From Nested Virtualization Architecture To 2D Security Management for Distributed Clouds

Alex Palesandro
Orange Labs, France
alex.palesandro@gmail.com

Aurélien Wailly
Orange Labs, France
aurelien.wailly@orange.com

Marc Lacoste
Orange Labs, France
marc.lacoste@orange.com

ABSTRACT

Distributed cloud computing is now moving user-centric. The promise of this evolution is: (1) unified resource management across IaaS providers, e.g., live migration independent from hypervisor technology; (2) increased customizability to choose the hypervisor services needed to realize fully à la carte, self-service clouds. However, a reality check shows that major interoperability and security barriers must be lifted, and complex trade-offs be found between functionalities provided and security assurance. Nested virtualization (NV) brings both a homogeneous interface for distributed hypervisor features and enhanced IaaS security, protecting VMs even in case of hypervisor compromise. However, practical NV adoption is still hindered by code additions to an already very large hypervisor TCB and by limited hardware support. New distributed IaaS architectures are therefore required.

In this paper, we argue that micro-hypervisor (MH) architectures are the key to overcome such NV limitations. From the previous challenges, we derive a set of design principles that a distributed IaaS architecture should satisfy. We compare possible architectural designs combining NV with micro-hypervisor features to find the best trade-off between user data privacy, modularity, TCB size, and compatibility with multiple platforms. We sketch how the resulting architecture may serve as a basis for 2D security management of a cloud of clouds: vertically, across layers for hardened IaaS security, and horizontally spanning cloud domains for end-to-end security.

1. INTRODUCTION

Distributed cloud computing (DCC) is now turning into a reality: modern hybrid cloud platforms enable to benefit seamlessly both from private cloud security and from public cloud flexibility. DCC is also moving user-centric: customers demand greater empowerment to deploy self-service clouds really adapted to their needs, and to move away from a provider-dominated landscape of highly heterogeneous closed platforms and its infamous vendor lock-in [3].

However, current *distributed cloud infrastructures (DCI)* suffer from two sets of limitations. These may be broadly classified as *horizontal* and *vertical* challenges. Horizontally, DCIs lack interoperability, mainly due to heterogeneity of IaaS services, hypervisor-specific and not compatible across providers. Control is also severely limited by monolithic infrastructures preventing fine-grained cloud customization by users. Vertically, security remains the last hurdle toward wide adoption of DCIs. Due to their complexity, infrastructure layers remain extremely vulnerable to attacks, making it difficult to achieve strong isolation and integrated protection.

Advances in *nested virtualization (NV)* were a major step forward to tackle such challenges [1]. By virtualizing the hypervisor through a thin software layer, NV brings enhanced isolation capabilities on the same physical machine, protecting VMs even in case of hypervisor compromise [26]. NV also provides a distributed homogeneous interface for commodity hypervisors [22, 23]. Practical NV adoption is still hindered by code additions to an already large hypervisor TCB (Trusted Computing Base) and by limited hardware support. There is also currently no NV architecture solving both vertical and horizontal challenges simultaneously. New distributed IaaS architectures are therefore required.

In this paper, we propose a new architectural approach for DCIs. We argue that advances in OS design bring significant improvements to traditional NV architectures: instead of relying on general-purpose hypervisors (GPHs) for both virtualization layers, new kernel designs allow to tackle more efficiently different sets of challenges separately in each layer. We make a position statement that *micro-hypervisor (MH) architectures* combined with NV benefits are a promising design for the next generation of DCIs.

This paper makes the following contributions. From previous challenges, we derive a set of design principles that a DCI architecture should satisfy. We compare possible architectural designs combining NV with MH features to find the best trade-off between user data privacy, modularity, TCB size, and compatibility with multiple platforms. We finally sketch how the resulting architecture may serve as basis for 2D security management of DCIs: vertically, across layers for hardened IaaS security, and horizontally spanning cloud domains for end-to-end security.

This paper is organized as follows. Section 2 presents the DCI challenges and design principles. Sections 3 and 4 review the NV ecosystem and analyze the limitations of traditional NV design. Section 5 presents the MH approach. Section 6 compares possible DCI architectures combining NV and MH according to the design principles. Finally, Section 7 presents a preliminary DCI system architecture based on the best resulting design with promising properties for 2D management of DCI security.

