Implementation and performance testing of ID layer nodes for hierarchized IoT network

Jordi MONGAY BATALLA[1], Mariusz GAJEWSKI[2], Waldemar LATOSZEK[2], and Piotr KRAWIEC[2]

[1]Warsaw University of Technology, Nowowiejska St. 15/19, 00-665 Warsaw, Poland
jordim@interfree.it

[2]National Institute of Telecommunications, Szachowa St. 1, 04-894 Warsaw, Poland
[m.gajewski;w.latoszek;p.krawiec]@itl.waw.pl

Abstract. Recent advances in technologies for smart devices are having a significant impact in IoT (Internet of Things) scenarios as, e.g., intelligent buildings. Sensor/actuator networks use small and non-intrusive devices consuming reasonable amount of energy and offering improved performance. On the other hand, highly specialized devices providing high reliability are interconnected by dedicated network infrastructure because of safety reasons. This article discusses early stage of the implementation of an innovative hierarchical network infrastructure for connecting IoT objects and services where the location of the nodes is closely related to the structure of the environment as it occurs in intelligent buildings/enterprises.

1 Introduction

Networks consisting of specialized sensors and actuators play a crucial role in currently under development intelligent buildings. There are two observable areas where the networked IoT objects are successfully used: energy saving and security. Both require sensors (i.e., passive infra-red, fume detectors, etc.) and actuators (i.e. light switches, window actuators, etc.) located in selected areas of a building. Their location is strictly dependent on the structure of the building and connections between them, which creates a hierarchical network that can be modeled as a tree topology, as shown in Fig. 1.

This article discusses implementation details together with test results of the deployed hierarchical network for connecting IoT objects where node location may be defined by the same environment structure, as it occurs in the case of intelligent buildings with fix nodes and fix sensors/actuators. The presented implementation is based on the ID Layer concept that we developed and presented in [1]. The discussed network node has been mostly developed in Linux kernel module and extends the solution proposed in Flexible Packet Forwarding (FPF) method [2].

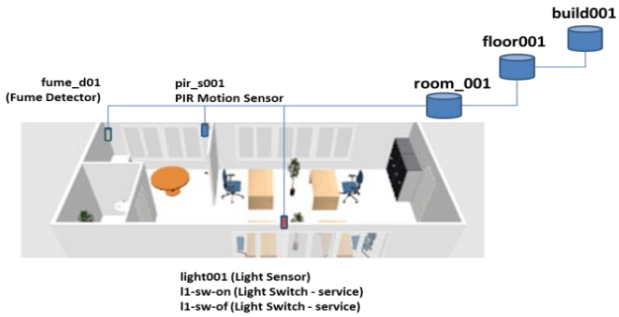

Fig. 1. Example of hierarchical network for data transmission in intelligent building

2 Context

Until recently, sensor networks were mainly the hermetic solutions based on specialized devices, dedicated network and proprietary protocols developed by suppliers. Over time, popular solutions have become factory standards widely used in the industry control [3] (e.g. Controller Area Network), but also in other areas [4]. For example, more and more smart devices have been equipped with standard wired or wireless Ethernet interfaces [5], thus they may share the same network infrastructure as other applications. Therefore, it is desirable to implement such nodes using layer-2 network that is backwards compatible with Ethernet.

In order to develop the location-oriented network topology, the key idea is to embed the physical network connectivity structure into a (logical) topological space (e.g. introducing a metric or Euclidean space). This approach was presented among others in VIRO (Virtual Id ROuting) [6] to illustrate how the novel topological perspective enables the development of the scalable resilient network routing algorithms. Another example of this approach is SEATTLE [7], which introduces OSPF-style shortest routing in layer 2 for inter-connecting objects and Ethernet switches. Such solutions are aimed at reducing network-wide flooding – often typical for Ethernet switches needed to forward packets whose locations are yet to be learned, especially in the case of wide layer-2 networks, which encompass small LAN networks.

