




## Article

# P4-KBR: A Key-Based Routing System for P4-Programmable Networks

Pilar Manzanares-Lopez , Juan Pedro Muñoz-Gea  and Josemaria Malgosa-Sanahuja 

Department of Information Technologies and Communications, Universidad Politecnica de Cartagena, E-30202 Cartagena, Spain; juanp.gea@upct.es (J.P.M.-G.); josem.malgosa@upct.es (J.M.-S.)

\* Correspondence: pilar.manzanares@upct.es; Tel.: +34-968-326534

**Abstract:** Software-defined networking (SDN) architecture has provided well-known advantages in terms of network programmability, initially offering a standard, open, and vendor-agnostic interface (e.g., OpenFlow) to instruct the forwarding behavior of network devices from different vendors. However, in the last few years, data plane programmability has emerged as a promising approach to extend the network management allowing the definition and programming of customized and non-standardized protocols, as well as specific packet processing pipelines. In this paper, we propose an in-network key-based routing protocol called P4-KBR, in which end-points (hosts, contents or services) are identified by virtual identifiers (keys) instead of IP addresses, and where P4 network elements are programmed to be able to route the packets adequately. The proposal was implemented and evaluated using bmv2 P4 switches, verifying how data plane programmability offers a powerful tool to overcome continuing challenges that appear in SDN networks.

**Keywords:** P4; routing; KBR; SDN



**Citation:** Manzanares-Lopez, P.; Muñoz-Gea, J.P.; Malgosa-Sanahuja, J. P4-KBR: A Key-Based Routing System for P4-Programmable Networks. *Electronics* **2021**, *10*, 1543. <https://doi.org/10.3390/electronics10131543>

Academic Editor: Martin Reisslein

Received: 2 June 2021

Accepted: 23 June 2021

Published: 25 June 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The idea of decoupling end-points (hosts, contents or services) from network addresses has been demanded in the last few years to provide support to enhanced services on networks. Different solutions have been proposed to adapt location-centric TCP/IP architecture to the content-centric paradigm.

In this way, Information Centric Networking (ICN) [1] was proposed as a way of solving the problems of the existing internet at the same time as moving the Internet towards a content distribution architecture. Named data networking (NDN) [2], one of the most popular ICN-based projects, allows the addressing in a packet to be keys/names/whatever, but it requires additional in-network infrastructure. NDN nodes maintain three specific data structures (cache memory or content store (CS), pending interest table (PIT) and forwarding information base (FIB). In addition, NDN defines specific routing mechanisms and implements the content retrieval using two new packet types: interest and data packets. Recently, the potential of SDN and the flexibility offered by programmable switches have provided new possibilities to the development of NDN solutions, moving the PIT and FIB structures to the programmable switches [3].

Achieving the rollout of NDN architectures in traditional IP networks is a slow and key challenge. However, the power of Software Defined Networking (SDN) and programmable switches such as P4 devices offer us new possibilities. In this work, we make use of the potential and flexibility of SDN and P4 to propose and implement a network-level key-based routing mechanism that does not require any additional data structures and that can co-exist with existing L2/L3 protocols. P4-KBR (P4 Key-Based Routing) defines a network-level routing protocol where end-points are identified by virtual identifiers (keys) instead of traditional IP addresses, and P4 network elements are configured to be able to route the packets adequately. This proposal is not a protocol just for content retrieval like NDN-based solutions. Neither it is an application-level key-based lookup solution, such as well-known p2p solutions, where forwarding tasks are implemented at the peers.

In fact, P4-KBR is a non-IP network-level routing protocol that can be used by higher level applications and systems. In addition, as said before, table entries generated by the P4-KBR solution can co-exist with table entries required for other routing/forwarding protocols based on IP or MAC addresses.

A previous version of P4-KBR called OFC-KBR was defined for OpenFlow-based networks [4]. OFC-KBR (OpenFlow Compatible Key-Based Routing) performance was evaluated in different topologies from real networks, validating the foundations of the solution. However, the analysis and evaluation of OF-KBR allowed us to identify a weakness related to the resource requirement in OpenFlow switches. Recent advances in data plane programmability has offered a workaround to solve this problem. P4 language [5] and P4-Runtime specification [6] reduce considerably the limitations of OpenFlow protocol by allowing more fine-grained and flexible programmability of device data planes. Thus, the proposed P4-KBR routing protocol implantation obtains an optimal use of network element resources.

P4-KBR continues to use the SDN architecture to enable the network management, using in this case the SDN ONOS controller [7] for the management of the P4-based devices. The description and review of the integration process of the ONOS controller and a P4-based networking solution is another contribution of this work. Most of the recent works focus directly on data plane programmability using P4, not paying special attention to the required integration of the control and data planes. The experimental results using the P4-KBR-based system demonstrate how P4 can be used to overcome limitations or give response to technical requirements of recent OpenFlow-based SDN networks.

The remainder of the paper is organized as follows. Some related works are introduced in Section 2. Section 3 provides background information on the elements of the P4 ecosystem. Section 4 defines the proposed P4-KBR solution and Section 5 describes the implementation of the P4-KBR system in detail. Section 6 validates and evaluates the implemented P4-KBR system. Finally, Section 7 concludes the paper.

## 2. Related Work

Almost two decades ago, key-based routing (KBR) solutions in conjunction with Distributed Hash Tables were highly used to create structured P2P networks such as Chord [8], Pastry [9], Tapestry [10] or Kademlia [11] offering content-sharing solutions in overlay networks. These networks implemented a routing service at application level where a message looking for a particular could be routed towards the node responsible for that key. More recently, key-based solutions have been focused on the proposal of key-value storage systems to give support to datacenter infrastructures or to implement web caching solutions [12–14].

Inspired by application level KBR/DHT solutions, we have proposed a key-based routing solution directly implemented at network layer that makes use of the potential of Software Defined Networking and the power and flexibility of programmable data planes. The keys, also named virtual identifiers, are obtained from textual names, whose format can be defined depending on the requirements of the service that is going to make use of the proposed routing solution. Routing tasks become independent from the IP addresses, offering service developers a named-based routing service.

VIRO (Virtual Id ROuting) [15] proposed a key-based routing solution in the context of SDN/Openflow networks. The idea behind VIRO is the introduction of a topology-aware, structured virtual id space onto which both physical identifiers (e.g., Ethernet MAC addresses) as well as higher layer addresses/names (e.g., IPv4/IPv6) are mapped. Taking advantage of such a topology-aware and structured vid space, VIRO employs a KBR algorithm to build routing tables, look up objects (names, addresses, vids, etc.) and forward packets. In order to implement a required address–name resolution mechanism, each host-node needs to maintain a local table that maps the physical identifiers to virtual identifiers. On the other hand, besides the remote SDN controller responsible for the management plane, VIRO implementation also requires that each node includes a local SDN controller to

run the VIRO packets forwarding functions. Another important drawback of this solution is the fact that the standard Ethernet frame structure must be modified to be able to include the ids and the required forwarding directives. Consequently, the authors needed to modify and extend the match and the actions of the Open vSwitch user and kernel spaces to be able to implement their solution.

Returning to key-value store solutions, the recent work TurboKV [3] presents a distributed key-value store architecture that leverages the power and flexibility of programmable switches. Implemented in P4 using bmv2 switches, this work designs an indexing scheme to manage the directory information records inside the programmable switches, and defines protocols and algorithms to route TurboKV client packets (search queries) to targeted storage nodes.

### 3. P4 Ecosystem

Programming Protocol-independent Packet Processor (P4) [5] is a representative data plane programming language that provides packet processing abstractions in networking elements in a target-independent way. The P4 language is used to program how packets are going to be processed by the data plane of programmable forwarding elements (e.g., hardware or software switches, network interface cards, routers, or network appliances), which are commonly known as P4 targets.