2. CHALLENGES AND DESIGN PRINCIPLES

2.1 Challenges

Deploying cloud applications able to transparently use resources of several heterogeneous platforms is still difficult. Multi-platform coordination is hard, primarily due to lock-in to different standards. Unfortunately, such lock-in is not only due to market strategic decisions, but also reflects heterogeneity of virtualization technologies. For example, each cloud provider may use different storage systems or formats for VM images. Different hypervisors have today a low degree of compatibility between them. This generally hin-

ders using some popular features like VM live migration across different platforms. Incompatibility is even greater considering each platform may adopt several types of virtualization techniques. This situation induces a tight dependency between the customer and the cloud provider. It may prevent guaranteeing service availability by relying on alternative providers, or setting up geographical optimizations according to cost or origin of traffic. This resulted in some dramatic data server failures in recent years, causing prolonged outage of service for many applications. We refer to this class of multi-platform issues as *horizontal challenges*.

Several concerns were also raised about user data privacy and platform integrity. For users, preserving the privacy of their data stored on the cloud platform is important. However, due to overprivileged hypervisors, a curious or malicious administrator may easily retrieve such data [26]. In a multi-tenant cloud, physical resource sharing between users may lead not only to VM data leakage but also to attacks against the IaaS [15, 17]. For the provider, a primary objective is thus to guarantee platform integrity. Unfortunately, commodity GPHs are not well adapted to observe deeply all layers of a running IaaS platform: running a GPH at the highest privilege level hampers detection and reaction on the hypervisor itself in presence of faulty or malicious behavior. Thus, both users and providers have a problem with overprivileged hypervisors. We refer to this class of layer-oriented issues as *vertical challenges*.

2.2 Design Principles

We identified the following key DCI design principles to meet the previous challenges:

DP1 User Data Privacy An administrator (or another platform user) should not be authorized to inspect and analyze user data and VM instances without explicit user consent. A trade-off should be found to allow providers to detect and react to potential threats, but also to prevent unlimited platform and user data access by curious or malicious operators. Thus, a platform administrator should not manage the infrastructure directly using the highest privilege level. Instead, his actions on user data objects should be validated by a mutually-trusted kernel run in the most privileged mode. This kernel should also enforce memory page encryption to prevent snooping when pages are swapped out.

DP2 Fail-Safe Modular Architecture The architecture should be fail-safe, e.g., for automatically migration of VMs or user-data during an attack, or to recover from software failures. The number of safety-critical components should minimal to reach a low attack surface. The overall design should also be highly modular to enable platform reconfiguration.

DP3 Small TCB The number of vulnerabilities and failures in the platform is directly linked with the code size run at the highest privilege level. A small TCB architecture will improve safety and integrity by design.

DP4 Inter-Platform Support The cloud architecture should support different platforms and providers. The user VMs should run (and potentially be migrated) unmodified over different IaaS platforms.

DP5 Compatibility with Legacy Evolution in the IaaS architecture should avoiding disrupting existing third party applications. Platform interfaces should be kept as close as possible to existing ones to prevent code rewriting notably for applications.

In addition to the previous functional principles, another set of principles should be considered regarding effective feasibility of implementing the DCI architecture. For instance in terms of platform usability:

DP* Minimal Additional Code Despite not being a strictly a technical limitation, the amount of code to be added and maintained to an existing architecture to meet functional requirements could become a show-stopper to its practical use.

In our analysis, we will consider not only the overall suitability of the architecture, but also the amount of effort needed to develop and deploy it. Table 1 summarizes the previous principles and the class of challenges they address.

Table 1: DCI Design Principles.

