The Network as a Computer with IPv6 Segment Routing: a Novel Distributed Processing Model for the Internet of Things

Andrea Mayer
Univ. of Rome Tor Vergata
andrea.mayer@uniroma2.it

Emanuele Altomare
Univ. of Rome Tor Vergata
altomare.emanuele@gmail.com

Stefano Salsano
Univ. of Rome Tor Vergata
stefano.salsano@uniroma2.it

Francesco Lo Presti
Univ. of Rome Tor Vergata
lopresti@info.uniroma2.it

Clarence Filsfils
CISCO Systems
cfilsfil@cisco.com

ABSTRACT

In this position paper, we propose a novel distributed processing model for the IoT, which improves the integration between IoT and Cloud. We give IoT developers an abstract machine that can be programmed using the Instruction Set Architecture of the AVR microcontrollers (the ones used in the Arduino). This abstract machine can be programmed by developers to access, following an I/O port driven approach, any 'thing' that is connected to the network. The distributed processing model is based on the extension of the network programming model offered by the Segment Routing protocol for IPv6 (SRv6). In the proposed model, the integration between IoT and Cloud is enhanced and simplified, reducing the need to split the applications in order to execute components into the Cloud or into the network edges. Thanks to the possibility to distribute the processing locally in the 'things', the new approach can be exploited to minimize latency, as well as to reduce the amount data to be exchanged between 'things' and Cloud.

CCS CONCEPTS

• **Networks** → **Programmable networks**; *In-network processing*;
• **Computer systems organization** → Sensors and actuators.

KEYWORDS

Segment Routing, SRv6, network programming, IoT, Cloud

1 INTRODUCTION

The Internet of Things (IoT) [10] is now moving from theory to reality. Billions of smart devices ('things') are being deployed and connected. The exchange and processing of data originated by these devices are a burden for the Cloud and for the network infrastructure that interconnects 'things' with each other and with the core of Internet. To address such problems *Smart Gateways* [11] and, more in general, the *Fog computing* [11], [7] concept play a key role in the overall scenario. Indeed, letting the data being processed near 'things' allows compressing and trimming useless information before sending them to the Cloud. Smart Gateways are a way of offloading part of the processing operations directly at the edges of the network. Anyway, Cloud and Smart Gateways have to be coordinated in order to decide how to slice processing units.

NGOSCPS 2019, April 15 2019, Montreal, Canada

In this position paper we advocate the introduction of a novel processing model that aims to improve the interaction and integration between IoT and the Cloud. We consider an Internet of Things relying on IPv6 networking. The proposed model is based on two main concepts: i) every IPv6 addressable device (things, network devices, hosts) can potentially execute operations that can be chained in a *network program*; ii) the network program and the execution state is carried in the IPv6 packets using and extending the SRv6 (IPv6 Segment Routing) *network programming model* [6]. We will refer to our proposed approach as SR-IoT.

IPv6 is considered as the network layer protocol that will support the IoT, thanks to its huge addressing space, its security enhancements and the possibility of direct communications without the limitation of NAT (Network Address Translation) needed for IPv4 [4]. In fact, IPv6 provides a convergence layer for IoT [8]. Our solution obviously relies on IPv6 and in addition it leverages a proposed extension called IPv6 Segment Routing (SRv6) [13], [12] which allows adding a sequence of *segments* in the IPv6 packet headers to influence the packet forwarding and processing within the network. The Segment Routing is a form of loose source routing, in which the originator of a packet can include a set of 'waypoints' (segments) in the packet header. The networking layer will route the packet through the sequence of waypoints until the final destination. In the SRv6 architecture the segments are expressed as IPv6 addresses. According to [6] the segments (IPv6 addresses) can represent instructions and not only network locations. Thanks to the huge IPv6 addressing space, it is actually possible to define the location and the instruction using a single segment/IPv6 address. In fact, an IPv6 address is 16 bytes (128 bits) long and typically only the leftmost 8 bytes are used to route the IPv6 packet toward a given node interface, representing the *locator* part of the address. The rightmost 64 bits are normally used to address the interface of an IPv6 host over a subnet. Hence, it is possible to use the rightmost 64 bits to define different instructions that need to be executed on a packet when a segment is reached. With Segment Routing, a sequence of locations/instructions can be added by the source to an IPv6 header, defining a *network program* to be applied to the packet. This concept is referred to as the *network programming model*. The instructions that are defined in [6] represent network level operations that can be applied to packets, like for example forwarding over a given output interface, encapsulating or decapsulating the packet, interacting with Operation and Management (OAM) functions in a network node. Our proposed approach extends this model by considering as instructions generic processing operations like

the ones defined in the instruction set of a computing architecture. Moreover the extended SRv6 packets do not only carry the program (i.e. the sequence of instructions) but also the runtime-context of a program such as registers, memory, etc. So our SR-IoT packets are fully-fledged processes (called also SR-IoT processes), the same that we can expect to be available in a simplified OS.

