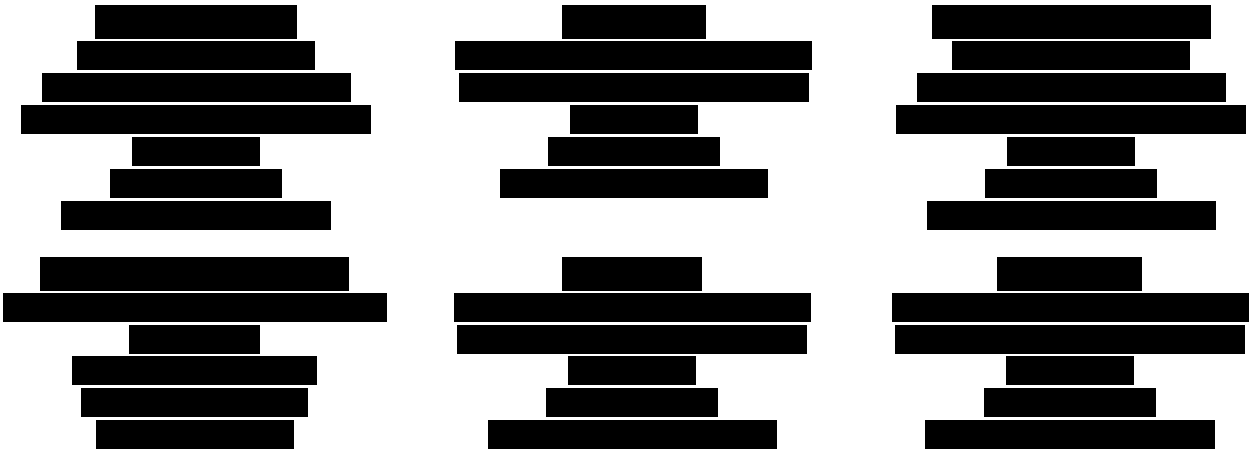
A novel SDN controller based on Ontology and Global Optimization for heterogeneous IoT architecture



ABSTRACT

Today, our modern living world is covered by ubiquitous systems that offer the ability to assess, understand and handle the environmental indicators of our urban environment as well as our delicate ecology and natural resources. The sudden increase of such systems creates the notion of Internet of Things (IoT). However, the variety of components and wide-area deployments in IoT create a disadvantage point: the heterogeneity problem. Such system has multiple heterogeneous wireless communication solutions with multiple access technologies such as bluetooth, wifi, zigbee, cellular, MANET, etc. Concretely, the effectiveness is revealed by a variety of access technologies that are working on a common core network with a common policy for every type of access network. The challenge is how to manage this heterogeneous network in a dynamic context with an open and distributed infrastructure. One of the most efficiency solutions for this issue is the Software-Defined Network approach (SDN). This paper proposed a new SDN-based architecture with a centralized controller that has a capacity of self-observing and adapting. The SDN controller has the ability to incorporate and support user requests to classify flow scheduling over task-level. Besides, the paper creates an Ontology for analyzing user's request and based on the Lagrange relaxation theory for a heuristic routing algorithm. The experimental works showed that the proposed solution yielded impressive and good results.

CCS CONCEPTS

• **Networks** → **Network services**;

SoICT '17, December 7–8, 2017, Nha Trang City, Viet Nam

ACM ISBN 978-1-4503-5328-1/17/12…$15.00

KEYWORDS

Software-defined network, Internet of things, Lagrange relaxation

ACM Reference Format:
Hai Anh TRAN, Duc TRAN, Linh Giang NGUYEN, Abdelhamid MELLOUK, Hieu MAC, and Van TONG. 2017. A novel SDN controller based on Ontology and Global Optimization for heterogeneous IoT architecture. In *SoICT '17: Eighth International Symposium on Information and Communication Technology, December 7–8, 2017, Nha Trang City, Viet Nam.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3155133.3155143

1 INTRODUCTION

Nowadays, there are 9 billion connected IoT devices and they expect a number of 24 billion for 2020 [11]. There is a wide variety of such devices like sensors, actuators, cameras, smartphones, etc. Such IoT devices are placed in variety of access networks in using different access technologies such as bluetooth, wifi, zigbee, cellular, MANET. Moreover, there are diversity of IoT applications implemented on top of these access networks such as building automation, smart cities, smart manufacturing, health care, automotive, etc. However, all these various elements (devices, access technologies, applications) use one common traditional core network infrastructure with traditional equipment and network protocols. The latter is not designed to support the high level of scalability, high amount of traffic and mobility. In other words, the traditional core network infrastructure cannot support different IoT tasks characterized by a very heterogeneous networking context. Concretely, we need to develop variety of policies for the core network to adapt heterogeneous IoT tasks. It is fascinating that these policies can be easily and dynamically implemented in the core network without the needs of a standard protocol.

