www.czasopisma.pan.pl PAN POLSKA AKADEMIA NAUK www.journals.pan.pl

INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2019, VOL. 65, NO.3, PP. 367-373
Manuscript received March 22, 2019; revised August, 2019. DOI: 10.24425/ijet.2019.129787

Performance of Routing Algorithm Remote Operation in Cloud Environment for IoT Devices

Valentyn Faychuk, Orest Lavriv, Bohdan Strykhalyuk, Olga Shpur, Ivan Demydov, and Roman Bak

***Abstract*—This paper proposes an advanced routing method in the purpose of increasing IoT routing device's power-efficiency, which allows to centralize routing tables computing as well as to push loading, related to routing tables computation, towards the Cloud environment at all. We introduced a phased solution for the formulated task. Generally, next steps were performed: stated requirements for the system with Cloud routing, proposed possible solution, and developed the whole system's structure. For a proper study of the efficiency, the experiment was conducted using the developed system's prototype for real-life cases, each represents own cluster size (several topologies by each size), used sizes are: 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27 and 29. Expectable results for this research – decrease the time of cluster's reaction on topology changes (delay, needed to renew routing tables), which improves system's adaptivity.**

***Keywords*—routing algorithm, IoT, IoT device, cloud environment**

I. Introduction

AT latest several decades of constantly increasing demand for intelligent and compact things, a new term has arisen – Internet of things, IoT [1]. Internet of things is strictly bounded with enlarged requirements to devices that can be used. Thus, for the field of information communication technologies, ICT, certain tendencies had grown up, arrowed towards solving myriad issues: simplicity, stiffness, fault tolerance, mobility, load balancing etc.

Commonly routers are robust and have stable power supplies (thus they must be statically installed). That is why common routing methods are expensive by means of computing, storing, and keeping routing table's actuality. Hence these routing methods can't be used in IoT clusters because of many redundant actions that drain battery resources. Critical influence these conditions reach in clusters with full or partial mesh topologies, which is usually the case for IoT.

II. Trends and Problem Statement

Using classical routing approaches introduces lots of problems for wireless IoT with its P2P networks [4]. Moreover, the EEPROM of the IoT device is too small and slow to hold routing tables. This motivates towards seeking new approaches.

Defying modern tendencies [2], [3], [5], this work proposes a fundamentally new approach – to centralize routing table's computing, which will decrease computational loading, applied to weaker devices in the cluster, and push determined loading center out of cluster bounds to the Cloud environment.

A. System requirements

In research, you can frequently notify term "cluster" – the network, composed of a certain number of devices that use their internally installed voltage sources (batteries) and can communicate as well as create brand new links when needed. According to that, the first requirement can now be stated – effective use of battery resources. The battery is mainly used in two cases: when the device performs computations and when it transmits or receives information (communicates). Term "coordinator" usually describes the device, which acts as an intermediate link in IoT hierarchy. However, in this research, its responsibility is to act like the cluster's controlling center. In the given case, it acts like the center of routing table's computation. The coordinator can use Cloud resources in its own purposes. Given amendment allows breaking the system into two self-reliable subsystems according to decomposition principle (coordinator, is even capable to compute everything needed on its own in case of losing the link with Cloud service).

B. Structure of the proposed routing principle

Now then, main parameters, which must be kept by the routing algorithm [6]–[8]: minimal complexity for defining further path to send packet inside nodes; minimal volume of local memory in the node, needed for routing process maintenance; and ability to transfer algorithm to the Cloud environment, that is algorithm must calculate all valuable parameters centrally. These requirements form the structure of the system. The system consists of three main parts: cluster, coordinator and the Cloud subsystem. But before we start to build system prototypes, it is obvious to consider proposed principle for routing messages inside the cluster. That's why in nearest subsections we review cluster and its functioning when the coordinator's along with Cloud's subsystems and their interaction will be covered later.

1) Routing table generation

Routing table, according to the proposed method, is formed using the Bellman-Ford algorithm [9], [10]. The example of distances matrix (it is the exact routing table) for the cluster, represented in figure 1, is offered in table 1. The matrix, in this case, describes minimal distances (in hops, just like "hop count" metrics in classical routing approaches) between each two IoT devices inside cluster for current topology. Each row in this matrix corresponds to a unique node and represents its unique vision of a network around it (consequence: two different nodes

All Authors are with Lviv Polytechnic National University, Ukraine (e-mail: valentyn.faichuk.tk.2015@edu.lpnu.ua, orest.a.lavriv@edu.lpnu.ua).

can't have equal rows). It is clear to see, that routing policy, in this case, is arrowed toward carrying packets through the smallest quantity of nodes (like in RIP protocol, again "hop count" metrics) during the routing process.