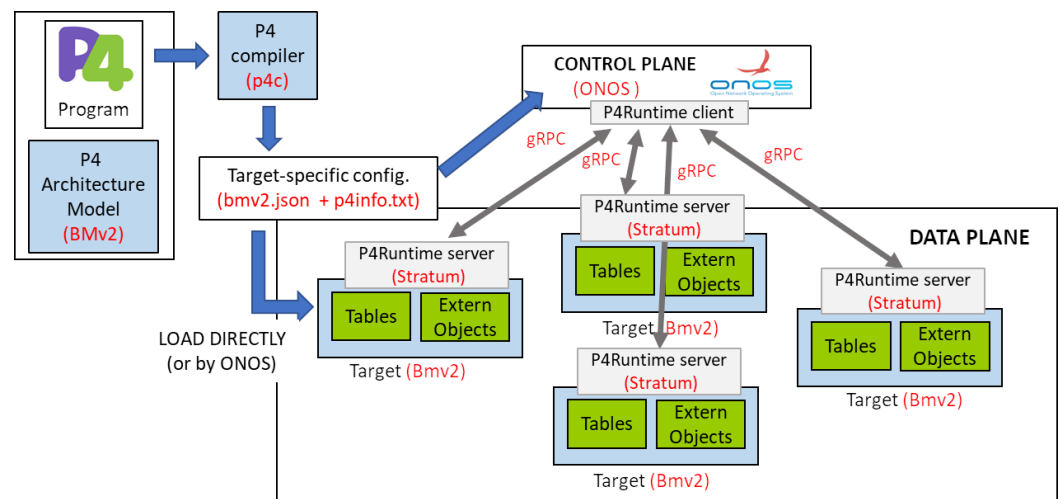
The P4 language offers end-users the ability to create custom protocols and algorithms in an agile and generic way. It allows the user to specify the format of packets (protocol's headers) to be recognised by network devices and the actions to be performed on incoming packets (e.g., forwarding, headers modification, adding protocol header). In addition, P4 offers stateful programming [6] based on the use of memory registers that can be accessed when processing a packet. Thus, forwarding procedures can be defined based on the network state, variations, and history of flow statistics at node level, without requiring the intervention of a controller.

The definition of a memory register that stores the switch vid and the possibility of defining new actions to be applied during processing of a packet through the pipeline are the key aspects that have made possible the implementation of the solution described in Section 4.3.

The P4 source code is not directly consumed by the network elements. A P4 program must be compiled for a particular platform. These platforms can be hardware-based (e.g., Barefoot Tofino, FPGA) or software-based (e.g., bmv2, eBPF/XDF, PISCES or P4rt-OVS). P4c [16] is a reference P4 compiler, that supports both P4 language versions P4-14 [17] and P4-16 [18]. A survey of P4 compilers can be found in [19].

Figure 1 depicts a typical workflow when programming a P4 target, showing the different elements involved in the P4 ecosystem. Target manufacturers provide a model of a specific hardware or software implementation, an architecture definition (that is, the P4 programmable blocks and an API to program the target) and a P4 compiler for the target. The P4 compiler maps the P4 source code into the target model, and it then produces a data plane configuration that implements the forwarding logic and an API to manage the state of the data plane objects from the control plane.

Another important and useful component of the P4 ecosystem is P4Runtime [6]. P4Runtime is a standard, open and silicon-independent protocol that enables us to control the P4 forwarding plane at runtime. Once a P4 program is deployed, it does not offer functions to interact with the P4 switch, for example, to add or remove entries in tables or to read counters. P4Runtime API has been developed for this aim. It follows a client–server model. A client integrated within the control plane interacts with the server to load the P4 program, write and read pipeline information (e.g., table entries and meters) and send and receive packets. P4Runtime uses a gRPCprotobuf-based language to define the API called `p4runtime.proto`.



**Figure 1.** P4 ecosystem.

As specified in the figure labeled in red, we have used ONOS (Open Network Operating System) [7], an open source SDN controller that includes support for P4Runtime, offering useful features such as: device discovery, P4 pipeline provisioning, match-action table operations (via existing ONOS APIs such as FlowRule, FlowObjective or Intent), controller packet-in and packet out, and counter reads.

Our proposal has been validated using the P4 reference software switch bmv2 [20] and Stratum [21], a open-source silicon-independent switch OS that exposes a set of next-generation SDN interfaces including P4Runtime. Bmv2 switches can also be controlled and programmed using a Thrift-based interface with a Command Line Interface (CLI).

Packet processing in bmv2 switches using P4 has the following phases. The input packets are first handled by the P4 programmable *Parser Stage* defined as a state machine. Each state extracts header fields (e.g., Ethernet, IPv4, TCP/UDP), which are declared in the P4 program and, depending on the field value, the state will change to another state.

Once the packet is parsed, it passes through the P4 programmable *Ingress pipeline* where match-action tables are used. A table compares the matching fields in flow entries with packet headers or metadata to determine the appropriate actions. These actions include the possibility of modifying the header fields of the current packet (stateless or transient information) and perform stateful operations that store state persistently, i.e., for more than a single packet. P4 allows the programmer to maintain stateful or persistent information in the form of counters (for counter events associated to entries in tables), meters (for data rate measurement) and registers (counters that can be operated from actions in a general way).

After the *Ingress pipeline*, the packet can be transferred to a non-P4 programmable block that represents the non-programmable traffic manager including fixed functions provided by the target vendor. Next, the packets pass through the *Egress pipeline*, which could define additional match-action tables for further modifications if necessary. Finally, the packets pass through the *Deparser* for serialization and then they exit the switch.

#### 4. P4-KBR

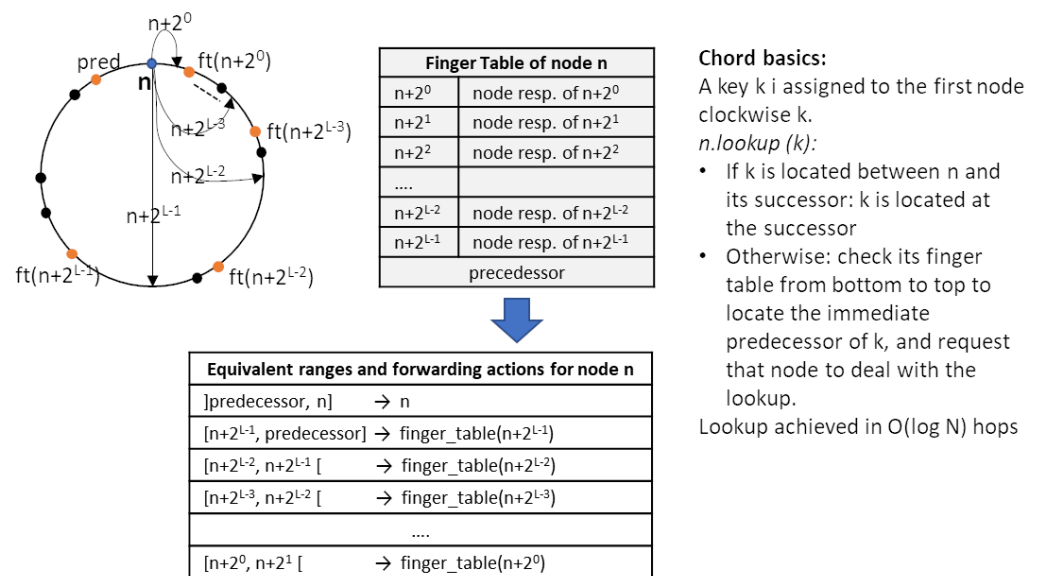
##### 4.1. Key-Based Routing Mechanism

The P4-KBR routing mechanism is inspired by Chord [8], a widely used application level DHT (Distributed Hash Table) protocol. The forwarding information used by Chord and its lookup procedure have been the foundations to define the routing mechanism of P4-KBR that, unlike Chord, is totally implemented at network level.