Sofware Defined Networking (SDN) is a good solution for the above issue [12, 22]. SDN is considered as a network architecture that is able to eliminate the inflexibility in traditional core network. Its idea of programmable networks facilitated network evolution in which the forwarding device is decoupled from control layer. Its

structure makes the behavior of the core network to be more flexible and adaptive to the requested quality of service of each access network, each application and each device. In addition, its centralized architecture gives capacity of collecting information/data and uses it to improve and adapt their policies instead of manually transforming these high level-policies into low-level configuration commands.

In order to overcome the heterogeneity issue above and profit the mentioned SDN's advantages, this paper proposed a layered SDN controller in IoT scenario. The SDN paradigm is a good solution to solve the heterogeneity issue of IoT environments with the following reasons: i) SDN architecture provides an obvious separation between services in the control plane and the data plane. Therefore, we can make decision about how traffic is handled in the control plane and choose mechanisms for forwarding traffic in the data plane; ii) SDN also balances the level of centralized control of SDN controller and the degree of decentralized operations through flow-based routing. This balance has been made by the interactions between controller and the forwarding devices.

The paper is structured as follows: In section 2, the paper presents some related works of SDN-based IoT architecture. Section 3 will focus on the proposed approach in describing and analyzing the layered SDN controller. In section 4, the paper describes the implementation and shows some experimental results. We end the paper with a conclusion and perspective.

2 RELATED WORK

This section will give an brief overview of how academic researchers today applied SDN to IoT scenario to improve the system performance and overcome the issues in IoT network.

Huang et al. [13] proposed a framework consisting of a set of nodes apply M2M protocol, a gateway to handle the devices which are not support M2M protocol, a set of another nodes and a controller to manage all of these types of devices. Once the routing information is changed, the controller transfer the new version to the agent in the objects to update the routing information on each one. Therefore, the durability of the IoT network will be improved. However, authors did not consider the QoS of different data flows. They applied the same routing policy for all type of flows.

In [17], authors used SDN to allow different objects from different networks to communicate with each other by using IPv6 and at the same time simplify the management and control operations of many object types by inserting an additional IoT controller over the SDN controller. After determining and establishing the forwarding rules, the IoT controller sends these rules to the SDN controller which forwards it to the forwarding devices. The purpose is to let different heterogeneous objects in the network to communicate with each others.

In [16, 20], the authors designed a SDN-based IoT framework in implementing NFV (Network Function Virtualization). Their idea is to exploit all advantages of SDN architecture to construct a SDN-based IoT framework that is able to meet the requirements of the characteristics of diversity and dynamics of a IoT context. However, the authors only give the general and theoretical ideas without experimental work. Vilalta et al. [23] proposed an end-to-end orchestration for IoT services using an SDN/NFV-enabled edge node under SDN but like the above proposals, they did not implement it and there was no experimental verification as a consequence.

Jararweh et al. [14] proposed the SDIoT architecture model in combining the idea of SDStore [5] and SDSec [2]. The architecture consists of three main components: 1) The physical layer where all the assets and hardware devices in the system reside. This layer is classified into some clusters such as sensor network cluster and database pool cluster; 2) The control layer plays also the role of a middleware. This layer consists of IoT controller, SDN controller, SDStore controller, and SDSec controller, that are entirely software-based controllers which abstract the control and management operations from the underling physical layer; 3) Application layer is simply combined many fain-grained user applications which simplifies the accessing and acting with the stored data by the end users through the Northbound APIs (N-APIs). Unfortunately, authors did not test their proposed work with experiments yet.

Chakrabarty et al. [3] proposed a SDN-based architecture, called Black SDN, for IoT system in focusing in the security issue. Black SDN is used to secure both the meta-data and the payload within each layer of an IoT communication packet. The authors consider the SDN centralized controller as a trusted third party for secure routing and optimized system performance management. However, the authors did not consider different types of attack scenario. With the same security purpose, Olivier et al. [19] proposed a SDN-based architecture for IoT context with distributed controllers. This proposed architecture consists of multiple domains where the SDN controllers are installed. The authors confirmed that such architecture has ability to guarantee the security of the entire network. However, they did not implement this architecture even in the simulation scenario.

It is clear that there is a lack of evaluation and testing with concrete experiments in all the above mentioned works. Moreover, they did not consider the system operation as a transparent processing task from the end-user (top-level) downto the IoT devices (the bottom level). This paper actualized this with the proposed architecture that plays the role of a bridge between the end-user and underlying layer.

3 PROPOSED SDN-BASED ARCHITECTURE FOR IOT SYSTEM

To overcome all the above issues, the paper proposes a layered SDN controller architecture (Fig. 1). The goal is to create a controller that plays a role of a middleware giving the transparency for user from the top layer to the bottom ones. The controller is developed based on an open source platform called *Floodlight* [1]. The paper decided to use the Floodlight platform due to the fact that Java is our most familiar language. The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks. Although the Floodlight project is discontinued, its latest version is sufficient for our development.

