Internet of Threads: Processes as Internet Nodes

Renzo Davoli
Computer Science and Engineering Department
University of Bologna
Bologna, Italy
Email: renzo@cs.unibo.it

***Abstract*—In the beginning, Internet and TCP/IP protocols were based on the idea of connecting computers, so the addressable entities were networking adapters. Due to the evolution of networking and Internet services, physical computers no longer have a central role. Addressing networking adapters as if they were the true ends of communication has become obsolete. Within Internet of Threads, processes can be autonomous nodes of the Internet, i.e., can have their own IP addresses, routing and QoS policies, etc. In other words, the Internet of Threads definition enables networked software appliances to be implemented. These appliances are processes able to autonomously interoperate on the network, i.e., the software counterpart of the Internet of Things' objects. This paper will examine some usage cases of Internet of Threads, discussing the specific improvements provided by the new networking support. The implementation of the Internet of Threads used in the experiments is based on Virtual Distributed Ethernet (VDE), Contiki and View-OS. All the software presented in this paper has been released under free software licenses and is available for independent testing and evaluation.**

***Keywords*-Internet; IP networks; Virtual Machine Monitors**

I. INTRODUCTION

The design of the Internet is dated back to the beginning of the 1960s. The main goal and role of the Internet was to interconnect computers. It was natural, though, to consider the network controller of computers as the *addressable entities*. The endpoints of any communication, those having an Internet Protocol (IP) address were the hardware controllers [2]. This *ancient* definition is still used in the modern Internet. Processes and threads are identified by their *ports*, a sub-structure of the addressing scheme, as they are considered dependent on the hardware (or virtual) computer they are running on.

In the common scenario when a client application wants to connect to a remote internet service, it first gets the IP address of the server providing the service, and then the client application opens a connection to a specific well-known (or pre-defined) port at that address.

Thus, the DNS maps a logical name (e.g., www.whitehouse.gov or ftp.ai.mit.edu) to the IP address of a network controller of a computer that provides the service. This emphasis on the hardware infrastructure providing the service is obsolete: the main focus of the Internet nowadays is on services and applications. The whole addressing scheme of the Internet should change to address this change of perspective.

By Internet of Threads (IoTh) we mean the ability of processes to be addressable as nodes of the Internet, i.e., in IoTh processes play the same role as computers, being IP endpoints. They can have their own IP addresses, routing and QoS policies, etc.

On IPv4, IoTh usage can be limited by the small number of available IP addresses overall, but IoTh can reveal all its potential in IPv6, whose 128-bit long addresses are enough to give each process running on a computer its own address.

This change of perspective reflects the current common perception of the Internet itself. Originally, Internet was designed to connect remote computers using services like remote shells or file transfers. Today users are mainly interested in specific networking services, no matter which computer is providing them. So, in the early days of the Internet, assigning IP addresses to the networking controllers of computers was the norm, while today the addressable entity of the Internet should be the process which provides the requested service.

For a better explanation, let us compare the Internet to a telephone system. The original design of the Internet in this metaphor corresponds to a fixed line service. When portable phones were not available, the only way to reach a friend was to guess where he/she could be and try to call the nearest line. Telephone numbers were assigned to places, not to people. Today, using portable phones, it is simpler to contact somebody, as the phone number has been assigned to a portable device, which generally corresponds to a specific person.

In the architecture of modern Internet services, there are already exceptions to the rule of assigning IP addresses to physical network controllers.

- Virtual Machines (VM) have virtual network controllers, and each virtual controller has its own IP address (or addresses). This extension is, from some perspectives, similar to IoTh. In fact, it is possible to run a specific VM for each networking service. This approach, although possible, would clearly be ineffective. It is highly resource demanding in terms of: main memory to emulate the RAM of the VM; disk space for the virtual secondary memory of the VM; time since the VM has to run an entire Operating System Kernel providing processor scheduling, memory managing, etc.
- Each interface can be assigned several IP service ori-

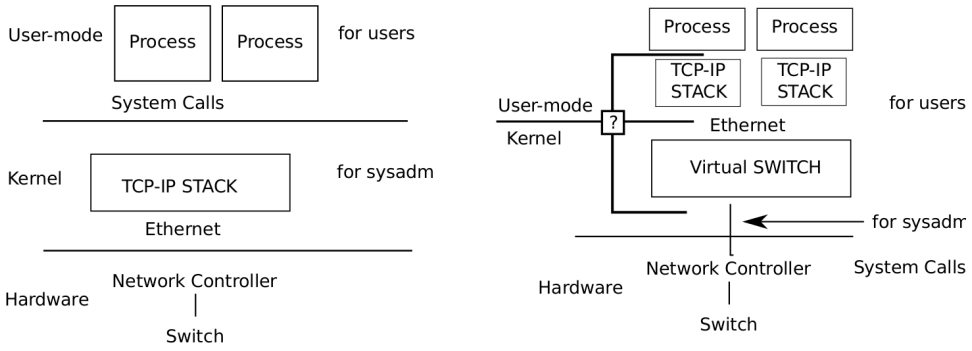

Fig. 1. Different perspectives on the networking support: the standard OS support is on the left side, IoTh is on the right side

ented addresses. Thus, it is possible to define different addresses, each one corresponding to a specific service. For example, if a DNS maps www.mynet.org to 1.2.3.4, and ftp.mynet.org to 1.2.3.5, it is possible to assign both addresses to the same controller. Addresses can be assigned to a specific process using the *bind* system call. This approach is commonly used in High Availability (HA) servers, where there is the need to migrate services from one host to another in case of faulty hardware or software [3]. This approach:

  - requires a complex daemon configuration (each socket must be bound to the right address) and network filtering (e.g., using *iptables* [4]) to prevent services from being reached using the wrong logical address (e.g., http://ftp.mynet.org);
  - provides the way to migrate service daemons in a transparent manner for clients, but it does require a non trivial procedure to delete addresses and network filtering rules on one server and then define the same addresses and filtering rules on the other, prior to starting the new daemon process.

