Edge-assisted Traffic Engineering and applications in the IoT

Nikos Fotiou, Dimitrios Mendrinos, George C. Polyzos
Mobile Multimedia Laboratory,
Athens University of Economics and Business
47A Evelpidon, Athens, 113 62 Greece
{fotiou,mendrinos16,polyzos}@aueb.gr

ABSTRACT

Software-Defined Networking (SDN) promises novel traffic engineering capabilities transforming every network from a mere packet forwarding fabric into an intelligent distribution medium. We extend the functionality of the core SDN component, the SDN controller, to perform path computations over "annotated" topologies. Specifically, we exploit the use of tags, which describe network node properties and capabilities, enabling a new type of network control applications and thus connecting user applications and traffic flows with network decisions and management. Tags can be set and modified in a number of different ways, supporting context awareness and cognition in the network in a lightweight and loosely integrated way. Intelligent applications running at edge nodes may request unicast, anycast, or multicast paths among nodes with specific tags or tag properties, realizing efficiently traffic engineering goals and supporting network slicing, virtualization, finer resource control, and easier management. We illustrate this new capability in the IoT domain by demonstrating how CoAP group communication can be implemented in a seamless, lightweight, and efficient way, releasing the constrained endpoints from the requirement to support IP multicast.

1 INTRODUCTION

Nowadays, it is evident that the Internet has evolved from a facility that interconnects static end-hosts, into a dynamic ecosystem which extends to our physical world. Smartphones, Things and home appliance with interconnection capabilities, portable computing devices, powerful personal workstations, all compose a powerful toolset that offers to end-users endless possibilities. Application providers create novel services that can be accessed through multiple devices, expand to multiple network locations, and generate vast amounts of traffic. Struggling to cope with this vivid environment, network operators seek to transform the traditionally static networks into an elastic, intelligent platform, that will not merely transfer packets from the "dump" edge to the "smart" core, but it will also provide in-network functions and services, making end-user experience even richer. A critical technology allowing the realization of this goal is Software-Defined Networking (SDN) [11].

SDN allows the use of centrally managed switches, enabling network operators to leverage the full capabilities of their network. These switches are programmable in the sense that using a specialized protocol (such as OpenFlow), they can be configured with rules to forward data "flows" towards their destinations through specific node and link paths, facilitating in this way load-balancing, service differentiation, in-network functions, and providing novel network services [5]. At the heart of an SDN-based network is an SDN controller, which is responsible for dynamically setting up switch flow tables based on configurations–usually expressed using a scripting or programming language–that capture the requirements and the business processes of the network operator.

In this paper, we propose an enhanced SDN-based architecture that extends traditional approaches in the following way: (i) network nodes are annotated with "tags" describing their properties and capabilities, (ii) SDN controllers act as "path computation elements" and are capable of calculating on-demand paths among or composed of nodes with specific tags or tag properties, (iii) intelligent applications, running on edge nodes, are able to request paths from the SDN controller, as well as to set/delete their own application-specific tags. These enhancements are supported by a Bloom filter-based forwarding technique that achieves seamless multicast communication. This technique requires minimum and fixed

MECOMM'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5906-1/18/08…$15.00
https://doi.org/10.1145/3229556.3229561

state in the switches, which can be installed during the bootstrap. Despite that the path requesting applications and the path computation process can be application aware and act based on the requirements of a specific application, or even based on the context of a specific user, our approach does not require from the SDN components to perform deep packet inspection (as for example in [3]), neither requires modifications to already established standards (as for example in [6]). As a matter of fact, our proof of concept implementation is based on readily available software switches and the OpenFlow 1.2 standard. We argue that the extensions proposed in this paper, combined with the Bloom filter-based forwarding, although simple, create many opportunities for novel services. In particular, our approach facilitates group/anycast communication paradigms, supports service chaining and composition, improves network management, and facilitates new services and applications. In order to demonstrate the efficiency of our approach, we present how a group communication protocol designed for the Internet of Things (IoT), i.e., the Constrained Application Protocol (CoAP) extension for group communication, can be efficiently implemented over an SDN network that supports tag-based path computation. Traditional CoAP group communication requires from CoAP endpoints to support the IP multicast protocol. Moreover, CoAP group related information is maintained by the endpoints, hence, the number of groups increases and managing them becomes hard. With our implementation, constrained endpoints can benefit from group communication using vanilla CoAP without the extensions, without modifications, and without the need to support IP multicast. Moreover, the creation of a new group becomes easier and the management of existing groups becomes more efficient. Finally, using our approach, service providers can easily leverage existing IoT deployments and offer new, group communication-based services.