In this vision, IoT, Cloud and Fog computing become part of an large distributed processing infrastructure. Processes are not tied and executed by single processing units (i.e: inside the Cloud or within a Smart Gateway) but they can move from one unit to another accordingly to their own execution flows.

The key factor that makes our approach different from other distributed execution models known in the literature such as opportunistic offloading of code, RPC, etc. is that 'processes' (i.e. programs and execution state) are encoded within IPv6 packets. We are not focusing on the efficiency in supporting the parallel execution over multiple nodes of processes that needs to exchange information. Our goal is to interact with multiple 'things', performing read and write operations and processing the collected data, minimizing the need of exchanges between the 'things' and the cloud. For these goals, we believe that embedding the program and the execution context in the network packet can be beneficial. Our proposal bear some similarity with the Active Networks concept [9], which was a popular research trend in the late '90s.

The content of this paper is as follows. Section 2 formalizes our novel and alternative distributed processing model. The proposed design and implementation of our first SRv6-Computing machine is described in section 3. Finally, we draw some conclusions in section 4.

2 DISTRIBUTED PROCESSING MODEL BASED ON SRV6

Because 'things' could be anything and placed anywhere, they are often handled by gateways that interconnect them directly to the Internet using IP protocol. Gateways could be 'dumb' and offer no additional computing capabilities or they can be 'smart'. 'Smart Gateways' offer some extra features for data processing before sending it to the Internet and then to the Cloud. In this way, an IoT application can rely also on some intelligence available at the edge of the network. To cope with this processing model, a IoT cloud application has to be decomposed into multiple software components and the developer has to decide where to run them. Our alternative distributed processing model aims to simplify the design of applications that deal with the IoT world. Developers do not have to split applications into Cloud and Edge processing parts anymore. Instead, we offer to IoT application developers an abstract computing machine (made of Cloud, Smart Gateways, IoT and Networks) named SR-IoT Computing Machine (SR-IoT CM) that can be programmed with its own instruction set. This abstract CM has a set of virtual I/O *ports* which represent 'things', network devices and hosts that are, directly or indirectly, interconnected to an IPv6 network. The mapping between the physical entities and the abstract CM is shown in Fig. 1. In the following subsections we give a big picture of the overall architecture of the SR-IoT CM and describe the tools and life-cycle of an SR-IoT process.

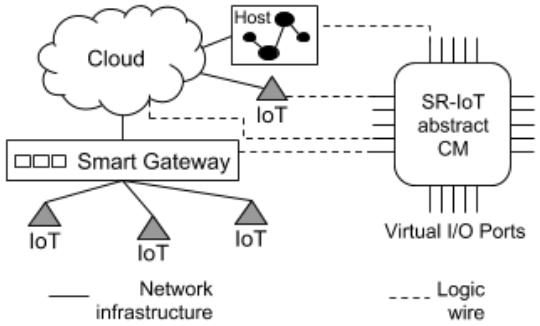

Figure 1: Mapping between physical devices and the abstract SR-IoT CM using virtual I/O ports.

2.1 Architecture and instruction set of the SR-IoT Computing Machine

The SR-IoT CM appears to developers as a single computing machine even though it leverages multiple computing units that can be spread all over the network. Computing units can be associated to any IPv6 addressable device: things (IoT device), network nodes, mobile and fixed hosts. In this scenario, each IPv6 node can also be seen like a proxy that is in charge of receiving the process embedded in the IPv6 packet and offloading the workload to its computing unit. The fundamental abstraction that relates the single computing machine with the physical computing units is the I/O *port*. The ports of all physical nodes that need to take part to a given computation task (process) are numbered in the same virtual numbering space. So from the point of view of the developer these ports belong to the same abstract computing machine. When the computation task will be compiled into a network program the virtual port numbers will be resolved into the physical computing unit addresses (IPv6 addresses) and physical port number.