Since the IoT scenario is very dynamic, the *State info DB* is the database that stores on the fly the state information of the network such as the topology, the link state, the joining or leaving nodes, etc. The user's commands that determine what user needs are placed at

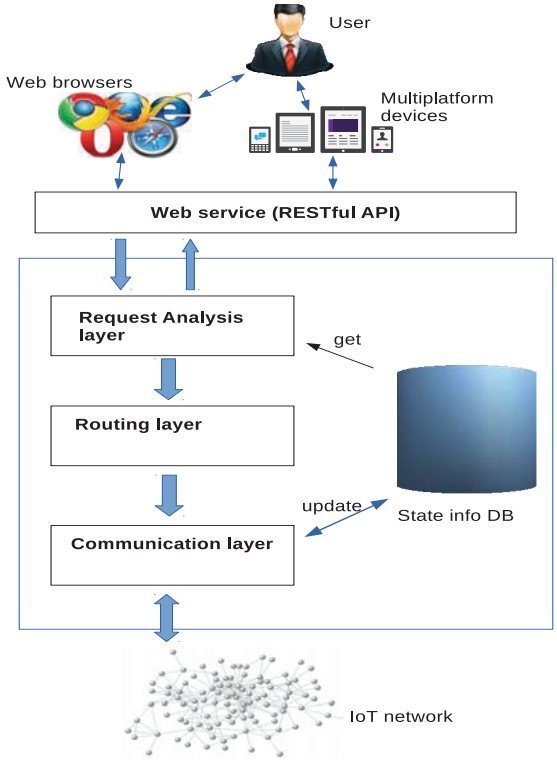


Figure 1: Layered SDN controller architecture.

the highest layer of abstraction. The paper constructs also a web service that provice RESTful API to end-users so that they can use the service in using any platform on any devices. The transparency offered by the proposed architecture is expressed by the independence from applications, devices and networks, which are exploited after to accomplish the required tasks. For example, the user's command is to count the number of people in the Room01. The controller will analyze this command and determine which service will be chosen. The service may be either to capture the video in the Room01 and count people inside, or activate the counting sensor in the door of this room. So, the devices are either the camera or counting sensors. In other words, the controller will map which devices and which application to complete the task. The lower layers will choose the networks and the path to route the data flows. Finally, all these taken decisions will be sent down to the communication layer, that will apply it to the selected devices. The operation of the Communication layer has been processed by the network emulator *mininet* [6] we used in the testbed. In this layer, it is essential to have an interface for communicating between data plane and control plane. There are variety of SBI protocols (e.g. ForCES [8]), but the most typical example is OpenFlow[18]. The paper has chosen OpenFlow because of its widely use and its advantages described in [18].

3.1 Resquest Analysis layer

As mentioned above, the IoT system always faces the challenge of heterogeneity of network and devices. Therefore, the Request

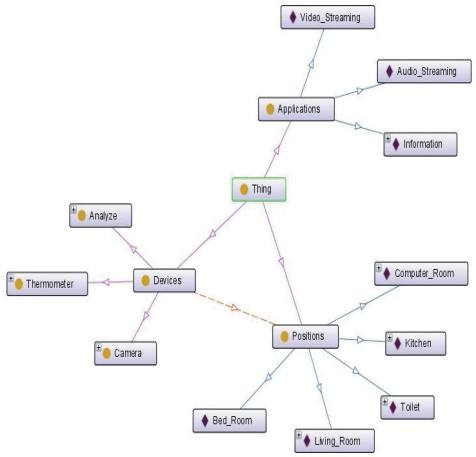


Figure 2: The ontology of the IoT system.

Analysis layer is used to provide an abstract layer to end-users so that the IoT works are independent of the underlying network and devices. In other words, this layer plays the role of a bridge between mission request and underlying layer.

This paper constructed an ontology and used semantic technology to describe the context of IoT network as well as the devices with its characteristics and capacity. The Figure 2 shows a compact version of the ontology. The latter possesses three main classes as follows:

- Devices: That describes all type of device in the system including the Camera, thermometer and analyzer (the PC).
- Positions: The positions where devices locate such as the kitchen, the computer room, the restroom, etc.
- Applications: The paper considers three applications: Video streaming, Audio streaming and Information (e.g. sensing info like temperature, humidity, light, etc.)

Based on the above classes in the ontology, the *Positions* and *Devices* are used to determine the sender and receiver nodes. The paper assigns each concrete application in *Applications* class the information of the requested maximum latency. Alternatively, the ontology is opened, therefore any other organizations can customize it to make it appropriate for their use.

Since the considered QoS metric is the maximum latency (delay) for each task, one of the three outputs of this layer is the minimum delta delay Δ_{delay}. The two remain output parameters are the sender and receiver node.

Concretely, the paper builds two modules for this layer: The module *User expression analysis* simply scans the expression text of the user and detect the keywords that are related to the constructed ontology. In turn, the *Onto analysis* module use these keywords as input and base on the Ontology data to determine the appropriate application and the IoT devices. Finally, it can give three outputs: *the delta delay* Δ_{delay}, *the sender node* and *the receiver node*. These outputs will be sent down to the *Routing layer*.

3.2 Routing layer

Based on the results calculated in the request analysis layer, the role of the Routing layer is to find an optimal path to route the data from the selected source to the destination nodes. The goal is to guarantee that the latency time does not overcome the calculated value Δ_{delay} and optimize the cost function in the same time.

