# A Simple Solution for IoT Experimentation in the Context of Future Internet Architectures

**Ramon P. S. Chaib**[1], **Antonio M. Alberti**[1]

[1]ICT Laboratory, Instituto Nacional de Telecomunicações
INATEL - João de Camargo 510, Centro, Santa Rita do Sapucaí
CEP 37540-000, Minas Gerais, Brazil. Phone: +55 35 3471 9218

ramonp@gec.inatel.br, alberti@inatel.br

***Abstract.*** *In the last decade, many approaches appeared to revolutionize Internet architecture from scratch. They are collectively called Future Internet Architectures. In this paper, we address the challenge of experimenting with Internet of things in this context. Using open-source tools, we developed a standard scenario capable to evaluate proposals in a small scale first (locally, in laboratory), and latter in large scale cloud-based testing platform. As a use case of the proposal, we discuss its application to the eXpressive Internet Architecture.*

## 1. Introduction

This work proposes a solution for testing IoT over future Internet architectures (FIAs) [1]. The aim is to test how the architecture behaves while interoperating with motes running Contiki [2], an open source operating system for the Internet of Things. Using Cooja, a network simulator that allows networks of Contiki motes to be simulated even in a hardware level [2], and Docker [3], a software container platform, we created a standard evaluation scenario to compare distinct Internet architectures with minor implementations of components for each of them. Unlike virtual machines (VMs), containers do not bundle a full operating system, they initialize only libraries and settings required to make the software work as needed. This enables efficient, lightweight, self-contained systems that guarantee software will always run the same, regardless of where it is being deployed [3].

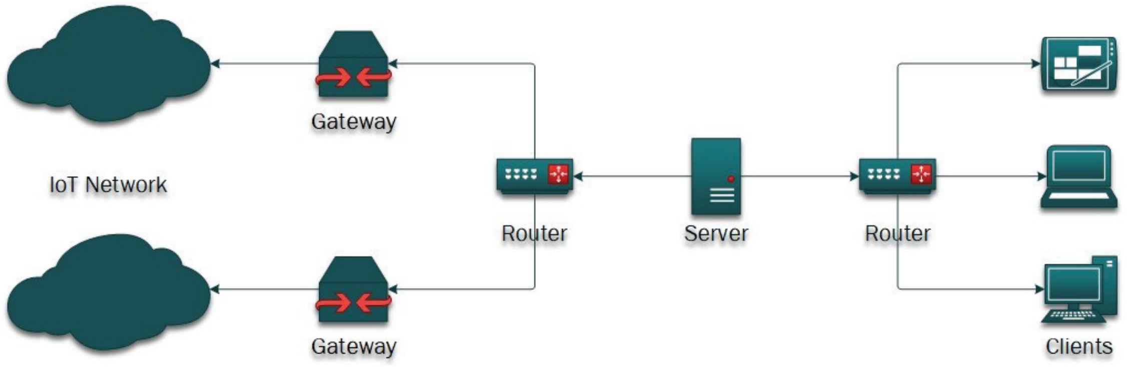
The first architecture to be tested is *XIA* (eXpressive Internet Architecture), an Future Internet Architecture with native support for multiple principals and the ability to evolve its functionality to accommodate new, as yet unforeseen, principals over time [4]. It will be testing interoperating with a 6LoWPAN IoT Network [5]. 6LoWPAN defines encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over LoWPANs. Specified by the IEEE 802.15.4 standard, LoWPAN is a simple low cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements [5].

The remainder of this article is organized as follows. Section 2 presents our system works and what we have to adapt for testing each architecture. Section 3 presents a study case where *XIA* [4] will be tested interoperating with a 6LoWPAN [5] IoT Network. Finally, Section 4 presents the final considerations, emphasizing both the main contributions of the article and future work.

## 2. Proposed Solution

Our system consist in a shell *script* that load five different types of *Docker Images* (*Client, Gateway, Client Router, Gateway Router, and Server*) connected in a star topol-

**Figure 1. Proposed solution to evaluate IoT in future Internet architectures.**



ogy. *Clients* are connected to the *Client Routers*, *Gateways* are connected to the *Gateway Routers*, *Client Routers* and *Gateway Routers* are connect to the *Server*.

There is only **one** *Server*, each *Server* can have **GR** *Gateway Routers* and **CR** *Client Routers*. Each *Gateway Router* can have **G** *Gateways*, while each *Client Router* can have **C** *Clients*. *Gateways* are running Cooja with **M** IoT motes, forming the *IoT Network*. **C CR G GR M** are input parameters of the *script*. The Figure 1 shows an example of the *script* configuration where the server is connected to **one** *Gateway Router* with **two** *Gateways* and **one** *Client Router* with **three** Clients. Each Gateway is simulating a *IoT Network*.

## 2.1. Echo Client/Server

The firmware of the motes is an *echo-server*, the *echo-server* receives data and replies the same data to the source. Each *Client* runs an *echo-client* that sends data to a mote randomly selected from a table previously stored in the *Server* and get its reply. The *echo-client* also make all the measures (delay, efficiency, load, errors, etc.). Everything will be stored and sent to the server at the end of the execution. Data size **S** and the number of requests **RQ** of each *Client* are also input parameters from the *script*.

## 2.2. Gateway Service

There is a *Gateway Service* running in the *Gateways*, the *Gateway Service* translate data from the *IoT Network* to the tested architecture. *echo-clients* and *echo-servers* will talk directly to it. In the initialization step, the *Gateway Service* store all motes addresses and send to a table in the *Server*.