Other approaches aimed at replacing the current global IP address space by flat identifiers, have been adopted by VRR [8], UIP [9] and ROFL [10]. They make use of several methods of hashed id assignment (mostly based on DHT), which produces an id-space completely independent of the underlying network topology. As a result, these methods perform routing based on logical distance to the id of the destination.

Real Time Control Systems consider not only topological addressing, but also transmission parameters such as packet delay and predictability of the response time. In such systems, the transmission is moderated by a controller/supervisor, which grants permissions to specific devices (i.e. sensors/actuators) by sending appropriate tokens. The order of polling is fixed according to the address table (with flat structure) stored in the controller and can have nothing to do with the physical placement of devices. The device with the next highest address is the logical neighbor, even when

they can be located at the extreme ends of a physical network. Example solutions encompass, among others, Profibus [11] and DeviceNet [12].

In turn, the ID Layer concept [1] assumes hierarchical addressing scheme for the same purpose, i.e. each level of address hierarchy is represented by one address segment. Forwarding process is based on analysis of particular address segments, however, this approach does not require physical node on each level of hierarchy since forwarding functionality may be performed also by virtualized nodes.

3 Implementation of ID Layer node

The main objective of the implementation of the ID_Layer node is to develop the ID layer in the network level instead of building an overlay network. The architecture proposed in [1] has a hierarchical structure, in which each node has a human-readable identifier related to its location. These nodes, called further as ID_Layer nodes, include connected objects (sensors/actuators) and also address the services offered by the objects. It is assumed that the name of each node, object and service is formed as an 8-ASCII extended character (word). The naming scheme uses hierarchical ID-based addressing scheme, which is created and managed in conjunction with the location of the IoT object. The human-readable names of nodes, object and services are also used for packet routing across the network.

All included functional blocks of ID-Layer node were created as software modules in user and kernel space of Linux operating system. Fig. 2 shows main building blocks of ID_Layer node and functional dependencies between them. The main modules are: (1) **Forwarding Administration Tool**, which is the configuration module. This module gives the possibility for an administrator to configure the Forwarding Table and to assign a node name; (2) **Forwarding Module**, which is responsible for sending a frame to the required node, regardless of whether the frame is a *data frame*, a *registration message* or a *resolution message*. More detailed definition of different types of frames is given in [1]. This module communicates with Registration Module (registration process), Resolution Module (resolution process) and the Forwarding Administration Tool during the initial node configuration procedure; (3) **Registration Module**, which is responsible for the registration of new objects/services in the node, to which the object/service are connected, while the (4) **Resolution Module** allows to obtain the information about all the objects/services registered in the specified node.

The implemented ID-Layer node performs functionalities of forwarding, registration of objects/services and resolution of services.

For forwarding frames, the ID address is included in the header of the ID frame together with the information about header length. Moreover, each node has assigned its own address by the administrator [1]. This allows to perform forwarding actions in the node only by comparing the ID with the address of the node without the necessity of running routing protocols.

Registration is the process by which objects (and basic/composed services offered by them) inform about the own characteristics to the closest network node. The net-

work nodes will maintain information about the connected objects and offered services.

At last, resolution is in charge of discovering the services offered by the objects and presenting them to the users (IoT applications).

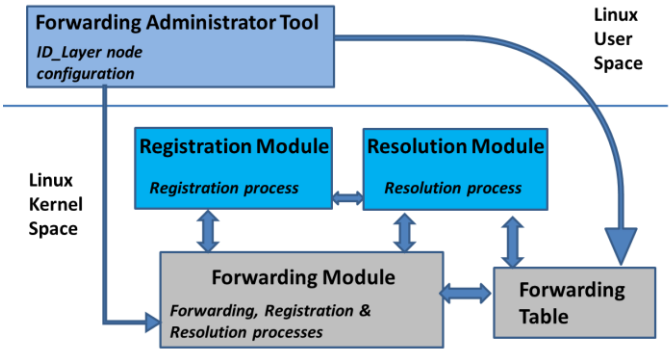

Fig. 2. General architecture of the ID_Layer node

Details of the functional processes performed by the modules are given in the next sub-sections, starting from the configuration of the ID-Layer node.

