Quantitative evaluation of Cloud-based network virtualization mechanisms for IoT

Giovanni Merlino
Università degli Studi di Messina, Italy
gmerlino@unime.it

Francesco Longo
Università degli Studi di Messina, Italy
flongo@unime.it

Salvatore Distefano
Università degli Studi di Messina, Italy
sdistefano@unime.it
Kazan Federal University, Russia
sdistefano@kpfu.ru

Dario Bruneo
Università degli Studi di Messina, Italy
flongo@unime.it

Antonio Puliafito
Università degli Studi di Messina, Italy
apuliafito@unime.it

ABSTRACT

Integration of the Internet of Things (IoT) with the Cloud may lead to a range of different architectures and solutions. Our efforts in this domain are mainly geared towards making IoT systems available as service-oriented infrastructure. Under Infrastructure-as-a-Service (IaaS) scenarios, network virtualization is a core building block of any solution, even more so for IoT-focused Cloud providers. Enabling mechanisms are required to support virtualization of the networking facilities for IoT resources that are managed by the Cloud. This work describes an approach to network virtualization based on popular off-the-shelf tools and protocols in place of application-specific logic, acting as a blueprint in the design of the Stack4Things architecture, an OpenStack-derived framework to provide IaaS-like services from a pool of IoT devices. We quantitatively evaluate the underlying mechanisms demonstrating that the proposed approach exhibits mostly comparable performance with respect to standard technologies for virtual private networks, or at least good enough for the kind of underlying hardware, e.g., smart boards, whilst still representing a more flexible solution.

Keywords

IoT; Cloud; OpenStack; network virtualization; VPN; reverse tunneling; performance evaluation

1. INTRODUCTION

In a typical Infrastructure-as-a-Service (IaaS) Cloud, users are able to create and bring up virtual machines (VMs), access the instances through ssh, VNC, or Web-based virtual console as well as to instantiate even topologically complex virtual networks among a set of VMs. Any solution meant to take advantage of IoT resources available outside the datacenter, if those are to be provided according to IaaS principles, would need to implement at least similar facilities.

In a heavily distributed ecosystem, such as IoT-related scenarios, many requirements diverge significantly in comparison to typical IaaS Cloud environments, such as the presence of nodes installed behind firewalls and/or NATs (especially when IPv6 deployments are not an option) or, more in general, the necessity to deal with any restricted environment with denied-by-default (institutional or corporate) security policies. Such constraints call for more powerful mechanisms to enable core functionalities for virtual infrastructure management, i.e., remote access to board-hosted resources and instantiation of virtual networks.

Any network virtualization [2],[5] mechanism for IoT infrastructure thus requires at least some form of reconfiguration capabilities for board-side networking facilities as well. Yet, in contrast to typically datacenter-oriented IaaS, the physical environment (cabling and media access setup, logical topologies and hierarchies, role allocation for equipment) in IoT scenarios is not always under control of the designer of the infrastructure, which may as well be opportunistically assembled, e.g., volunteer-contributed.

This paper describes a rationale and some mechanisms to enable such functionalities when dealing with the unique requirements and challenges dictated by IoT environments, e.g., embedded boards and other constrained devices. In particular, network virtualization is addressed here to provide suitable facilities on top of Cloud-managed IoT resources in a technology agnostic fashion, still taking into account the limitations of smart devices, while at the same time suitable to be mapped onto an IaaS-focused solution, as investigated in [3] in terms of a device-centric approach for sensor-hosting nodes.

2. STACK4THINGS

In our infrastructure-oriented approach we envisioned a *virtualization* layer for boards, able to provide access to I/O pins, such as GPIO, as a Service through RESTful interfaces, and to send (predefined or custom) commands to the envi-

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266600

ronment running on the board. Here, we propose our implementation of such an approach, namely the Stack4Things [7] framework, from now on referred to as S4T, designed by extending the relevant OpenStack subsystems to smoothly integrate and leverage as much existing functionalities as possible.

In terms of hardware options with regard to boards, we confine the discussion on purpose to relatively smart embedded devices, such as modern Raspberry PI or Arduino-based hybrid systems, hosting either kind of (low power) microprocessor (MPU) and/or micro-controller (MCU) units. Such kind of boards is suited to host a minimal Linux distro, e.g., OpenWRT, able then to host a number of tools and runtime environments, such as Node.js or Python, where the *S4T lightning-rod*, the node-side component of the S4T architecture, runs on the MPU and interacts with the OS tools and services of the board, and with sensing and actuation resources through I/O pins. It represents the point of contact with the Cloud infrastructure. This is ensured by a WAMP and WebSocket-based communication between the S4T lightning-rod and its Cloud counterpart, namely the *S4T IoTronic service*, which is characterized by the standard architecture of an OpenStack service.