- Linux Containers (LXC), as well as Solaris Zones [5], [6], allow system administrators to create different operating environments for processes running on the same operating system kernel. Among the other configurable entities for containers, it is possible to define a specific network support, and to create virtual interfaces of each container (flag CLONE_NEWNET of clone(2)). The definition and configuration of network containers, or zones, are privileged operations for system administrators only. This feature appears very close to the intents of IoTh. However, in LXC the creation of a networking virtual interface, thus the setting of a new IP address, is a system administration operation. In fact, it requires the process to own the CAP_NET_ADMIN capability, the same required to modify the configuration of the physical controller of the hosting computer. In IoTh, the creation of a networking environment for a process is as simple as a library function call. In this way, a process defines its IP address(es) as an ordinary user operation. IoTh provides Network Access Control to prevent abuses of networking services at the virtual Data-Link layer (in general a Virtual Ethernet). A process can define its interfaces and its IP addresses as the owner of a Personal Computer (PC) can assign any IP address to the interfaces of their PC connected to a Local Area Network (LAN). If the IP address is wrong, or inconsistent with the addresses running on the LAN, that PC cannot communicate. And even if the address is correct, it is possible to set up firewalls to define what that PC can do and what cannot be done. In IoTh each process can play the role of the PC in the example. In the same way, virtual firewalls can be set up to define which are the permitted networking services.

The paper will develop as follows: Section II introduces the design and implementation of Ioth, followed by a discussion in Section III. Related work is described in Section IV. Section V is about usage cases. Section VI introduces an innovative way to assign IP addresses and Section VII is about practical examples. Section VIII discusses the security issues related to IoTh and Section IX provides some performance figures of a proof-of-concept implementation. The paper ends with some final considerations about future work.

II. Design and Implementation of IoTh

The role and the operating system support of the Data-Link networking layer must be redesigned for IoTh. Processes cannot be plugged to physical networking hubs or switches as they do not have hardware controllers (in the following the term *switch* will be used to reference either a switch or a hub, as the difference is not relevant to the discussion). On the other hand, it is possible to provide processes with virtual networking controllers and to connect these controllers to virtual switches. Figure 1 depicts the different perspectives on the networking support. The focus of Fig. 1 is to show how IoTh changes the Operating System (OS) support for networking: what is provided by the hardware vs. what is implemented in software, what is shared throughout the system vs what is process specific and what is implemented as kernel code vs. what runs in user-mode.

The typical networking support is represented on the left side of Fig. 1. Each process uses a networking Application Program Interface (API), usually the Berkeley sockets API [7],

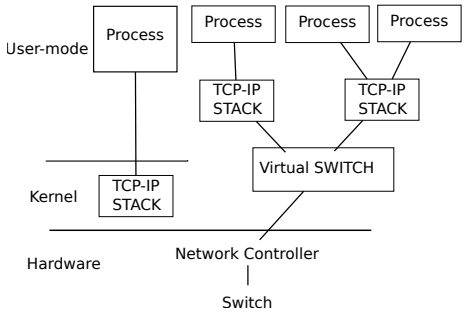

Fig. 2. A more complex scenario of IoTh usage

to access the services provided by a single shared stack, or by one of the available stacks for zones or LXC (see Section I). The TCP-IP stack is implemented in the kernel and directly communicates with the data-link layer to exchange packets using the physical LAN controllers.

In IoTh, represented on the right side of the figure, unprivileged processes can send data-link packets using virtual switches, able to dispatch data-link packets from process to process and between processes and virtual interfaces (e.g., tuntap interfaces) of the hosting OS. Virtual switches can also be interfaced to physical networking controllers, but this latter operation is privileged and requires specific capabilities (CAP_NET_ADMIN).

So, the hardware-software boundary has been moved downwards in the IoTh design. In fact, the data-link networking (commonly the Ethernet) includes software components in IoTh, i.e., virtual switches, for unprivileged user processes. In IoTh the virtual switches are shared components between user processes, while the TCP-IP stacks (or, in general, the upper part of the networking stacks, from the networking layer up) are process specific. It is also possible for a group of processes to share one TCP-IP stack, but in the IoTh design this is just an implementation choice and no longer an OS design issue or system administration choice.

The kernel/user-mode code boundary is flexible in IoTh: both the virtual ethernet switches and the TCP-IP stacks can be implemented in the kernel or not. A virtual switch can be a standard user-mode process or a kernel service, while a TCP-IP stack is a library that can be implemented in the kernel to increase its performance.

Figure 2 shows a more complex scenario that is more consistent with a real usage case of IoTh. In fact, the traditional support for networking and the IoTh approach co-exist in a system. Following the telephony systems example of the introduction (see Section I), fixed line services and portable phones interoperate.

The OS running on a computer still needs a computer specific TCP-IP stack and IP addresses to be used when a system administrator needs to configure some systemwide service. In the telephone service metaphor we use fixed lines when we want to be sure to call a specific place. Internet services (like ftp, web, MTA, etc.) can be reached using their own IP addresses. These services are the portable phones of the metaphor.

IoTh can be used for networking clients, too. Several virtual switches running on the same hosting system can be connected to different networks and can provide different services, e.g., can have different bandwidths or routing. Additionally, several virtual switches can be active on the laptop of a professor attending a conference, one connected by an encrypted Virtual Private Network (VPN) to his/her University's remote protected intranet and another directly connected to the physical, maybe wi-fi, LAN of the conference center.

Users of IoTh enabled OS, like the professor in the example, will be able to choose between the available networking services in their applications (browser, voip clients, etc.) just as they currently choose the printer.

III. Discussion

All the concepts currently used in Local Area Neworking can be applied to IoTh networking.

Virtual switches define virtual Ethernets. Virtual Ethernets can be bridged with Physical Ethernets, so that workstations, personal computers or processes running as IoTh nodes are indistinguishable as endnodes of Internet communication. Virtual Ethernets can be interconnected by Virtual Routers. It is possible to use DHCP [8] to assign IP addresses to processes, to use IPv6 stateless autoconfiguration, [9], to route packets using NAT [10], to implement packet filtering gateways, etc. This is a non-exaustive list of protocols and services running on a real Ethernet that work on a virtual Ethernet, too. It is also possible to create complex virtual networking infrastructures composed by several virtual Ethernets such as virtual application level firewalls with De Militarized Zone (DMZ) networks and virtual bastion hosts.

IoTh can support the idea of network structure consolidation in the same way that the Virtual Machines provided the idea of Server consolidation. Complex networking topologies can be virtualized, thus reducing the costs and failure rates of a hardware infrastructure. Today it is common for large companies to substitute their servers with virtual machines, creating, in this way, their internal cloud, or moving their servers to an external system-as-a-service (SaaS) cloud. This choice permits a more flexible and economically effective management of the servers and, at the same time, all the investments in terms of software can be preserved, as it is possible to move each server to a virtual counterpart, while mantaining the same software architecture: operating system type and version, libraries, etc. IoTh adds one more dimension to this consolidation process: it is possible by IoTh to virtualize not only each server as such, but also the pre-existing networking infrastructure.

Network consolidation is just an example of IoTh as a tool for compatibility with the past. In this example each process joining the virtual networks is just a virtual machine or a virtual router or firewall. The granularity of an Internet node is flexible in IoTh. A virtual machine can be an Internet node, but each browser, bit-torrent tracker, web server or mail transport agent (MTA) can be an Internet node, too.