The width of the routing table is equal to the size of the cluster (quantity of devices inside the cluster). As a result, lengths of different rows are equal, which allows to element-wisely compare them for the routing process. This also imposes certain limitations on the cluster size: for big clusters, rows become too large to be transmitted and compared effectively, especially by the weak IoT devices. However, the solution to this problem is the subject of a separate investigation.

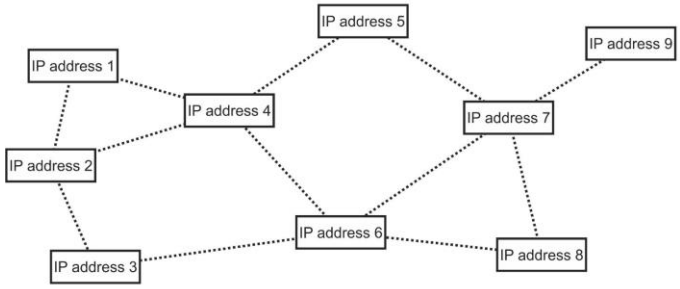

Fig. 1. Example of a cluster of eight IoT devices

TABLE 1
BELLMAN-FORD MATRIX FOR THE CLUSTER, SHOWN IN FIG. 1

	IP 1	IP 2	IP 3	IP 4	IP 5	IP 6	IP 7	IP 8	IP 9
IP 1	0	1	2	1	2	2	3	3	4
IP 2	1	0	1	1	2	2	3	3	4
IP 3	2	1	0	2	3	1	2	2	3
IP 4	1	1	2	0	1	1	2	2	3
IP 5	2	2	3	1	0	2	1	2	2
IP 6	2	2	1	1	2	0	1	1	2
IP 7	3	3	2	2	1	1	0	1	1
IP 8	3	3	2	2	2	1	1	0	2
IP 9	4	4	3	3	2	2	1	2	0

2) Routing process itself

When the hop needs to transfer a packet to the other hop inside its cluster, and it knows IPv6 address of the destination, it first sends to the coordinator request for the sequence of distances for the destination (routing row). If the device with the destination's address is still inside the cluster, the coordinator returns the actual routing row inside distances matrix, which corresponds to the IPv6 address of the destination. After receiving this row, hop saves it into cash and attaches it to the packet header. Each distances matrix have its unique identifier, which also must be attached to a packet header. This is done to avoid comparing routing rows that belong to different routing tables in further nodes.

To define the path for packet transmission, the node must have rows from distances matrix of all its neighbors. These information devices get from the coordinator, whenever it distributes among nodes corresponding rows. Along with that, they save in cache memory routing rows for all their neighbors in addition to their own. This is done to decrease service traffic during routing. When an intermediate node receives a packet, it looks for destination's routing row, attached to the header, then it takes routing rows of its neighbors from the cache memory and compares them with destination's row (Fig. 2). When comparisons are done, device evaluates resulting differences. The bigger is the difference between routing rows (from single routing table), the bigger is the distance between that neighbor and a destination node within the current topology. It is clear to see, that after comparing differences, the device will choose a neighbor, whose routing row differs from the row of destination minimally.

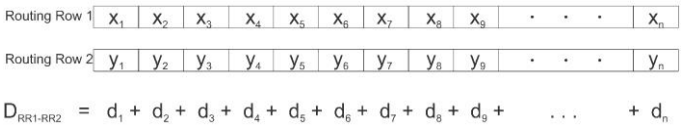

Fig. 2. Process for getting the difference between two routing rows, where the difference between two corresponding members $d = abs(x - y)$

III. EXPERIMENT INFRASTRUCTURE

Ability to centralize computations in proposed routing principle allows to push loading to the Cloud environment [5], [11]. However, along with centralization, the need for an exhaustive description for input and output parameters occurs. Apparently, the structure for output data (fig. 3, data structure, generated by Bellman-Ford algorithm) determines itself: it is distances matrix and could be represented as a list of rows (row, in its turn, is the list of calculated distances).