The structure of the remainder of this paper is as follows. In Section 2 we present the design of our solution. In Section 3 we present our implementation of an Internet Service Provider's (ISP) network that interconnects CoAP endpoints, which we selected as a use case in order to illustrate more details of our solution, as well as in order to highlight its advantages. Finally, in Section 4 we provide our conclusions and plans for future work.

2 DESIGN

Our design assumes the network of a single ISP implemented using SDN technology. ISP's clients are connected to the network through "intelligent" edge-nodes.

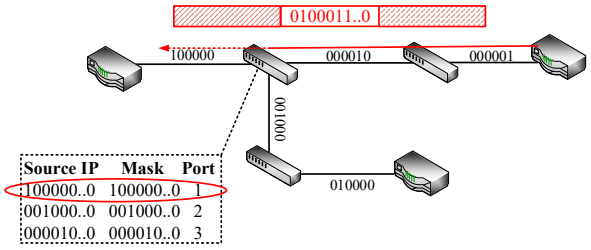

Figure 1: Example of Bloom filter-based forwarding.

2.1 Bloom filter forwarding

For all in-network traffic (i.e., for the portion of the network surrounded by edge-nodes) Bloom filter-based forwarding is used. In particular we assume the Bloom filter-based forwarding approach described by Reed et al. in [9]. To realize this, each link of the SDN network is assigned a unique identifier, which is a fixed-size Bloom filter-based vector. Each link identifier is mapped to an arbitrary bitmask with length equal to two IPv6 addresses and corresponding OpenFlow [1] rules are installed at its adjacent switch interfaces. The forwarding identifier that realizes the communication between edge-nodes is a Bloom filter encoding the link identifiers of the path that a packet should follow, by simply ORing these identifiers. An interesting property of this type of forwarding is that by ORing the encodings of two paths $A \rightarrow B$ and $A \rightarrow C$, the encoding of a multicast path from A to B and C is derived. The solution in [9] encodes Bloom filters in the IPv6 source and destination fields of an IPv6 packet and uses the OpenFlow arbitrary mask match in switches to make forwarding decisions. Therefore, this forwarding technique is implemented using standard OpenFlow mechanisms and without modifying the structure of network packets; to a network monitoring tool, these packets appear to be ordinary IPv6 packets, but with "strange" addresses. Moreover, the appropriate OpenFlow rules related to Bloom filter-based forwarding can be installed when a switch joins the network. With these rules, the switch will not require any further information set by the controller in order to forward a packet, achieving significant gains in terms of network overhead reduction.

Figure 1 illustrates a simplified example of Bloom filter-based forwarding. As it can be seen each link is identified by a (binary) identifier. The path that a packet should follow is encoded in a Bloom filter, constructed by ORing the identifiers of the links that compose the path. For simplicity reasons we assume that the Bloom filter is stored in the Source IP field of the packet (padded with zeros to match the field's length). Each switch port is configured with a network mask and a source IP, both of which match the identifier of

the link connected to that port. For each incoming packet, every switch performs the following actions:

- It retrieves the Source IP address included in the incoming packet
- For each port (except the port from which the packet arrived) it ANDs the source IP of the packet with the network mask associated with the port
- If the output of the AND operation matches the source IP associated with the port, then the packet is forwarded to the link connected to that port

These actions can be achieved using standard OpenFlow rules. For more details interested readers are referred to [9].