By way of summary, the upper part of Figure 2 shows the basic concepts of Chord. In particular, the finger table of a node is shown, and how it is used by the Chord lookup mechanism to forward a packet to the node responsible for the searched key.

The Chord lookup procedure, which checks the finger table from bottom to top to locate the next node towards the one responsible for the searched key, can be adapted to become a network-level key-based routing mechanism totally implemented at network level in P4 networks (or in OpenFlow-based networks as the predecessor OFC-KBR). This network level implementation results in a set of Match–Action flow table entries that will be inserted into each network device. The match fields of each entry determine a range of virtual identifiers, and the action fields indicate what to do when a packet destined to a value included in the range is received.

The SDN controller makes use of its global knowledge of the network to identify the node responsible of each finger table entry and to perform the translation, configuring the flow table of each network element adequately.



**Figure 2.** The upper part of the figure depicts the Chord basics: the definition of the finger table and the lookup mechanism. At the bottom of the figure, the set of value ranges and associated actions after transforming the lookup mechanism into a network level solution is shown.

The table at the bottom of Figure 2 shows the set of key ranges that are obtained for a network element and the forwarding action that should be done depending on the key value the packet is destined for. For a node  $n$ ,  $finger\_table(n+2^i)$  refers to the value of the finger table entry  $n+2^i$ , that is, the network element responsible of key value  $n+2^i$ , and  $\rightarrow x$  means the action *send to node x*. At the control plane, once the ranges and associated actions are obtained by the controller, it will instruct each network element to install the adequate flow entries.

Instead of defining a shim P4-KBR header to contain the source and destination key values (vids) of a packet, the proposed implementation of P4-KBR uses IPv4 address fields to store the vid values. Consequently, the destination IPv4 address is the matching field in the flow table entries used for routing tasks. Depending on the matched entry, a receive packet will be forwarded to the next network element towards the network element responsible for the destination vid.

As can be seen in the example shown in Figure 3, but without loss of generality, most of the obtained ranges cannot be expressed by bitmasked values directly. For example, the range  $[303, 319[ = [000100101111, 000100111110]$  cannot be expressed by one bitmasked value. This fact causes a technical difficulty in OpenFlow-based networks, as it was identified in [4], because OpenFlow specification only enables the use of bitmasks to express ranges of values in the IPv4 source and destination matching fields. To solve this problem in OpenFlow networks, proposed the use of the prefix expansion algorithm [22] was proposed, converting each obtained range into a set of bitmasked equivalent ranges.

Using the previous example, the range  $[303, 319[$  would be expressed by the set of bitmasked ranges:  $[303]$ ,  $[304, 311]$ ,  $[312, 315]$ ,  $[316, 317]$  and  $[318]$ . This is a practical solution, but it involves the insertion of a relatively high number of flow tables entries in the nodes.

Offering a greater potential and flexibility, P4Runtime does support the insertion of table entries where a specific match key field is defined for a RANGE match. A range is defined by a low bound and high bound, where the low bound must be less than or equal to the high bound [6]. Thus, the implementation of P4-KBR routing proposal in P4-based networks directly solves the problem of the high occupation of devices tables that was identified in OpenFlow-based networks.

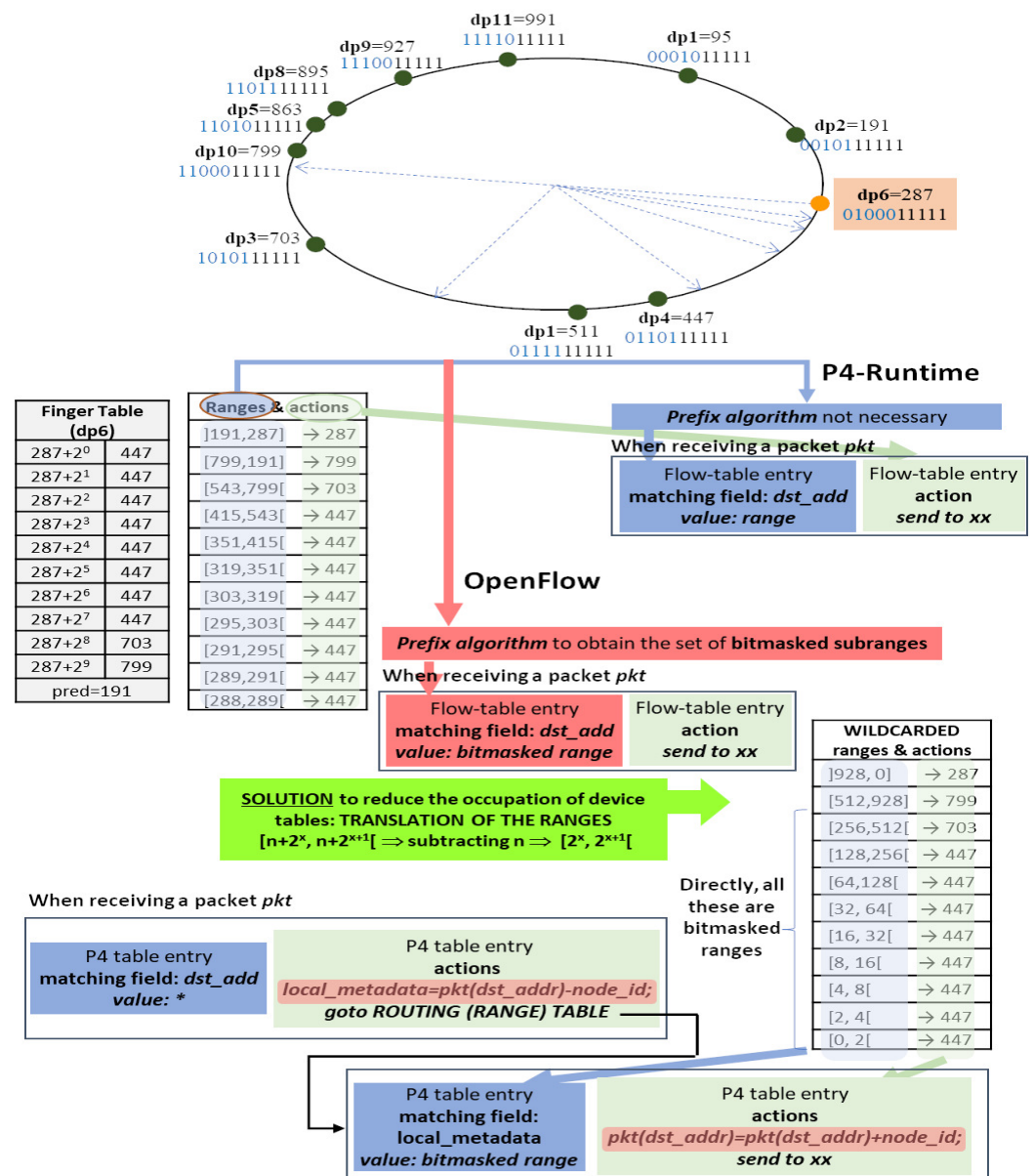
#### 4.2. Use of MPLS to Create the Virtual Links

As seen before, P4-KBR is based on the creation of a virtual space, where network elements and hosts/contents/services (from now, end-points) involved in the communication are identified by virtual identifiers (vids).

