Technical Fundamentals of Cloud Systems

Cloud Computing

Universiteit Leiden

Key constituent

- The key constituent of Cloud Computing is virtualization: the ability to create *virtual machines*.

- "*A virtual machine is taken to be an efficient, isolated, duplicate of the real machine*".

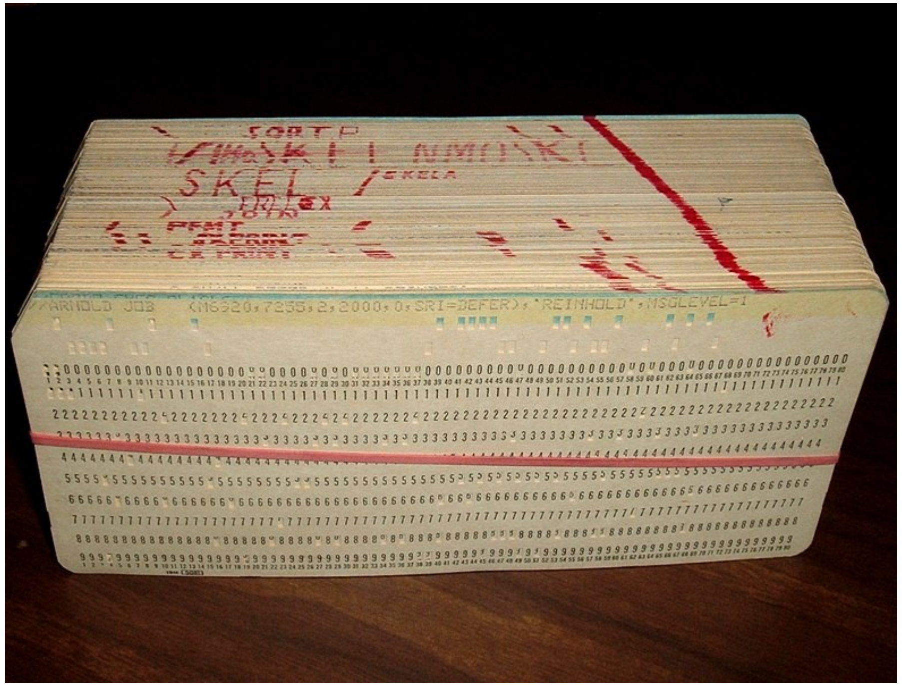
  – Popek and Goldberg. Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7

Key constituent

- The key constituent of Cloud Computing is virtualization: the ability to create *virtual machines*.

- "*A virtual machine is taken to be an efficient, isolated, duplicate of the real machine*".
  - Popek and Goldberg. **1974.** Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7 (July **1974**) 412-421.
  - So, not at all a new concept!!

A little history …



ArnoldReinhold [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)]

- Fifties and sixties: a computer was very large (classroom times 4) and very expensive.
- One computer per organization.
- Operation based on *batch processing*.
- Peripheral computers were used to prepare *jobs* to be submitted to the batch queue.
- Users could not directly interact with the real machine; always had to wait turn for their job to be processed.

A little history … (2)

- Slowly but surely computers became more powerful.
- First work on timesharing was done beginning of the sixties at MIT. Possible to interrupt a job in order to perform some interactive work.
- IBM CP/CMS provided the first implementation of a virtual machine in 1964.
  - Mainframe computer: IBM System/360 Model 40, equipped with a special additional device that we now know as an MMU.
  - "Second-generation time-sharing system"

A little history … (3)

Image source: http://en.wikipedia.org/wiki/File:Bundesarchiv_B_145_Bild-F038812-0014,_Wolfsburg,_VW_Autowerk.jpg

Image source: http://en.wikipedia.org/wiki/File:IBM_System360_Model_30.jpg

CP/CMS

- CP: Control Program
  - Operating System (OS) that simulates multiple copies of the machine it is running on.
- CMS: Conversational Monitor System
  - Interactive OS for a single user.
  - Able to run under CP or directly on the hardware! This made no difference.
  - Time sharing: give each user its own copy of the CMS.

Vision for the future

- In 1981, R. J. Creasy shared the following vision for the future:

  " As larger, faster, and less expensive machines become available, the software systems supporting interconnected virtual machines can move smoothly to collections of real machines."

  R. J. Creasy (1981): *The Origin of the VM/370 Time-Sharing System*.

So, what actually happened?