Through IoTh, there is no difference between local and remote Inter Process Communication (IPC) services. A process can have its own IP address(es) and can interoperate with other processes using standard protocols and standard ports. Several processes running on the same host can use the same port, since each one uses different IP addresses. The same IPC protocols can be used regardless of the host on which the process is running: nothing changes, whether the communicating processes are running on the same host or on different, perhaps remote, computers. This allows a simpler migration of services from one machine to another.

Each process in IoTh can have its user interface implemented as an Internet service. This means, for example, that it is possible to create programs which register their IP addresses in a dynamic DNS, and where their web user interface is accessible through a standard browser.

IoTh is, from this perspective, the software counterpart of Internet of Things (IoT [11]). In IoT hardware gadgets are directly connected to the network. IoT objects interact between themselves and with users through standard Internet protocols. IoTh applies the same concept to processes, i.e., to software objects as if they were virtual IoT gadgets. These IoTh-enabled processes using internet protocols to interoperate can be called networked virtual appliances. If they were implemented on specific dedicated hardware objects, they would become things, according to the definition of IoT.

IV. Related Work

IoTh uses and integrates several concepts and tools already available in the literature and in free software repositories.

A. Virtual Ethernet Services

IoTh is based on the availability of virtual data-link layer networking services, usually virtual Ethernet services, as Ethernet is the most common data-link standard used. Virtual Ethernet networking was first introduced as a networking support for virtual machines. Originally, each virtual machine monitor program was provided with its specific virtual networking service. Some of them were merely an interface to a virtual networking interface (tuntap [12]). The kernel of the hosting operating system had to manage the bridging/switching or routing between virtual machines and real networks. User-mode Linux (UM-L [13]) introduced the idea of network switch as a user process interconnecting several UM-L VMs.

Virtual Distributed Ethernet (VDE [14]) extended the virtual switch idea in many ways:

- VDE created a virtual plug library to interconnect different types of VMs to the same virtual switch. Currently, User-Mode Linux, qemu [15], kvm [16], virtualbox [17] natively support VDE in their mainstream code. Virtuallyi, any VM that supports tuntap can also be connected to VDE using the virtual tuntap library.
- VDE switches running on different hosts can be connected by virtual cables to form an extended virtual Ethernet LAN. All the VM connected to one of the interconnected switches regard the others as if they all were on the same LAN.
- VDE provides support for VLANs, Fast spanning tree for link fault tolerance, remote management of switches, etc.
- VDE is a service for users: the activation of a VDE switch, the connection of a VM to a switch, or the interconnection of remote switches, are all unprivileged operations.

VDE switches were first implemented using user-level processes, but there is an experimental version of faster VDE switches running as kernel code (kvde_switch). Although this implementation runs as kernel code, kernel switches can be started and managed by unprivileged users and processes. kvde_switch is based on Inter Process Networking (IPN [18]) sockets, a general purpose support for broadcast/multicast IPC between processes.

The idea of a general purpose virtual Ethernet switch for virtual machines has been implemented by some other projects, too:

- OpenVswitch [19] is a virtual Ethernet switch for VMs implemented at kernel level. OpenVswitch has VLAN and QoS support. It has been designed to be a fast, flexible support for virtual machines running on the same host. It does not support distributed virtual networks, and requires root access for its configuration.
- Vale [20] is a very fast support for virtual networking, based on the netmap [21] API. It uses shared memory techniques to speed-up the communication between the VMs. Vale, like OpenVswitch, does not directly support distributed networks and must be managed by system administrators.

B. TCP-IP stacks

As described in the introduction, the TCP-IP networking stack is generally unique in a system and it is considered as a shared systemwide service provided by the kernel. The implementation of the TCP-IP stack can be found in the kernel source code of all the free software kernels. In Linux, the IPv4 stack is in the directory /net/ipv4 and the IPv6 stack is in /net/ipv6. Alpine [22] is an early approach to network virtualization for protocol development. In fact, Alpine used a customized BSD kernel running as a user process to serve as a partial virtual machine monitor to virtualize the system calls for networking. TCP-IP stack implementations as libraries are common as software tools for embedded system design. Many manufacturers of embedded system platforms provide their own TCP-IP libraries as part of their development kits. Some of these libraries have been released as free software. Unfortunately, many of these libraries provide minimal or partial implementation of the networking stacks to be used for specific purposes only and are tailored or optimized for a specific embedded system hardware architecture. Adam Dunkels wrote two general purpose and free licensed TCP-IP stacks for embedded systems: uIP [23] and LWIP (Light Weight IP) [24]. uIP is a very compact stack for microcontrollers having limited resources, while LWIP is a more complete implementation for

powerful embedded machines. LWIP was initially designed for IPv4, but a basic support for IPv6 has recently been added. In 2005, when LWIP did not support IPv6 yet, VirtualSquare labs created a fork of LWIP, named LWIPv6 [25]. LWIPv6 then evolved independently and is now a library supporting both IPv4 and IPv6 as a single hybrid stack, i.e., differently from the dual-stack approach, LWIPv6 manages IPv4 packets as a subcase of IPv6 packets. When LWIPv6 dispatches an IPv4 packet it creates a temporary IPv6 header, used by the stack, which is deleted when the packet is later delivered. LWIPv6 is also able to support several concurrent TCP-IP stacks. It has features like packet filtering, NAT (both NATv4 and NATv6), slirp (for IPv4 and IPv6) [26].

C. Process/OS interface

In this work, we use two different approaches to interface user processes with virtual stacks and virtual networks. A way to create networked software appliances is to run entire operating systems for embedded computers as processes on a server. Contiki [27], or similar OSs, can be used to implement new software appliances from scratch. This approach cannot be used to interface existing programs (e.g., an existing web server like Apache) to a virtual network, unless the software interface for networking is completely rewritten to support virtual networking.

ViewOS [28] is a partial virtualization project. View-OS virtualizes the system calls generated by the programs, so unmodified binary programs can run in the virtualized environment. ViewOS supports the re-definition of the networking services at user level. Processes running in a View-OS partial virtual machine, where the networking has been virtualized, will use a user-mode stack instead of the kernel provided one. Server, client and peer-to-peer programs can run transparently on a View-OS machine as if they were running just on the OS, but using a virtualized stack instead of the kernel stack.

Another project provides network virtualization in the NetBSD environment: Rump Anykernel [29]. The idea of Rump is to provide user-mode environments where kernel drivers and services can run. Rump provides a very useful structure for kernel code implementation and debugging, as entire sections of the kernel can run unmodified at user level. In this way, it is possible to test unstable code without the risk of kernel panic.