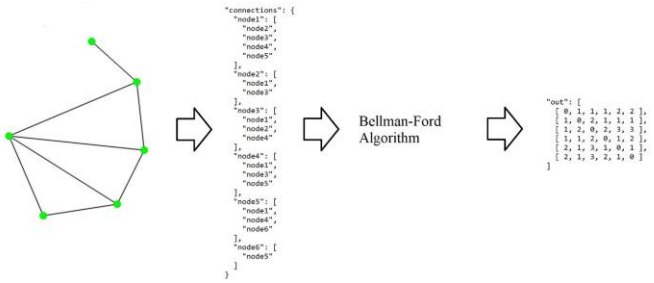

Fig. 3. Connections map and distances matrix descriptions for the presented cluster in JSON format, nodes in the cluster are encountered counterclockwise

Now, let's describe what the structure of input data looks like. For routing table's calculation, it is enough that the algorithm has only connections map of a cluster. The most convenient way to describe this map in order to send it through the internet to the Cloud is JSON (JavaScript Object Notation – data exchange format [12]) (fig. 3, data structure next to the cluster). Therefore, connections map object consists of a core (outer) list, whose keys are cluster's separate nodes. Values, related to these keys contain local (inner) lists of other nodes, connected to the node that is denoted by this key. The most integral parts of this JSON description is text strings (node's names).

A. Structural diagram of the experiment and principle of cooperation between coordinator and Cloud subsystem

We propose to experimentally distinguish two cases (fig. 4): when the coordinator tries to compute matrix by itself, and when the coordinator uses Cloud service. These cases are generally denoted with Simulated device 1 and Simulated device 2. Such a strange designation derives from a simplified view on the system's structure. The cluster is represented by a model, which changes its state (topology). Coordinator gets these changes and after a certain delay returns response – new distance's matrix. The format of the data object, which describes topology, is realized in terms of JSON according to the structure, shown on fig. 3. Distance matrix format is also JSON (fig. 3). Finally, the

abstraction of Simulated device is a function, which receives input arguments and after a certain delay, responds with a result. The most generally, for system Simulated device 1 the name of coordinator is nothing else, then Simulated device 1.

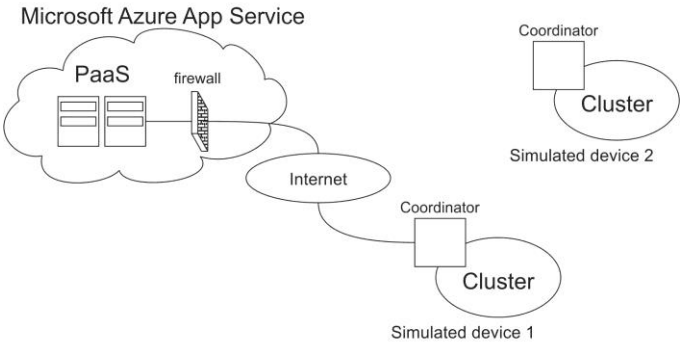

Fig. 4. Simplified experimental chart

To embody the routing table's computing service, Microsoft Azure App service was chosen [13], [14]. Type of this office – «platform as a service», fig. 5. Platform as a service (PaaS) – is a place for development and introduction of custom products, located entirely within the Cloud environment. Azure App service has plenty of resources, which allows in future without excessive headache turn from simple services, like the one, which is developed for this research, to more complicated [15]. Microsoft Azure App service gets paid as you go (Metered Usage).

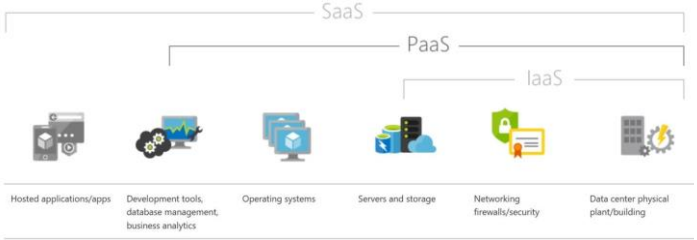

Fig 5. Direct services description, which can be used via Microsoft Azure PaaS [16]

Python programming language was chosen for structural elements because of its simplicity and flexibility. During development, we used development tools DevOps, also presented by Microsoft Azure. The application is based on the Flask [17], the popular Python framework. Using this approach allows to simplify and enforce Cloud service's development.

1) Coordinator's structure

As stated, there are two coordinator's types: the one that calculates routing tables by itself and the one, which uses remote Cloud service. System, which contains cluster, which has a link to the Cloud environment, that uses remote resources, called Simulated device 1.