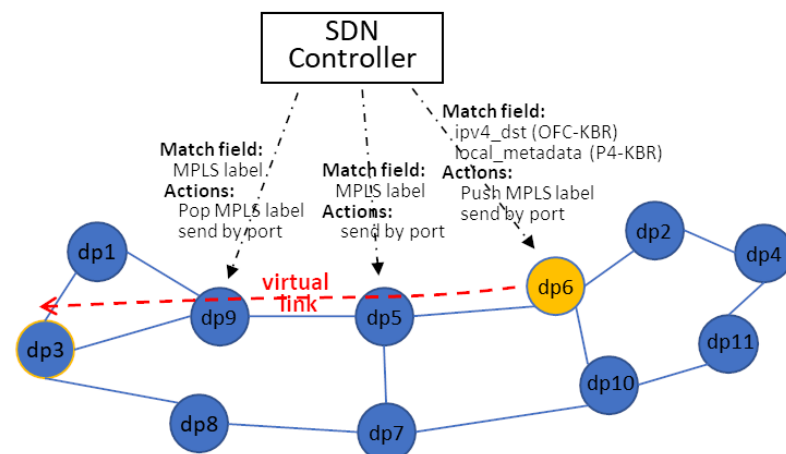
In a theoretical way, vids are defined as L-bits strings that comprise two parts of L/2 bits. In the case of a network element, the first L/2 bits are the result of applying a hash function to the device identifier and the last L/2 are set to one. In the case of end-points, the first L/2 are the result of applying the hash function to the device identifier to which the end-point is connected and the last L/2 bits are the result of applying the hash function to its name (a flat or a hierarchical textual name, depending on the service that is going to use P4-KBR).

According to the vid assignment scheme, location of nodes in the virtual ring is not linked to the physical topology. For this reason, although the action *send to node x* defined at node n establishes a virtual link between the node n and the node x, it is likely that both nodes are not directly connected. On the other hand, this action has to be translated into a P4-compatible action. In particular, the action should indicate the adequate output port to forward the packet toward destination x.

In order to translate a virtual link between two nodes not connected physically into a physical path, avoiding the appearance of loops during the forwarding process, we proposed the creation of a tunnel using the MPLS technology [23], as it is summarized by an example shown in Figure 4. This figure corresponds to the physical topology of the nodes that resulted in the virtual topology shown in Figure 3. As can be seen there, for example, the third of the wildcarded ranges defined at dp6 is associated to the action *send to node 703 (dp3)*. Due to the fact that there is not a direct link between dp6 and dp3, a MPLS tunnel is configured by the installation of adequate flow table entries in the involved nodes.



**Figure 3.** Ranges of dp6 cannot be expressed as bitmasked ranges, so a larger number of bitmasked subranges are obtained using the prefix algorithm. P4 and P4Runtime support the inserting of table entries using RANGE match as match field type. On the other hand, P4 language also allows the implementation of a solution based on the use of LPM matching. The “Translation of ranges solution” is also implemented and evaluated in this work.



**Figure 4.** Physical path implementation of a virtual link using MPLS technology.

#### 4.3. Translation of Ranges Solution

The prefix extension algorithm was proposed as a useful tool that allows programmers to implement this key-based routing protocol in OpenFlow networks [4]. However, as described before (see the example in Section 4.1), this prefix extension algorithm involves the insertion of a relatively high number of flow table entries in the nodes.

A way to address this drawback, which we called the “translation of ranges solution”, was theoretically defined in [4]. However, its implementation was left pending awaiting new versions of the OpenFlow standard that would allow it.

Although the problem of non-bitmasked ranges is solved in P4 networks thanks to the definition of the RANGE matching format, it is interesting to point out that data plane programmability and P4 language do also offer us an adequate framework for implementing the “translation of ranges solution”, as described below.

“Translation of ranges solution” was deduced after analyzing the structure of the Chord finger table (see Figure 2) and how it affects the definition of the key value ranges. It can be concluded that the drawback of the bitmasked representation of the ranges does not exist if  $n = 0$ . In this case, each one of the last  $(L - 1)$  ranges could always be represented by one bitmasked value. Making use of this feature, the problem of ranges that have to be expanded would not exist if, after obtaining the ranges associated to any node  $n$ , they were moved or translated by subtracting the vid of the node to the limits of each range (see green box in Figure 3).

However, the use of these translated ranges instead of the original ranges makes two additional actions necessary during the routing mechanism (marked in red in Figure 3): one that must be applied when a packet is received, and other that must be applied before a packet is forwarded. In other words, because the value used in the matching field (that is, the destination address) has been shifted, when receiving a packet, and before looking up the matching entry, the destination address of the packet has to be modified by subtracting the vid of the node. Likewise, once the adequate sending action has been obtained from the matching entry, and before forwarding, the destination address of the packet must be restored by adding the vid value.

Unfortunately, the action set defined by OpenFlow standards does not include these specific actions, preventing the implementation of the “translation of ranges solution”. However, as is described below in Section 5.3, the P4 language provides an enhanced programmability of data planes offering new tools that do make its implementation possible.

## 5. P4-KBR Implementation

### 5.1. Data Plane

The design of the P4-KBR data plane is presented in Figure 5. After being parsed in the *Ingress Parser* module, packets are passed to the *Ingress Pipeline* module, where

four match–action tables are defined: the *join\_process* table, the *host\_name\_resolution* table, the *key\_routing* table and the *mpls\_forward* table, as described below.

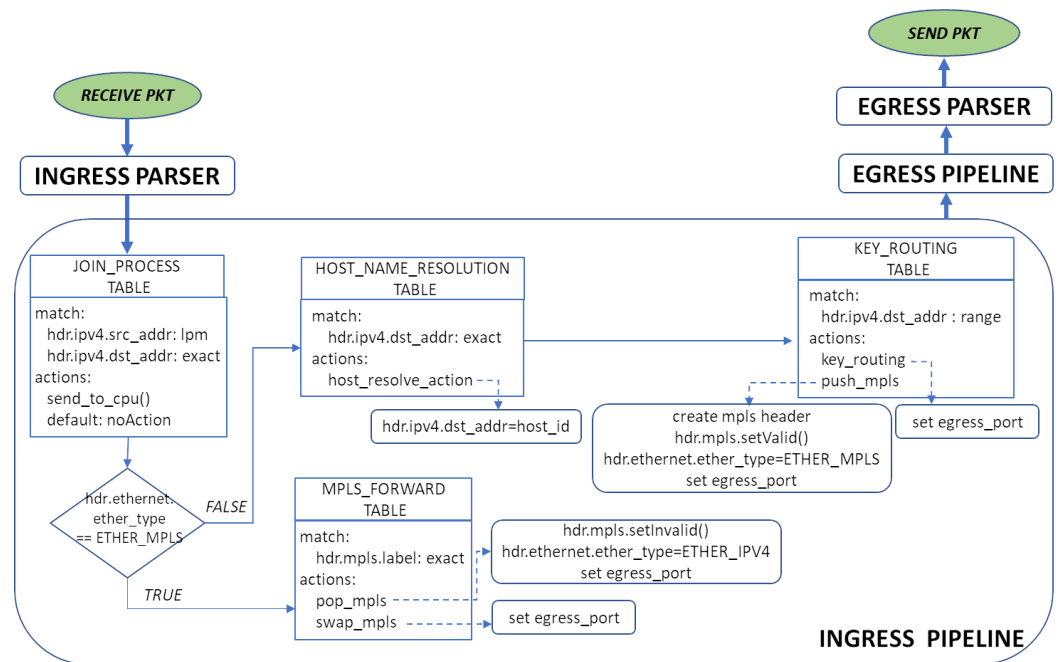


Figure 5. Tables and actions in the ingress pipeline.