At the same time, Rump provides a way to run kernel services, like the TCP-IP stack, at user level. It is possible to reuse the kernel code of the stack as a networking library, or as a networking deaemon at user level. Antti Kantee, the author of Rump, named this idea Anykernel. In Rump, it is possible to run each device driver or system service in three different ways: as kernel code, as user-mode code embedded in the application process, or as a user-mode server. These three operational modes respectively correspond to three kernel architectures, when applied to all the drivers and services: monolithic kernels, exokernels and microkernels. Then an Anykernel is a kernel where the kernel architecture is flexible and can be independently decided on each driver or server.

D. Multiple Stack support

Some IoTh applications require the ability for one process to be connected to several TCP-IP stacks at the same time. The Berkeley sockets API has been designed to support only a single implementation for each protocol family.

ViewOS and LWIPv6 use an extension of the Berkeley sockets API, *msocket* [30], providing the support for multiple protocol stacks for the same protocol family. A new system call *msocket* is an extended version of the *socket* system call: *msocket* has one more (leading) argument, the pathname of a special file which defines the stack to use. *msocket* is back compatible with the standard Berkeley sockets API: it is possible to define and modify the current default stack of a process. The *socket* system call will use the current default stack. The default stack is part of the process status, and it is inherited in the case of *fork* or *execve*. Existing programs using only *socket* can work on any available stack, one at a time.

V. Usage Cases

This section describes some general usage cases of IoTh. A complete description of the experiments, including all the details to test the results, can be found in Section VII.

A. Client Side usage cases

- Co-existence of multiple networking environments. This feature can be used in many ways. For example, it is possible to have a secure VPN connected to the internal protected network of an institution or a company (an intranet) on which it is safe to send sensitive data and personal information, and a second networking environment to browse the Internet.
  As a second example, technicians who need to track networking problems may find it useful to have some processes connected to the faulty service, while a second networking environment can be used to look for information on the Internet, or to test the faulty network by trying to reach the malfunctioning link from the other end.
- Creation of networking environments for IPC. Many programs have web user interfaces for their configuration (e.g., CUPS or xmbc). Web interfaces are highly portable and do not require specific graphics libraries to run. Using IoTh it is possible to create several Local Host Networks (LHN), i.e., virtual networks for IPC only, to access the web interfaces of the running processes. LHN can have access protection, e.g., an LHN to access the configuration interface of critical system daemons can be accessible only by *root* owned processes. All daemons can have their own IP address, logical name and run their web based configuration interface using port 80.
  Let us take the CUPS example. The web interface of CUPS is available at the port 631 on localhost, and allows users to read the status of the printers. System administrators can add or reconfigure printers, CUPS asks for a password authentication for these operations. Using IoTh it is possible to define the FQDN *cups.localhost*

to be a static IP address in the file /etc/hosts. An IoTh version of CUPS could join two LHN using *msockets* and use the same IP address on both. One virtual network is for system administrators, the other for users. User browsers can be connected to the LHN for system administrators if they are enabled, otherwise they can use the other LHN. All the configuration options will be disabled by CUPS for all accesses from the user LHN. The same approach as CUPS could be used for many system daemons providing a web interface. In this way, the LHN for system administration works as a single-sign-on channel for all the daemons. In fact, system administrators will not be required to be authenticated by a password, as only processes owned by a privileged account can join that LHN.

- Per user IP addresses. It is possible to use IoTh to assign an IP address to each user working on the same server. This is useful, for example, in critical environments, where tracking the responsibility of all the activities on the network is required. In multiuser and multitasking operating systems using a shared TCP-IP stack it is not generally possible, unless the configuration forces the system to log the mapping between each TCP connection or UDP datagram, and the owner of the process which requested the networking operation. It would be clearly a very expensive procedure in terms of processing power and storage requirements. This problem has been described in detail in User Level Networking (ULN) [31]. Through IoTh, each user can be given their own VDE, where their TCP-IP stacks or their processes can be connected. The router of user VDEs can assign IP addresses, or a range of IP addresses, to each user. In this way, there will be a direct mapping between each IP address and the user responsible for it. Likewise, it is possible to track the personal computers connected to a public network. Because of the distributed nature of VDE, the IP addresses of a user can be assigned to processes running on different hosting computers.
  IP addresses assigned per user allows for the implementation of differentiated services. Users or classes of users can be assigned different QoS and access constraints.

B. Server side usage cases

- Virtual hosting is a well-known feature of several networking servers: the same server provides the same kind of service for multiple domains. Apache web server is one of the most common examples of daemons supporting virtual hosting. The same Apache process can work as a web server for many domains. The target IP address used by the client, or the address specified in the GET request of the html protocol, can be used by Apache to assign each request to a domain.
  IoTh generalizes this idea. It is possible to run several instances of the same networking daemon, giving each one its IP address. It is possible to run several pop, imap, DNS, web, MTA, etc., daemons, each one using its own stack. All the daemons will use their standard port numbers.
  It may be objected that it is already possible to obtain a similar result by assigning all the IP addresses to the networking controller of the server (or to a controller) and configuring each daemon to bind its specific IP address. In this way, one shared TCP-IP stack manages all the addresses, and each daemon selects the one to use. Unfortunately, this makes this approach complex and prone to error for system administrators.
  For example, the difference between this approach and the IoTh method is clear when the IP address of a daemon must be modified. Several configuration files must be edited in a consistent manner, using one shared stack to complete the required operation: /etc/network/interfaces, the daemon's configuration files and, in well configured systems, the *iptables* directives of the firewalling rules.
  Using IoTh it is sufficient to change the daemon's address.
  It is possible to envisage daemons designed for IoTh that run several concurrent networking stacks and provide specific services, depending on the stack they receive the requests from.
- Service migration in IoTh is as simple as stopping the daemon process on one host and starting it on another one. In fact, a daemon process can have its embedded networking stack, so its IP address and its routing rules are just configuration parameters of the daemon process itself. A VDE can provide a virtual Ethernet for all the processes running on several hosts. Stopping the daemon process on one server and activating it later on a second server providing the same VDE is, in the virtual world, like unplugging the Ethernet cable of a computer from a switch and plugging it into a port of another switch of the same LAN (the ports of both switches should belong to the same VLAN).
- With IoTh it is possible to design network daemons which change their IP addresses in a dynamic way. One Time IP address (OTIP) [32] applies to IP addresses the same technique used for passwords in One Time Password (OTP) services [33]. In OTP, the password to access a service changes over time, and the client must compute the current password to be used to access the service. This is common for protecting on-line operations on bank accounts. When a customer wants to use his/her account, the current password must be copied from the display of a small key-holder device. OTIP uses the same concept to protect private services accessible on the Internet. A daemon process changes its IP address dynamically over time and all its legitimate users can compute its current IP address using a specific tool, and connect. Port scan traces and network dumps cannot provide useful information for malicious attacks, because all the addresses change rapidly.
  This method has mainly been designed for IPv6 networks. In fact, the current server address can be picked up as one

of the valid host addresses available on the local network, most of the time among 2^{64} possible addresses. Clearly, a 2^{64} address space is too large for attackers to try a brute force enumeration attack on all the available addresses, and even if they eventually succeeded, the retrieved addresses would have to be exploited before their validity expires and the servers move to new addresses.

