Rottmann Ana Sokolova Rainer Trummer
Department of Computer Sciences
University of Salzburg, Austria
firstname.lastname@cs.uni-salzburg.at

Joshua Love Raja Sengupta
Center for Collaborative Control of Unmanned Vehicles
University of California, Berkeley
jlove@me.berkeley.edu, sengupta@ce.berkeley.edu

Abstract

Data center cloud computing distinguishes computational services such as database transactions and data storage from computational resources such as server farms and disk arrays. Cloud computing enables a software-as-a-service business model where clients may only pay for the service they really need and providers may fully utilize the resources they actually have. The key enabling technology for cloud computing is virtualization. Recent developments, including our own work on virtualization technology for embedded systems, show that service-oriented computing through virtualization may also have tremendous potential on mobile sensor networks where the emphasis is on information acquisition rather than computation and storage. We propose to study the notion of information-acquisition-as-a-service of mobile sensor networks, instead of server farms, for cyber-physical cloud computing. In particular, we discuss the potential capabilities and design challenges of software abstractions and systems infrastructure for performing information acquisition missions using virtualized versions of aerial vehicles deployed on a fleet of high-performance model helicopters.

1 Introduction

Imagine a fleet of autonomously flying high-performance quadrotor helicopters equipped with cameras and laser range finders gathering data for information acquisition tasks such as search-and-rescue missions [13, 16] and environmental monitoring [9]. Inspired by data center cloud computing [1], the helicopters do not directly execute any mission code but instead work as servers hosting virtual abstractions of networked autonomous vehicles that perform the actual missions. Virtual autonomous vehicles (or virtual vehicles, for short) form virtual mobile sensor networks whose nodes can be created and deployed dynamically at flight time and then migrate from one real vehicle to another in order to aggregate information as efficient and fast as possible before being terminated at the end of their mission. Virtual vehicles specify their exact resource requirements such as the required sensors and actuators as well as the necessary CPU and communication bandwidth. Virtual vehicles, once admitted to fly on a real vehicle by a virtual vehicle monitor (VVM), are temporally isolated from each other guaranteeing their resource and performance requirements. Virtual vehicles will acquire sensor data and may choose to carry it as their "payload" or stream it live to their client depending on the available communication resources.

Similar in spirit to virtual machines, virtual vehicles provide a robust, mobile, secure, and safe execution and information acquisition platform enabling what we call cyber-physical cloud computing (CPCC). Here, cloud computing becomes a metaphor for information acquisition as a service of mobile sensor networks, rather than the traditional notion of platform- or software-as-a-service. The distinction of service and platform through virtualization again enables clients to minimize usage of resources to what they really need, and providers to maximize utilization of the resources they actually have. In particular, CPCC should eventually be able to absorb hundreds of real vehicles, provided by tens of organizations with different operating procedures, and handle thousands of concurrent requests for search, surveillance, tracking, pictures, video feeds, etc.

CPCC has the potential for a number of exciting new capabilities for clients and providers of mobile sensors alike but also creates a range of interesting challenges in software, systems, and control engineering. Potential capabilities include large-scale, high-level programming and efficient load balancing of mobile sensor net-

*This work is supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design, the Austrian Science Funds P18913-N15 and V00125, the United States Office of Naval Research, and an NDSEG Fellowship through the Air Force Research Office.

Figure 1: The JAviator quadrotor helicopter [6].

works. Interesting challenges are, in software and systems engineering, virtual vehicle migration to and isolation on real vehicles as well as, in control engineering, multiplexing flight control. For example, multiple virtual vehicles on the same real vehicle may require different flight plans that need to be properly negotiated by the VVM. The problem may be simplified by having real vehicles follow flight plans that are pre-determined by the vehicle provider. In this case, virtual vehicles do not alter the flight plans of real vehicles but may still migrate from one real vehicle to another. Vehicle migration across wireless links is nevertheless challenging and may only be possible with small-footprint, light-payload virtual vehicles. Vehicle isolation is another key challenge that involves non-trivial, real-time process scheduling and management.

Next, we discuss our existing hardware infrastructure, which we plan to use as CPCC platform: a fleet of ten high-performance quadrotor helicopters called JAviators [6]. Then, in Section 3, we describe a Xen-based version of our virtual machine monitor [4], which we are currently enhancing for CPCC. Finally, in Section 4, we discuss potential CPCC capabilities and challenges in more detail.