- Virtualization capabilities continued to be used on IBM mainframes (up to today).
- However, many software moved to local (desktop) computers. Sometimes supported by on-premise server systems.
  - Do note the lack of a global high speed network that people had to deal with in the past!

So, what actually happened? (2)

On-premise (commodity) server systems were underutilized.

- Due to cost (ownership, power utilization) it became interesting to consolidate servers using virtualization.
- It was made possible to virtualize the commodity x86 platform.
- Add to this the major advances in the development of a global high speed network (the Internet) from 1995 onwards.
- And cloud computing was born:
  - Amazon announced EC2 (Elastic Compute), an IaaS product, in 2006.
  - and many others followed.

Retrospect

Interesting turn of events:

- from centralized mainframes,
- to minicomputers,
- to desktop computers,
- to laptops,
- to tablet and smartphone satellite stations connected to global cloud infrastructure.

Virtualization of machines

- The architecture behind the IBM mainframes was designed to support virtualization.
- Trap-and-emulate method:
  - Guest OS ran in “user mode” of the VMM.
  - If the guest attempted to execute a privileged instruction, a trap occurred.
  - CP took over and emulated the privileged instruction.

Virtualization of machines (2)

- Also the virtualization of devices was foreseen.
- CP could attach real devices such as tape drives to virtual machines.
- Virtual disks were mapped onto disk volumes.
- Virtual card puncher and reader were implemented via a spool subsystem.
  - (Just like printers these days: prepare the entire job and send it to a printer queue).

Virtualization of machines (3)

- Later, it was found that emulating **all** privileged instructions was too expensive.
- IBM System/370 was the successor of the System/360 and implemented the "SIE" instruction.
  - SIE: Start Interpretive Execution.
  - Within this mode, the hardware could emulate most privileged instructions without interference of the hypervisor. These instructions would operate on shadow system registers configured by the hypervisor.
  - In some cases, still a trap to hypervisor necessary.

Osisek, D. L., et al. 1991. ESA/390 interpretive-execution architecture, foundation for VM/ESA. IBM Systems Journal 30, 1 (1991), 34–51.

Virtual Machine Monitor

- The CP acted as Virtual Machine Monitor (VMM).
- Popek and Goldberg defined the following three characteristics of a VMM:
  1. A VMM creates a program environment that is equivalent to that of the original machine.
  2. Within this environment, programs run only slightly slower in the worst case.
  3. The VMM has full control of all system resources.

Popek and Goldberg. 1974. Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7 (July 1974) 412-421.

Virtual Machine Manager (2)

Something is considered a VMM if it adheres to the following three properties:

- ***Equivalence***: the program (guest) is executed in a way such that it is indistinguishable whether a control program is present; all privileged instructions can be executed as meant by the programmer.
- ***Efficiency***: all normal instructions are executed directly by the hardware without interference by the control program.
- ***Resource control***: a program cannot grant itself additional resources; this can only be done through the allocator of the control program.

Hypervisors

- VMMs are these days usually referred to as hypervisors and implement “system VMs”.
  - Contrary to process VMs such as JVM and .NET VM.
- Three types of hypervisors are distinguished:
  - **Type-0:** a firmware hypervisor implemented in the hardware.
  - **Type-1**: a bare metal hypervisor runs directly on top of the hardware.
  - **Type-2**: hypervisor runs on top of a host operating system.

Virtualizing x86

- Now, we would like to virtualize x86, because many are using commodity x86 hardware and not mainframes.
- The x86 architecture cannot be virtualized the “classical way”.
  - No traps are generated when attempting execution of privileged instructions in user mode. Some of these instructions may simply segfault.
  - Some instructions exhibit different behavior depending on being run in user or kernel mode; so always user-mode execution is taken, even in virtual kernel mode.
  - 'real' and 'virtual 8086' modes allow a guest OS to determine it is not running in privileged mode; so the execution environment is not equivalent to “bare metal” execution.

Virtualizing x86 (2)

- One could consider interpreting the x86 instructions, but this is very slow and violates one of the VMM criteria.
- VMWare's solution: binary translation.
  - Translate executable code, including privileged instructions to a code using only user-mode instructions.
  - Privileged instructions are replaced with specific code to emulate the instruction.

Virtualizing x86 (3)

The guest OS maintains its own page table.

- The guest cannot modify the MMU, the host must do this.
- Further, note that guest physical addresses are actually virtual addresses from the host's point of view.
  - gVA → gPA
- So, the translation is not complete!
  - We need a subsequent translation gPA / hVA → hPA.
- How to solve this?