C. Other usage cases

IoTh allows us to use several networking stacks. These stacks can be several instances of the same stack, or different stacks. In fact, it is possible to have different implementations of TCP-IP stacks or stacks configured in different ways, available at the same time. Processes can choose which one is best suited to their activities.

This feature can be used in different ways:

- Using an experimental stack as the single, shared stack of a remote computer can partition the remote machine in cases of a malfunctioning of the stack itself. IoTh enables the coexistence of the stack under examination with a reliable production stack, which can be used as a safe communication channel.
- Processes can have different networking requirements. For example, communicating peers on a high latency link need larger buffers for the TCP sliding window protocol. It is possible to configure each stack and fine tune its parameters for the requirements of each process, as each process can have its own stack.

VI. Hash-Based Addresses

The deployment of IoTh based services requires the definition of several IP addresses. In fact, the number of IP addresses used by IoTh can be orders of magnitude higher than the number of IP addresses currently assigned only to hardware or virtual controllers.

Although each (numerical) IP address could be defined by some form of autoconfiguration (stateful or stateless [8], [9]), the real problem is to provide the mapping between each Fully Qualified Domain Name (FQDN) of a service and the corresponding IPv6 address of the process providing the requested service. In other words, the management of a high number of IoTh addresses by the standard Domain Name Service (DNS) procedures can be complex, error prone and time consuming for system administrators.

In [35], we propose a novel IP address self-configuration scheme based on a 64-bit hash encoding of the FQDN to be used as the host part of the IPv6 address. The complete IPv6 address will then be composed by the network prefix of the (virtual) LAN followed by the hash encoding of the FQDN.

The above referenced paper shows how the name resolution of hash-based addresses can be managed automatically, requiring system administrators the provide solely a unique FQDN for each service. The hash-based IP addressing self assigning method is completely consistent with the name resolution protocols in use on the Internet [34]. So, existing networking clients and DNS implementations can seamlessly interoperate with DNS providing addresses using the hash-based name resolution.

It is worth noting that hash encoding might generate collisions. Thus, multiple FQDNs can correspond to the same hash-based IP address. The same problem can arise in OTIP generated addresses (see Section V) where two processes could temporarily acquire the same address. In both [35] and [32] there is a discussion about the statistical relevance of the problem. Being an instance of the well-known birthday paradox problem, the probability of hash collision can be estimated as follows:

$$Pr[(n, m)] \approx 1 - e^{-\frac{m^2}{2n}} \quad (1)$$

where m is the number of elements choosing the same random key amongst n possible keys.

Figure 3 shows the probability function (1) plotted for up to 1Mi (i.e., 2^{20}) computers. The probability of two addresses colliding is less than one in 30 million for a LAN connecting more than 1 million hosts. In more realistic cases, network connecting less than one thousand nodes, the probability has the order of magnitude of one in $3 \cdot 10^{14}$.

Although very unfrequent, collisions of hash-defined addresses may happen, but such collisions can be detected by DNS servers and reported to system administrators who can change some of the FQDNs of the services to solve the problem. On the contrary, for OTIP it is not possible to completely avoid the collision problem, which, however, could only cause extremely unlikely temporary unreachability state of the services involved in the collision.

VII. Practical examples

This section presents some practical experiments on IoTh. These experiments are based on several tools which have been designed or extended to provide a working proof-of-concept of IoTh. For an independent testing of the results published in this paper, the source code of all the software is available under free software licenses on public repositories.

The tools used for the experiments include

- Virtual Distributed Ethernet: vde_switch, vde_plug;
- LWIPv6: the stack and sliprv6;
- Contiki: including the VDE interface;
- View-OS: umview or kmview, umnet (network virtualization), umnetlwipv6 (interface to lwipv6), umfuse (virtual file system support), umfusefile (single file virtualization).

A. Example 1: client side IoTh

.

This example shows how to use IoTh to run a browser on its own network.

First of all, start a VDE switch, a ViewOS VM and connect the VDE to a remote network. It is sufficient to have an unprivileged user account on far.computer.org, slirpvde6 is able to route the entire subnet.

```
$ vde_switch -daemon -sock /tmp/vde1
$ umview xterm &
$ dpipe vde_plug /tmp/vde1 = \
```

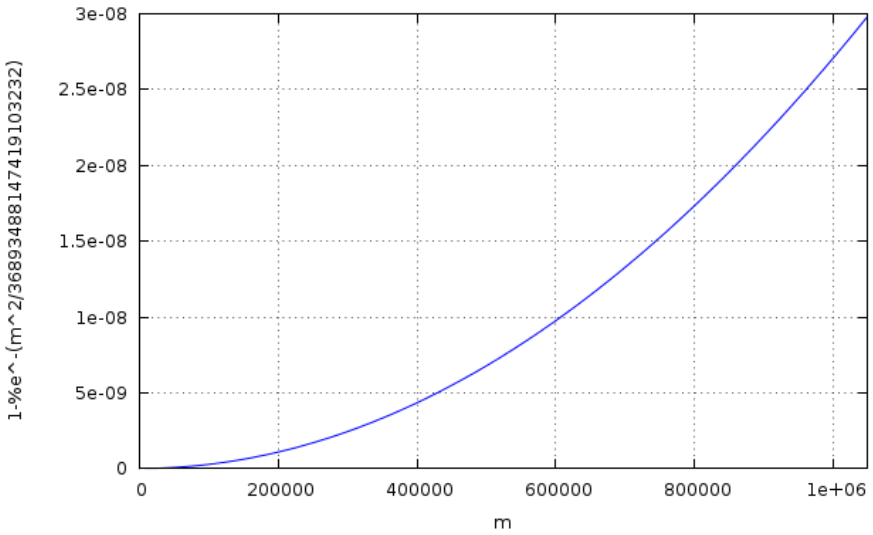

Fig. 3. Probability of address collision in a 64 bits hash

```
> ssh far.computer.org slirpvde6 -D -N -
```

The new xterm is running in the View-OS VM. In that xterm type:

```
$ um_add_service umnet umfuse
$ mount -t umnetlwipv6 none /dev/net/default
$ mount -t umfuseramfile -o ghost \
> /etc/resolv.conf /etc/resolv.conf
$ ip link set vd0 up
$ /sbin/udhcpc -i vd0 -q \
> -s ~/etc/udhcpc/default.script
$ firefox new
```

The first command loads the View-OS modules for network and file system virtualization. View-OS uses the mount operation to load a new network stack. It becomes the default stack for the VM because the target of the mount operation is `/dev/net/default`. `umfuseramfile` is used to virtualize `/etc/resolv.conf`: in this way, the virtual file is writable in this View-OS VM, and it is possible to define the DNS server to be used by the VM.