2.2 Tags

Each node in the network (i.e., both edge and internal nodes) is associated with a set of tags. These tags may represent network related attributes (e.g., network address, network function), physical location (e.g., room number, geolocation information), application layer functionality or context (e.g., service description, security properties) and many other types of information. Moreover, SDN controllers learn (using mechanisms which are out of the scope of this paper) the network topology, all link identifiers, as well as the tags associated with each node.

Controllers may also recognize types of tags (e.g., integer, geo-location, video quality) and perform some basic operations (e.g., arithmetic comparisons, compare dimensions, ordering, etc).

2.3 Paths

SDN controllers are capable of computing paths (and determine the appropriate Bloom filter identifier) among nodes with specific tags or tag properties, as well as paths passing through or composed of nodes with specific tags. Specific algorithms are required for this and typically this functionality is provided with extra network apps with particular capabilities. A node in the network can send a packet to a (set of) node(s) identified by (some) specific tags. The node may specify that the packet should reach all nodes (multicast), or at least one node in the set or the best available node (anycast). Moreover, the origin may define whether the destination node(s) should be associated with all tags or with some of the tags, or even define an ordered list of tags, which should be reflected in an ordered list of nodes that must be part of the composed path (service chaining). For example, a node may request that a packet should be initially forwarded to a firewall, then to a compression service, and finally to a particular edge-node. If the sender node knows the path towards the destination node(s), then it simply encodes it in a Bloom filter and the aforementioned forwarding procedure is used. If the node does not know the path to the desired destination(s), it creates a special packet, it sets the Ethernet source and destination fields of the packet to a pre-defined constant value and adds the desired tags in the packet's payload. The first SDN switch that receives this special packet will not have a rule to switch it; hence, it forwards it to the SDN controller. The SDN controller extracts the tags, and given its knowledge of the underlying SDN topology, it calculates the appropriate Bloom filter, as well as a Bloom filter for the reverse path (if applicable), and returns this information to the origin switch as an Openflow PacketOut message. Finally, the switch forwards this message to the node that originally sent the packet. That node can now use the provided Bloom filter for all subsequent communications.

3 USE CASE: COAP GROUP COMMUNICATION

3.1 Background

CoAP [10] is a lightweight protocol, designed to be the "HTTP of the IoT." The CoAP interaction model is similar to the client-server model of HTTP: a CoAP client requests a resource from a server; if the resource is available, the server responds, otherwise it simply ACKnowledges (ACK) the request and responds asynchronously when the resource becomes available. CoAP resources are identified by a URI, similar to HTTP URIs. CoAP group communication is a CoAP extension defined in RFC 7390 [8]. This extension allows clients to retrieve or set resources from a group of servers e.g., retrieve the temperature from all sensors of a building, turn on all the lights of a smart city and so forth. When CoAP group communication is used, URI-hosts (e.g., coap://floor1.building6), are mapped to an IP multicast group in which all related CoAP endpoints are members. In order to implement this behavior in a legacy network (a) DNS servers should be modified, and (b) all CoAP servers should join a priori all possible IP multicast groups.

3.2 Implementation

As a proof of concept, we used the mininet network emulator [4], the open vswitch programmable switch [7], and the POX network controller [2] to emulate an SDN-based network of an ISP that allows tag-based path computation. The network topology (including link identifiers and node tags) is included into a JSON-encoded configuration file. A Topology Manager helper class parses this file and provides methods for creating a path among nodes with specific tags. This helper class, implemented using the NetworkX python library, is used by the SDN controller whenever a path is requested by a network node. Moreover, this class is also used by the controller every time a new switch joins the