3.1 Configuration of ID_Layer node

The Forwarding Administration Tool located at the user space communicates with the Forwarding Module (kernel space) for basic node configuration in the following areas: Forwarding Table configuration and node name configuration.

The Forwarding Table configuration is performed by adding new entries in an appropriate data structure maintained by the ID_Layer node in the kernel space. The following sample command is performed to configure one entry:

./cf_tool add_ethernet *room001 eth4 aa:11:b0:c0:00:01*

where:

room001 – name of the next node

eth4 – name of the interface through which the frame will be sent

aa:11:b0:c0:00:01 – destination MAC address

In order to perform a complete configuration of the node, it is necessary to assign a node name. It is performed by issuing the following command:

./cf_tool add_name *floor111.build111.room111*

3.2 Forwarding process

The data forwarding process is performed by the Forwarding Module using the Forwarding Table, which is a data structure that stores necessary information about routes to the adjacent nodes. The forwarding process applies only to data frames and *resolution messages*, in which the destination address means the domain name of the node, while the *registration messages*, for which the fixed destination address (*.locathst*) is established, are forwarded without querying the Forwarding Table.

Data and resolution frames contain similar formats, as far as addressing concerns (ID frame header is presented in Fig. 3). The format of the ID frame header contains destination and source addresses as well as Message info (more information about data format can be found in [1]). Both, the source and the destination addresses, contain different levels (corresponding to different hierarchical levels) separated by dots (e.g., *build001*. *floor001.room0001*). Each level contains 8 bytes. In the 2-byte *message info* field, the first four bits define the message type, the next bit identifies whether the message is multicast or unicast. Finally, 11 bits indicate the length of the message in bytes.

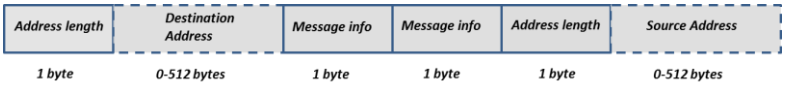

Fig. 3. Exemplary ID_Layer frame

When a frame arrives to the node, this compares the destination address with the entries of the Forwarding Table and forwards the frame, following the rule inserted into the Forwarding Table. The forwarding operation is preceded by a validation of the destination address. The aim of this step is to check whether the appropriate part of the destination address contained in the frame is consistent with the node name assigned by the administrator. For example, a frame with address floor001.room0001 should not arrive from the parent node interface (interface where the parent node is connected) to a node with address .floor002, but it may arrive from the child node interface (this case would be the case when the frame should be directed to the destination through the parent node). Each level of the destination address is compared with the corresponding level of the own node name set previously by the administrator. In the case of a failed name validation, the frame is forwarded to the node at a higher level of hierarchy (parent node). In the case of positive validation, the frame is forwarded according to the forwarding rule set in the Forwarding Table**.** From the implementation's point of view , there are two options for performing such a validation. The first option consists of converting the destination address of the frame as well as the address of the node to integer type and comparing the integers. In the second option, both the destination address and the own node name are stored and compared as character variables. It is supposed that the first option is quicker since the number of comparisons is proportional to the number of levels, whereas the second option requires a number of comparisons proportional to the number of characters (which is equal to 8 times the number of levels). In the test experiments presented in the next section, we will compare the performance of the two options.

If the validation process finishes positively, then the forwarding process goes ahead by searching the relevant part of the domain name that will be used during the forwarding procedure. The information about the own node name allows the algorithm to find the relevant part of the name. If the node name is *floor001.build001* and the destination address set in the frame header is *floor001.build001.room0001*, then the forwarding will be based on the last part of the address (*room0001*).

The next step of the algorithm is to find an appropriate entry in the Forwarding Table for the relevant part of the domain name.

The Forwarding Table consists of entries with 3 fields: *destination_MAC*, *dev_name* and *next_node*, as shown in Table 1.