Then the following commands activate the virtual controller vd0, start a DHCP client to request a dynamic address for vd0, and start a browser.

All the connections of the browser will be seen from the Internet as if they were generated by `far.computer.org`

Figure 4 shows an example in which one browser is connected to the local networking service (the one in Bologna), while a second browser uses IoTh and is connected to an American network (the one in Kansas City).

B. Example 2: server side virtual network appliance

This example shows how to start virtual networked appliances. More specifically, the programs used in this example are a simple web server, based on Contiki, and a simple virtual network-attached storage (NAS), based on LWIPv6.

The source code for the Contiki example is included in the subversion *(svn)* repository of VDE on sourceforge [36] as a patch to the Contiki source tree. From the patched version of Contiki it is possible to generate the test program named `webserver-example.minimal-net-vde`.

As a first step, launch a VDE switch connected to a tap interface and assign an IP address to it.

```
$ sudo vde_switch -d -s /tmp/vde2 -tap tap0
$ sudo ip link set tap0 up
$ sudo ip addr add 192.168.100.1/24 dev tap0
```

Then start the Contiki web server:

```
export CONTIKIIP=192.168.100.2
export CONTIKIMASK=255.255.255.0
export CONTIKIVDE=/dev/vde2
webserver-example.minimal-net-vde
```

From a browser it is now possible to reach the Contiki web server at its own IP address, as shown in Figure 5.

The next step shows how to migrate the Contiki web server to another hosting machine. Then compile the Contiki web server on the other host, or copy the executable file, if the architecture of the remote machine is compatible with the local one. Open a new terminal window and create a VDE cable to the remote machine (far.machine).

```
$ dpipe -d vde_plug /tmp/vde2 = \
> ssh far.machine vde_plug /tmp/vde2[]
```

Open an ssh connection on the remote machine and start the Contiki web server using the same sequence of commands described here above. Now reload the web page on your browser. Nothing seems to change. The page is overwritten by an identical one, but that page is now provided by the Contiki server running on the remote machine.

The NAS example uses LWIPv6. The source code for this example is available from the wiki site of VirtualSquare [37].

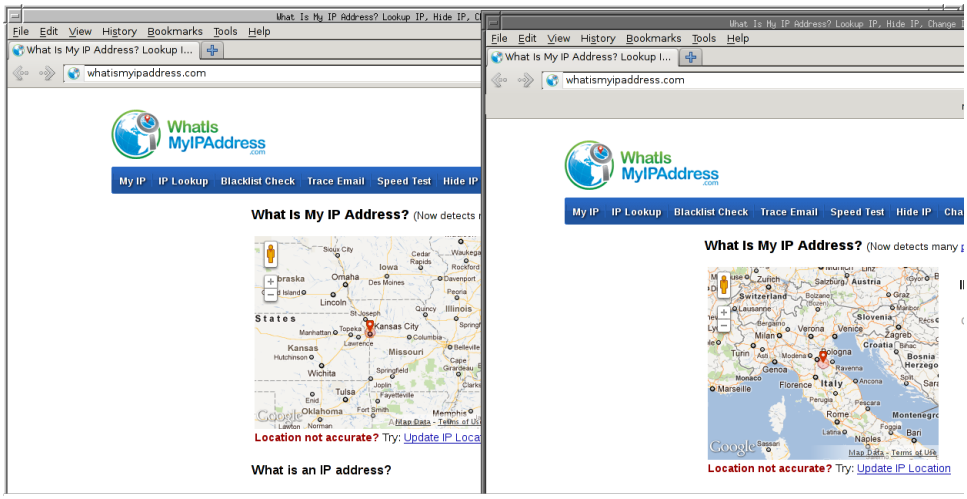

Fig. 4. Client side example of IoTh: the browsers are using two different stacks. The one in Kansas City uses a user-mode TCP-IP stack and a virtual networking switch.

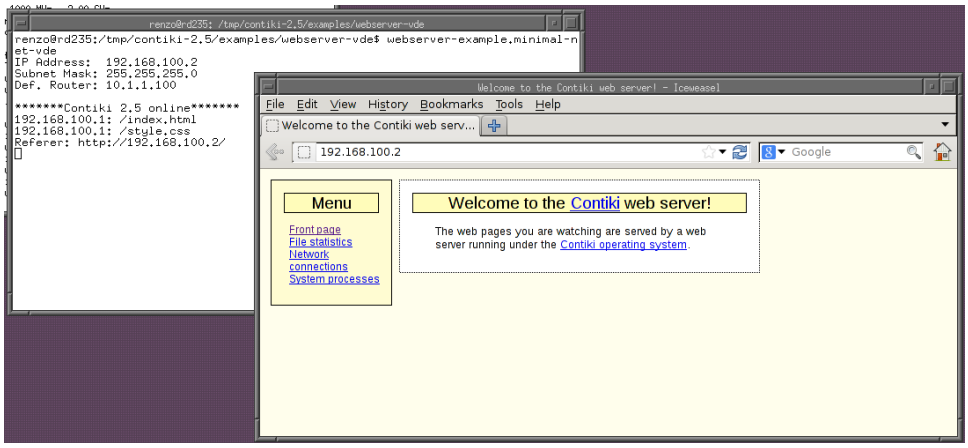

Fig. 5. A server side example of IoTh usage: the web page of the browser in foreground is provided by the contiki based virtual networked appliance started in the terminal window in background.

On a computer where the LWIPv6 library has already been installed (it can be generated from the sources available on the svn repository or simply installed as a packet for Debian Sid users), compile the webnas4 example.

Change your current directory to a subtree of the file system which does not contain private data, as its contents will be exported, and start webnas4 as follows:

```
$ /path/to/webnas4 \
> 192.168.100.3/24+02:01:02:03:04:05 /dev/vde2
```

All the contents of the current directory are now exported to the VDE network and accessible via web by loading the page `http://192.168.100.3` on the browser.