Id	Design Principle	Class of Challenges
DP1	User Data Privacy	Vertical
DP2	Fail-Safe Modular Architecture	Vertical
DP3	Small TCB	Vertical
DP4	Inter-Platform Support	Horizontal
DP5	Compatibility with Legacy	Horizontal
DP*	Minimal Additional Code	–

3. RELATED WORK

A growing body of research has investigated building architectures and infrastructures integrating a subset of those design principles leveraging NV [1, 7, 12, 22, 26]. The distributed cloud feature set may be expanded using two levels of virtualization instead of a single one. This design allows to specialize hypervisor functions per layer, each layer addressing a subset of features, simplifying overall development compared to a GPH. NV development on x86 architecture was initially hindered by severe performance issues [9]. However, several works have shown feasibility of realizing low-overhead NV using dedicated implementation architectures [1], or to recover performance loss exploiting multi-level consolidation [22]. NV slowdown can now be considered an acceptable price for advanced features – hardware manufacturers are also developing new hardware primitives to reduce this overhead [13].

However, so far no architecture has addressed both vertical and horizontal set of challenges simultaneously. XenBlanket [22] does not investigate cloud architectural security concerns (DP2, DP3), limiting its focus to providing new features on a coordinated multi-provider cloud. Cloudvisor [26] proposed a NV-based security architecture, but cannot run more than one nested hypervisor at a time, which is a critical requirement for a multi-provider cloud platform (DP4). In the super-cloud vision [23], each user runs its own L1 hypervisor instance to realize a blanket layer overcoming incompatibilities between different providers. However, security issues have not been addressed.

Other approaches based on a single layer of virtualization do not fulfill all principles either. Self-Service Clouds (SSC) [2] introduces a complete security architecture to overcome major security and privacy issues of currently deployed cloud infrastructures. The aim is to reconcile user-centric and provider-centric requirements of the cloud. Despite being similar to the super-cloud approach (DP4), SSC introduces new components that have to be directly managed by the user. This change of interface with user VMs prevents moving straightforwardly to the new platform without having to change its applications (DP5).

To reduce the TCB size, DeHype [24] presented a dissected version of KVM, moving a great part of the code to user space. DeHype satisfies all vertical design principles, re-interpreting a micro-

kernel design in a KVM setting. It increases robustness and security of a cloud platform leveraging KVM (DP1, DP2, DP3), but does not address horizontal challenges for a multi-provider service.

4. CURRENT NESTED VIRTUALIZATION

4.1 Approach

Nested virtualization (NV) refers to a system architecture with two layers of virtualization: the guest OS virtualizes a nested guest. The concept may be generalized to an arbitrary number of nested guest layers, leading to recursive virtualization [9, 16]. (Full) virtualization is able to execute tasks in a deprivileged context that normally requires a privileged execution. With NV, this approach may be extended to virtualization itself: a guest hypervisor may be run with nested guest VMs without requiring any change to any of them.

Two layers should be distinguished when migrating to an NV architecture: (1) the hypervisor running above the hardware (*L0 hypervisor*); and (2) the nested hypervisor (*L1 hypervisor*). Nested guests are generally referred to as *L2 guests*. Unfortunately, virtualization comes with a price due to the extra layer of indirection introduced: the computational overhead is generally acceptable when dealing with single-layer virtualization, but grows exponentially when adding further levels [9]. This performance overhead prevented NV adoption during many years until was finally proposed an architecture tending towards more reasonable overheads [1].

NV feasibility opened plenty of new possibilities and avenues for research that have been largely discussed in the literature. Such works all tend to diversify hypervisor functionalities, introducing new features but also keeping existing ones. Each layer clearly addresses different sets of issues:

- L0 deals with vertical aspects of the platform. It aims to guarantee the best performance and strongest level of isolation. L0 represents the last line of defense of the platform and a privileged point for monitoring of its global status.

- L1 deals with the horizontal feature set. It should provide the widest possible support for different platforms and the ability to virtualize using different techniques e.g., para-virtualization, hardware-assisted virtualization, dynamic binary translation.

4.2 Limitations

Today, NV presents several limitations related to hardware support and TCB size.