The main goals of IoTronic lie in extending the OpenStack architecture towards the management of sensing and actuation resources, i.e., to be an implementation of the SAaaS paradigm. However, in this work we focus on the mechanisms providing network virtualization facilities as described in the following section.

3. NETWORK VIRTUALIZATION FOR IOT

Our approach to network virtualization is based on enabling mechanisms in terms of custom layering and board-side tunneling facilities, to be coupled with the corresponding Cloud-side adaptations.

3.1 Tunneling and layering model

As remote infrastructure, boards are possibly going to be available over very restrictive, IPv4-only deployments. The only assumption that can (for all purposes, always) be considered true is outgoing Web traffic being permitted, i.e., board-initiated communication over standard HTTP/HTTPS ports. We thus resorted to an HTTP-borne mechanism for bidirectional connectivity and reachability of internal services, namely WS. WebSockets [4] as channels between a browser and a server are considered standard facilities for bidirectional communication and in particular server-pushed messaging. One of the main advantages of WS is that it is network agnostic, by just piggybacking communication onto standard HTTP interactions. This is of benefit for those environments which block Web-unrelated traffic using firewalls. Less explored is the creation of generic TCP tunnels over WS, a way to get client-initiated connectivity to any server-side local (or remote) service.

In early work [9] we devised a design and implementation of a novel *reverse* tunneling technique, as a way to provide server-initiated, e.g., Cloud-triggered, connectivity to any board-hosted service, or any other node on a contributed resource network, e.g., a WSN. In particular the latter may enable typical IoT scenarios, e.g., Machine-to-Machine (M2M) interactions, by supporting these patterns in a device-centric [3] fashion. This way, a gateway acting not only as a proxy for access to data gathered from mostly passive resources, but also as a relay to activate remoting toward nodes in a masqueraded network is available, thus allowing to explore options beyond protocols such as Traversal Using Relays around NAT (TURN) [8].

3.2 Server-less mesh implementation

Thank to the availability of an always-on WAMP control channel, the above mentioned scenario can also be reproduced in a fully peer-to-peer fashion. In such a case, one board, out of the set of IoT devices that need to be connected over a certain virtual network, is selected by the Cloud to act as a central node in a star topology, according to a ranking for suitability in taking that role, on the basis of current availability of on-board resources and load. An ad-hoc variation of the TCP *hole punching* [13] technique, in which the Cloud would preliminarily act as the required third party in the phase of connection establishment, is exploited to allow all the other devices belonging to the virtual network to connect to the chosen central node by establishing WebSocket-based reverse tunnels. On top of such a peer-based WebSocket-based infrastructure, the virtual network is built, in a star topology. Once the virtual network is formed, network communication is fully peer-to-peer and does not require any further action from the Cloud until the (virtual) network needs to be torn down. Moreover, other topologies may be enabled as well, such as a full-mesh at layer 2 among all the boards selected to be interconnected over a Cloud-initiated virtual network.

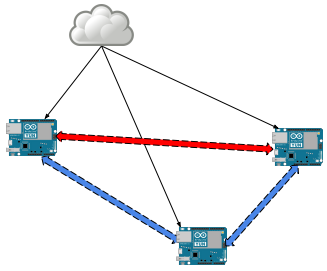

Figure 1: Cloud-enabled server-less star and mesh topologies.

In Figure 1 a simple 3-node network (plus the Cloud) is depicted, where the thick double-sided blue arrows correspond to the star topology, when the node in the middle of the figure is chosen as the center of the star, whilst a similar arrow in red stands for the missing link among peers to establish a full mesh. The thin single-sided black arrow represents both (transient) WS-based connection establishment, and just steady command streams afterwards, having the Cloud at that point already hole-punched the nodes and subsequently torn down its tunnels to those.

In that case the topology would be a tree over the set of bridges exposed by the boards, dynamically instantiated by means of, e.g., a protocol for automatic configuration of bridges belonging to the Spanning Tree family. In this case the virtual network would not feature any single point of failure, nor have a node be the bottleneck for all the generated traffic inside the network itself. On the other hand, this configuration requires a longer setup phase, due to the establishment of an higher number of tunnels, $n(n-1)/2$, in proportion to the number of nodes n, compared to the star topology, $n-1$, and more importantly a slightly higher

baseline in terms of requirements, as each node would host a virtual bridge, not just the central node.

4. A USE CASE

In order to test and validate the proposed solution, we developed experiments on a smart city environment. #SmartME [1] is a crowdfounding project that aims at collecting a number of field-deployed networks of sensors and actuators in Messina into a federated Smart City infrastructure, provided as-a-service [10], actually an instance of a so-called Software-Defined City [11]. Thanks to such an infrastructure, it will be possible to collect data and information to build services for citizens, who may take part in this network through the involvement of smartphones and other mobile devices by which it will be possible to interact with objects and may even themselves turn into data producers.