To accomplish this purpose, the paper proposes a routing algorithm that uses the concept of aggregated costs and finds the optimal multiplier based on Lagrange relaxation. The interest of this algorithm is that it can give a lower bound on the theoretical optimal solution along with the result. The experimental results showed that the difference between the cost and the lower bound is small. In addition, the proposed routing algorithm takes into account the trade-off between the e2e delay and the quality of the found path. Gary et al. [10] proved that a routing problem is NP-complete in the case where the number of QoS metrics that should be minimized are more than or equal to two. Therefore, the paper tried to define a simpler problem instead of attack the complex problem. For us, the delay along the path should be acceptable, and the cost of the path should be as low as possible. Concretely, the routing task is to find a path that is minimal for a cost and the delay remains under a given bound. The delay bound is determined in the Request Analysis layer. In other words, the paper formulates the problem to the Delay Constrained Least Cost path problem (DCLC) [21].

The DCLC problem is formulated as follows: A communication network is modeled as a directed and connected graph $G = (V, E)$, where E denotes the set of directed links and V represents the set of nodes (e.g. switches, routers) connected by directed links. Any node is reachable from any other node in this graph. Every directed link $e = (u, v) \in E$ has a delay $D(e)$ and cost $C(e)$ correlated with it. The link delay $D(e)$ reflects a measure of the delay a packet experiences when passing through the link e. The link cost $C(e)$ represents some other metrics to optimize such as the loss-rate, bandwidth, jitter, etc. The Eq. 1 has been chosen for the link cost.

$$C(e) = w_1 * l_e + w_2 * b_e \tag{1}$$

where w_1 and w_2 are weight constants of loss-rate (l_e) and bandwidth (b_e) metric of the link e, respectively.

The constrained minimization problem is represented as follows:

$$\min_{r \in R'(x,y)} \sum_{e \in r} c(e) \tag{2}$$

where R' is the set of routing path from source x to destination y for which the e2e delay is bounded by Δ_{delay} determined by the Resquest Analysis layer. We have also $R'(x, y) \subseteq R(x, y)$. A $r \in R(x, y)$ is in $R'(x, y)$ if and only if:

$$\sum_{e \in r} d(e) \leq \Delta_{delay}$$

In order to solve the NP-complete problem of DCLC, the paper uses a heuristic algorithm based on the theory of Lagrange relaxation. The latter is considered as a common technique for determining lower bounds and finding solutions for this problem. The idea is based on the heuristic of minimizing the modified cost function. The cost function in the Eq. 1 is modified in adding the influence of delay with a parameter γ to form an aggregate weight for each link (Eq. 3).

$$c_\gamma = c_{\text{old}} + \gamma.d \tag{3}$$

where d is the delay,
c_{old} is the cost that is calculated in using the Eq. 1.

So, for a given γ, the minimal path, denoted by r_γ, can be found in using Dijkstra's algorithm w.r.t. the calculated cost (Eq. 3). The optimal path is denoted in using Dijkstra's algorithm w.r.t. cost with parameter γ as $\mathcal{R}_\gamma$. If the total delay of the path $\mathcal{R}_\gamma$, denoted by $D(\mathcal{R}_\gamma)$, is less or equal than Δ_{delay}, it is the optimal solution. Otherwise, if $D(\mathcal{R}_\gamma) > \Delta_{delay}$, the value of γ is increased in order to increase the influence of the delay factor in the cost function in Eq. 3.

Before describing the algorithm, the relationship between the parameter value, the cost and the delay of the path can be illustrated by the following lemmas.

Lemma 3.1. *If* $0 \leq \gamma_1 \leq \gamma_2$ *then* $D(\mathcal{R}_{\gamma 1}) \leq D(\mathcal{R}_{\gamma 2})$ *and* $C(\mathcal{R}_{\gamma 1}) \geq C(\mathcal{R}_{\gamma 2})$.

Proof. See [9]. □

Therefore, the Lemma 3.1 showed that the larger the parameter, the larger the cost, while the smaller the delay. It shows that as long as the resulting shortest path does not violate the Δ_{delay}, a smaller parameter γ will definitely give rise to a better solution. The next lemma is used to find the smallest parameter with the corresponding shortest path that does not violate the Δ_{delay}.

Lemma 3.2. *If* $\gamma_1 < \gamma_2$, $D(\mathcal{R}_{\gamma 1}) \neq D(\mathcal{R}_{\gamma 2})$, $\gamma' = \frac{C(\mathcal{R}_{\gamma 1}) - C(\mathcal{R}_{\gamma 2})}{D(\mathcal{R}_{\gamma 2}) - D(\mathcal{R}_{\gamma 1})}$, *then* $C(\mathcal{R}_{\gamma 1}) \geq C(\mathcal{R}_{\gamma'}) \geq C(\mathcal{R}_{\gamma 2})$, $D(\mathcal{R}_{\gamma 1}) \leq D(\mathcal{R}_{\gamma'}) \leq D(\mathcal{R}_{\gamma 2})$.