In principle, every instruction of our SR-IoT CM is composed of i) a locator code and ii) operation code (Fig. 3-a). The operation code of each instruction has to be locally executed by a computing unit, while the the locator part is meant to be used by the network to deliver the packet to an IPv6 node. This means that also the network takes part in the execution of the program, by routing a packet (process) from a IPv6 node to the next one on the basis of the active segment specified into the SRv6 header (SRH). In this way the network allows executing the instructions according to the order in which they are specified into the program. While this functionality could have been implemented at application level, it is very convenient to have a standard implementation of SRv6 in the networking level. Such implementation avoids redesigning and implementing the needed features, it ensures interoperability among 'things' and with network devices and in general it is more efficient to perform such functions in the networking level. From the implementation point of view, we can reuse the Linux implementation of SRv6 as a fundamental component our prototype.

The actual coding of the instructions will be optimized so that a sequence of multiple instructions that need to be executed on the same computing unit can be coded without repeating the locator part in the instruction code (Fig. 3-b). The generation and optimization of the sequence of instructions is performed by a tool chain that takes as input the code written by the application developer,

for example using the assembly language of the SR-IoT computing machine. The application developer will only need to deal with the application logic and with the virtual I/O port numbers of the 'things' he wants to access, and the tool chain will take care about the locator IPv6 addressing.

2.2 Tool chain

2.2.1 Compile, assemble and links. Given a program written using the operation instruction set, the first components of the tool chain compile, assemble and link it. The outgoing artifact of this phase consists of a binary file, called also Operation Binary Code (OpBC), which contains all the code that will be executed on computing units. However, OpBC can not be pushed into the network yet because instructions in OpBC are referred to the single abstract SR-IoT computing machine and they are not resolved into instructions that can be executed by the physical computing units.

2.2.2 Locator tool. The Locator is the component in charge of filling, according to a given strategy, the locator part of an instruction in an OpBC and of resolving the abstract I/O port numbers into the physical port numbers of the computing units. It carries out three tasks: i) it decides on which node the instruction will be forwarded and executed, ii) it encodes into the locator part of the instruction the address (location) of the node in the network, iii) it remaps the port numbers from virtual to physical ones. The Locator can accomplish task i) because of its knowledge of the resources and the network, i.e. the map of virtual port numbers into IPv6 addresses and physical port numbers. Indeed, this mapping could be managed by the owner of the Cloud infrastructure that offers the SR-IoT CM as a service. In this vision, the SR-IoT CM could be seen as a Platform as a Service (PaaS). The infrastructure owner is aware of the location of 'things' that can be reached through the network. As a result, he can map those 'things' with the network nodes to which they are linked to. Concerning tasks ii) and iii), for each instruction, the Locator encodes the IPv6 address of the network node directly into the locator part and properly remaps the port numbers that are contained as parameters in the instructions from the virtual port numbers to the physical port number. At this stage, the produced binary file is fine to be executed by the SR-IoT CM but it needs to be properly encoded into an SRv6 packet. We note that according to the format shown in Fig. 3-a, each instruction includes the locator of the node (e.g. an IPv6 prefix of 64 bits) and the instruction code itself that can be 16 bits or 32 bits.

2.2.3 SRv6 Packet builder. The last fundamental step that is required to turn an OpBC into an executable consists in encoding all the program instructions into an IPv6 (SRv6) packet. The simplest approach would be to encode each instruction as a segment (IPv6 address) into the Segment Routing Header (SRH) of the packet, directly using the format shown in Fig. 3-a. In this simple approach, the active segment points to the next instruction as well as to the destination address of the IPv6 packet. The network is able to route the packet and let it reach the target computing unit thanks to the locator part. If there are multiple instructions to be executed on a node, the node will recognize its own IPv6 address in the locator part of the following instructions and the instructions will continue to be processed locally. When a different IPv6 prefix is found in the

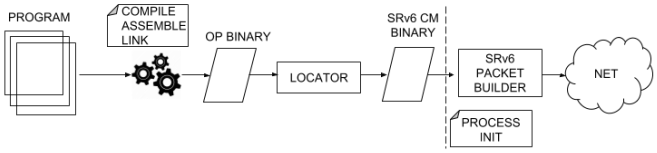

Figure 2: SR-IoT CM compilation and execution flow.