Virtualization x86 (4)

- The VMM marks the page table of the guest OS as read-only, such that it can catch every write operation.
- For every gVA → gPA translation that the guest OS adds, the VMM ensures the corresponding gPA → hVA → hPA translation.
  - Shadowing the guest's translations.
- The translation gVA → hPA is stored in the host page table that is accessed by the CPU's MMU.
- This is called shadow paging.
  - It works; but a very expensive technique.

x86 extensions for virtualization

- Intel and AMD introduced extensions for virtualization to their implementation of the x86 architecture:
  - Intel VT-x (earlier known as Vanderpool)
  - AMD-V (earlier known as Secure Virtual Machine (SVM))
- A new execution mode ("ring") is introduced: guest mode.
  - Within this mode a guest can execute privileged instructions. The hardware executes these based on shadow registers.
  - Exit from guest mode when a certain condition occurs, such as HLT, CR3 load/store, page fault, device I/O.
  - On exit, the VMM can take control and emulate the instruction if necessary.
  - Modeled after "SIE mode" introduced on IBM System/370.

x86 extensions for virtualization (2)

- Relies on “vmrun” and “vmexit” instructions, which are quite expensive in the number of cycles.
  - You want to avoid “exit” operations.
- A 2006 paper from VMware compares hardware and software VMM performance. (Note: this is before the introduction of nested paging extensions).
  - Take it with a grain of salt?
  - Both software and hardware VMM perform well on compute-bound codes.
  - Advantage software VMM: replace vmexit/vmrun instructions with fast function calls through binary translation. Excels on heavy I/O & process context switching.
  - Advantage hardware VMM: can run system calls without invoking VMM. Good for workloads with many system calls.

K. Adams and O. Agesen. 2006. A comparison of software and hardware techniques for x86 virtualization. ASPLOS 2006. ACM.

x86 extensions for virtualization (3)

- Further micro-architectural improvements required.
- Around 2008, introduction of hardware MMU support: AMD "nested paging", Intel "extended page tables".
- Idea: VMM maintains a "nested page table".
  - As a result, the guest can maintain its page table without triggering traps.
  - The MMU walks both the "guest page table" as well as the "host page table".
  - MMU virtualization.

x86 extensions for virtualization (4)

- Guest has a page table “gPT”: gVA → gPA
  - Can be modified without hypervisor involvement.
- Hypervisor takes care of nPT: gPA → hPA
- Hardware performs a page walk on gPT and nPT to perform the full translation gVA → hPA.