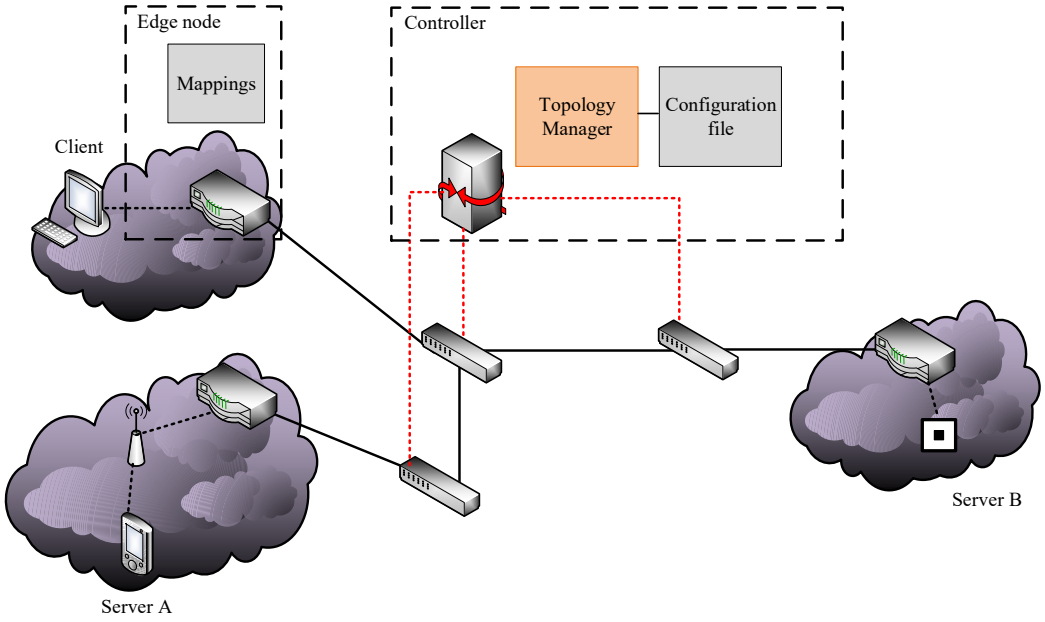

Figure 2: Reference architecture.

network in order to install the appropriate rules related to the Bloom filter-based forwarding.

In addition to the emulated network, we implemented CoAP clients and servers using the libcoap library. Edge-nodes of the ISP's network act as CoAP proxies and CoAP clients are configured to use an edge-node as a proxy. Hence, whenever a CoAP client wishes to send a CoAP (group) request it simply forwards it to an edge-node. Moreover, CoAP servers are associated with "group tags" that have application specific semantics (e.g., "buiding6", "floor1"). This association is implemented in the corresponding edge-nodes and CoAP servers are oblivious about it. Hence, tag-server (de-)association is implemented without communicating with the respective servers. A group name, included in the URI-host field of a CoAP group request, is interpreted as a list of tags and it should be forwarder to all servers associated with these tags. For example, a request for coap://floor1.building6 should be forwarded to all CoAP servers associated with the tags "floor1" and "building6". Edge-nodes that are connected to a CoAP server maintain mappings of group names to IPv6 addresses. These mappings are used for forwarding CoAP requests to the appropriate CoAP server. Moreover, and since CoAP servers cannot handle Bloom filters, edge-nodes replace the Bloom filter included in the transmitted packet with the server's IPv6 address. The implemented architecture is illustrated in Figure 2.

Given the above description, a CoAP group communication-based transaction is implemented by executing the following procedures.

3.2.1 CoAP request. The CoAP client sends the CoAP request to its proxy edge-node. This edge-node extracts the URI-host and splits it into tokens. If this is the first request for that URI-host the edge node executes the "Path Request" procedure, otherwise it executes the "Request Forwarding" procedure.

3.2.2 Path request. The edge node creates a packet with Ethernet destination 00:00:00:00:00:01 and includes in its application payload the list of tokens. Then it forwards this packet to the network. The first programmable switch that receives this packet forwards it to the SDN controller. The SDN controller extracts the token list and requests from the Topology Manager to compute a path from the sender edge-node to all edge-nodes connected to a CoAP server associated with all the desired tags. Then, it creates a new packet that includes in its payload the constructed Bloom filter and sends it back to the programmable switch; the switch in return forwards the received packet back to the sender edge-node.