locator part of the active segment in the segment list, the packet will be forwarded to the next computing unit. The drawback of this simple approach is the waste of header space. In fact, an IPv6 address is 16 bytes (128 bits) long, and the locator part is typically an IPv6 prefix of 6 bytes (48 bits) or 8 bytes (64 bits), leaving 10 or 8 bytes for the instruction code. For example, if the instruction code is 2 bytes, 8 bytes or 6 bytes will be wasted. Assume that the operation codes are of 2 bytes (16 bits) and consider a program of 100 instructions. If each instruction is simply coded in an IPv6 address, 16 bytes will be used for each instructions and in total $100 * 16 = 1600$ bytes will be used to represent a program of 200 bytes. Therefore, the SRv6 packet builder uses the more compact representation shown in Fig. 3-b. Each IPv6 address can carry multiple instructions when a sequence of consecutive ones need to be executed in the same computing unit. In this encoding, a two-bytes field called INFO, is added in each IPv6 address to include the information needed to properly decode the instructions (e.g. the length of the Locator part and the number of instructions contained). With the optimized encoding, let us consider the program of 100 instructions, assuming that it needs to be executed in 4 computing units (25 instructions for each unit) and a locator of 6 bytes (48 bits) is used. The first IPv6 address will carry a locator of 6 bytes and 4 instructions of 2 bytes, the other 3 IPv6 addresses will carry 7 instructions and no locator: 25 instructions are encoded in 4 IPv6 addresses. In total we need $(4 * 4) * 16 = 256$ bytes to encode our program instead of 1600 bytes for the uncompressed case, with a compression factor of 6.25.

2.3 Process life-cycle

Any OpBC needs of its own runtime-context to be run on computing units. The runtime-context includes registers, stack as well as the memory that will hold all the information required for the program evaluation. Hence, the payload of any SRv6 packet that encodes an OpBC is also initialized with the suitable runtime-context. In this way, each program along with its own environment becomes a fully-fledged process that migrate from one node to another. Thereafter, the SRv6 packet is sent through the network to reach, as specified into the SRH, the first node.

The execution of the process begins as soon as the SRv6 packet arrives at the node whose address matches with the active segment. In this case, program and data are extracted and they are sent to the suitable computing unit. Once all the instructions (local to that unit) are performed, the node updates the payload of the packet with the current runtime-context that reflects the outcome of the operations. It increases also the `segment left` field in the SRH in the same way as a CPU increases its program counter. Afterwards, the network node sends the SRv6 packet to the next node (if any) and this procedure repeats again until the packet reaches the last

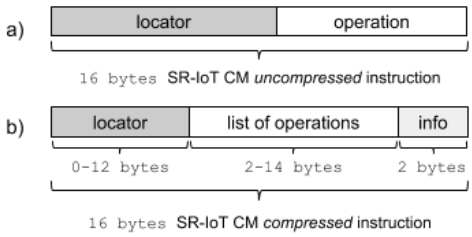

Figure 3: Encoding of SR-IoT CM instructions into IPv6 addresses.

one. In our model, the final node is always represented by a collector that allows us to gather the results of the program execution.

2.4 Feasibility considerations

Due to the way in which our SRv6 CM is implemented, the whole SR-IoT process, including the program instruction and the context, must fit into an IPv6 packet which, generally, does not exceed 1500 bytes. This represents the major limitation of our solution. Assuming 100 instructions over 10 nodes the program will be encoded with 320 bytes, leaving around 1100 bytes for the context, which includes the virtual RAM, CPU registers, stack.

3 DESIGN AND RESULTS

We decided to test our novel distributed processing approach trying to build a first prototype of both: i) tools needed to generate the OpBC and ii) a computing unit that is in charge of executing the binary code. The following of this section is divided into subsection 3.1 that aims to describe some key aspects about the tools we used and designed to generate the OpBC. Instead, in subsection 3.2 we give some insights about how we realized the computing units and their interaction with the network nodes which are in charge of intercepting SRv6 packets in the network.