Table 1. Data structure of Forwarding Table entries

Name of variable	Type of variable	Description
destination_MAC	uint8_t [48]	Destination MAC address of the node interface to which the frame is sent
dev_name	char [5]	Name of the outgoing interface
next_node	char [8]	Domain address of the next node

The information about the *next_node* (destination node address) is used to calculate a specific *index* of the entry in the Forwarding Table. In order to add the appropriate entry to the table, the algorithm converts the 8-byte long *next_node* name to 1-byte numerical value according to the following iteration algorithm (1):

$$index = ((37 * index) + ch \rightarrow name[i] \& 0xff) \quad (1)$$

where *index* is the value of the converted *next_node* (1 byte), *i* is an iteration variable and *ch→name[i]* is the i-level of the domain node name.

Then, the *find_entry* function queries the Forwarding Table about the interface connected to the value *index*. On the basis of the information contained in this entry, the frame is sent to the next node.

Fig. 4 shows the sequence diagram of the forwarding process.

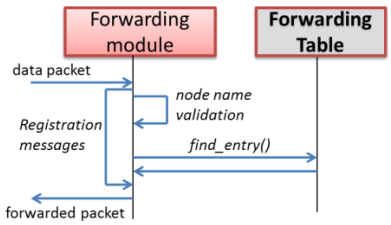

Fig. 4. Sequence diagram of the forwarding process

3.3 The registration process

The registration process illustrated in the sequence diagram presented in Fig. 5 is initiated upon the receipt of a specific *register message* [1] by the Forwarding Module. Then, the Forwarding Module reads the appropriate *message info* field [1] placed in the message header and redirects the process to the Registration Module without querying the Forwarding Table. In the Registration Module, the function *register_handle* is called. This function maintains main data structure with information about currently registered objects or services (object/service identifier –*id* and full

address of the object/service - *ObjectAddress*). If the data structure does not yet store the entry with the demanded *id* of the object/service, then a new entry is created with the data contained in the *register message*. Finally, the Registration Module passes to the Forwarding Module the necessary information used for sending *response message* to the object/service (that sent the *register message*) in order to confirm the registration process.

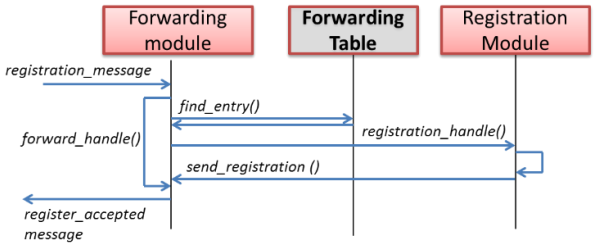

Fig. 5. Sequence diagram of the registration process

3.4 Resolution process

The resolution process illustrated in the sequence diagram of Fig. 6 is initiated by the user (IoT application) in order to retrieve information about registered objects/services.

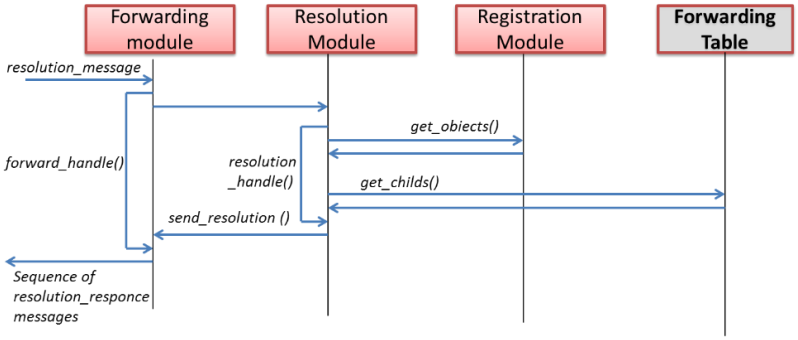

Fig. 6. Sequence diagram of the resolution process

For this purpose, the user's application sends a resolution message, which is forwarded in accordance with the forwarding algorithm until the destination node. When the message reaches the destination node, the Forwarding Module of this node extracts the message info and, based on this information, redirects the process to the Resolution Module where the function resolution_handle is executed. In the next step, the Resolution Module retrieves information about registered objects/services of the queried node from the Register Module (get_object function) and retrieves