Proof. See [9]. □

The Lemma 3.2 shows that with $\gamma' = \frac{C(\mathcal{R}_{\gamma 1}) - C(\mathcal{R}_{\gamma 2})}{D(\mathcal{R}_{\gamma 2}) - D(\mathcal{R}_{\gamma 1})}$, the shortest path $\mathcal{R}_{\gamma'}$ must have a delay between the delays of $\mathcal{R}_{\gamma 1}$ and $\mathcal{R}_{\gamma 2}$, and the cost is also between the costs of these two paths.

The heuristic algorithm is based on these Lemmas above. The *least cost path* (found by using Dijkstra's algorithm w.r.t. cost, $\mathcal{R}_c$) is first obtained, and if his delay is not greater than Δ_{delay}, then it must be the optimal solution. Otherwise, the *least delay path* (found by using Dijkstra's algorithm w.r.t. delay, $\mathcal{R}_d$) is found, and if it is greater than the Δ_{delay} then there is no solution for that. If none of the above conditions is true, the algorithm begins an iterative procedure. In each iteration, either $\mathcal{R}_d$ is updated with a better solution having a lower delay or $\mathcal{R}_c$ is updated with a better solution having lower cost.

The Algorithm 1 is described in detail as follows: First, the paper uses the original cost function (Eq. 1) to find the *least cost path* in using Dijkstra's algorithm. If the delay of this path meets the delay requirement Δ_{delay}, so it is the optimal path. Otherwise, we find the *least delay path* and examine if the delay of this path is greater than Δ_{delay}, we conclude that there is no solution and the algorithm stops. If not, we start the loop. Now, it's time to add the delay to the cost function (Eq. 3). The parameter γ is assigned to the initial value:

Algorithm 1 Lagrange relaxation-based routing algorithm

Require: $source, dest, c, d, \Delta_{delay}$
Ensure: Optimal path
1: $\mathcal{R}_c \leftarrow$ Dijkstra($source, dest, c$)
2: **if** $d(\mathcal{R}_c) \leq \Delta_{delay}$ **then return** $\mathcal{R}_c$
3: **end if**
4: $\mathcal{R}_d \leftarrow$ Dijkstra($source, dest, d$)
5: **if** $d(\mathcal{R}_d) > \Delta_{delay}$ **then return** "No solution"
6: **end if**
7: **while** true **do**
8: $\gamma := \frac{C(\mathcal{R}_c)-C(\mathcal{R}_d)}{D(\mathcal{R}_d)-D(\mathcal{R}_c)}$
9: $\mathcal{P} \leftarrow$ Dijkstra($source, dest, c_\gamma$)
10: **if** $C_\gamma(\mathcal{P}) = C_\gamma(\mathcal{R}_c)$ **then return** $\mathcal{R}_d$
11: **else if** $D(\mathcal{P}) \leq \Delta_{delay}$ **then**
12: $\mathcal{R}_d \leftarrow \mathcal{P}$
13: **else**
14: $\mathcal{R}_c \leftarrow \mathcal{P}$
15: **end if**
16: **end while**

Table 1: Computer specifications for testbed

	CPU	RAM	OS
Computer 1	Intel Core i5-4200U 1.6GHz	4GB	Windows 10 64bit
Computer 2	Intel Core i3-4150 3.5GHz	8GB	Linux

$$\gamma := \frac{C(\mathcal{R}_c) - C(\mathcal{R}_d)}{D(\mathcal{R}_d) - D(\mathcal{R}_c)}$$

The Dijkstra's algorithm is used w.r.t the new value of cost c_γ. If the value of cost of the new path found is equal to the cost value of the *least cost path*, the optimal path is the *least delay path*. If not, if the delay of the new path found meets the delay requirement Δ_{delay}, the *least delay path* is updated to the new path. Otherwise, the *least cost path* is updated to the new path. After, the loop is repeated.

4 EXPERIMENTAL RESULTS

In order to have the experimental results, the paper used a simulation testbed to evaluate the performance of the proposed controller with different scenarios.

Concerning the concrete testbed, the paper used two computers with specifications described in Tab. 1. The controller program and the web service are installed in the Computer 1. In the Computer 2, the paper used *mininet* [6] to emulate the network topology with nodes representing Openflow-enabled switches and IoT devices. Tab. 1 describes the specifications of these two PCs. The user is able to send request to the Computer 1 in using Restful API through any platforms, any devices. The paper developed also a mini Android application that sends the RESTful-based request to the web service of the system.

In order to evaluate accuracy of the *Request Analysis layer*, we asked 200 students to install the developed mini Android application and send the 30 request to the system (they can eventually use their web browser), 10 requests per application. There are totally 6000 requests (2000 requests per application). In the testbed, the Android device specification can be ignored because our mini Android application is very lightweight with only one simple operation of sending the request to our web service and receiving the result in following the RESTful API.

Table 2: Evaluation of accuracy for three applications

Application name	Total num	Exact num	Prop
Video app	2000	1987	99.3 %
Audio app	2000	1992	99.6 %
Information app	2000	1893	94.6 %

Tab. 2 shows that majority of user requests have been analyzed exactly for Video streaming (99.3%) and Audio streaming (99.6%) applications. The faults mostly come from the Information application. That is explained by the fact that the *Request Analysis layer* is based on the analysis and processing of text strings, therefore it is easier for the concrete application name like video or audio. For the information application that includes variety of sensing information such as temperature, humidity, etc. is more difficult to identify the user task. However, the proportion exactitude of 94.6% is acceptable.