3.2.3 Request Forwarding. Now the edge-node knows the Bloom filter of the path towards all destinations. It creates a new packet and it uses the IPv6 source and destination fields

to encode the Bloom filter. Moreover, it adds in the application payload CoAP request. Finally, it sends this packet in the network. The packet reaches all destination edge-nodes using Bloom filter-based forwarding. Each node extracts the CoAP request and forwards it to the appropriate CoAP server.

3.2.4 Response Forwarding. Each CoAP server generates a response and forwards it to the corresponding edge-node. The edge-node selects the appropriate Bloom filter, creates a new packet, and forwards it in the network. This packet includes the Bloom filter in the IPv6 address fields and the CoAP response in the payload. This packet eventually reaches the edge-node in which the CoAP client is connected. The latter node extracts the CoAP response and forwards it to the client.

The above procedure is illustrated in Figure 3.

3.3 Evaluation

From the description of our implementation, it is evident that our solution provides the following advantages:

CoAP endpoints do not need to support IP multicast. As mentioned in [10] the recommended approach for implementing CoAP group communication is by using IP multicast. However, IP multicast is feared of becoming a barrier for the adoption of CoAP group communication for many (obvious) reasons

- CoAP endpoints must be preconfigured with all group names and the corresponding IP multicast addresses
- IoT devices should be enhanced to support IP multicast
- Network management will become harder in order to not result in conflicting IP multicast addresses

With the proposed solution, all these drawbacks are overcome.

DNS does not have to be modified. Similarly, RFC 7390 states that URI hosts should be translated into IP multicast addresses using DNS servers. This implies that DNS servers should be configured with all possible group names and the corresponding IP multicast addresses. Even worse, if group names are artificially constructed by applications, DNS servers should implement the application logic used for the group name computation. With our solution, not only DNS servers do not have to be modified, but DNS is not used at all! Indeed, the mapping from a group name to a Bloom filet-based forwarding identifier is performed by the controller using the provided configuration file.

Group management becomes more efficient. By having all groups centrally managed it becomes easier to create a new group, as well as to modify existing ones. For instance, a tag can be assigned to/removed from a CoAP server without informing it. Hence, a server can be added to or removed from a group with no additional communication overhead. Even better, a CoAP server does not have to be aware of the tags with which it is associated. This results in simpler applications running in CoAP endpoints.

Service creation and composition becomes easier. With the proposed solution, a service provider may offer new services using existing resources simply by updating the controller's configuration file. For example, consider the case of an already deployed IoT network composed of temperature sensors deployed in multiple buildings of a smart city. A service provider may provide aggregation services simply by creating a configuration file that contains groups and associated sensors.

Network functions can be easily integrated. Supposedly, a network provider offers add-on network services (e.g. caching). These services are also associated with tags. A node may request from the controller to create a path through a service point.

In the implementation presented in this section, edge-nodes request paths by simply providing a list of tags. Nevertheless, our implementation allows nodes to specify path creation criteria using attribute-values pairs (e.g., location=floor1). This can be extended to richer expressions, for example, *quality >= standard*, *located in [co-ordinates]*. What is more, our implementation allows the association of tags with links. In the future this feature can be used to provide, for example, QoS restrictions.

4 CONCLUSION

In this paper, we proposed an enhancement to SDN-based architectures that allows tag-based topology management and traffic engineering. With our approach, each network node is associated with a set of tags and the SDN controller holds the role of the path computation element, i.e., it is responsible for computing paths among (or composed of) nodes identified by a set of tags. Moreover, the proposed solution leverages Bloom filter-based forwarding to achieve seamless traffic engineering. By using tag-based topology management combined with Bloom filter-based forwarding, SDN-based architectures achieve significant gains, including support for multiple communication modes (such as multicast and anycast), service composition, improved management, and better support for new applications. As a proof of concept, we presented our implementation of an emulated network that supports tag-based path creation in order to enable CoAP group communication. Our implementation exhibits significant advantages compared to an approach