Let's consider the structure of Simulated device 1 in more detail (fig. 6). The coordinator has a direct link to its cluster, through which it receives information about topology changes. After notifying change it saves renewed topology state within its local database. Next according to the HTTP protocol [18] it composes the request toward Cloud in the purpose of performing remote computations. To make request coordinator get the cluster's last diameter and new topology state. Then it stuffs the request body with input parameters. Request body – is a JSON text string that is to be sent to the Cloud service by address http://api_bfalgorithm.azurewebsites.net/calculate in the purpose of receiving a response with prepared distances matrix. The request body consists of 3 key-value pairs. The value for key "version" is the version of used API (application programmable interface) – the set of rules and principles, which describes a conversation between devices through the Internet [19]. Next key is "depth". It is, strictly speaking, the depth of algorithm penetration, or the maximal distance between two nodes in topology that can be measured by the algorithm. To make the "depth" term apparent – think of it as of an analogy for TTL field within packets, transmitted by an existing routing algorithm. The sense of restricting "depth" is also the same – to decrease time, needed for distances matrix calculation. To fill in depth "depth" we propose to use last cluster's diameter plus one, because one integral change in topology can't lead to diameter expansion more, than by one. This approach optimizes the algorithm's performance for a specific topology. In the next key-value pair, the key is "connections", and the value is the topology object in JSON format (fig. 3). The only one header, obvious to be present in the request is stated as Content-Type: application/json and is used to point information type. Also, as we are sending to the service information, the type of request must be POST.

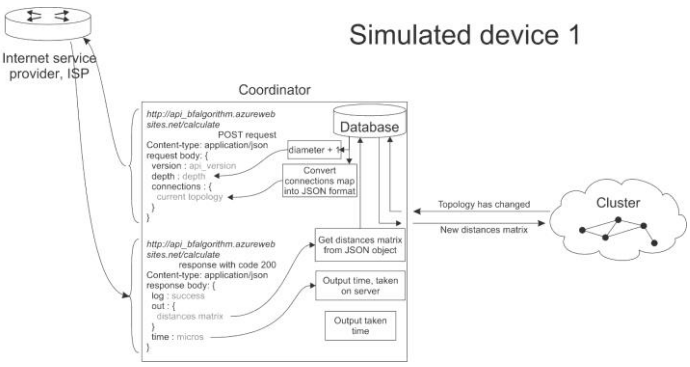

Fig. 6. Structure of the coordinator, connected to the Cloud environment through ISP – internet service provider

In this case, two delays are worth measuring: delay on a server-side and total delay before receiving a response. Healthy response status is 200. Obvious headers: Content-Type: application/json. Response body similarly contains a JSON object. The first key-value pair in the body is a log, it is used to account about how the method was executed in the Cloud environment. That is, if the algorithm had been executed successfully or if not, then why (an incorrect API version, an incorrect format of the request etc.). The value for the key "out" is the resulting distances matrix in JSON format (fig. 3). In the pair where key equals to "time" the value equals to the delay for BF algorithm performance (time, spent for algorithm execution in Cloud environment). If the value of key "log" is "success", then coordinator writes into its database received distances matrix. Only after coordinator successfully gathered distances matrix, it can respond to the topology changes.

In contrast with the previous case, Simulated device 2 doesn't contain any Cloud service and its coordinator tends to perform computations self-reliably. Hence, such a coordinator doesn't

need any API (can avoid request formation). However, after being informed about the integral change in topology, it pushes change into its database. Then it pulls the latest topology map as well as the latest known cluster's diameter, adds 1 to it, and pushes these parameters toward the input of BF function. After the algorithm has finished, the coordinator does the same as in the previous case. The only measurement is the total delay before getting distances matrix ready. As we can admit the interface for both coordinators is the same, but the realizations are different.

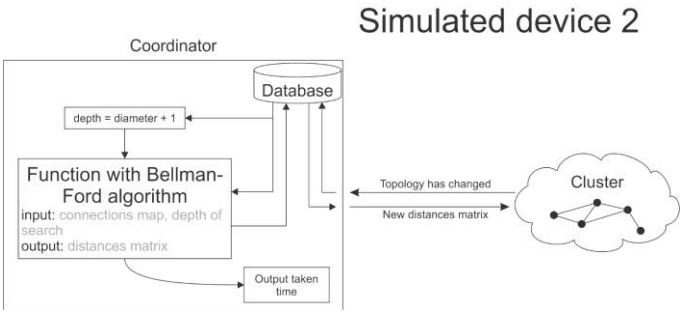

Fig. 7. The internal structure of self-dependent coordinator

2) Structure of the Cloud environment

The most complicated fraction of the experimental assembly is Cloud environment (fig. 8). The major part of this subsection is dedicated to the method, which is responsible for processing "calculate distances matrix" requests along with sending corresponding responses. Recalling the structure of coordinator, API is based on HTTP protocol (request-response). For experimental purposes and further clarity let us name this method "calculate". This method processes only POST JSON requests, otherwise error message gets triggered.