Concerning the performance of the routing layer where we applied the routing algorithm (we call it shortly **LARE** from now on), the paper implemented some other routing algorithms that also focus on the DCLC problem:

- The Constrained Bellman-Ford (**CBF**)routing algorithm: In [24], Widyono proposed a routing algorithm based on a breadth-first search that help to exploit paths in increasing delay while updating lowest cost path in each visited node. The algorithm runs until either the highest constraint is exceeded or it cannot improve the paths anymore.
- The Multi-Criteria Routing algorithm (**MCR**) proposed by W.C. Lee et al. [15]: This routing algorithm is based on heuristics of a ranked metrics in the network. The algorithm determines alternately the shortest path for each metric until the best path is found, or it fails for all metrics.
- The routing algorithm proposed by Cheng et al. [4]: This algorithm combines the problem of finding the least cost and least delay paths by modifying the cost function (we call that **MCF** algo for short). In other words, authors try to calculate a simple metric from multiple requirements based on a single cost aggregated as a combination of weighted QoS metrics.

The comparison of the experiment is based on some measures of the implemented algorithms as follows:

- *Number of fails*: In other words, it is the average of the number of unreachable nodes. This is the case where the path cannot be found with a given delta delay Δ_{delay}. This measure is used to assess the efficiency of the algorithm in finding the destination nodes.
- *Delay*: This is the average of delay time of the paths from a source node to all reachable destination nodes.
- *Cost*: The average cost of the paths from a source node to all reachable destination nodes.

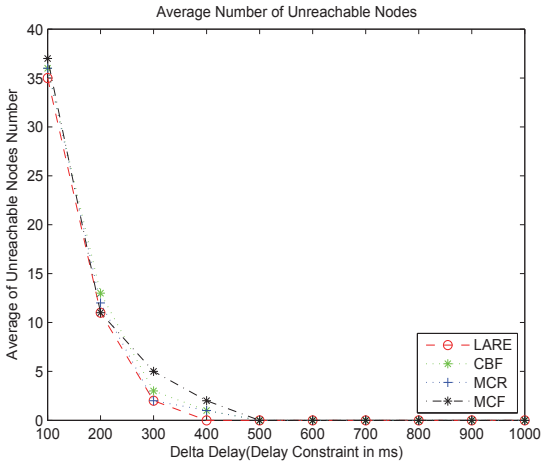


Figure 3: Average Unreachable Node Number.

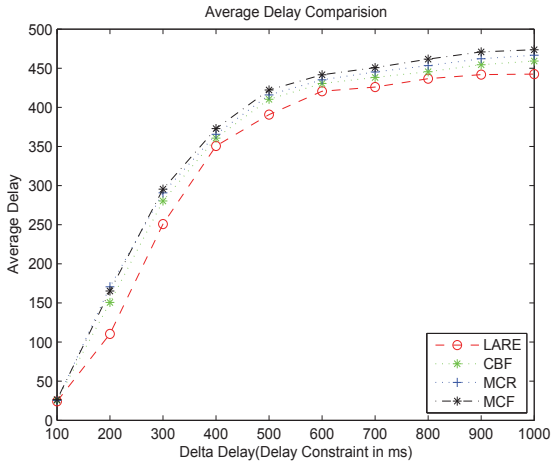


Figure 4: Average Delay.

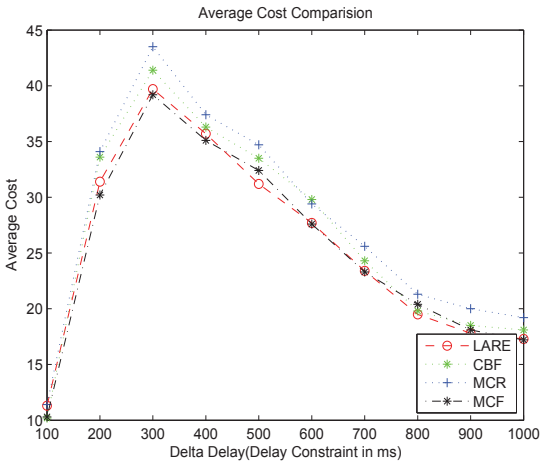


Figure 5: Average Cost.

Concerning the network topology, first, the paper tested the routing algorithm on a simple network with 17 nodes to verify its functionality. After, the paper worked on NTT network topology [7] that is modeled on the exact characteristics of a real-world network, the former NTT (Nippon Telephone and Telegraph company) fiber-optic corporate backbone.

Following the NTT network topology, there are totally 37 nodes in the testbed. In order to evaluate the performance of routing algorithms, we vary the value of delta delay Δ_{delay} from 100 to 1000 (ms). In the Figure 3, we can see the average of unreachable node number in varying the delta delay from 100 to 1000 ms. With the delta delay smaller than 200ms, almost the algorithms find small number of destination nodes, therefore the number of unreachable nodes is quite high. After the point of 400ms, all algorithms is able to find all the paths.