2 Helicopter

We have designed and built, entirely from scratch, a high-performance quadrotor helicopter called JAviator (*Java Aviator*) [6] as shown in Figure 1. In particular, we have designed and built the frame and the onboard power electronics as detailed below. The motors were built for us according to our specifications. The sensors, the rotor blades, the computer board, and the battery are off-the-shelf components. The JAviator has originally served as our "flying testbed" to demonstrate flight control software written in Java with our collaborators at IBM Research. In the meantime, we have also used the JAviator to demonstrate flight control software and systems infrastructure written in C. We are currently able to fly the JAviator navigated manually but with automatic attitude and altitude control. Right now there are ten identical machines of which five are fully equipped and ready to fly. We have recently started working on automatic position control for autonomous indoor (by ultra-wide-band localization) and outdoor (by differential GPS) multi-vehicle flight.

The JAviator is, as the term "quadrotor" suggests, a 4-rotor helicopter. The advantage of a 4-rotor aircraft is that it can be built without a means for cyclic pitch control, as required in conventional single-main-rotor designs. A quadrotor is therefore mechanically simpler and less expensive to build. However, quadrotors can only be flown by computer, controlling at least its attitude and altitude, which makes them an ideal platform for demonstrating software capabilities. Compared to most other quadrotor research aircraft that are in wide use today, we aimed to go beyond the prototypical stage of usually ungainly looking contraptions and tried to provide something professional looking right from the beginning. For this purpose, we developed an aircraft that not only stands out with its high robustness and payload capacity, but also achieves a high degree of utilization and demonstrative impact. One special feature of the JAviator that makes it unique is its fully symmetrical airframe, resulting from a similar top and bottom frame. The JAviator has an overall diameter (diagonal over rotors) of 1.3 m and an empty weight of 2.2 kg, including the battery and all onboard electronics. Due to the incorporation of custom-built brushless motors, which are significantly stronger than conventional motors, the JAviator's propulsion system generates a maximum lift of 5.4 kg, which translates into a theoretical maximum payload of 3.2 kg. Without payload the average flight time is around 30 min, a maximum of 40 min was achieved during tests while hovering in ground effect.

The JAviator is currently equipped with an inertial measurement unit, one ultrasonic and two laser dis-

tance sensors, and a so-called Robostix-Gumstix stack, which serves as the onboard computer system. The Robostix contains an Atmel *ATmega128* processor clocked at 16 MHz and provides the necessary communication interfaces to connect to the many different sensors. It further provides the required PWM units for driving the four motors. The Gumstix features an Intel *XScale PXA270* CPU clocked at 600 MHz, a 128-MB RAM, and a 32-MB flash memory. The operating system on the Gumstix is a Linux system running a kernel version with real-time extensions and support of high-resolution timers, which we modified to work with the Gumstix. Attitude and altitude control is done on the Gumstix, which receives the necessary sensor data from the Robostix and then sends the computed actuator updates back to the Robostix. The JAviator ground station comprises a laptop computer, a 4-axis joystick for piloting the helicopter, and a separate logging workstation. All ground computers run Linux and are connected via WLAN to the onboard computers. All relevant material including hardware blueprints, pictures, and videos is available at `javiator.cs.uni-salzburg.at`.

The JAviator is designed to carry significant payload such as additional sensors and even non-embedded computers. We are now working on mounting at least a webcam and a small-formfactor server-grade multi-core board connected to the webcam and the onboard Gumstix onto the JAviator. The result is that the JAviator will become a flying server with two key physical capabilities that a traditional server does not have: non-trivial sensing and three-dimensional mobility. The JAviator server will be hosting our VVM as well as any virtual vehicles "flying" on the JAviator. Ideally, the JAviator server will eventually replace the Gumstix, if the monitor can provide sufficient real-time guarantees for low-level control (in the order of 100Hz rates and 1ms latencies). Attitude, altitude, and position control will then be performed by a virtual vehicle rather than the Gumstix. The Robostix is, however, even more difficult to replace since most sensors and actuators require special I/O devices not available on off-the-shelf, non-embedded hardware.