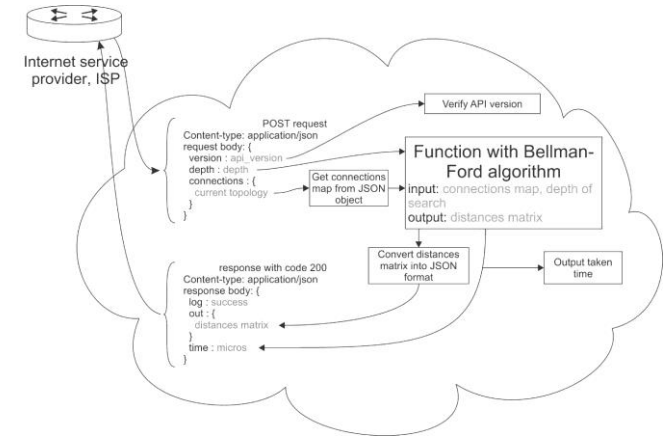

Fig. 8. Internal structure and example of correct work of a method that computes distances matrices in Cloud environment

"Calculate" method captures JSON object and verifies, whether API version is correct. If not – the response is "log": "failure, invalid API". Further, BF function converts the connections map into distances matrix. The time to compute distances matrix must be measured and also can be saved to the database in the purpose of collecting statistics. The healthy response must contain three obvious key-value elements: pair "log": "success", "out": distances matrix, and a "time": delay time to execute BF function. Finally, the method transmits this response is with code 200 towards the source of the request.

B. Practical realization of the system

Coordinators within Simulated device 1 and Simulated device 2 are realized programmatically. Python programming language Python of version 3.6.6 was used. Additional modules are simplejson (functions that load JSON object from text string into a python's dictionary data set and vice versa), requests (functions for making requests to the Cloud environment), time (functions for performing time measurements). Simulator programs get their input values in a form of JSON formatted topologies. Program with convenient GUI for making a cluster and exporting its topology into JSON format (fig. 9) is written in C++ programming language, version 11. GUI uses OpenGL framework [20], whilst other functions use standard template library STL features.

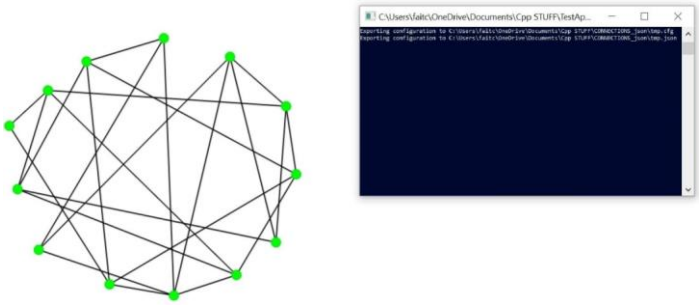

Fig. 9. The graphical user interface of a program for generating clusters and exporting their topologies in JSON format

13 different cluster sizes (devices quantity in the cluster) of 3 types (by means of topology) for performing experimental measurements were developed using this program.

Cloud service was developed using Microsoft Azure App Service [14]. To implement the application, we use the Python of version 2 along with the Flask framework, latest version [17]. All resources are offered via "pay as you go" payment method. Virtual machine parameters: 1 Core, 1.75GB RAM. The development environment is Azure DevOps. We also use source control (control version system), based on Git for versioning. In this case, developers have their own copies of the repository on their local equipment [15], [21]. Thus, further development of a service can be done by separate teams. Services of Azure DevOps is CI/CD (continuous integration, continuous delivery). The occupied resources are located in North Europe because it is the most probable scenario [22].

IV. Efficiency for Remote Calculation of Routing Tables Research

A. Mathematical reasoning of the proposed principle

In this section, we will operate the term performance or efficiency or gain (all these terms are interchangeable) from using remote computations instead of local ones.

Formerly, let us generally explain, what the "gain" means for us. Ideally, to organize effective dynamical routing within a cluster, the integral topology changes must trigger immediate reactions. Though, in real systems the delay is obvious. It directly influences the efficiency of the routing process, i.e. its relation to the routing policy. The ability to predict cluster state (model its further behavior) is so crucial because it influences the level of control over the cluster. Thus, the fact that reaction delay ought to be minimized is apparent. For the proposed

principle, the delay depends directly on the performance of the BF algorithm (as well as any other investigated algorithm). Two major scenarios of performing BF algorithm are local and remote. Hence, efficiency represents the ratio between two delays of routing table's computation: for local scenario and for remote scenario (certainly for the same cluster).