First, to support NV, the hypervisor has to provide an exact copy of hardware virtualization primitives. Full virtualization with hardware assistance (HAV) is the most popular technique, but requires quite a complex dedicated support. HAV primitives assist instruction emulation, memory address translation, and management of devices. Both Xen and KVM propose nested VMX/SVM features that allow to efficiently handle nested hardware virtualization on single-layer hardware primitives. Full virtualization is currently the only virtualization technique supported for L1 virtualization due to its ability to run unmodified guest OSes – and thus hypervisors. Paravirtualization would on the contrary require an ad hoc modified hypervisor to act as L1 (thus violating DP*). Despite still poor NV performance and not being yet stable on modern hypervisors [25], such feature is under heavy development and will hopefully be in a usable state soon.

Second, due to current monolithic GPH architectures, each feature directly introduced within the hypervisor such as security enhancements enlarges the TCB [26]. Therefore, a huge amount of code runs in highly privileged mode and must be trusted. Several works discuss and analyze the overall TCB of modern GPHs, always bigger than 100 KLoC [18, 20]. This approach directly violates the DP3 principle. Moreover, the previous architecture does not improve overall stability of the infrastructure. The L0 hypervisor faces the same issues as single-layer GPH architectures: a large TCB implies also that a failure experienced in one of its critical components may trigger a generalized failure of the whole platform (violating the DP2 principle). Besides, in standard GPH architectures, the administrator has the entire control of the platform and could potentially snoop and inspect user VM status and data (violating the DP1 principle). Such designs notably do not take advantage of the possibility to use more modular architectures with only a tiny security-critical kernel.

Overall, current NV architectures lack some of the dedicated features found in modern hypervisors and the impossibility to rely on a tiny TCB (violating principles DP4 and DP3). If the first one is currently being addressed by hypervisor developers, the second simply has not be achieved with modern hypervisors so far.

5. MICRO-HYPERVISORS

5.1 Approach

To solve the TCB size issue, the idea of *micro-hypervisor (MH)* architecture was introduced [18, 19, 20]. Such hypervisor design draws inspiration from evolution in OS architecture (e.g., extensible, micro-, exo-, component-based kernels). Within the hypervisor are distinguished security-critical modules from non-sensitive code. The aim is to expel as much code as possible from the TCB make the hypervisor ultra-thin. Some hypervisor components (e.g., device drivers, VM address space management) may be virtualized outside the MH, isolated, and restarted in case of compromise [18]. The utmost MH removes the virtualization layer altogether [19].

MH architectures may be seen as convergence of traditional hypervisor and micro-kernels (MK). From the MK design principles legacy, the MH is guided by TCB minimization. Thus, it features a minimal kernel running in privileged mode, offering only a few fundamental services (e.g, scheduling, IPCs, memory management, VMX extensions) to other modules running in less privileged domains. All other components, such as device drivers, are moved out in user-space. The main benefit of such increased modularity is to easily define groups of user-space services, each group sharing a common a set of privileges. Trust relationships between groups may be arbitrarily defined, mirroring those between sets of principals or applications sharing the system. This allows MK to implement different "personalities": device drivers being run in user mode, such fine-grained control allows great flexibility to implement device sharing patterns. While the interest of realizing such "personalities" in traditional MKs somewhat lessened as system services, drivers, and even applications had to be completely re-designed (hypervisors offering better primitives for abstraction with the notion of VM), such flexibility still remains a major benefit of coupling micro-kernel design with virtualization [8].

MHs are particularly well adapted to sastisfy vertical design principles. For instance, MH design increases reliability as services crashing in user space may be restarted without bringing down the entire system (DP2). The attack surface is also lowered by the small TCB size, the minimal kernel only containing security-critical components and forming a last line of defense (DP3). MH might also address some horizontal principles. "Personalities" are concretely a set of processes handling specific functions running in user-space. Implementing such features outside the kernel makes it easier for each user to rely on different instances of such personalities running concurrently, possibly in a distributed manner (DP4).