x86

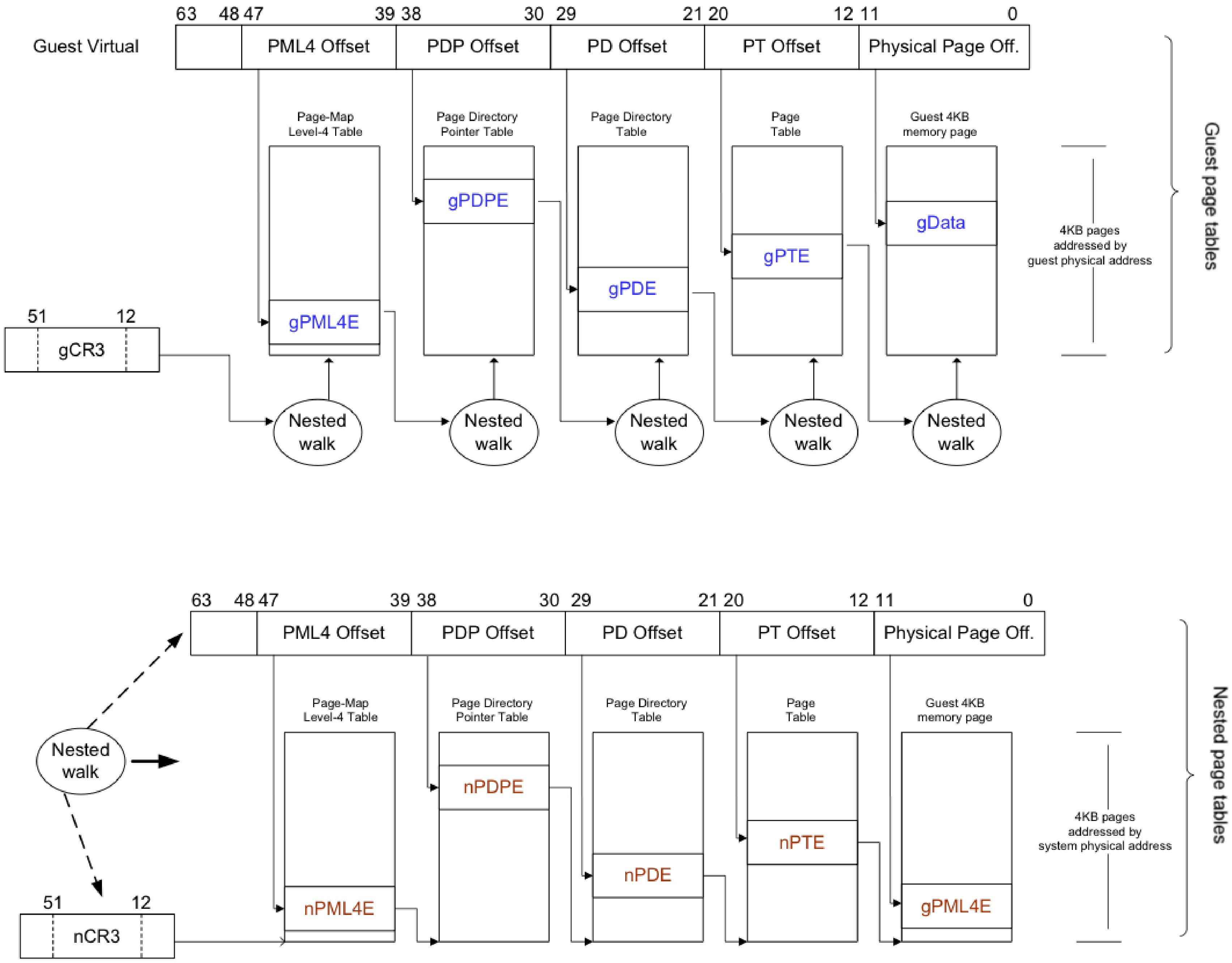


Figure from: AMD-V Nested Paging White Paper, AMD Inc., 2008.

Nested virtualization

- Can we nest this? Run a VM in a VM? Or a hypervisor within a hypervisor.
- Need hardware support; e.g., IBM interpreted SIE on System/370.
- Or try to resolve this in software.
  - On x86, traps always trap to the lowest-level hypervisor.
  - Make this hypervisor aware of nesting and forward the trap to the correct guest.
  - Implementation details: Ben-Yehuda, Muli, et al. 2010. The turtles project: Design and implementation of nested virtualization. OSDI 10.

Hypervisor implementations

- Hosted virtualization (type-2)
  - Desktop: VMWare Workstation, Virtual Box, Parallels Desktop.
  - Linux KVM: Kernel-based virtual machine
- Bare-metal virtualization (type-1)
  - VMware ESX
  - Xen
    - Hypervisor kernel does not have drivers; this is delegated to the dom0 kernel (first kernel that is started by the hypervisor).
  - Microsoft HyperV
- Hardware virtualization (type-0)
  - IBM LPAR (as found in mainframe systems)

Linux KVM

- KVM is a kernel interface that can be used to configure the address space of a guest VM.
- Qemu is used to run guest VMs and uses the KVM interface if available.
  - Qemu can emulate hardware devices, but preferably “paravirtualized” drivers are used (see next slide).
  - Qemu “cores” appear as processes in the host OS. The host OS scheduler is used.
- Virtual disks are mapped to local files or partitions, or even remote disks (iSCSI, Ceph RBD).

Device emulation vs. paravirtualized devices

- Virtual machines need access to (virtualized) hardware devices such as disks or network interfaces.
- Full emulation of such devices means that the actual real hardware needs to be accurately emulated.
  - To start with, this is complex, as the hardware device needs to be accurately mimicked for the device driver within the guest OS kernel to work.
  - Hardware devices are typically controlled through memory-mapped I/O or privileged instructions. This often results in many traps to the hypervisor.
- So, although hardware will work out of the box, it comes with a performance penalty.

Device emulation vs. paravirtualized devices (2)

- Paravirtualized devices are a solution to this problem.
  - The device is provided by the hypervisor with a particular interface.
  - The guest OS needs a special device driver to communicate with this device. This device driver will access the interface provided by the hypervisor.
  - As a result, the guest and hypervisor can communicate with each other through efficient means such as hypervisor calls or a shared data structure and avoid expensive traps.
- Desktop solutions such as VMware and VirtualBox often come with special drivers for the guest OS to improve graphics performance; this can be seen as an example.
- Linux KVM provides virtio interfaces, that use “virtqueues” to facilitate communication between guest and hypervisor.