In the efficiency design next factors must be considered: algorithm itself, or rather its algorithmic complexity; ratio between times, needed to perform one integral operation in Cloud and in IoT device (generally it depicts how much the IoT device is faster than the Cloud environment); transmit time for one integral value between IoT device and Cloud environment.

Algorithmic complexity of an algorithm is the mathematical law, which outlines delay increase with data amount's (length of input array) enlargement [23], fig. 10.

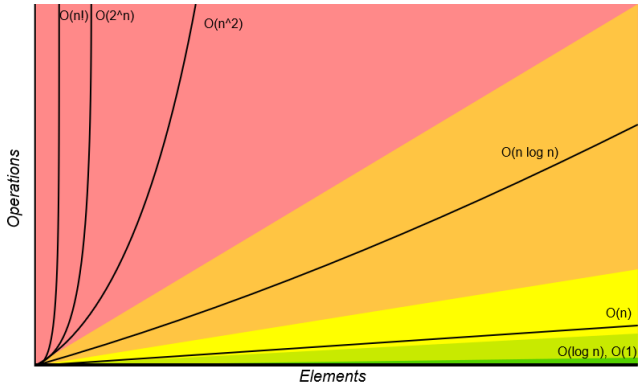

Fig. 10. Time on performing algorithms of a different algorithmic complexity versus the number of input values (in programming, size of input array) [24]

The attitude of the time for performing one integral operation in the Cloud environment to the time for processing it in IoT device It is a non-dimensional indicator of the certain algorithm running time decrease using the Cloud environment, having its values between 0 and 1 (providing that Cloud environment is more powerful than IoT device which is almost always the case). We propose to gather it empirically by comparing the measured delays, introduced by one operation within the IoT device and Cloud, formula (1).

$$p = \frac{P_{device}}{P_{cloud}} = \frac{t_{cloud}}{t_{device}} \in (0 \cdots 1) \tag{1}$$

where P_{device} and P_{cloud} – are computational capacities of coordinator and Cloud environment, t_{device} and t_{cloud} – are the durations, introduced by an integral operation, similarly.

The time spent to transmit one value between IoT device and Cloud environment – is the time of passing one integral part of information through one direction (from coordinator to Cloud or vice-versa). As it depends on a myriad of factors, we propose to measure this value also empirically (transmit a large amount of information between terminals, measure transmission time, divide time by values quantity), formula (2).

$$\tau = \frac{T_{\Sigma}}{N}, [ms] \tag{2}$$

where T_{Σ} – is the delay, introduced by passing values through the network, N – values quantity.

The efficiency, being a ratio between the local computation and remote computation times, depends on these parameters via relations, described by formula (3).

$$\varepsilon(n) = \frac{T(n)}{p \cdot T(n) + 2\tau \cdot n} \in (0 \cdots \infty) \tag{3}$$

where $T(n)$ – is the delay of performing algorithm over n values inside the IoT device, n – the number of input values.

The numerator is the time of local algorithm execution. The first term of the denominator is the time to perform an algorithm in the Cloud environment. The second term of the denominator is the delay, introduced by data transmission (here we assume, that the amount of data in request roughly equals the amount of data in response). For cases of medium to low clusters, the difference between request and response sized can be neglected. On fig. 11 formula (3) is tabulated (efficiency vs cluster size) for different algorithmic complexities of possible routing algorithms.

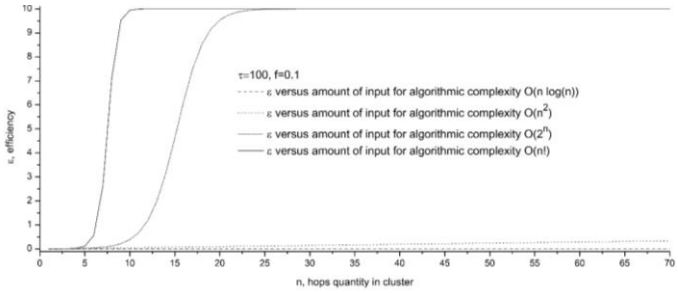

Fig. 11. Tabulated efficiency for next algorithmic complexities, curves from bottom to top: quasilinear, quadratic, exponential, factorial