4.1 Description

For our network virtualization approach, #SmartME represents an interesting testbed and case study, thanks to a development, deployment and testing platform integrated with networks of sensors, actuators or other smart devices already deployed to the district area, and to this day employed exclusively for specific application domains and a limited number of purposes.

The virtual networking subsystem has been put to test in this context by exploiting the #SmartME testbed, considering two main scenarios, one based on clients (e.g., boards) behind residential xDSL gateways, e.g., at home, and another one based on clients belonging to the (Fast) Ethernet-switched network of the university campus, respectively.

The first one is thus about connectivity scenarios over a Wide-Area Network, whereas the second one over a LAN, respectively.

4.2 Quantitave evaluation

Here we provide an evaluation focusing on specific key performance indices, namely latency and throughput.

Table 1 and Table 2 report on the set of experiments that have been conducted. The experiments are based on the *iperf3* [14] tool for measuring throughput and the ubiquitous *ping* tool to gauge latency, the latter by means of ICMP echo requests to obtain Round-Trip Time values as estimation of delay. In both cases, the test setup consists in having a server ready, and generating traffic over TCP (iperf) or ICMP requests (ping) from the client, which collects partial and final statistics. Values in the table are averages computed over a number of 1000 samples, where each chunk of 10 samples represent a single run for the tool. We did not include variance values because they are negligible.

The first column indicates the kind of technology employed and under which topology: in particular, *direct* refers to tests between two hosts directly connected, over WAN or LAN, respectively. The *vpn-p2p* abbreviation refers to an OpenVPN server to which an OpenVPN client is connected, under the same roles as the two aforementioned hosts. Same happens for *s4t-p2p*, in this case with two hosts controlled by the S4T Cloud which, by first hole-punching through in the setup phase, connects one to the other directly over a WebSocket tunnel in combination with SOCAT. In the *vpn-srv* and *s4t-srv* cases, the OpenVPN server and the S4T Cloud, respectively, enable two clients to act as the two hosts, connected over a bridge which resides on the server (or the Cloud).

For the sake of comparing under the most relevant conditions, OpenVPN has been tested in TCP mode, and various parameters (e.g., MTU of the TUN interfaces, MSS for both iperf and the TCP tunnels) had already been tuned for S4T in the implementation phases, as also discussed below. As latencies are absolutely aligned in the various scenarios, or at least very predictable (e.g., natural increase for multi-hop setups, such as server-based OpenVPN or S4T Cloud-based virtual network for two clients), we will focus the discussion on the more interesting and insightful values obtained for the throughput metric, albeit it is actually latency that may be considered the most relevant metric for the use case under consideration, as it is key for near real-time (e.g., multimedia) applications, and the most reliable metric in general for embedded systems, considering that throughput is naturally more susceptible to other factors, e.g., high CPU load or RAM usage, differences in the media interface, etc.

iPerf3 works by repeatedly sending an array of *len* bytes for *time* seconds, where *len* by default is 128KB for TCP, and the default for *time* is 10 seconds. Even if not reported in the table, retransmissions, equal to about 500 in case of SOCAT-based TUN-over-TCP, i.e., where the *s4t-p2p* scenario is stripped of the WS tunnel, in a 10-second period, ramp up to almost 5000 for the same time interval when also piping over WS, i.e., the full end-to-end *s4t-p2p*, thus roughly increasing retransmissions ten-fold. Throughput thus decreases correspondingly to about a tenth, from almost 90Mbps to about 10Mbps to be precise.

This degradation in the performance with regard to throughput derives from a limited CWND (Congestion Window) on average, less than 10KBytes, compared to 2-3M under ideal conditions, i.e., corresponding to about 90Mbps throughput as seen in the table. The problem then lies in a relatively high number of retransmissions, leading to a comparatively small congestion window. Some workarounds exist, either to lower retransmissions in this scenario, such as disabling Nagle's algorithm, or to make TCP more aggressively reactive to retransmissions, ramping up speed more quickly, such as switching to a different congestion control policy, in particular *Scalable* [6], both of which have been applied during testing.

Otherwise, it is intrinsic to how TCP works (i.e., the streaming model) the inability to fully control how TUN-inbound datagrams are encapsulated and delivered as payload in the data stream. This means that there are bound to be packet losses (and thus retransmissions) due, for instance, to the splicing of two (IP) datagrams as payload of a single (TCP) segment. When that happens, the payload cannot be passed without errors to the (outbound) TUN on the other side of the tunnel, which expects the delivery of single (whole) datagrams, and either rejects or truncates what it gets due to mismatches between the MTU and the size of such segments

In this sense, the presence of a tunnel based on WebSockets exacerbates the behaviour by introducing a further chance to introduce randomness in the delivery by virtue of TCP-level decapsulation and subsequent re-encapsulation, the latter as a result of the TCP channel established over WebSockets. This as a result of trading off raw performance, at the price of high application-level complexity and an (internal) ad-hoc architecture, as is the case for OpenVPN, with the simplicity and flexibility of off-the-shelf tools act-

Table 1: Throughput and latency measurements: xDSL.