In addition to the key-based routing mechanism itself, we define a joining of end-points mechanism and a distributed name resolution service to implement a host/service/content-centric communication service that makes use of the P4-KBR routing mechanism. The *join\_process* table and the *host\_name\_resolution* table are used to implement both mechanisms. Regarding the first one, when an end-point wants to join the system, it sends a specific *join\_packet* (a packet whose source key is the vid of the joining end-point, and the destination is the vid of the switch to which it is connected). By matching the *join\_process* table, a *join\_packet* will be identified, and then it will be sent to the controller (*send\_to\_cpu()* action) so that it can install the *host\_name\_resolution* table entries required by the name resolution mechanism. This second mechanism allows the communication between end-points just needing to know the end-point's name and not its virtual identifier (see [4] for a detailed explanation of both mechanisms).

As detailed in Section 4.3, the key contribution of this paper is focused on the network-level key-based routing mechanism. The *key\_routing* table and the *mpls\_routing* table are used for this purpose, being populated by the controller after obtaining the set of equivalent ranges and forwarding actions. As can be seen in the figure, the set of possible actions as result of a *key\_routing* table matching are the forwarding to the next virtual node by setting of the required egress\_port (if the next hop in the key-based routing path is a physically connected node) or the forwarding to the next virtual node through the corresponding mpls tunnel (if the next hop in the key-based routing path is not a physically connected node). As can be deduced, the *mpls\_forwarding* table is used to maintain the required mpls tunnels.

Turning to the *join\_process* table, the *send\_to\_cpu* action invokes the forwarding of the *join\_packet* to the controller using the P4Runtime API. Packets sent to the defined CPU\_PORT within the P4 device are used by the P4Runtime server to populate the PacketIn message that will be sent to the controller. Here, the standard annotation *@controller\_header("packet\_in")* is used to defined the PacketIn message metadata fields, in this case just the original ingress port whereby the packet was received.

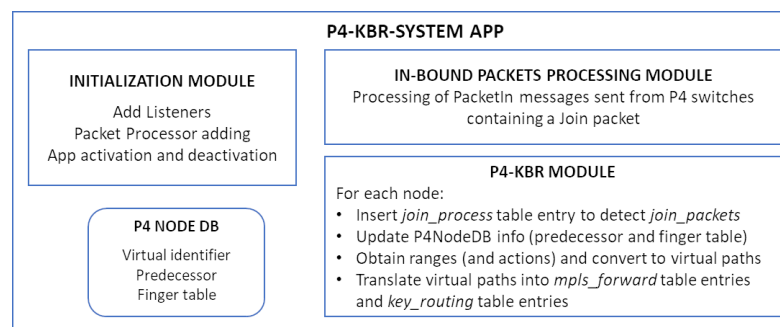
Any additional table is defined in the egress pipeline. During the egress pipeline, the egress parser serializes the header objects back into the packet, and then the packet is forwarded through the corresponding output port which is defined by the egress port value.

## 5.2. Control Plane

Our proposal uses ONOS to implement the SDN control plane. ONOS controller allows programmers to use existing pipeline-agnostic apps (apps that program the network using pipeline unaware abstractions known as FlowObjectives) but also to write and use new pipeline-aware apps (apps that are dedicated for a given dataplane).

In this work, we used the built-in network discovery service to obtain the network information, which is necessary to configure the P4 switches adequately. We also coded a pipeline-aware app called the P4-KBR-system app, which implements the P4-KBR-based communication system.

The P4-KBR-system app is described in Figure 6. The initialization module obtains a unique application ID and adds a packet processor to the list of packet processors in order to process the received PacketIns.



**Figure 6.** Description of the pipeline-aware P4-KBR-SYSTEM app.

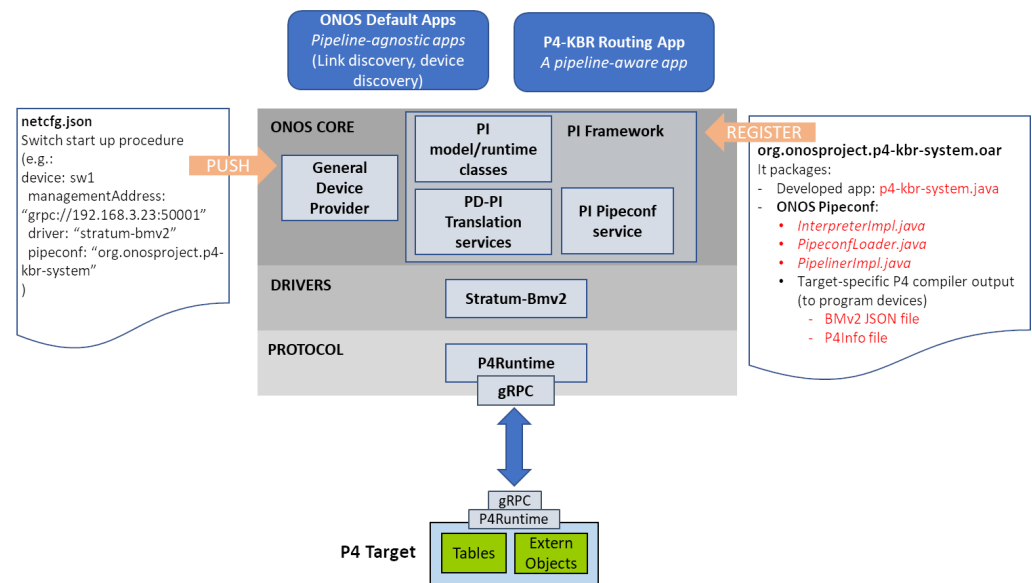
The P4NodeDB maintains information about each network element (mainly, its virtual identifier, predecessor and finger table), information that is used by the P4-KBR module and the Inbound packet processing module to configure the tables defined in the ingress pipeline of each P4-switch.

The inbound packet processor module processes the incoming packets. In particular, this module processes the PacketIn messages containing the *join\_packets* sent by the P4-switches during the end-point joint procedure. After obtaining the *join\_packet*, this module will include an entry in the *key\_routing* table of the P4-switch that generates the PacketIn to enable it to forward packets to the joined end-host. It will also include an entry in the *host\_name\_resolution* table of the P4-switch in charge of performing the end-point name resolution.

The last module implements the core of the P4-KBR functionality. The following tasks are executed for each node in the network topology. Firstly, a *join\_process* table entry is inserted to allow the network device to detect any *join\_packet* sent by a connected end-point, leading to the sending of a PacketIn to the controller. Next, this module will make use of the discovered network topology (obtained by the build-in ONOS service) to update the P4Node database with the information required to obtain the wildcarded ranges and the associated actions. The last step is to translate the obtained ranges into the set of required *mpls\_forward* table entries and *key\_routing* table entries that are necessary to implement the forwarding actions associated to each range.

Although ONOS was designed considering fixed-function dataplanes and fixed and well-known forwarding behaviors, the integration of P4 programs with ONOS was possible thanks to the develop of the PI Framework (Program/Pipeline Independent Framework) (see Figure 7). In order to allow ONOS to manage and deploy a pipeline on the devices, it is necessary to define an element called *ONOS pipeconf* that, in fact, is an ONOS app (.oar,

ONOS app archive) that includes some required Java files, the P4 compiled output and the P4 Info file.



**Figure 7.** ONOS and P4. Pipeline Independent (PI) framework allows the ONOS to manage and deploy a pipeline-aware app.

The PipelineInterpreter is the ONOS driver behavior used to map ONOS internal data structures (i.e., OpenFlow-derived ONOS header/actions) to P4 program-specific entities. In our proposal, Packet-In messages are used to implement the join process. Therefore, by implementing *InterpreterImpl.java*, the ONOS metadata name of the inport is adapted to be consistent with our P4 program.