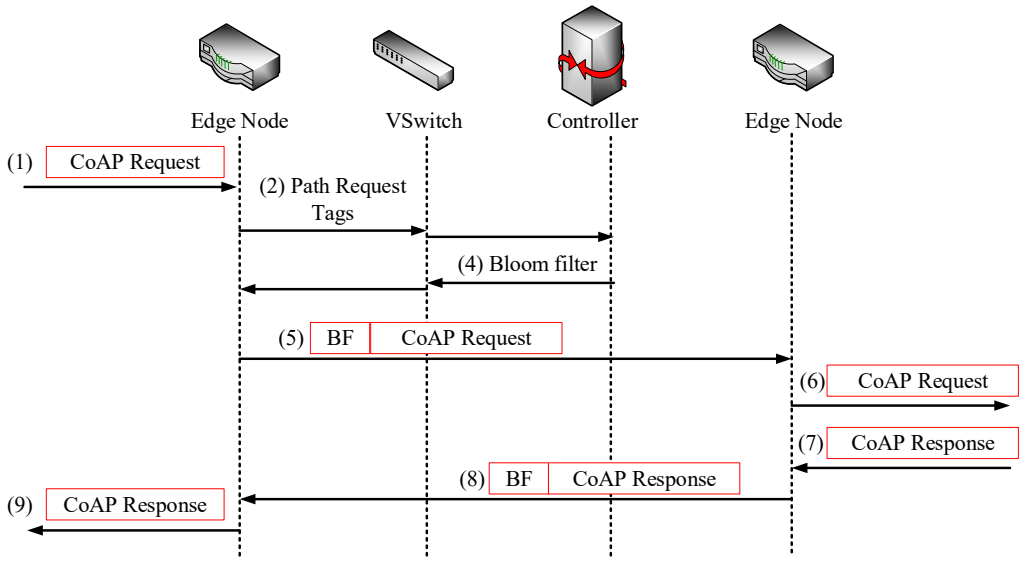

Figure 3: A CoAP group communication transaction.

that follows the implementation recommendations of the related RFC.

The work presented in this paper can be expanded in various directions. Firstly, we assumed that SDN controllers learn the network topology, as well as the link identifiers, through a configuration file, but topology discovery solutions can instead be investigated. Secondly, we left tag management as future work. Moreover, Bloom filter-based forwarding can be further improved: future work can consider ephemeral Bloom filters to counter security attacks (e.g., DoS attacks), or to handle user mobility. Similarly, Bloom filter techniques resilient to path failures can be investigated. Finally, our IoT implementation can be extended to consider incorporating solutions such as the CoAP resource directory, which can enhance tag management.

ACKNOWLEDGMENTS

This research was supported by the AUEB Research Center.

REFERENCES

[1] 2011. OpenFlow Switch Specification v1.2.0. OpenFlow. (2011). https://www.opennetworking.org/ (last accessed 6 Apr. 2018).

[2] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.* 38, 3 (July 2008), 105–110.

[3] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia. 2013. SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming. In *2013 Second European Workshop on Software Defined Networks*. 87–92.

[4] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*. ACM, New York, NY, USA, Article 19, 6 pages.

[5] A. Lara, A. Kolasani, and B. Ramamurthy. 2014. Network Innovation using OpenFlow: A Survey. *IEEE Communications Surveys Tutorials* 16, 1 (First 2014), 493–512.

[6] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and T.V. Lakshman. 2014. Application-aware Data Plane Processing in SDN. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*. ACM, New York, NY, USA, 13–18.

[7] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 117–130.

[8] A. Rahman and E. Dijk. 2014. *Group Communication for the Constrained Application Protocol (CoAP)*. RFC 7390. IETF.

[9] M. J. Reed, M. Al-Naday, N. Thomos, D. Trossen, G. Petropoulos, and S. Spirou. 2016. Stateless multicast switching in software defined networks. In *2016 IEEE International Conference on Communications (ICC)*. 1–7.

[10] Z. Shelby, K. Hartke, and C. Bormann. 2014. *The Constrained Application Protocol (CoAP)*. RFC 7252. IETF.

[11] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. 2015. A Survey on Software-Defined Networking. *IEEE Communications Surveys Tutorials* 17, 1 (Firstquarter 2015), 27–51. https://doi.org/10.1109/COMST.2014.2330903