Topology/Technology	iperf: *throughput [**Mbps**]*	ping: *latency (RTT) [**ms**]*
direct	0.993	85.2
vpn-p2p	0.964	88.95
vpn-srv	0.811	136.8
s4t-p2p	0.405	83.38
s4t-srv	0.335	123.6

Table 2: Throughput and latency measurements: campus network.

Topology/Technology	iperf: *throughput [**Mbps**]*	ping: *latency (RTT) [**ms**]*
direct	92.5	0.23
vpn-p2p	88.02	0.725
vpn-srv	81.75	1.501
s4t-p2p	10.65	1.813
s4t-srv	4.017	3.579

ing as separate subsystems and taking care of different facets of the communication model, in line with the UNIX philosophy of using one (good) tool for each job.

Fortunately results are still absolutely decent, and ensure the viability of such a solution for embedded systems where the throughput of any modular solution, when fully in userspace, is typically lower anyway, since it may be capped by the CPU maxing out, as is the case for the Node.js-based reverse tunneling, possibly due to the current limitations of the V8 engine under MIPS.

When considering the case of WAN-level connectivity, e.g., featuring significantly lower bandwidth and naturally higher delay, it may be noticed that throughput for the S4T-based setups degrades less sharply, only down to about 50%, as can be seen in Table 1. Moreover, there is room for improvement considering that, compared to plain TCP sockets, WebSockets support delimiting payloads according to a specific *message*-oriented semantics for delivery, a facility which may indeed be exploited in this very sense.

5. CONCLUSIONS

Our opinion is that a novel approach is required for IoT and Cloud integration, and models and mechanisms which are agnostic to field deployments and topologies are essential to IoT infrastructure management and service provisioning.

Our approach to Cloud-enabled virtual networking for IoT tries to provide a blueprint for a combined solution, where VPN-like behavior, albeit the most easily advertised functionality and the one most easily picked up for a comparison, is actually just one out of many features. Other useful ones include always-on centralized control by means of WS-based command streams, bypassing restrictive firewall policies by piggybacking onto HTTP, relaying traffic through the Cloud for NAT traversal, Cloud-initiated hole punching to support server-less star and tree topologies for peer-to-peer networks, or even exposing internal services through reverse tunneling.

Moreover, functional requirements aside, performance has been shown to be mostly comparable and in all cases absolutely acceptable considering inherent limitations of the hardware platforms under consideration.

In terms of performance, the aforementioned considerations about the results will be the starting point for further improvements to the design, in particular resorting to facilities such as the WebSockets message-based semantics to partly overcome limitations intrinsic to the communication model, or even modifying some of the system-level tools, e.g., SOCAT, by employing low-overhead simple bytestream-oriented protocols on top of TCP, such as SLIP [12], to mark packet boundaries within the (TCP) payload.

6. REFERENCES

[1] D. Bruneo, S. Distefano, F. Longo, and G. Merlino. An IoT testbed for the Software Defined City vision: The #SmartMe project. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, May 2016.

[2] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[3] S. Distefano, G. Merlino, and A. Puliafito. Device-centric Sensing: an alternative to data-centric approaches. *IEEE Systems Journal*, 2015.

[4] I. Fette and A. Melnikov. The WebSocket protocol. RFC 6455.

[5] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: a survey. *Communications Surveys Tutorials, IEEE*, 15(4):1888–1906, Fourth 2013.

[6] T. Kelly. Scalable TCP: improving performance in highspeed Wide Area Networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, Apr. 2003.

[7] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito. Stack4Things: a Sensing-and-Actuation-as-a-Service framework for IoT and Cloud integration. *Annals of Telecommunications*, pages 1–18, 2016.

[8] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around Nat (turn): Relay extensions to Session Traversal Utilities for Nat (stun). RFC 5766.

[9] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito. Enabling mechanisms for Cloud-based network virtualization in IoT. pages 268–273, 2015.

[10] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito. Stack4Things: Integrating IoT with OpenStack in a Smart City context. 2015.

[11] G. Merlino, D. Bruneo, F. Longo, A. Puliafito, and S. Distefano. Software Defined Cities: a novel paradigm for Smart Cities through IoT clouds. pages 909–916, 2015.

[12] J. Romkey. Nonstandard for transmission of IP datagrams over serial lines: SLIP. STD 47, RFC Editor.

[13] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) communication across Network Address Translators (NATs). RFC 5128.

[14] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. iPerf: the TCP/UDP bandwidth measurement tool. *http://software.es.net/iperf/*, 2005.