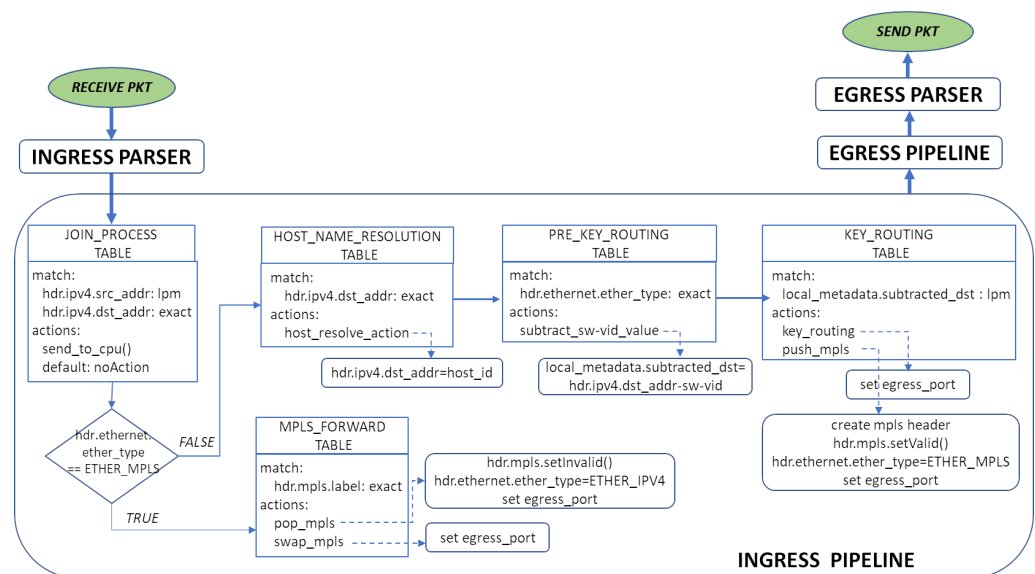
In addition, related to the pipeline-specific behaviour, *PipeconfLoader.java* and *PipelinierImpl.java* are included. The first one, *PipeconfLoader.java*, is a component that registers the pipeconf at app activation. *PipelinierImpl.java* is an implementation of Pipelinier driver behavior. The Pipelinier is a wrapper used to map forwarding objectives defined by built-in apps to pipeline-aware P4 tables.

As said before, we make use of the built-in lldpprovider app to discover the topology information. Therefore, in our case, Pipelinier implementation is used to map the forwarding objective used by the lldpprovider app to a P4 table called *link-discovery* table. Due to the fact that this table is not related to P4-KBR functionality, this table has not been represented in Figure 5. This table identifies the lldp (link layer discovery protocol) packets generated by the lldpprovider app, sending them to the controller along with certain local topological information.

### 5.3. Data Plane: “Translation of Ranges Solution”

The use of RANGE matching implemented by P4 enables the most efficient implementation of P4-KBR routing solution. However, we consider it might be of interest for readers the description of how the potential of P4 programming could offer an alternative solution to implement P4-KBR if only LPM matching (that is, value/mask matching) would be available.

Figure 8 shows the *Ingress Parser* module in this case. The main difference from the Ingress Pipeline defined in Section 5.1 is the inclusion of the *pre\_key\_routing*.



**Figure 8.** “Translation of ranges solution”: Tables and actions in the ingress pipeline

As detailed in Section 4.3, the “translation of ranges solution” requires the subtraction and the addition of a value to the destination address of a received packet. Neither of these actions are included into the set of available OpenFlow actions defined by the OpenFlow standard. However, the P4 programmable data plane allows us to implement the required translation of the keys by the definition of the *pre\_key\_routing* table and the *subtract\_sw-vid\_value* action. This table will be in charge of executing the shift of the destination address of the packet, a necessary task before looking up in the *key\_routing* table, which is populated by the controller after obtaining the adequate set of equivalent ranges and forwarding actions.

The shift of the destination address and the lookup in *key\_routing* table are based on the use of a user-defined metadata field called *local\_metadata.subtracted\_dst*. This metadata field will store the shifted destination value generated during the execution of the P4 program, in particular as a consequence of the *subtract\_sw-vid\_value* action defined in the *pre\_key\_routing* table.

Summing up, the possibility offered by P4 of storing and manipulating data pertaining to each received packet as a user-defined structure, and the flexibility in the definition of tables (matching fields and actions) have made it possible to overcome the limitations associated with OpenFlow standard.

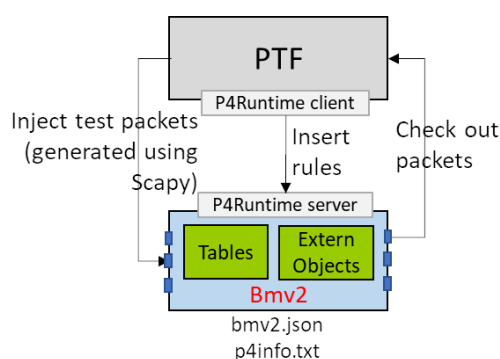
## 6. Testbed and Experimental Results

The P4-KBR system was implemented in a virtualized environment. We use Mininet to create network topologies that consist of bmv2 P4 switches and network hosts. Mininet is executed in a Docker container and ONOS controller is launched in other container. P4Runtime interface is used by ONOS to manage the network elements.

With this testbed, we want to verify the correct functionality of the P4 program and the P4-KBR system implementation as a whole and, also, to corroborate the improvement obtained by the P4-based solution compared with the previous OpenFlow-based solution.

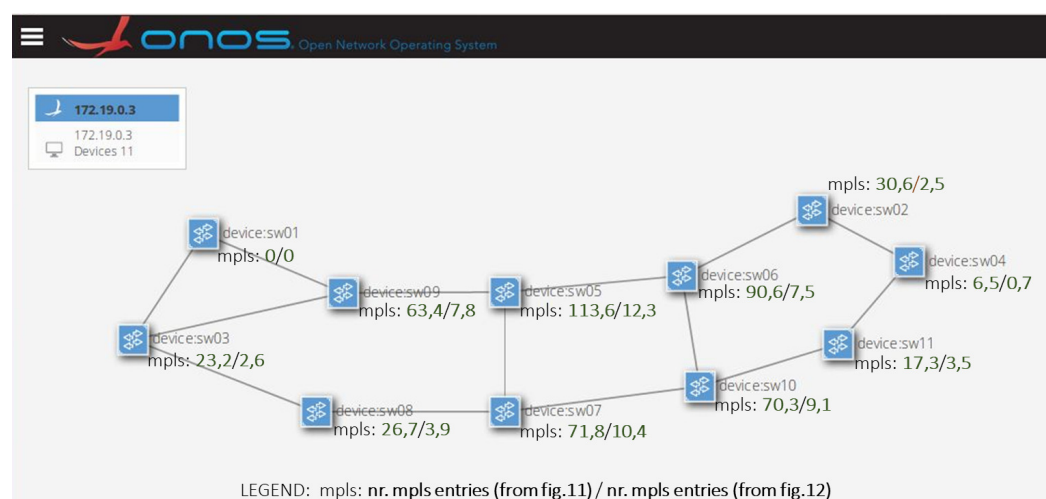
As a first step, a python-based data plane test framework (PTF) [24] is used to validate the P4 program (see Figure 9). PTF allows programmers the definition of unity tests in devices with P4-Runtime support. For each test, a set of rule(s) is inserted in the bmv2 switch by the PTF. Then, using the Scapy library [25], synthetic packets are created and injected into the P4 bmv2 switch. Finally, PTF checks if packets are forwarded through the expected port according to the program behavior being tested. The main objective of our PTF tests was to verify the correct functionality of the key-based routing. The LPM matching-based implementation which involves the adequate programming of *subtract\_sw-*

*id\_value* action used in the *pre\_key\_routing* table was checked. The RANGE matching-based implementation was also checked.