B. Experimental results estimation and analyzation

The main reason to conduct an experiment – is to determine if the proposed method can be effectively used on practice. Bellman-Ford algorithm used in our experiment has some distinctions from its reference realization. In the etalon realization of a Bellman-Ford algorithm, its algorithmic complexity (memory access operations are discarded) is cubical $O(n^3)$ [9]. For the developed algorithm, fig. 12, however, the measured algorithmic complexity in case of processing clusters with less than 29 devices is around $O(n^4)$. The Simulated device 2 was tuned to work 10 times slower than the Cloud does $p = \frac{t_{cloud}}{t_{device}} = 0.1$.

```
def BF_algorithm(connections, depth=10):
    return [[distance(from_node,to_node,connections,depth) for to_node in connections.keys()] for from_node in connections.keys()]

def distance(current_node,to_node,connections,m_depth,depth=0,passed_nodes=[]):
    if current_node==to_node: return depth
    depth+=1
    if depth>m_depth: return m_depth+1
    elif current_node in passed_nodes: return m_depth+1
    else:
        ex_passed_nodes = deepcopy(passed_nodes)
        ex_passed_nodes.append(current_node)
    distance_to = m_depth+1
    for next_node in connections[current_node]:
        tmp = distance(next_node,to_node,connections,m_depth,depth,ex_passed_nodes)
        if tmp<distance_to: distance_to=tmp
    return distance_to
```

Fig. 12. Realized Bellman-Ford algorithm

Results for measuring the time spent to calculate different matrixes locally and remotely are presented on the fig. 13 in form of a plot (dotted line is the local scenario – Simulated device 2, solid line is the remote scenario – Simulated device 1). Each value on the plot is averaged among 50 real measurements of a studied value [25]. There is no need to introduce errors on this graph by now because plotted curves are mainly needed to represent the sense of how does the duration increases with increasing cluster size under different scenarios. This plot allows to make a very important conclusion: for different boundary conditions (right and left plot margins) it is better to use different approaches. For very small clusters it is better to compute routing tables using local resources. But for medium to large clusters that is much better to use remote Cloud service. And in case of an infinite cluster, the use of remote service gives $\frac{1}{p}$ gain over local scenario.

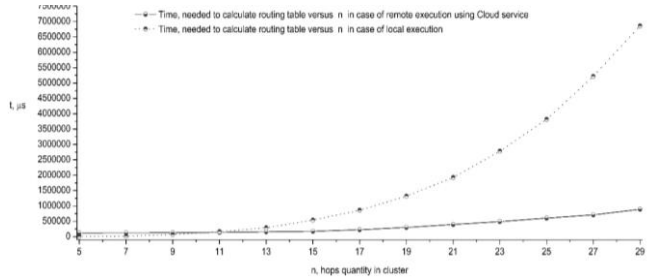

Fig. 13. Time to compute distances matrix versus cluster sizes in Simulated device 1 (solid line) and in Simulated device 2 (dashed line)

For now, we can derive the curve of the efficiency versus cluster size (fig. 14). Parameters for mathematical approximation: algorithmic complexity is $O(n^4)$, coefficient $p = \frac{t_{cloud}}{t_{device}} = 0.1$, the transmit time was selected empirically.

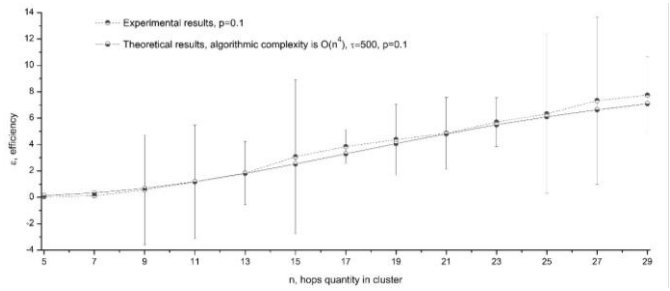

Fig. 14. The efficiency of using Cloud environment for routing tables computation versus cluster size derived theoretically (solid line) and experimentally (dashed line)

CONCLUSION

Internet of things has plenty of unique conditions that post enlarged demands related to effective battery resource usage in IoT devices. These restrictions make classical routing approaches insufficient. In this research, we propose the solution for effective routing in IoT systems and then improve it using Cloud service. Also, the results of efficiency evaluation are presented.

Estimated sizes of cluster are: 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27 and 29. Reached results: proposed adopted for IoT routing method; it was optimized for the minimal reaction time in case of topology changes using Cloud service; the optimization gain was characterized in two ways: mathematical and experimental. Meanwhile, these results allow to inspect the applicability of the proposed approach to IoT systems in practice.

The peculiarity of this research is a practical realization of the fully functional research prototype. Prototyping process includes API development, Microsoft Azure App Service and Microsoft Azure DevOps resources occupation, services as well as simulators development and many more.