3 Virtual Vehicle Monitor

Virtualization allows multiple operating systems to share a single, physical machine through encapsulation in so-called domains created by a virtual machine monitor or hypervisor. A virtual vehicle monitor (VVM) is essentially a virtual machine monitor enhanced for CPCC. The architecture of our virtualization system is depicted in Figure 2. The VVM is based on the open-source Xen hypervisor [2]. Similar to a virtual machine monitor, the main tasks of the VVM are domain scheduling, I/O handling, and memory isolation.

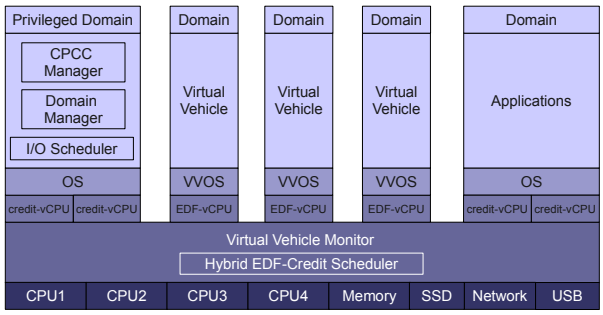

Figure 2: Virtualization architecture for CPCC.

In addition to the spatial domain isolation provided by the Xen hypervisor, the VVM provides temporal domain isolation using enhanced scheduling mechanisms as described below. The VVM runs at least one privileged domain that contains device drivers and enables other domains to perform I/O by handling their I/O requests. Moreover, the privileged domain contains domain management tools and a CPCC manager. The domain management tools provide functionality to create and destroy domains, initiate migration of domains, and modify the scheduling parameters of domains. The CPCC manager connects the VVM with other VVMs in the cyber-physical cloud. Based on the domain management tools its main job is to create and destroy virtual vehicles and manage migration between different VVMs. The VVM also supports domains running general purpose operating system instances, e.g. a Linux system that runs legacy applications such as a web server.

In the following we discuss the main elements of our virtualization system in more detail.

CPU Scheduling In the Xen hypervisor the unit of scheduling is a so-called virtual CPU (vCPU), which is an abstraction of a physical CPU. The default scheduler in Xen, called credit scheduler, is tuned for high throughput and good fairness among all active vCPUs in the system. However, the credit scheduler does not provide low latency or guaranteed CPU shares.

We have developed a hybrid EDF-credit scheduler for our VVM that can schedule a vCPU using either the standard credit scheduler (credit-vCPU) or alternatively an earliest-deadline-first scheduler (EDF-vCPU) [5]. The hybrid EDF-credit scheduler provides low latency and allows to specify a guaranteed CPU share for EDF-vCPUs while still achieving high throughput for credit-vCPUs. The implementation of the hybrid EDF-credit scheduler uses a run-queue for each physical CPU and applies work stealing and dynamic migration of vCPUs in order to balance the workload across all available physical CPU cores. Moreover, it supports dynamically switching a

vCPU from a credit-vCPU to an EDF-vCPU and vice versa. All scheduling parameters are adjustable through the domain management tools.

Virtual Vehicle Domains Virtual vehicles are hosted inside domains running a special-purpose virtual vehicle operating system (VVOS). The memory footprint of a virtual vehicle domain, including the VVOS, the application code and the data, should be small enough to achieve minimal down-time when migrating. Additionally, small domains provide fast deployment when creating and starting new virtual vehicles. Virtual vehicles run on EDF-vCPUs for temporal isolation, while legacy applications, which do not require strong temporal guarantees, run in domains that are deployed on credit-vCPUs.

We are currently working on a VVOS to bootstrap and host the run-time infrastructure for virtual vehicles. It is a single-address-space operating system that implements basic thread scheduling and device drivers for virtual I/O devices only. A virtual I/O device provides a well-defined interface that abstracts the low-level details of the underlying real I/O device, which is accessible by privileged domains only. A virtual device connects to a privileged domain that contains the device driver for the real I/O device. The privileged domain accesses the I/O device on behalf of the connected domain. Moreover, since multiple virtual vehicles can share a single device, the privileged domain performs I/O scheduling for all incoming I/O requests from various virtual vehicles.