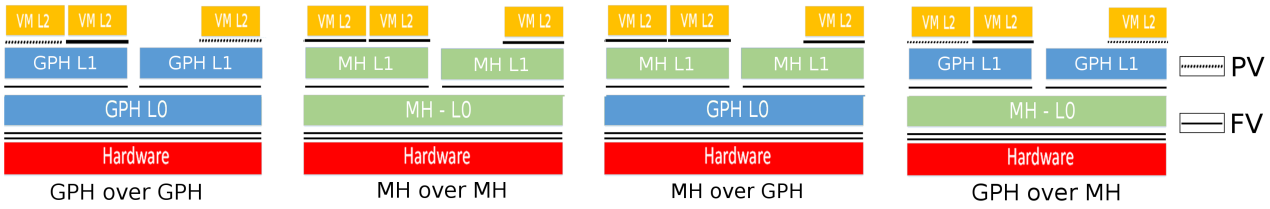

Figure 1: Different Types of Nested Architectures.

5.2 Limitations

The MH approach also showed several limitations. First, support for hardware is lacking, notably in terms of device drivers. Fortunately, this issue may be addressed leveraging I/O passthrough techniques, despite some resulting exclusive allocation challenges. Second, several cloud issues are simply not addressable with GPHs. However, GPHs are now widely adopted and foundation of existing cloud platforms. While a GPH exports mature APIs to upper software layers and is also tightly integrated with cloud toolkits, MHs lack such API stability and integration, requiring explicit dedicated code to be written – thus violating DP5. Third, for the sake of kernel size, MHs could lack several virtualization features commonly available on modern GPH hypervisors. For instance, XMHF does not support multiple guests. Such limitations applicable for MH architectures alone may be lifted partially or totally when applying MH design to NV architectures.

6. ARCHITECTURE COMPARISON

Adding an extra layer of virtualization enables to introduce a wide range of new features in the IaaS infrastructure, but does not reduce the TCB size. Introducing MH design within a nested IaaS architecture could represent a new approach to improve NV benefits by overcoming such limitations. In what follows, we analyze the four possible architectures shown in Figure 1 to achieve nesting of GPH and MH, highlighting their potential, and assessing their benefits and weaknesses w.r.t. to the design principles of Section 2, as well as their cost of implementation.

6.1 GPH/GPH

The first architecture only features GPHs in both layers to serve as point of comparison. This type of architecture enables several remarkable features related to VM management flexibility such as live migration over different platforms, and transparent migration of virtualized clusters across data centers, fulfilling principle DP4. More generally, assuming complete and correct support of virtualization primitives for nested hardware in mainstream hypervisors, this architecture satisfies all horizontal design principles.

However, it fails to safisfy vertical design principles as TCB size is that of currently deployed NV architectures. The interest of other MH-based architectures will thus be to introduce isolation and privacy improvements by reducing the TCB, while still keeping horizontal features.

6.2 MH/MH

This architecture is considered only to highlight the improvements of a MH design compared to a GPH/GPH architecture. An MH/MH implementation would meet the vertical requirements missing in the previous approach. As in single-layer MH architectures, hypervisor disaggregation and minimization overcomes issues posed by monolithic system architectures and over-privileged administrative VMs (DP1). The smaller resulting TCB also improves resilience of the platform against a large set of software failures (DP2) and reduces the surface of attack of the most privileged part of the IaaS infrastructure (DP3).

However, the interest this class of architecture remains limited as it cannot overcome any of the generally cited issues of single-layer MHs such as compatibility with legacy hypervisor architectures (DP5). While several horizontal properties are theoretically achievable, they would require a large development effort to implement the relevant features such as interoperable advanced resource management. Moreover, the lack of nested VMX support prevents testing in practice this type of architecture, as MH usually rely on VMX primitives to create nested HVM guests.

Overall, with this architecture, if vertical challenges are well addressed as in single-layer MHs, it does not provide any improvement for horizontal challenges, despite NV being used to bridge gaps between IaaS platform eco-systems.

6.3 MH/GPH

The third architecture aims to address security concerns. It combines a GPH as L0 and a MH as L1. This approach is interesting as it may be easily deployed without requiring any extra code addition to the GPH. Hardware support is not a real barrier as existing GPH-based platforms could potentially export nested VMX features to guests. Even if this is not yet the case in modern cloud platforms, this lack may be considered temporary due to the experimental status of such features.