VM Live Migration

- In modern data centers, the ability to “live migrate” active VMs is important.
- Approach:
  - Copy pages.
  - Suspend & migrate CPU state.
  - Redirect network traffic to new physical host.
  - Copy final pages.
  - Finalize & clean up.

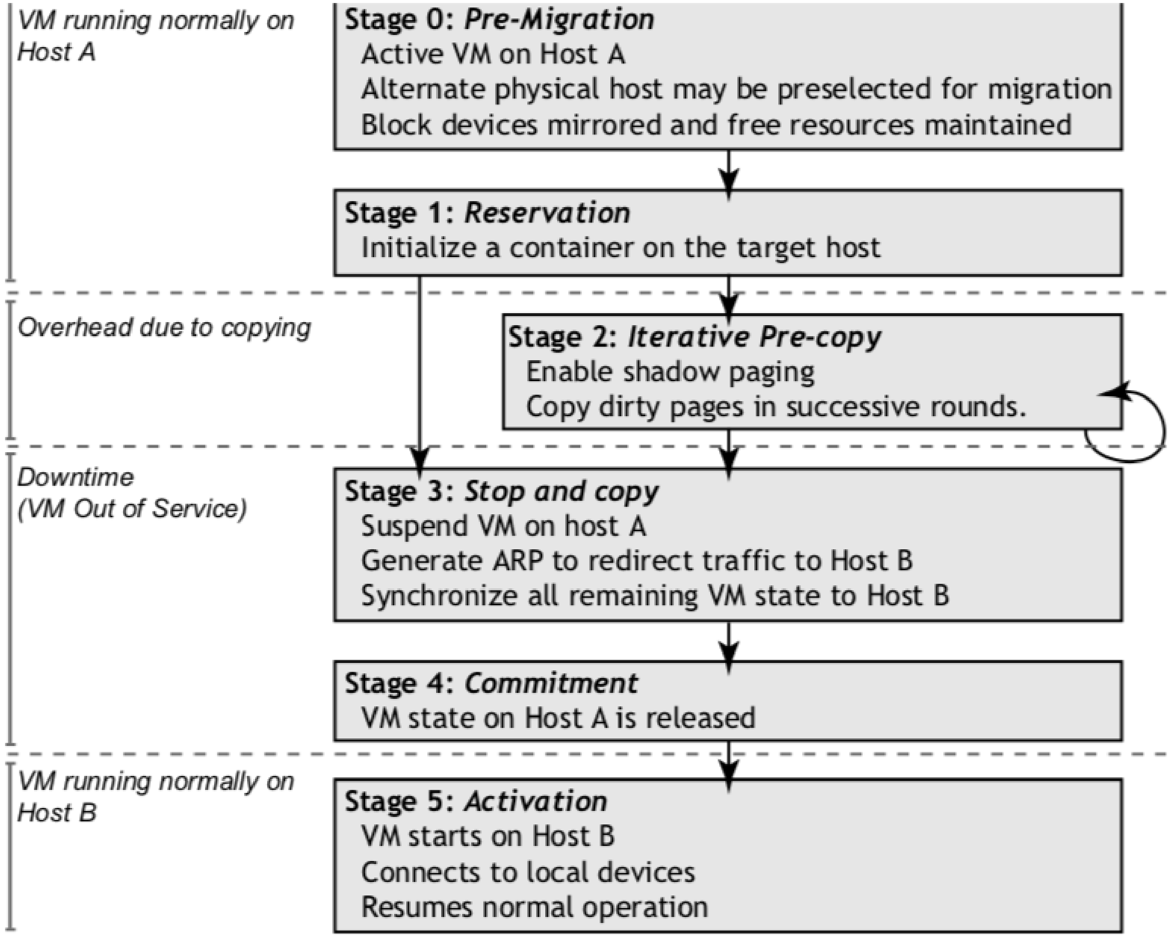


Figure 1: Migration timeline

Figure from: C. Clark et al. 2005. Live Migration of Virtual Machines. In: NSDI '05: 2nd Symposium on Networked Systems Design & Implementation. USENIX Association.

Universiteit
Leiden