Internet of Threads

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 Internet was designed to connect computers or, more precisely, networking controllers. In fact, IP addresses were assigned, and most of the time are still assigned, to the hardware interfaces [1].

In a typical situation when a client application wants to connect to a server daemon it first uses a Domain Name Server (DNS) to resolve a logical name of the server to an IP address. Usually the DNS maps the logical name, which is a readable specification of the required service (e.g., www.whitehouse.gov or ftp.ai.mit.edu), to the IP address of a hardware network controller of a computer able to provide the requested service.

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 most of the time 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).
- Each interface can be assigned several IP service oriented addresses. 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. Services can be assigned to a specific process using the *bind* system call.
- Linux Containers (LXC), as well as Solaris Zones [2], [3], 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.

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 discusses the security issues reated to IoTh and Section VII provides some performance figures of a proof-of-concept implementation. The paper ends with some final considerations about future work.

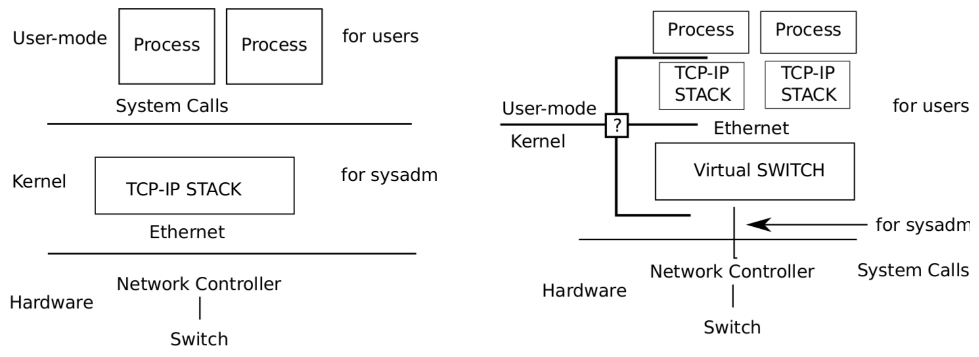

Fig. 1. Different perspectives on the networking support: the standard OS support is on the left side, IoTh is on the right side

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 rlevant 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 [4], 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 stack, 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.

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 to assign IP addresses to processes, to use IPv6 stateless autoconfiguration, to route packets using NAT, to implement packet filtering gateways, etc.

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. 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 network 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.

By IoTh, TCP-IP networking also becomes an Inter Process Communication (IPC) service. 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 use Internet protocols as its user interface. This means, for example, that it is possible to create programs which register their IP addresses in a dynamic DNS, and which have their web user interface accessible through a standard browser.

IoTh is, from this perspective, the software counterpart of Internet of Things (IoT [5]). In IoT hardware gadgets are directly connected to the network. They interact between objects and with the 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.

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

- VDE [6] is a general purpose, distributed support for virtual networking. A vde_switch is the virtualized counterpart of an Ethernet switch. virtual machines (or virtual appliances) can be connected to a vde_switch using the vde_plug library. Remote vde_switches can be connected together to form extended LANs. 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 provides support for VLANs, fast spanning trees for link fault tolerance, remote management of switches, etc.
- OpenVswitch [7] 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 [8] is a very fast support for virtual networking, based on the netmap [9] 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. Adam Dunkels wrote two general purpose and free licensed TCP-IP stacks for embedded systems: uIP [10] and LWIP (Light Weight IP) [11]. 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 [12]. 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 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).

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[13], 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[14] 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. 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[15]. 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.

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 Socket API, msocket[16], providing the support for multiple protocol stacks for the same protocol family.

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 the Technical Report [17].

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.

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. 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.. deamons, each one using its own stack. All the deamons will use their standard port numbers.
- Service migration in IoTh is as simple as stopping the daemon process on one host and starting it on another one. In fact, a deamon 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 have the same untagged VLAN).
- With IoTh it is possible to design network daemons which change their IP addresses in a dynamic way. One Time IP address (OTIP) applies to IP addresses the same technique used for passwords in One Time Password (OTP) services. 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. OTIP uses the same concept to protect private services accessible on the Internet. A deamon 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. A public demo of OTIP was given during FOSDEM2012[18].

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 testing 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. 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. The interaction between

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

processes connected to a VDE and the other networks (or the entire Internet) can be regulated by specific configurations of the virtual routers used to interconnect that VDE.

It is also possible to consider the positive effects of IoTh with respect to protection from external attacks. Port scanning [19] 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 deamons 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 deamons 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 damages it may cause in cases when the daemon is compromised (e.g., by a buffer overflow attack).

VII. 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 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 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

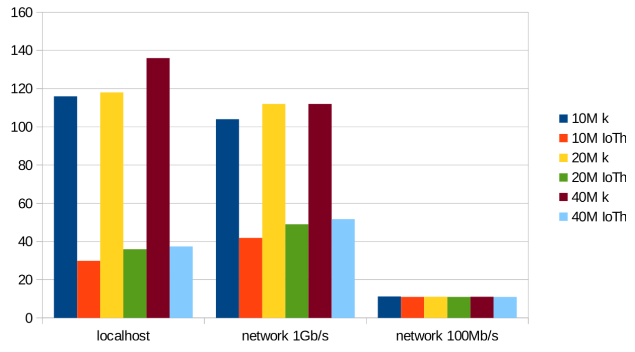

Fig. 2. A graphical view of Table I data

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 of Fig. 2 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.

VIII. 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 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. 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 [20]. 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.

REFERENCES

[1] 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 04.15.2013

[2] 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.

[3] LXC team, "lxc linux containers," http://lxc.sourceforge.net/ 04.15.2013.

[4] IEEE and The Open Group, "Posix.1 2008," http://pubs.opengroup.org/onlinepubs/9699919799/ 04.15.2013.

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

[6] 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.

[7] Open vSwitch team, "Open vswitch," http://openvswitch.org/ 04.15.2013.

[8] 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 04.15.2013

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

[10] 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.

[11] A. Dunkels, L. Woestenberg, K. Mansley, and J. Monoses, "Lwip," http://savannah.nongnu.org/projects/lwip 04.15.2013.

[12] R. Davoli, "Lwipv6," http://wiki.virtualsquare.org/wiki/index.php/LWIPV6 04.15.2013, 2007.

[13] 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, pp. 455–462.

[14] 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, 2008, pp. 287–296.

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

[16] 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*, 2012, pp. 588–593.

[17] R. Davoli, "Internet of threads, technical report," http://www.cs.unibo.it/∼renzo/iothtr.pdf 04.15.2013, Computer Science and Engineering Department. University of Bologna, Tech. Rep., 2013.

[18] ——, "Video of the public "internet of threads" demo, fosdem 2012," http://video.fosdem.org/2012/maintracks/k.1.105/Internet_of_Threads.webm 04.15.2013, 2012.

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

[20] R. Davoli, "Virtual square tutorial disk image," http://wiki.virtualsquare.org/wiki/index.php/Virtual_Square_Tutorial_Disk_Image 04.13.2013.