However, there is no TCB reduction as the code size running in L0 kernel space is as big as before. Besides, adopting a MH as L1 prevents using inter-platform enhanced features. As already seen for MH design, the price of having a thin hypervisor is normally paid with a lack of certain features. Thus, this architecture cannot provide any new horizontal features leveraging NV. Therefore, this approach can only provide the same features as those deployed today, with an increased but not sufficient level of isolation.

6.4 GPH/MH

This last approach achieves a smaller TCB than other architectures, due increased modularity brought by the micro-kernel-like design of the L0 layer – satisfying DP3. Leveraging a GPH like Xen or KVM for the L1 layer provides the user with an environment seeming identical to those used in production today – satisfying DP5. The L1 GPH is normally integrated with modern cloud toolkits like OpenStack, and will thus support third-party applications – satisfying DP4. Therefore, this architecture achieves a stronger level of isolation due to vertical use of NV to introduce an additional hardware-protected security monitor. Thanks to the GPH, the MH layer remains transparent to the user and should not impact the functional properties of the IaaS platform.

This class of architecture may suffer from two types of limitations. First, nested VMX extensions must be exposed to the L1

Table 2: Architecture Comparison.

Id	Functional Design Principle	Class of Requirement	GPH/GPH	MH/MH	MH/GPH	GPH/MH
DP1	User-Data Privacy	Vertical	×	✓	×/✓	✓
DP2	Fail-Safe modular architecture	Vertical	×	✓	×	✓
DP3	Small TCB	Vertical	×	✓	×	✓
DP4	Inter-Platform support	Horizontal	✓	×	×	✓
DP5	Compatibility with Legacy	Horizontal	✓	×	×	✓
DP*	Minimal Additional Code	–	✓	×	✓	✓

hypervisor to let him "nest" another HVM guest. This requirement is implemented by GPHs like Xen and KVM, but usually not included in the key features to be provided by a MH: implementing nested VMX may be estimated to around 1KLoC, to be compared with the size of a typical MH (2-20 KLoC) [20]. More generally, the MH quest towards ever increasing minimalism could represent a serious limitation to preserving functions of existing platforms: the constraint of privileged code size may force MH developpers to drop some features considered as non-essential. This may apply to NV-related primitives, but also to other types of features. For instance, XMHF may only virtualize a single L1 instance at a time. While this may be acceptable if the platform objective to enforce strict resource access control in a single-provider private cloud, it hinders having a platform model enabling the user to run its own hypervisor as L1 to set up a self-service cloud independently from the underlying provider – as in that case, multiple L1 hypervisors would need to run concurrently on the L0.

Second, device drivers should be correctly managed. Drivers are a major MH issue, due to costs of development or of adaptation to a new architecture. Such challenges may be addressed using device assignment: the L1 GPH hypervisor manages the physical device with the proper driver now running in a lower level of privilege. To be workable, this approach would require exclusive allocation to be simply implementable, e.g., using SR-IOV [6].

6.5 Discussion

Overall, how do such architectures compare to the two main reported benefits of NV, i.e., vertical, through an increased level of isolation between hypervisor and customer VMs, and horizontal to easily achieve multiple provider platform interoperability at system level? Table 2 summarizes the previous analysis.

The GPH/MH architecture appears as the most interesting to expand the NV promised benefits. Through a solid and secure MH as L0, it is the only one able to reduce the platform TCB. This would bring several benefits towards a more secure infrastructure. First, a guarantee that the platform administrator could not lose control of the system thanks to an extra security layer. Second, certification perspectives: Klein et al. [10] showed that less than 10 KLoC kernels could be formally verified, under several conditions, which is far from be achievable today with standard GPH/GPH approach where the TCB is an order of magnitude bigger than MHs [18].

However, the effort to set up a GPH/MH is strongly dependent on the technique for L2 virtualization. As already seen, to support L2 guests with hardware-assisted virtualization which is becoming mainstream, the MH needs to export nested VMX primitives. On the other end, adopting paravirtualization or dynamic binary translation for L2 does not requires particular hardware support, and thus would not require MH patching to set up a running platform. However, paravirtualization supports only a subset of system features, which could be an issue as general solution.