For the use of the remote Cloud service, performed measurements showed a rapid increase of gain in cases of bigger cluster sizes, e.g. in case of 29 IoT devices in the cluster, the gain is $\varepsilon = 7.739$ for experimental assembly and $\varepsilon = 7.092$ for theoretical approximation. At the same time, the biggest mismatch between theory and experiment is $\Delta = 0.549\,\%$. The convenience of this research is that Cloud performs computations 10 times faster, that IoT device does. In case of infinite cluster size (right border condition), efficiency tends to be 10. Therefore, in the case of 45 hops in the cluster, the efficiency must be $\varepsilon = 9.011$, while for 99 hops it equals $\varepsilon = 9.898$ (derivative lowers as the gain approaches the ratio between Cloud's and IoT device's speeds). Also, the issues for further studies are: using databases to improve routing table's calculation performance in Cloud, adjusting (varying) routing policy remotely in Cloud; predicting further cluster states in Cloud; machine learning investigation in the purpose of improving administration strategies, used by Cloud.

REFERENCES

[1] A. Murtala, Z. S. Subashini, P. Vetrivelan, and S. Proceedings, Lecture Notes in Electrical Engineering 493 Wireless Communication Networks and Internet of Things, Springer Singapore, vol. VI, 2019.
[2] L. M. Camarinha-Matos, S. Tomic, and P. Graça, Eds., Technological Innovation for the Internet of Things, vol. 394. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
[3] A. Suzdalenko and I. Galkin, "Instantaneous, Short-Term and Predictive Long-Term Power Balancing Techniques in Intelligent Distribution Grids," 2013, pp. 343–350.
[4] S. Rani and S. H. Ahmed, Multi-hop Routing in Wireless Sensor Networks. Singapore: Springer Singapore, 2016.
[5] M. Klymash, N. Peleh, O. Shpur and I. Lutsiuk, "Clustering model of cloud centers for big data processing", 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, 2018.
[6] A. Orda and R. Rom, "Optimal Routing with Packet Fragmentation in Computer Networks," Int. J. Networks, vol. 29, no. 1, pp. 11–28, 1997.
[7] P. Kuila and P. K. Jana, Clustering and Routing Algorithms for Wireless Sensor Networks, 2018.
[8] M. Klymash, O. Lavriv, T. Maksymyuk and M. Beshley, "State of the art and further development of information and communication systems", IEEE International Scientific Conference "Radio Electronics and Info Communications", 2016.
[9] O. K. Sulaiman, A. M. Siregar, K. Nasution, and T. Haramaini, "Bellman Ford algorithm - In Routing Information Protocol (RIP)," J. Phys. Conf. Ser., vol. 1007, no. 1, 2018.
[10] J. B. Singh and R. C. Tripathi, "Investigation of Bellman-Ford Algorithm, Dijkstra's Algorithm for suitability of SPP," 2018.
[11] L. Wang, Y. Cui, I. Stojmenovic, X. Ma, and J. Song, "Energy efficiency on location based applications in mobile cloud computing: A survey," Computing, vol. 96, no. 7, pp. 569–585, 2014.
[12] W. Jackson, JSON Quick Syntax Reference, Apress, 2016.
[13] R. Reagan, Web Applications on Azure. Berkeley, CA: Apress, 2018.
[14] S. Machiraju and S. Gaurav, Hardening azure applications : techniques and principles for building large-scale, mission-critical applications, Apress, 2019.
[15] L. Carlson, Programming for PaaS. O'Reilly Media, Inc, 2013.

[16] "What is PaaS? Platform as a Service | Microsoft Azure." [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-paas/. [Accessed: 01-Feb-2019].
[17] M. Grinberg, Flask Web Development, 2014.
[18] B. Swen, "Outline of initial design of the Structured Hypertext Transfer Protocol," J. Comput. Sci. Technol., vol. 18, no. 3, pp. 287–298, May 2003.
[19] S. Patni, Pro RESTful APIs, 2017.
[20] S. Guha, Computer Graphics Through OpenGL® : from theory to experiments, CRC Press, 2019.
[21] K. Geisshirt, A. Olsson, R. Voss, and E. Zattin, Git version control cookbook : leverage version control to transform your development workflow and boost productivity, 2014.
[22] R. Reagan, Web applications on Azure : developing for global scale, Apress, 2018.
[23] M. Ravasi and M. Mattavelli, "High-level algorithmic complexity evaluation for system design," J. Syst. Archit., vol. 48, no. 13–15, pp. 403–427, 2003.
[24] "Algorithmic complexity." [Online]. Available: https://devopedia.org/algorithmic-complexity. [Accessed: 01-Feb-2019].
[25] Anna G. Chunovkina, Leonid A. Mironovsky, Valery A. Slaev, Metrology and theory of measurement, 2013.