I/O Scheduling Virtual vehicles not only require temporally deterministic execution of their program code but also temporally deterministic handling of their I/O requests. In order to support given latency and throughput requirements of a virtual device, the privileged domain applies I/O scheduling among all virtual devices currently connected to a real device. The I/O scheduler in the privileged domain uses a hierarchical token bucket traffic-shaping algorithm to regulate the packet flow from a virtual device to the real device.

Virtual Vehicle Migration Domain migration in general and virtual vehicle migration in particular is a key mechanism for CPCC. For example, it is possible for a single virtual vehicle to change its geographical location much faster than it is for a real vehicle. A virtual vehicle can simply migrate at the speed of the communication link to a real vehicle that is closer to its target location.

Currently, our VVM supports migration of running domains as provided by the Xen hypervisor [3]. Since our virtual vehicle domains have a small memory footprint and do not have much dynamically changing state the live-migration mechanism of Xen should be sufficient. However, using a low-bandwidth wireless connection between real vehicles for migration could require more sophisticated algorithms. For instance, migration overhead can be reduced by applying memory compression algorithms on the migrating domain [11].

4 CPCC Capabilities and Challenges

Traditional virtualization technology provides resource and data isolation and is therefore one of the key enabling technologies for scalable and secure cloud computing. Instead of maintaining and running their own machines, clients are only required to specify (and pay for) their exact performance requirements, which are then met by adequate virtual machines in the cloud. For clients, the advantage, besides reduced costs, is more flexibility and capabilities and, for providers, more control over system utilization and reliability. Moreover, virtualization reduces the developement and maintainance effort [12, 14].

Virtualization in CPCC, if adapted properly, may also provide similar benefits but also entirely new capabilities. For example, the notion of a virtual vehicle may enable multiple clients to share a single real vehicle but use it for different purposes. A virtual vehicle flying on a given real vehicle may provide interesting flight dynamics when migrating to other, possibly distant vehicles at the speed of their communication links. Moreover, the involved real vehicles may have entirely different flight and sensor capabilities and even be operated by different providers. In the event of an imminent real vehicle crash (as in physical crash), virtual vehicles may "evacuate" to nearby real vehicles in order to save their potentially valuable information payload.

Traditional virtualization technology typically implements machine or process models that are only deterministic in terms of functional behavior. Repeated execution of sequential programs in these models is guaranteed to compute the same output for the same input. However, other non-functional properties such as execution time and energy consumption may vary and may even be unbounded (mostly due to complex resource sharing). Virtualization technology for CPCC must therefore include solutions that provide deterministic behavior beyond computing mathematical functions. Determinism with respect to non-functional properties such as time, space, and energy is key to compositionality and thus scalability of the involved engineering methods and tools [4].

Traditional data centers utilize virtualization technology and, in particular, virtual machine migration mostly for load balancing (LAN migration [3]) but also for accommodating increasing communication demand such as high throughput and low latency requirements (WAN migration [17, 18]). With CPCC the actual number and

location of server machines becomes even more relevant since sensor location and live communication are key factors. Moreover, servers also move in space now, not just virtual machines. Already challenging problems related to resource allocation, scheduling, and management become even more difficult in this context.

Next, we discuss our plans for developing CPCC on three levels of abstraction: (1) virtual vehicles as the key CPCC infrastructure, (2) virtual information aggregators forming virtual networks of virtual vehicles that provide information-acquisition-as-a-service for CPCC, and (3) a mission language for scalable CPCC programming.

CPCC Infrastructure. A virtual vehicle is an abstraction of a real vehicle. It provides resources for computation and communication, such as a scripting engine for our mission language and a wireless link, as well as access to sensors and actuators. In order to fly a virtual vehicle, it must be instantiated by providing a program implemented in our mission language that determines its functional and timing behavior. For example, the program may specify not only what to do but also how long a particular activity is supposed to take. During flight, the vehicle maintains its program state as well as its control state such as its geographical location and speed. Interestingly, similar to virtual machines in data centers, it may not be necessary for virtual vehicles to know on which real vehicle they are currently flying, or whether it is actually sharing a real vehicle with other virtual vehicles.