The Figure 4 shows the average delay curves of 4 algorithms. We can see that with the low values of delta delay (100-300 ms), the number of reachable nodes is small.

The destination nodes are close to the source nodes, therefore that makes the delay values low too. After the value of 400 ms of delta delay where almost all the paths can be found for all source and destination nodes, the average delay value falls into a steady state. Now, we can see that the LARE algorithm gives the best result in term of average delay. The MCF algorithm gives the worst value due to its policy of combining all metrics. In fact, it's difficult to select an appropriate aggregating weights.

Figure 5 gives us the average cost of the paths found by the 4 algorithms. As we explained above, in the beginning ($\Delta_{delay} < 300$) where it is impossible to find path for all destination nodes and the found paths are very short, therefore the costs are small too. The algorithms find more paths when we increase the delta delay, therefore the average cost increases too. After the moment where all destination can be found ($\Delta_{delay} \geq 300$), the average cost of the paths will decrease with the delta delay due to the fact that the algorithms can now find the paths with lower costs and higher delta delay. We can see that the LARE algorithm gives almost the same good result as MCF. This result is exactly what we expected because MCF focuses on optimizing the cost value. The MCR has the worst result due to the fact that it only find the best path for one metric in ignoring the cost value.

The paper also tested the capacity of supporting the stress of the system in comparing with traditional core network (without SDN architecture) that applies the simple best-effort policy. The paper tried to stress the system in generating incrementally the request number from 100 to 2000 requests per second.

We fixed the delta delay to the value of 1000ms to make sure that all paths can be found. Figures 6 and 7 show that the proposed system with the SDN architecture is obviously better than the traditional one in term of delay and cost. We can see that even in the most stressed state (request number = 2000), the proposed system can obtain a good result of average delay (694ms) and cost (33). Otherwise, in the same point, the traditional system got a average delay of 935ms and cost of 69. This superiority of performance in term of both delay and cost can be explained by the fact that the

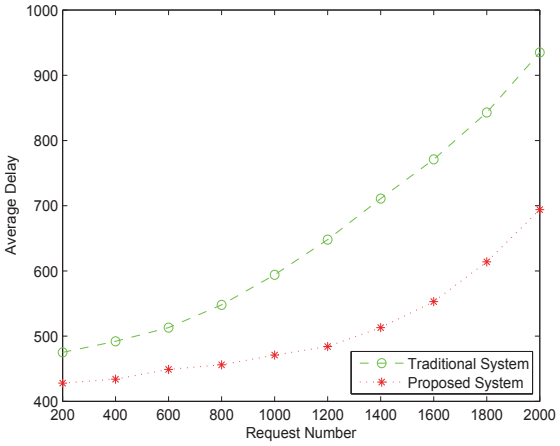


Figure 6: Capacity of supporting stressed network in term of delay.

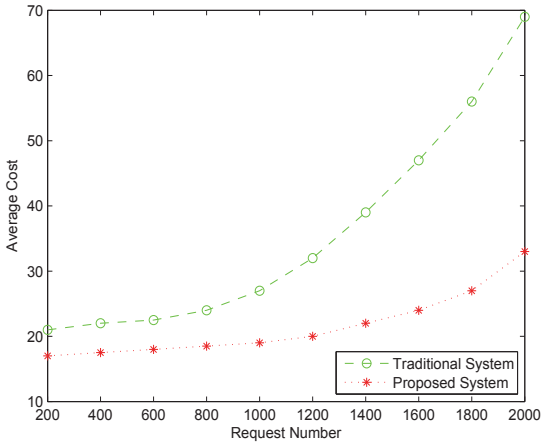


Figure 7: Capacity of supporting stressed network in term of cost.

core network of the traditional system is implemented a naive and simple policy (the best-effort). Otherwise, The proposed system possess a layered architecture with the Request analysis layer that analyzes the user requests and classifies the request types in order to give an appropriate network constraint (delta delay). The Routing layer in the proposed system applies a heuristic routing algorithm basing on the Lagrange relaxation theory. That is the reason the proposed system is able to support the stress condition with a huge number of requests in a moment.

The paper tested also the system performance while providing the video application. Concretely, video flows are generated (video streaming) in using the open source software VLC between nodes in the network created by mininet. The test lasts five minutes. The result of delay and jitter is calculated by cumulating for each chosen path. In Figure 8, we can see that the proposed system

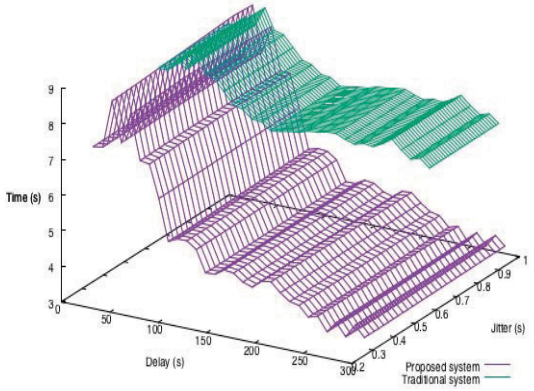


Figure 8: Video streaming application.

gives a better result in term of delay and jitter. The average end-to-end delay of the proposed system is about 3.5s while traditional system is about 7.3s. In term of the jitter parameter, the proposed system gives the result of 0.4s while the traditional system is about 0.6s. This is exactly the results we expected because the proposed system has well differentiated the data flow's type (e.g. video, audio, information data).