It is possible to use static global IP addresses for the Contiki address and for the NAS example, and to define a default router for both as follows:

```
$export CONTIKIDR=192.168.100.1
```

for Contiki and the option `route:192.168.100.1` for webnas4, using a convenient global IP prefix instead of 192.168.100. In this way, both virtual network appliances can be reached by any Internet user. These examples use IPv4 to shorten the address in the commands and to make them more readable, although they can be modified to use IPv6 instead.

VIII. Security Considerations

Several aspects of security must be taken into consideration in IoTh.

It is possible to limit the network access possibilities of an IoTh process and restrict the network services it can use. In fact, each IoTh process must be connected to a virtual local network to communicate, and virtual local networks have access control features. In VDE, for example, the permission to access a network is defined using the standard access control mechanisms of the file system. Each VDE network can be restricted to specific users, groups using the file permissions or Access Control Lists (ACL). The interaction between processes connected to a VDE and the other networks (or the entire Internet) can be regulated by specific configurations of

TABLE I
COMPARISON IN BANDWIDTH (MB/S) BETWEEN A KERNEL STACK AND IOTH

	10MB kernel	10MB IoTh	20MB kernel	20MB IoTh	40MB kernel	40MB IoTh
localhost	116	29.9	118	35.9	136	37.4
network 1Gb/s	104	41.9	112	49.0	112	51.7
network 100Mb/s	11.2	11.0	11.1	11.0	11.1	11.0

the virtual routers used to interconnect that VDE.

Limiting the IoTh process access to networking is just one aspect of IoTh security. It protects the environment from the effects of faulty, buggy or malicious processes.

It is also possible to consider the positive effects of IoTh with respect to protection from external attacks. Port scanning [38] is a method used by intruders to get information about a remote server, planned to be a target for an attack. A port scan can reveal which daemons are currently active on that server, then which security related bugs can be exploited.

This attack method is based on the assumption that all the daemons are sharing the same IP stack and the same IP addresses. This assumption is exactly the one negated by IoTh. Port scanning is almost useless in IoTh, since an IP address is daemon specific, so it would reveal nothing more than the standard ports used for that service. When IoTh is applied to IPv6, the process IP address on a VDE network can have a 64-bit prefix and 64 bits for the node address. A 64-bit address space is too large for a brute force address scan to be effective.

There are also other aspects of security to be considered regarding the effects of IoTh on the reliability of the hosting system. Daemon processes run as unprivileged user processes in IoTh. They do not even require specific capabilities to provide services on privileged ports (CAP_NET_BIND_SERVICE to bind a port number less than 1024). The less privileged a daemon process is, the smaller the damage it may cause in cases when the daemon is compromised (e.g., by a buffer overflow attack).

In some cases, IoTh can limit the effects of Denial of Service (DoS) attacks. In fact, DoS attacks may succeed by overloading not only the communication channels, but also the TCP-IP stacks of the target machine. In this latter case, in IoTh, a maximum load boundary for the daemon process can confine the effects of the attack to the target service, which is overloaded by the high rate of requests, while the other daemons running on the same host would be much less affected.

IX. PERFORMANCE OF IOTH

IoTh provides a new viewpoint on networking. As this paper has shown in the previous sections, IoTh allows a wide range of new applications. IoTh flexibility obviously costs in terms of performance. A fair analysis of IoTh performance has to consider the balance between the costs of using this new feature and the benefits it gives. In the same way, processes run faster on an Operating System not supporting Virtual Memory, but, for many applications, the cost of Virtual Memory is worthwhile because you can run a greater number of processes. The IoTh approach can co-exist with the standard management

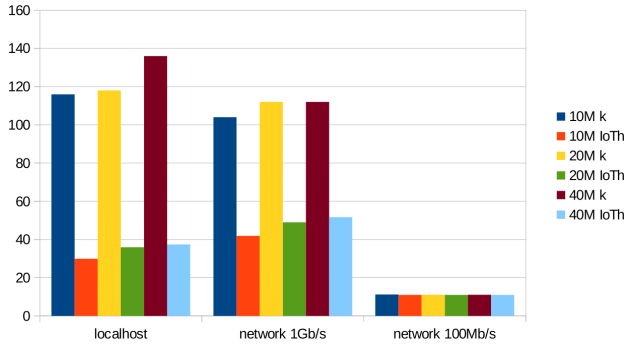

Fig. 6. A graphical view of Table I data

of IP addresses and services. System administrators can decide which approach is more suitable for each service.

Table I shows the comparison of the bandwidth of a TCP connection between the Linux Kernel TCP-IP stack implementation and a IoTh implementation based on VDE and LWIPv6. The program used for the test is the NAS example of the previous section. The test set includes the measure of the bandwidth for file transfers of 10MB, 20MB and 40MB between processes running on the same host, on hosts connected by a 100Mb/s LAN and by a 1Gb/s LAN. The test environment consists of two GNU-Linux boxes (Debian SID distribution), Linux 3.2 kernel, NetXtreme BCM5752 controller, dual core Core2Duo processor running at 2Ghz, HP ProCurve Switches 1700 and 1810G. The files have been transferred using wget.

From the table and from the graph (Fig. 6) it is possible to see that IoTh can reach a sustained load of about 50MB/s, so the overhead added by the new approach is appreciable only on very fast communication lines. On a 100Mb/s LAN the difference is minimal. The improved performance for larger file transfers is caused by the constant startup cost (socket opening, http protocol, etc.), which is distributed on a longer operation. On localhost or on fast networks, the bandwidth of IoTh is about a quarter to a half of the bandwidth reached by the kernel.

It is worth considering that, in this test, both VDE and LWIPv6 run at user level. These are the performance values of the less efficient implementation structure of IoTh. Kernel level implementations of the TCP-IP stack library, and of the virtual networking switch engine, will increase the performance of IoTh.

X. CONCLUSION AND FUTURE WORK

IoTh opens up a range of new perspectives and applications in the field of Internet Networking.

IoTh unifies the role of networking and IPC, so it can play an important role in the design of future applications: distributed applications and interoperating processes can use the same protocols to communicate.

The research on IoTh is currently working in two different directions: the design of new IoTh applications and the evolution of the infrastructures to support IoTh.

The challenge of supporting new IoTh based services creates a need to analyze the TCP-IP protocols, in order to evaluate if and how these protocols, designed for physical networks, need to be modified or updated to be effective in IoTh. An example of a question that needs to be evaluated is whether the DNS protocol can have specific queries or features for IoTh.