## 2.3. Docker Images

All *Docker Images* must have the tested architecture running with all necessary procedures for its operation configured in the initialization step. *Docker Images* are started in the following order: *Server, Gateway Routers, Client Routers, Gateways, Clients*.

- *Server*: Configured as a router of the tested architecture. Connects all routers, store test results and address tables.
- *Client Routers/Gateway Routers*: Configured as routers of the tested architecture.
- *Gateways*: Configured as hosts of the tested architecture, starts Cooja and the *Gateway Service* in the initialization step.
- *Clients*: Configured as hosts of the tested architecture, starts *echo-client* on the initialization step.

## 3. Study Case: XIA / 6LoWPAN

In the first test, a *6LoWPAN* service will be used by a purely *XIA* application. For this, *IPv6* addresses need to be bound to *XIA* identifiers (*XID*) [4]. The *XIA* architecture defines several *XIA* identifier types with distinct semantics of communication, processing that is required to forward packets, and intrinsic security properties [4]. In this case, a *SID* (Service Identifier) type will be used. *SIDs* support communication with services [4].

### 3.1. XIA Docker Images

All *Docker Images* will be running *XIA Prototype* [4] in Ubuntu Linux containers. *XIA Prototype* allows users to Run Sample Applications over the *XIA* network and write *XIA* applications. The prototype is implemented on top of the *click modular router* [6], which will create some overhead and increase the *Docker Images* complexity.

- *XIA Server*: First image to be initialized. It is configured as an *XIA router*. It starts with an empty address table that will be populated by the *Gateway Services* later.
- *XIA Gateway Routers*: Will be initialized after the *XIA Server* has been fully started. They are configured as *XIA routers* and their only function is to route packets between *XIA Gateways* and the *XIA Server*.
- *XIA Client Routers*: Will be initialized after the *XIA Server* has been fully started. They are configured as *XIA routers* and their only function is to route packets between *XIA Clients* and the *XIA Server*.
- *XIA Gateways*: Will be initialized after all routers have been fully started. Configured as *XIA hosts*. First they start Cooja and then the *Gateway Service*.
- *XIA Clients*: The last images to be initialized. They copy the address table stored on the *XIA Server* and then initialize the echo-client.

### 3.2. 6LoWPAN echo-server/Cooja

A simple 6LoWPAN echo-server that receives a package and replies the same to the source. It will be running in the Cooja motes over a simulated 6LoWPAN IoT Network.

### 3.3. XIA/6LoWPAN Gateway Service

The *Gateway Service* will list all IPv6 addresses of the motes and assign a *SID* to each one creating a table. Another table containing only the *SIDs* will be concatenated to the address table on the *Server*. After that the *Gateway Service* stays alive waiting to route packets between *echo-clients* and *echo-servers*. When a packet is forwarded to a *echo-server*, the *Gateway Service* waits for the response to forward back to the source *echo-client* until a preset timeout. Any packet to the same *echo-server* received during this period will be buffered until it can be forwarded. If a timeout happens the *Gateway Service* replies to the *echo-client* an error message indicating packet loss on the *IoT Network*.

### 3.4. XIA Echo Client

The *echo-client* generate the messages and send to the *echo-servers* (*SID*) randomly chosen from the address table. For each generated message, the *echo-client* waits for its response, measuring: latency, packets lost in the *XIA network* (determined by a timeout), packets lost in the IoT network (determined by the Gateway Service error message) and errors (corrupted packets). Everything will be stored in a log and sent to the server at the end of execution for analysis.

## 4. Concluding Remarks

Once a test scenario is developed, running the tests will require just a single command in the terminal or minor implementations to test other proposals. *echo-client* and *Gateway Service* need to adapted using the new architecture APIs. *Docker Images* need to be adapted as well, but they are easy to prepare and similar to one another. The *echo-server* and the *script* remains the same. When a small test is successful in a single computer, we can move for larger tests using computational clouds just choosing new values for the topology. By employing open-source tools Docker and Cooja, this solution has very low cost and covers all network overheads since its components are actually implemented. Other applications can be developed as long as it is possible to implement a gateway capable of translating the messages among the architectures.

As a future work, we will be testing *Linux XIA* [7], a native implementation of *XIA* in the Linux kernel. Unlike *XIA Prototype*, *Linux XIA* does not use the *click modular router*, which will reduce the overhead and facilitate the preparation of the *Docker Images*.

## References

[1] P. Stuckmann and R. Zimmermann, "European research on future internet design," *IEEE Wireless Communications*, vol. 16, no. 5, pp. 14–22, October 2009.

[2] Contiki. (2017, Mar.) Contiki website. [Online]. Available: http://www.contiki-os.org/

[3] Docker. (2017, Mar.) Docker website. [Online]. Available: https://www.docker.com/

[4] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "Xia: Efficient support for evolvable internetworking," *NSDI'12, USENIX Association*, pp. 23–23, 2012.

[5] G. Montenegro, C. Schumacher, and N. Kushalnagar, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," RFC 4919, Aug. 2007.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[7] M. Machado, C. Doucette, and J. W. Byers, "Linux xia: An interoperable meta network architecture to crowdsource the future internet," *ANCS '15, IEEE Computer Society*, pp. 147–158, 2015.

## Acknowledgments

This work was partially supported by Finep, with resources from Funttel, Grant No. 01.14.0231.00, under the Radiocommunication Reference Center (Centro de Referência em Radiocomunicações- CRR) project of the National Institute of Telecommunications (Instituto Nacional de Telecomunicações - Inatel), Brazil. The authors also would like to thank CNPq, CAPES, FINATEL and FAPEMIG.