3.1 Atmel AVR instruction set

During the design and implementation of the SR-IoT CM we decided not to 'reinvent the wheel', but we exposed a well-known ISA to the application developer: the Atmel AVR [2] instruction set. The choice to use this ISA was dictated by the simplicity of this architecture and by the familiarity that developers, who usually program 'things', have with this chip as it is integrated into most of the Arduino-like platforms. We leveraged the AVR-GCC toolchain [1] to generate the binary code starting from a program written in C or asm. After that, we wrote from scratch the Locator tool and SRv6 Packet Builder in order not only to craft SRv6 packets but also to initialize them properly with the suitable runtime-context and send them through the network.

3.2 SIMAVR computing unit

We decided to simulate the AVR μC through the SIMAVR [3], an open source simulator that represents our computing unit. We have patched SIMAVR in order to integrate it with the underlying layer, consisting of the software components that we developed (middleware).

The changes to the SIMAVR were mainly related to the control of the Program Counter (PC). As we mapped instructions on SRv6 segments, we always need to check if the next instruction pointed by the PC will be performed on the same computing unit or on a different one. Therefore, the modified SIMAVR asks the middleware if it has to execute the next instruction. If it receives a positive answer, it proceeds, otherwise it immediately stops execution by returning control to the middleware. The middleware modifies the SRv6 packet updating the active segment and the IPv6 destination address and transmits it on the network, in order to allow the execution of the process to continue through the infrastructure. When a packet is received from the network layer, it is passed to the middleware that takes care of i) adapting the contents of the packet to run on the SIMAVR emulator and ii) starting computation.

4 CONCLUSIONS

The novelty of our approach to network programming resides in the fact that we modeled a complex infrastructure as a single logical machine. This is going beyond the Fog computing paradigm where there is a marked difference between 'edge' (IoT) and 'core' (Cloud) nodes. Indeed, in our computing model we offer to IoT developer the transparency of the location of 'things' that are accessible through the abstraction of virtual of I/O ports. The developer only needs to code the application logic without worrying about how the code will be executed on the network and computing units.
Thanks to Locator component, the owner of the infrastructure can enforce its management policies. Therefore, the interests of the major stakeholders can be satisfied: i) the Service Provider (SP) who wants to provide a service and ii) the Infrastructure Owner who wants to manage and maintain the infrastructure.

An added value, in our opinion, is the usage of the emerging technology like SRv6 to execute the sequence of instructions that compose the programs, transporting the execution status from node to node. SRv6 is a new standard whose implementation in Linux was introduced since the kernel version 4.10 and it has been already implemented in several systems [5]. This will allow to achieve good performance and also to simplify the integration of our solution in existing environments.

REFERENCES

[1] 2019. AVR-GCC. https://gcc.gnu.org/wiki/avr-gcc/
[2] 2019. AVR microcontrollers. https://en.wikipedia.org/wiki/AVR_microcontrollers
[3] 2019. SIMAVR. https://github.com/buserror/simavr
[4] B. Ray. 2015. *3 Reasons Why IPv6 Is Important For the Internet Of Things.* Blog post. https://www.link-labs.com/blog/why-ipv6-is-important-for-internet-of-things
[5] C. Filsfils *et. al.* 2019. *SRv6 interoperability report.* Internet-Draft. IETF. https://tools.ietf.org/html/draft-filsfils-spring-srv6-interop
[6] C. Filsfils, P. Camarillo (ed.) et al. 2019. *SRv6 Network Programming.* Technical Report. https://tools.ietf.org/html/draft-filsfils-spring-srv6-network-programming
[7] Sheng Wen Ivan Stojmenovic. 2014. The Fog computing paradigm: Scenarios and security issues.
[8] Mark Kelly Keith Nolan. 2018. IPv6 Convergence for IoT Cyber-Physical Systems.
[9] K. Psounis. 1999. Active networks: Applications,Security, Safety, and Architectures. *IEEE Communications Surveys* 2, 1 (1999), 2–16.
[10] L. Atzori, A. Iera, G. Morabito. 2010. The Internet of Things: A survey. (2010).
[11] Eui-Nam Huh Mohammad Aazam. 2014. Fog Computing and Smart Gateway Based Communication for Cloud of Things.
[12] S. Previdi (ed.), C. Filsfils (ed.) et al. 2018. Segment Routing Architecture. IETF RFC 8402. https://tools.ietf.org/html/rfc8402
[13] S. Previdi (ed.) et al. 2016. IPv6 Segment Routing Header (SRH). Internet-Draft. http://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-02