The real vehicle can be controlled through the privileged domain similarly to other hardware such as CPUs and wireless cards. Again, I/O scheduling becomes necessary as several virtual vehicles may co-exist on any real vehicle. However, there are many CPCC applications, such as search-and-rescue [13, 16] or environmental monitoring [15], where it is desired to give the virtual vehicles some authority to change or modify the trajectory of a real vehicle. The correct methods and levels of abstraction at which to do this is an ongoing area of research.

One option is to allow scheduled control of individual actuators by virtual vehicles through the privileged domain. Each actuator device would have a driver that the privileged domain exposes to virtual vehicles through virtual devices. This low-level control approach becomes complicated as certain sets of actuators must be controlled consistently to retain vehicle stability. For instance, allowing virtual vehicle A to control altitude while virtual vehicle B controls attitude may result in undesired behaviors or even instability and a crash of the real vehicle.

Alternatively, a virtual vehicle could control the real vehicle through a set of high-level control behaviors provided as services by the real vehicle [10]. The privileged domain would connect the real vehicle's services to different virtual devices that represent the different control services. These high-level control services (e.g. track line, go to point) would have beneath their invocation a robust and stable low-level real-time controller that the real vehicle's operator has developed, tested, and validated. This allows the real vehicle's operator to guarantee their vehicle's operation and protect their investment from poorly written virtual vehicles, while at the same time allowing more abstract and more compact virtual vehicles to be written, which would ease both development and communication. A vehicle scheduler in the privileged domain would then schedule the access of the virtual vehicles to the real vehicle. This could be implemented such that only one virtual vehicle is in control of the entire real vehicle at any one segment of time, or it could attempt to execute a combination of non-conflicting virtual vehicle behaviors simultaneously, e.g. fly to point A and search area C, which contains point A.

CPCC Service. A network of virtual vehicles may be used for collaborative missions to gather sensor data. A virtual vehicle network (VVN) may specify the collaborating virtual vehicles and their requirements on the communication performance in the network. The specified communication performance, for instance, may limit the freedom on which real vehicles the virtual vehicles of the VVN are deployed and migrated since some of the real vehicles might be too far apart to communicate at the specified rate. Alternatively, a VVN may also specify geographical proximity directly as requirement if communication performance is not an issue, e.g. for formation flight [7].

A VVN is created by instantiating the participating virtual vehicles on real vehicles based on their performance as well as initial geographical location and flight plan requirements. A virtual vehicle network monitor (VVNM) running on a central server, similar to a name server, will be in charge of tracking real vehicles by maintaining a database of their CPCC managers' state (resources, geographical location, flight plan, and identifiers of the hosted virtual vehicles). The database is only updated by real vehicles through their CPCC manager, which report their state back to the network monitor across a possibly low-bandwidth but long-range communication link. All other communication, in particular virtual vehicle communication and migration traffic, may be done across possibly independent, high-bandwidth yet short-range links. For this purpose, non-trivial ad-hoc networking protocols and a distributed VVNM may increase performance and reliability, but may cost additional communication and computation due to the required synchronization of information.

In order to find a suitable real vehicle to migrate to, a virtual vehicle may ask the VVNM for migration targets based on available resources, location, flight plan, and other virtual vehicles (but not real vehicles). For example, a virtual vehicle may migrate because of the availability of certain sensors in a specific location, or to follow another virtual vehicle. This is where the distinction of virtual and real vehicles, that is, service and platform, pays off. Virtual vehicles work with more abstract and thus more robust notions of resources and control state while real vehicles may improve robustness independently of the virtual vehicles' behavior.

CPCC Programming. A mission language is a coordination language for describing the information acquisition behavior of mobile sensors. A mission is a distributed program written in a mission language. The execution of a mission is successful if the desired sensor data is made available as data streams with specified transmission rates in a central location such as a server. For example, a search-and-rescue mission may involve multiple mobile sensors streaming high-resolution still images along with temperature data from a range of different locations to a central server. The key challenge in the design of a mission language is to capture the creation, update, and termination of missions for mobile sensors. We have already obtained promising results with the design of a mission language called the Collaborative Sensing Language (CSL) [10]. We are developing CSL by further adapting its semantics to a service-oriented notion of virtual vehicles and networks described above. Here, the emphasis is on identifying the correct abstractions for dealing with concurrent and changing sets of real and virtual vehicles and networks. CSL, in its existing form, is based on the notion of tasks and the assignment of tasks to vehicles [8]. Virtual vehicles are essentially a richer, fully programmable and service-oriented form of tasks. They will allow 'arbitrary tasks' to be generated from the provided services of the real vehicles. An extension of CSL will incorporate the enriched semantics and offer information-acquisition-as-a-service rather than the existing platform-oriented view.