5 CONCLUSION

In this work, we have proposed a layered architecture of a SDN controller designed for a heterogeneous IoT environment with the variety of devices and applications/services. The system is considered as a bridge to overcome the gap between abstract high level tasks and specific low level network/device. Experimental results showed that the proposed routing method yielded better performance when compared with related ones. The proposed controller enables flexible, efficient and effective handling tasks on users requests, network and resources. Concerning the future works, we are in the process of implementing the controller into a real testbed with real devices (e.g. Openflow-enabled switches).

6 ACKNOWLEDGMENTS

[redacted]

REFERENCES

[1] [n. d.]. The Floodlight Open SDN Controller project. http://www.projectfloodlight.org/floodlight/. ([n. d.]).
[2] Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, Andy Rindos, et al. 2015. Sdsecurity: A software defined security experimental framework. In *Communication Workshop (ICCW), 2015 IEEE International Conference on*. IEEE, 1871–1876.
[3] Shaibal Chakrabarty, Daniel W Engels, and Selina Thathapudi. 2015. Black SDN for the Internet of Things. In *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE, 190–198.
[4] Shigang Chen and Klara Nahrstedt. 1998. On finding multi-constrained paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, Vol. 2. IEEE, 874–879.
[5] Ala Darabseh, Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. 2015. Sdstorage: a software defined storage experimental

framework. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 341–346.

[6] Rogério Leão Santos de Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, and Ligia Rodrigues Prete. 2014. Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE, 1–6.

[7] G. Di Caro, F. Ducatelle, and L.M. Gambardella. 2005. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications* 16, 5 (2005), 443–455.

[8] Avri Doria, J Hadi Salim, Robert Haas, Horzmud Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. 2010. *Forwarding and control element separation (ForCES) protocol specification*. Technical Report.

[9] G Feng and C Doulgeris. 2001. Fast algorithms for delay constrained leastcost unicast routing. *shortened version presented on INFORMS* (2001).

[10] Michael R Gary and David S Johnson. 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. (1979).

[11] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645–1660.

[12] Fei Hu, Qi Hao, and Ke Bao. 2014. A survey on software-defined network and openflow: from concept to implementation. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2181–2206.

[13] Hai Huang, Jiping Zhu, and Lei Zhang. 2013. An SDN_based management framework for IoT devices. In *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET*. IET, 175–179.

[14] Yaser Jararweh, Mahmoud Al-Ayyoub, Elhadj Benkhelifa, Mladen Vouk, Andy Rindos, et al. 2015. Sdiot: a software defined based internet of things framework. *Journal of Ambient Intelligence and Humanized Computing* 6, 4 (2015), 453–461.

[15] WC Lee and Michael Hluchyj. 1994. Multi-criteria routing subject to resource and performance constraints. In *ATM Forum*, Vol. 94. 0280.

[16] Jie Li, Eitan Altman, and Corinne Touati. 2015. A General SDN-based IoT Framework with NVF Implementation. *ZTE communications* (2015), 1–11.

[17] Pedro Martinez-Julia and Antonio F Skarmeta. [n. d.]. *Extending the Internet of Things to IPv6 with Software Defined Networking*. Technical Report. Euchina-fire. http://www.euchina-fire.eu

[18] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[19] Flauzac Olivier, Gonzalez Carlos, and Nolot Florent. 2015. New Security Architecture for IoT Network. *Procedia Computer Science* 52 (2015), 1028–1033.

[20] Nathalie Omnes, Marc Bouillon, Gael Fromentoux, and Olivier Le Grand. 2015. A programmable and virtualized network & it infrastructure for the internet of things: How can nfv & sdn help for facing the upcoming challenges. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*. IEEE, 64–69.

[21] Douglas S Reeves and Hussein F Salama. 2000. A distributed algorithm for delay-constrained unicast routing. *IEEE/ACM Transactions on Networking (TON)* 8, 2 (2000), 239–250.

[22] Vishwapathi Rao Tadinada. 2014. Software Defined Networking: Redefining the Future of Internet in IoT and Cloud Era. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 296–301.

[23] Ricard Vilalta, Arturo Mayoral, David Pubill, Ramon Casellas, Ricardo Martínez, Jordi Serra, Christos Verikoukis, and Raul Muñoz. 2016. End-to-End SDN orchestration of IoT services using an SDN/NFV-enabled edge node. In *Optical Fiber Communication Conference*. Optical Society of America, W2A–42.

[24] Ron Widyono et al. 1994. *The design and evaluation of routing algorithms for real-time channels*. International Computer Science Institute Berkeley.