On the other hand, IoTh requires an efficient infrastructure, able to provide a virtual networking (Ethernet) service to processes. This support must be optimized by integrating the positive results of currently available projects and then extending them to provide new services. For example, the speed of Vale [20] should be interfaced with the flexibility of VDE. There are several features in use on real networks that could be ported on virtual networks, e.g., port trunking. The research should also consider new efficient ways of interconnecting the local virtual networks to provide a better usage of virtual links, both for efficiency and for fault tolerance.

All the software presented in this paper has been released under free software licenses and has been included in the Virtual Square tutorial disk image [39]. This disk image can be used to boot a Debian SID GNU-Linux virtual machine. All the software tools and libraries used in this paper have already been installed and the source code of everything not included in the standard Debian distribution is also available in the disk image itself.

ACKNOWLEDGMENTS

I would like to express my gratitude to all the software designers and developers of the VirtualSquare Lab who have shared my daydreaming about the virtualization of everything, patiently following all my brainstorming. This paper is an extended version of [1], published by IARIA in the proceedings of The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013). I wish to thank all the anonymous reviewers of IARIA for their detailed comments and suggestions.

REFERENCES

[1] R. Davoli, "Internet of threads," in *ICIW 2013, The Eighth International Conference on Internet and Web Applications and Services*. IARIA, 2013, pp. 100–105.

[2] J. Postel, "DoD standard Internet Protocol," RFC 760, Internet Engineering Task Force, Jan. 1980, obsoleted by RFC 791, updated by RFC 777. [Online]. Available: http://www.ietf.org/rfc/rfc760.txt 02.25.2014

[3] F. Piedad and M. Hawkins, *High Availability: Design, Techniques and Processes*, ser. Harris Kern's Enterprise Computing Institute Series. Prentice Hall, 2000.

[4] P. McHardy *et al.*, "Netfilter," http://www.netfilter.org.

[5] D. Price and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads," in *Proceedings of the 18th USENIX conference on System administration*, ser. LISA '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 241–254.

[6] LXC team, "lxc linux containers," http://lxc.sourceforge.net/ 02.25.2014.

[7] IEEE and The Open Group, "Posix.1 2008," http://pubs.opengroup.org/onlinepubs/9699919799/ 02.25.2014.

[8] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC 2132 (Draft Standard), Internet Engineering Task Force, Mar. 1997, updated by RFCs 3442, 3942, 4361, 4833, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc2132.txt

[9] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462 (Draft Standard), Internet Engineering Task Force, Dec. 1998, obsoleted by RFC 4862. [Online]. Available: http://www.ietf.org/rfc/rfc2462.txt

[10] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022 (Informational), Internet Engineering Task Force, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

[11] K. Ashton, "That 'Internet of Things' thing," *RFID Journal*, vol. 22, pp. 97–114, 2009.

[12] M. Krasnyansky, "Universal tun/tap device driver," 1999, linux Kernel Documentation: Documentation/networking/tuntap.txt.

[13] J. D. Dike, "User-mode linux," in *Proc. of 2001 Ottawa Linux Symp. (OLS)*, Ottawa, 2001.

[14] R. Davoli, "Vde: Virtual distributed ethernet," in *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, ser. TRIDENTCOM '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 213–220.

[15] F. Bellard, "Qemu project," http://fabrice.bellard.free.fr/qemu/.

[16] A. Kivity, "kvm: the Linux virtual machine monitor," in *OLS '07: The 2007 Ottawa Linux Symposium*, Jul. 2007, pp. 225–230.

[17] Oracle inc., "Oracle vm virtualbox," https://www.virtualbox.org/.

[18] R. Davoli, "Inter process networking (ipn)," http://wiki.virtualsquare.org/wiki/index.php/IPN, 2007.

[19] Open vSwitch team, "Open vswitch," http://openvswitch.org/ 02.15.2014.

[20] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," University of Pisa, Italy, Tech. Rep., 2012. [Online]. Available: http://info.iet.unipi.it/∼luigi/papers/20120608-vale.pdf 02.25.2014

[21] L. Rizzo, "Revisiting network i/o apis: the netmap framework," *Commun. ACM*, vol. 55, no. 3, pp. 45–51, 2012.

[22] D. Ely, S. Savage, and D. Wetherall, "Alpine: A user-level infrastructure for network protocol development," in *Proc. of 3rd USENIX Symp. on Internet Technologies and Systems (USITS01)*, March 2001, san Francisco.

[23] A. Dunkels, "Full tcp/ip for 8-bit architectures," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 85–98.

[24] A. Dunkels, L. Woestenberg, K. Mansley, and J. Monoses, "Lwip," http://savannah.nongnu.org/projects/lwip 02.25.2014.

[25] R. Davoli, "Lwipv6," http://wiki.virtualsquare.org/wiki/index.php/LWIPV6 02.25.2014, 2007.

[26] K. Price, "Slirp, the ppp/slip-on-terminal emulator," http://slirp.sourceforge.net 02.25.2014.

[27] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[28] L. Gardenghi, M. Goldweber, and R. Davoli, "View-os: A new unifying approach against the global view assumption," in *Proceedings of the 8th international conference on Computational Science, Part I*, ser. ICCS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 287–296.

[29] A. Kantee, "Flexible operating system internals: The design and implementation of the anykernel and rump kernels," 2012, doctoral Dissertation, Aalto Univerisity, Finland.

[30] R. Davoli and M. Goldweber, "msocket: multiple stack support for the berkeley socket api," in *SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012, pp. 588–593.

[31] A. Pira, E. Tassi, and R. Davoli, "Managing User Level Networking-Personal IP networks," in *Proc. of ICN 04 - International Conference on Networking*, April 2004.

[32] R. Davoli, "Otip: One time ip address," in *ICSNC 2013, The Eighth International Conference on Systems and Networks Communications*. IARIA, 2013, pp. 154–158.

[33] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238 (Informational), Internet Engineering Task Force, May 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6238.txt

[34] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[35] R. Davoli, "Ipv6 hash-based addresses for simple network deployment," in *AFIN 2013, The Fifth International Conference on Advances in Future Internet*. IARIA, 2013, pp. 15–20.

[36] ——, "Vde sourceforge home page," http://vde.sourceforge.net.

[37] R. Davoli *et al.*, "Virtual square wiki," http://wiki.virtualsquare.org/.

[38] Fyodor Vaskovich (Gordon Lyon), "The art of port scanning," *Phrack*, vol. 7, no. 51, 1997.

[39] R. Davoli, "Virtual square tutorial disk image," http://wiki.virtualsquare.org/wiki/index.php/Virtual_Square_Tutorial_Disk_Image 02.25.2014.