information about all child nodes from the Forwarding Table (get_child function), which enables the identification of the objects/services registered in the nodes of lower hierarchy. In the final phase of the resolution_handle function, the Resolution Module sends the appropriate information to the Forwarding Module. The latter builds a resolution response message, in which the source and the destination addresses are interchanged. Moreover, information about registered objects/services and names of the child nodes are placed into the information field of the resolution response message. Finally, these messages are sequentially forwarded back to the requester.

4 Performance tests of forwarding process

Even if the advantages of ID addressing and ID Layer forwarding are numerous for IoT applications [1], the proposed solution risks fail in the case when the implementation does not fulfill the requirements of performance necessary for forwarding a large amount of packets. The aim of the presented here performance tests is to show that the deployed solution is efficient enough to be used in IoT scenarios. More precisely, we will demonstrate that the ID-Layer node performance is comparable to IP router implemented on Linux OS (software development). Moreover, the test deal with scalability issues show the behavior of the ID_Layer node for increasing up to 8 number of the domain levels, i.e., for increasing hierarchy atomization.

The testbed /consists of one System Under Test (SUT) and one tester. The SUT is the ID_Layer node installed on HP ProLiant DL360G6 server, which runs Linux Operating System. The tester is the Spirent TestCenter (equipped with CM-1G-D4 card). The tester and the SUT are connected by two 1 Gbps Ethernet links in ring topology, as proposed in the benchmarking for testing network interconnect devices presented in RFC 2544 [13]. We performed tests for the following frame size: 96B, 112B, 128B, 160B, 256B, 384B, 512B, 1024B and 1518B, and the stream was the maximum allowed by the interfaces, i.e., 1 Gbps. In these conditions, we measured the frame losses observed in the SUT due to overload of the server.

The results presented below shows the Frame Loss Ratio for different frame sizes and increasing number of domains. First of all, let us remark that the software IP router implemented on Linux OS was installed in the same hardware and the Frame Loss Ratio of the IP router was, at least, 20 times higher than the ID_Layer node (for all frame sizes), even in scenarios with 8 domain levels. Note that the IP router performance is not affected by level complexity because of the same nature of IP addresses. For clarity purposes, we did not present the values of Frame Loss Ratio in the figure.

As one of the major features of the forwarding process is the validation of the destination address in the ID_Layer frame, we compared two approaches for implementation, which is described above. As stated above, this address is composed of the levels of the domain name separated by dots. The introduction of this functionality results in the need for additional computational effort caused by parsing the destination address.

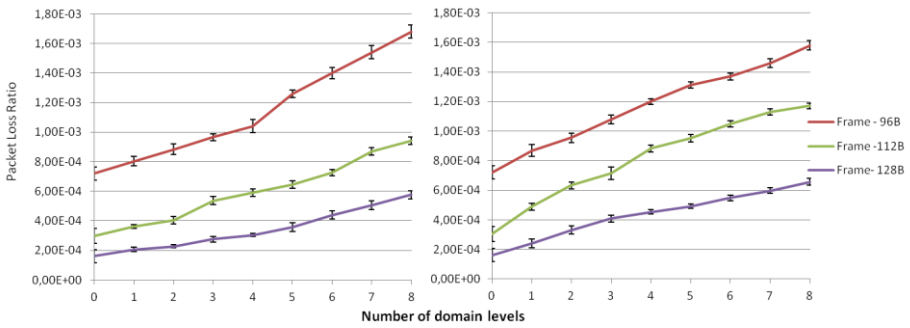

Fig. 7. Test results for frames 96B, 112B, 128B frames (both options of forwarder implementation):
a) name validation with conversion of variables. *b)* name validation without conversion of variables (0 number of domain levels means forwarding without validation name)

The results revealed that the increasing number of levels of the domain name in the destination address causes a higher Frame Loss Ratio for both options of the ID_Layer forwarder (i.e., name validation with and without conversion of variables). The increase of Frame Loss Ratio is approximately linear for each test scenario. In addition to this, it can be noted that, for the same tests (frames with the same number of domain levels), there is no significant difference between the values of Frame Loss Ratio for both implementations of the forwarder. .