Therefore, a MH-based architecture could increase the feature set of current NV architectures, mostly vertically. The main remaining barrier is to come up with an MH implementation supporting HAV nested guests. We are working on a GPH/MH implementation aiming to lift this limitation using paravirtualization.

7. TOWARDS 2D SECURITY MANAGEMENT

We now sketch how the previous enhanced-NV design enables to build a DCI security architecture meeting both vertical and horizontal requirements. Such DCIs will face 2D threats, vertical spanning layers, and horizontal, spanning domains – DCIs being both multi-layer and multi-domain. Vertically, attacks currently target several infrastructure layers (e.g., VMs, hypervisor). Unfortunately, most existing defenses are generally for single layers only. This makes it difficult to grasp the overall extent of an attack. Horizontally, threats may propagate through the network from cloud to cloud. Unfortunately, most existing defenses are for single clouds only. Lack of interoperability in security policies and mechanisms is thus a major barrier to unified hybrid cloud security. Overall, DCI security requires 2D protection.

We consider the 3-layered DCI architecture shown in Figure 2. The bottom and top layers are similar to a single-cloud IaaS.

- At the bottom is the *provider-centric* view of the DCI: each cloud provider contributing to the DCI is in control of compute, storage, and network resource provisioning.
- At the top is the *user-centric* view of the DCI: customer VMs are arranged in a set of *self-service clouds* (SSCs) where security and other services may be fully customized by users, independently from underlying providers.
- The middle layer is a distributed resource abstraction and security management plane we call the *security super-cloud*.

The super-cloud has two main functions. Its northbound interface federates cloud resources in a provider-independent manner and enables to customize their security to realize the SSCs. While many orchestration architectures have been proposed at service or middleware levels (e.g., multi-Iaas platforms), our architecture aims to achieve federation at system-level to fully take advantage of underlying provider platform openness (if available) for finer-grained control and increased flexibility.

The super-cloud southbound interface is in charge of 2D DCI security management. It provides a unified vision of security across layers of the DCI, guarantees a homogeneous level of security across clouds independently from underlying provider mechanisms, and integrates both dimensions smoothly. Despite many defense mechanisms being available in this 2D landscape, they remain difficult to configure and to reconcile, the current "by hand" approach for supervision being a nightmare for security administrators. To overcome such complexity, we consider here a security monitoring infrastructure providing 2D detection and reaction to threats with a high degree of automation.

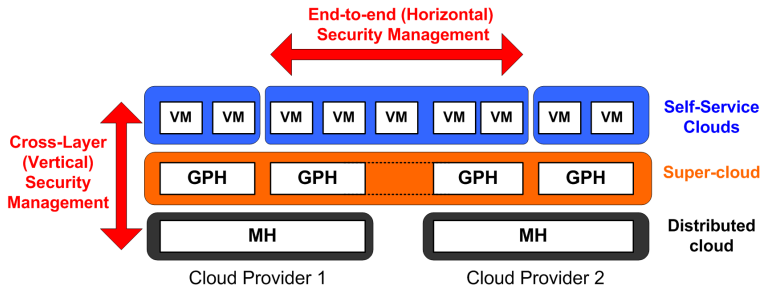

Figure 2: DCI System Architecture.

In terms of system architecture, we consider a NV-based design to realize the super-cloud above the DCI as in [23]. We assume the provider-layer to be MH-based. This design choice provides a solid foundation to meet all vertical features, notably smaller TCB size and enhanced DCI security such as guaranteeing VM security even if intermediate layers are compromised. Note that this assumption does not seem unreasonable for the coming years as: (1) despite most current IaaS platforms still being GPH-based, there is a strong trend towards making the hypervisor more minimal and flexible, as witnessed by disaggregation of hypervisors of the mainstream IaaS platforms [4, 14]; (2) MHs themselves are becoming component-based, with possibility of interoperability to avoid any further risk of IaaS lock-in as shown by first multi-MH OS frameworks [20].

The super-cloud layer is GPH-based, both to be able to realize the SSCs by defining blanket layers and to guarantee security interoperability and other horizontal features.