**Figure 9.** Unity test environment using the Plane Test Framework (PTF).

Now, we focus on evaluating the performance of P4-KBR solution in a network controlled by ONOS. The tests were performed on the Abilene network topology (see Figure 10), a well-known topology from the Internet topology zoo [26].

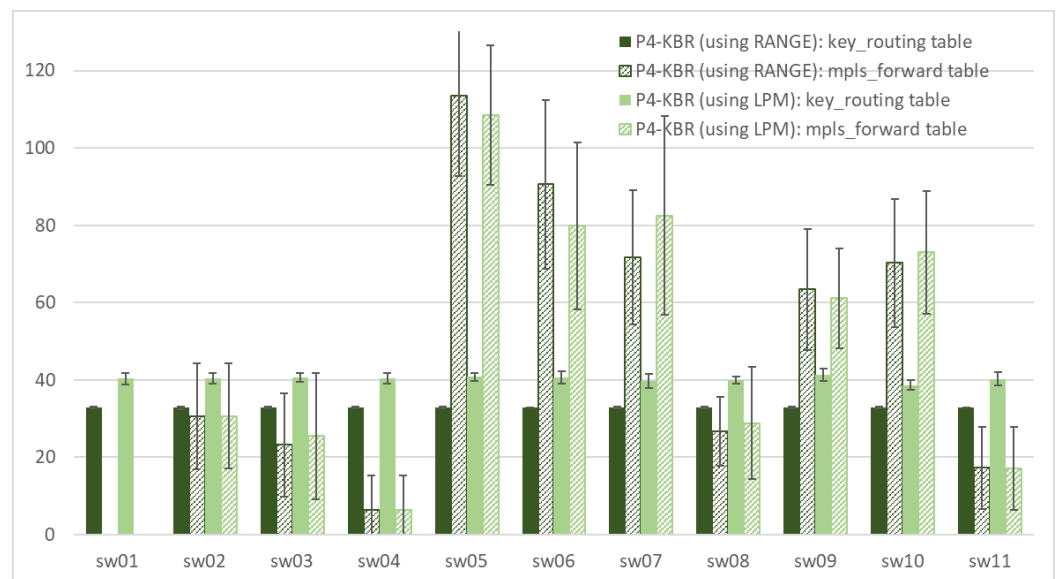


**Figure 10.** Abilene network topology from the Internet network topology zoo [26].

Bmv2 is meant to be used as a tool for developing, testing and debugging P4 data planes and control plane software written for them [20]. As such, the performance of bmv2 (in terms of throughput and latency) is significantly less than that of a production-grade software switch like Open vSwitch, and it also presents some constraints related to the amount of devices used in a virtualized network scenario using mininet and ONOS.

Even not being a large topology, results shown below let us to verify the functionality of P4-KBR system and to demonstrate the improvement provided by this proposal over OFC-KBR solution.

Figure 11 shows the number of entries in the *key\_routing* table and the *mpls\_forward* table of each bmv2 device. A total of 10 simulations were carried out, assigning a different set of switch identifiers to the nodes for each simulation. The dark green values correspond to the optimal P4-KBR system implementation, that is, using RANGE matching. The light green values correspond to the "translation of ranges solution" implementation, using LPM matching.



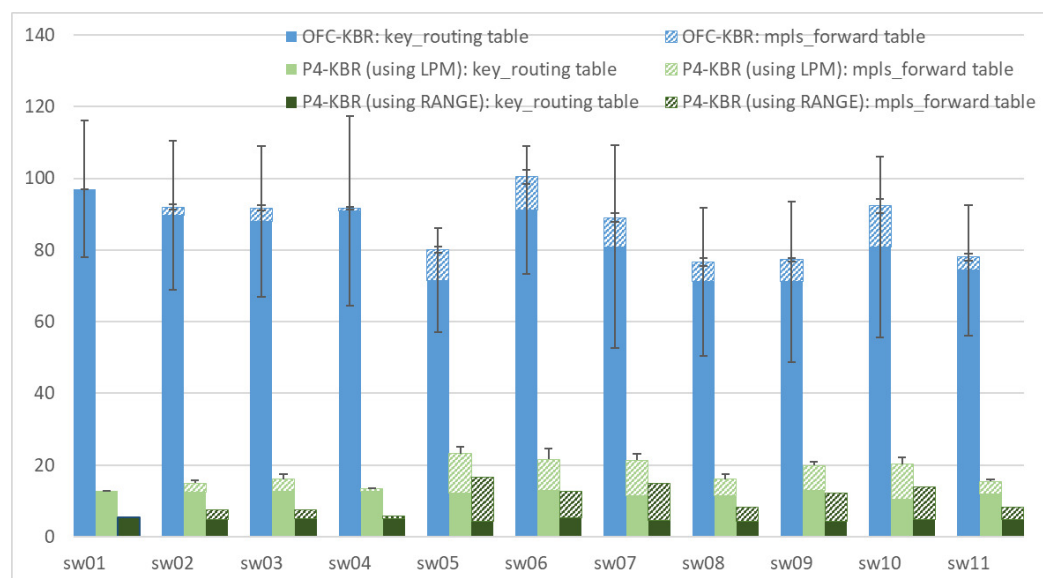
**Figure 11.** P4-KBR system. Occupancy of the device tables: number of entries in the *key\_routing* and *mpls\_forwarding* tables.

As expected, in the first case, the number of *key\_routing* table entries is 32, being  $L = 32$  the number of bits used to define the virtual id space. With the RANGE matching type, each of the obtained ranges (which are generally described in Figure 2) and their associated actions just require a table entry independently of the range limits.

In the second case, the number of the *key\_routing* table's entries is also close to 32, but it is a bit larger. The last  $L - 1$  ranges (that is, from range  $[n + 2^0, n + 2^1[$  to range  $[n + 2^{L-2}, n + 2^{L-1}[$  range) are translated to an equivalent bitmasked range (that is, range  $[2^0, 2^1[$  to range  $[2^{L-2}, 2^{L-1}[$  range) which just require a table entry for each one. However, the first two ranges, which involve the node's predecessor in their limits, cannot be expressed as bitmasked ranges despite the range movement. The prefix expansion algorithm is applied to convert them into a set of bitmasked ranges.

Regarding the occupancy of the *mpls\_forward* table, this value depends on the physical location of the node in the topology. Bmv2 devices such as sw05, sw06, sw07, sw09, and sw10 are more likely to be included in mpls tunnels used to implement the virtual paths than others such as sw01 and sw04. The relation between the physical location of devices and the number of entries is shown in Figure 10. The mean number of entries in the *mpls\_forward* table of the experiments with optimal P4-KBR system is shown next to each device.

Figure 12 compares the occupancy of the tables required by the P4-KBR routing system compared to the OpenFlow-based solution (OFC-KBR) [4]. In this case, trying to optimize even more the table occupation of the nodes, the implementation of P4-KBR and OFC-KBR systems merge the wildcarded ranges with the same forwarding action when possible. That is, if the action associated to more than one wildcarded range defines the forwarding to the same node, and the ranges can be merged using the adequate bitmask, just one table entry will be required. This merging process will be more likely in small topologies than in large topologies.



**Figure 12.** OFC-KBR versus P4-KBR. Occupancy of the device tables: number of entries in the *key\_routing* and *mpls\_forwarding* tables.