5 Summary

We proposed information-acquisition-as-a-service of mobile sensor networks for cyber-physical cloud computing (CPCC), and presented a fleet of high-performance model helicopters as possible target platform and virtualization technology enhanced for CPCC. We also discussed potential capabilities and design challenges of software abstractions and systems infrastructure for CPCC information acquisition missions.

References

[1] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the clouds: A Berkeley view of cloud computing. Tech. rep., EECS Department, University of California, Berkeley, Feb 2009.

[2] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proc. SOSP* (2003), ACM.

[3] CLARK, C., FRASER, K., HAND, S., HANSEN, J., JUL, E., LIMPACH, C., PRATTN, I., AND WARFIELD, A. Live migration of virtual machines. In *Proc. NSDI* (2005), USENIX.

[4] CRACIUNAS, S., KIRSCH, C., PAYER, H., RÖCK, H., AND SOKOLOVA, A. Programmable temporal isolation in real-time and embedded execution environments. In *Proc. IIES* (2009), ACM.

[5] CRACIUNAS, S., KIRSCH, C., PAYER, H., RÖCK, H., AND SOKOLOVA, A. Programmable temporal isolation through variable-bandwidth servers. In *Proc. SIES* (2009), IEEE.

[6] CRACIUNAS, S., KIRSCH, C., RÖCK, H., AND TRUMMER, R. The JAviator: A high-payload quadrotor UAV with high-level programming capabilities. In *Proc. GNC* (2008), AIAA.

[7] GIRARD, A., AND HEDRICK, K. Formation control of multiple vehicles using dynamic surface control and hybrid systems. *International Journal of Control 76*, 9 (2003), 913–923.

[8] GODWIN, M., SPRY, S., AND HEDRICK, J. Distributed collaboration with limited communication using mission state estimates. In *Proc. ACC* (2006), IEEE.

[9] HART, J., AND MARTINEZ, K. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews 78* (2006), 177–191.

[10] HEDRICK, K., JARIYASUNANT, J., KIRSCH, C., LOVE, J., PEREIRA, E., SENGUPTA, R., AND ZENNARO, M. CSL: A language to specify and re-specify mobile sensor network behaviors. In *Proc. RTAS* (2009), IEEE.

[11] JIN, H., DENG, L., WU, S., SHI, X., AND PAN, X. Live virtual machine migration with adaptive, memory compression. In *Proc. CLUSTER* (2009), IEEE.

[12] LEVIS, P., AND CULLER, D. Maté: a tiny virtual machine for sensor networks. In *Proc. ASPLOS* (2002), ACM.

[13] MCGEE, T., AND HEDRICK, K. Guaranteed strategies to search for mobile evaders in the plane. In *Proc. ACC* (2006), IEEE.

[14] MÜLLER, R., ALONSO, G., AND KOSSMANN, D. A virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev. 41*, 3 (2007), 145–158.

[15] RATHINAM, S., DE ALMEIDA, P. P., KIM, Z., JACKSON, S., TINKA, A., GROSSMAN, W., AND SENGUPTA, R. Autonomous searching and tracking of a river using an UAV. In *Proc. ACC* (2007), IEEE.

[16] RYAN, A., AND HEDRICK, J. A mode-switching path planner for UAV-assisted search and rescue. In *Proc. CDC-ECC* (2005), IEEE.

[17] TRAVOSTINO, F., DASPIT, P., GOMMANS, L., JOG, C., DE LAAT, C., MAMBRETTI, J., MONGA, I., VAN OUDENAARDE, B., RAGHUNATH, S., AND WANG, P. Seamless live migration of virtual machines over the MAN/WAN. *Future Generation Computer Systems 22*, 8 (2006), 901–907.

[18] WOOD, T., RAMAKRISHNAN, K., VAN DER MERWE, J., AND SHENOY, P. CloudNet: A platform for optimized WAN migration of virtual machines. Tech. Rep. TR-2010-002, University of Massachusetts, Jan 2010.