To implement 2D security management proper, we intend to rely on self-protection frameworks which have been defined for both cross-layer [21] and cross-domain [11] IaaS and multi-IaaS integrated security monitoring. We are currently investigating how they may be integrated within the previous system architecture, notably to guarantee security SLAs [5].

8. CONCLUSION AND FUTURE WORK

This paper reviewed the main DCI design challenges and how they could be addressed by combining NV with different hypervisor architectures. We also presented a preliminary system architecture combining GPHs and MHs for 2D security management of DCIs.

Ongoing work focuses on refining this architecture to add self-protection features based on the VESPA framework. The objective is to determine where to place detection, decision, and reaction components for the different autonomic security loops, both within layers and across clouds. We are also implementing a proof-of-concept prototype of this security architecture using Xen as L1 GPH and NOVA as L0 MH. Xen seemed the most suitable hypervisor for L1, notably due to its support of different virtualization techniques and ability to run without HAV support. NOVA appears promising for L0 due to its clean design and code availability. With this platform, we are currently focusing on implementing vertical properties (e.g., cross-layer self-protection) in a single cloud setting, with extension to horizontal properties (e.g., inter-cloud self-protection) in a distributed cloud setting.

9. REFERENCES

[1] M. Ben-Yehuda, M. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The Turtles Project: Design and Implementation of Nested Virtualization. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[2] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-Service Cloud Computing. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[3] Cloud Security Alliance. The Notorious Nine Cloud Computing Top Threats in 2013, February 2013.

[4] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[5] CUMULUS Project. http://cumulus-project.eu/.

[6] Y. Dong, Z. Yu, and G. Rose. SR-IOV Networking in Xen: Architecture, Design and Implementation. In *USENIX Workshop on I/O Virtualization (WIOV)*, 2008.

[7] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus. HVX: Virtualizing the Cloud. *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[8] G. Heiser and B. Leslie. The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors. In *ACM SIGCOMM Asia-Pacific Workshop on Systems (APSys)*, 2010.

[9] B. Kauer, P. Verissimo, and A. Bessani. Recursive Virtual Machines for Advanced Security Mechanisms. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) Workshops*, 2011.

[10] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[11] M. Lacoste, A. Wailly, A. Tabourin, L. Habermacher, X. Le Guillou, and J.-P. Wary. Flying Over Mobile Clouds with Security Planes: Select your Class of SLA for End-to-End Security. In *International Conference on Cloud and Utility Computing (UCC)*, 2013.

[12] C. Liu and Y. Mao. Inception: Towards a Nested Cloud Architecture. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[13] J. Nakajima. Making Nested Virtualization Real by Using Hardware Virtualization Features, 2013.

[14] Nguyen, Anh and Raj, Himanshu and Rayanchu, Shravan and Saroiu, Stefan and Wolman, Alec. Delusional Boot: Securing Hypervisors Without Massive Re-engineering. In *7th ACM European Conference on Computer Systems (EUROSYS)*, 2012.

[15] D. Perez-Botero, J. Szefer, and R. B. Lee. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *International Workshop on Security in Cloud Computing*, 2013.

[16] G. J. Popek and R. P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[18] U. Steinberg and B. Kauer. NOVA: A Microhypervisor Based Secure Virtualization Architecture. In *ACM European Conference on Computer Systems (EUROSYS)*, 2010.

[19] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[20] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, and A. Datta. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. *IEEE Symposium on Security and Privacy*, 2013.

[21] A. Wailly, M. Lacoste, and H. Debar. VESPA: Multi-Layered Self-Protection for Cloud Resources. In *International Conference on Autonomic Computing (ICAC)*, 2012.

[22] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize Once, Run Everywhere. In *ACM European Conference on Computer Systems (EUROSYS)*, 2012.

[23] D. Williams, H. Jamjoom, and H. Weatherspoon. Plug into the Supercloud. *IEEE Internet Computing*, 17(2):28–34, 2013.

[24] C. Wu, Z. Wang, and X. Jiang. Taming Hosted Hypervisors with (Mostly) Deprivileged Execution. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[25] R. Yongjie. Nested Virtualization Test Report for Xen 4.3-RC1.

[26] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.