As can be seen, considering the Abilene topology, the number of entries in the *key\_routing* table is reduced to an average value of 4.78 in the case of the optimal P4-KBR system implementation (using RANGE matching) and 12.17 in the case of “translation of ranges solution” P4-KBR (using LPM matching). The number of entries in the *mpls\_forward* table is also reduced in both cases. As previously commented, the occupancy of the *mpls\_forward* table also depends on the location of the device in the topology (see the values in green in Figure 10). In any case, it can be observed that P4-KBR system improves the occupation of tables substantially compared with the OpenFlow-based solution OFC-KBR. The obtained values indicate an average saving in the number of entries of 88.15% by the optimal P4-KBR (using RANGE matching). This saving is lightly reduced to 79.5% when LPM matching is used in the implementation.

## 7. Conclusions

In this paper, we propose, implement and evaluate an in-network key-based routing protocol called P4-KBR. This non-IP-based routing protocol extends SDN networks to the non-host centric paradigm, implementing all the forwarding tasks at network level without the need of any additional infrastructure.

An efficient implementation of this solution, in terms of requirements of device resources, was possible thanks to the potential and flexibility offered by data plane programmability. P4 and P4Runtime opened the data plane programmability, paving the way for new protocols, solutions and applications without restrictions linked to standardized interfaces.

The proposal was implemented and evaluated using Mininet, an ONOS controller and a network composed of bmv2 P4 switches. In addition, this work includes a detailed review and description of the integration process of the ONOS controller and a P4-based networking solution.

**Author Contributions:** Conceptualization, P.M.-L., J.P.M.-G. and J.M.-S.; methodology, P.M.-L., J.P.M.-G. and J.M.-S.; software, P.M.-L.; validation, P.M.-L., J.P.M.-G. and J.M.-S.; formal analysis, P.M.-L., J.P.M.-G. and J.M.-S.; investigation, P.M.-L.; resources, P.M.-L.; data curation, P.M.-L.; writing—original draft preparation, P.M.-L., J.P.M.-G. and J.M.-S.; writing—review and editing, P.M.-L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been funded by the Spanish State Research Agency (AEI), under project grant AriSe2: FiNe, (Ref. PID2020-116329GB-C22 / AEI / 10.13039/501100011033).

**Data Availability Statement:** All data included in this study are available upon request by contact with the corresponding author.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Ahlgren, B.; Dannewitz, C.; Imbrenda, C.; Kutscher, D.; Ohlman, B. A Survey of Information-Centric Networking. *IEEE Commun. Mag.* **2012**, *50*, 26–36. [CrossRef]
- Zhang, L.; Estrin, D.; Burke, J.; Jacobson, V.; Thornton, J.D.; Smetters, D.K.; Zhang, B.; Tsudik, G.; Claffy, K.; Krioukov, D.; et al. Named data networking (NDN) project. *PARC Technical Report NDN-0001*. 31 October 2010. Available online: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.306.9771&rep=rep1&type=pdf> (accessed on 18 June 2021).
- Eldakiky, H.; Du, D.H.-C.; Ramadan, E. TurboKV: Scaling Up The Performance of Distributed Key-Value Stores with In-Switch Coordination. *arXiv* **2020**, arXiv:2010.14931.
- Flores-de la Cruz, A.; Manzanera-Lopez, P.; Muñoz-Gea, J.P.; Malgosa-Sanahuja, J. OpenFlow Compatible Key-Based Routing Protocol: Adapting SDN Networks to Content/Service-Centric Paradigm. *J. Netw. Syst. Manag.* **2019**, *27*, 730–755. [CrossRef]
- Bosshart, P.; Varghese, G.; Walker, D.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; et al. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
- P4Runtime Specification. Version 1.3.0. The P4.org API Working Group. July 2020. Available online: <https://p4lang.github.io/p4runtime/spec/v1.3.0/P4Runtime-Spec.pdf> (accessed on 15 June 2021).
- ONOS (Open Network Operating System). Available online: <https://opennetworking.org/onos/> (accessed on 15 June 2021).
- Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.R.; Kaashoek, M.F.; Balakrishnan, H. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Trans. Netw.* **2003**, *11*, 17–32. [CrossRef]
- Rowstron, A.; Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 12–16 November 2001; pp. 329–350. [CrossRef]
- Zhao, B.Y.; Huang, L.; Stribling, J.; Rhea, S.C.; Joseph, A.D.; Kubiawicz, J. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Sel. Areas Commun.* **2004**, *22*, 41–53. [CrossRef]
- Maymounkov, P.; Mazières, D. Kademlia: A peer-to-peer information system based on the xor metric. In Proceedings of the First International Workshop on Peer-to-Peer Systems, Cambridge, MA, USA, 7–8 March 2002; pp. 53–65. [CrossRef]
- Chen, Y.; Lu, Y.; Yang, F.; Wang, Q.; Wang, Y.; Shu, J. FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), Lausanne, Switzerland, 16–20 March 2020; pp. 1077–1091. [CrossRef]
- Trajano, A.F.R.; Fernandez, M.P. Two-phase Load Balancing of In-Memory Key-Value Storages Using Network Functions Virtualization (NFV). *J. Netw. Comput. Appl.* **2016**, *69*, 1–13. [CrossRef]
- Jin, X.; Li, X.; Zhang, H.; Soulé, R.; Lee, J.; Foster, N.; Kim, C.; Stoica, I. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17), Shanghai, China, 28–31 October 2017; pp. 121–136. [CrossRef]
- Dumba, B.; Mekky, H.; Jain, S.; Sun, G.; Zhang, Z.L. A Virtual Id Routing Protocol for Future Dynamics Networks and Its Implementation Using the SDN Paradigm. *J. Netw. Syst. Manag.* **2016**, *24*, 578–606. [CrossRef]
- P4c. Available online: <https://github.com/p4lang/p4c> (accessed on 15 June 2021).
- The p4 Language Specification Version 1.0.5, The P4 Language Consortium. 2018. Available online: <https://p4lang.github.io/p4-spec/p4-14/v1.0.5/tex/p4.pdf> (accessed on 15 June 2021).
- The p4<sub>16</sub> Language Specification Version 1.0.0, The P4 Language Consortium. 2017. Available online: <https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.pdf> (accessed on 15 June 2021).
- Stubbe, H. P4 Compiler & Interpreter: A Survey. Available online: [https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2017-05-1/NET-2017-05-1\\_07.pdf](https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2017-05-1/NET-2017-05-1_07.pdf) (accessed on 18 June 2021).
- Behavioral Model (bmv2). Available online: <https://github.com/p4lang/behavioral-model> (accessed on 15 June 2021).
- Stratum-Enabling the Era of Next-Generation SDN. Available online: <https://github.com/stratum/stratum> (accessed on 15 June 2021).
- Srinivasan, V.; Varghese, G.; Suri, S.; Waldvogel, M. Fast and scalable layer four switching. *ACM SIGCOMM Comput. Commun. Rev.* **1998**, *28*, 191–202. [CrossRef]
- Rosen, E.; Viswanathan, A.; Callon, R. Multiprotocol Label Switching Architecture, RFC 3031; Available online: <https://dl.acm.org/doi/pdf/10.17487/RFC3031> (accessed on 18 June 2021).
- PTF: Packet Test Framework. Available online: <https://github.com/p4lang/ptf> (accessed on 15 June 2021).
- Scapy. Packet crafting for Python2 and Python3. Available online: <https://scapy.net> (accessed on 15 June 2021).
- Knight, S.; Nguyen, H.X.; Falkner, N.; Bowden, R.; Roughan, M. The internet topology zoo. *IEEE J. Sel. Areas Commun.* **2011**, *29*, 1765–1775. [CrossRef]