In Fig. 7, we presented results only for frames not bigger than 128B, but the tests were performed for other frame sizes according to the test assumptions. The results of the other tests confirmed that the forwarder transmits frames with bit rate equal to the maximum link bitrate - 1Gb/s for frames larger than 160 Bytes independently of the test scenario (number of domain levels). For the frame size equal to 160B, the Frame Loss Ratio parameter does not exceed the level of 10^{-6}. This means that the main limitation of the forwarder performance is the performance of network interface as frame loss occurs only for very small frames.

It can be concluded that the size of the address contained in the frame header has a small impact on the value of the Frame Loss Ratio parameter and, on the other hand, the performance of the ID_Layer node is satisfactory (at the same level as software IP router).

5 Conclusions

More and more smart devices used in current developments are equipped with Ethernet interfaces, which allow sharing the same network infrastructure between different applications. This approach allows to reduce implementation costs and does not adversely affect the functionality of the intelligent building solutions.

The based on the ID layer concept implementation was developed over the Linux operating system. It engages mainly kernel resources to improve effectiveness of the solution.

We conducted performance tests of the prototype aimed at checking the effectiveness of the implemented solution for different lengths of the address field. In result, we calculated the limit performance of the node, which could be located on different levels in hierarchy tree. The test results presented above confirm the usefulness of kernel based approach for ID layer implementation. In particular, it is able to serve high frame rates and can be implemented as additional functionality of Linux based network nodes.

Further works will cover, besides the completion of routing functionality according to assumptions in [1], development of system for centralized domain names management.

References

1. J. Mongay Batalla and P. Krawiec, Conception of ID layer performance at the network level for Internet of Things, Springer Personal and Ubiguitous Computing, 2013
2. A. Bęben, J. Mongay Batalla, P. Wiśniewski and G. Xilouris, "A Scalable and Flexible Packet Forwarding Method for Future Internet Networks", IEEE Globecom 2014. The source code for Flexible Packet Forwarding (FPF) method may be found in: http://tnt.tele.pw.edu.pl/software_fpf.php
3. Website of CAN in Automation (CiA), available on http://www.can-cia.de/
4. Świątek, P., Klukowski, P., Brzostowski, K., Drapała, J.: Application of wearable smart system to support physical activity. In: Advances in Knowledge-based and Intelligent Information and Engineering Systems,pp. 1418–1427. IOS Press (2012)
5. Hunt J., Building automation migrates towards Ethernet and wireless, Industrial Wireless Book, Issue 69 (April 2012), available on http://www.iebmedia.com/wireless.php?id=8593&parentid=74&themeid=255&hft=69&showdetail=true&bb=1
6. Sourabh J, Yingying C, Zhi-Li Z (2009) VEIL: A "Plug-&-Play" virtual (ethernet) Id layer for below IP networking. 1st IEEE workshop on below IP Networking 2009 (In conjunction with IEEE Globecom 2009)
7. C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: a scalable ethernet architecture for large enterprises. SIGCOMM, 2008
8. M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTS. SIGCOMM Computer Communication Rev., 2006
9. M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica. Rofl: routing on flat labels. In SIGCOMM, 2006.
10. B. Ford. Unmanaged internet protocol: taming the edge network management crisis. SIGCOMM Computer Communication Rev., 2004.
11. Website of PROFIBUS and PROFINET International (PI), available on http://www.profibus.com
12. Website of global standards development and trade association for Common Industrial Protocol or "CIP" – and the network adaptations of CIP – EtherNet/IP, DeviceNet, CompoNet and ControlNet. Website available on http://www.odva.org/
13. Bradner and McQuaid, "Benchmarking Methodology for Network Interconnect Devices". Requests For Comments RFC 2544. 1999.