Implementation of Routing Protocols Using the P4 Language

João Rodrigues Felício
joaofelicio98@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2022

Abstract

Software Defined Networking (SDN) is an exciting technology that changed the way operators configure and manage networks bringing much more space for innovation through network programmability. In 2013, the first programmable chip was prototyped, enabling operators to change the data plane without modifying the hardware. These new chips motivated the development of the P4 language to program the data plane. These tools made it possible to implement new protocols in the data plane, running on programmable hardware, instead of waiting for the long development cycles of chip manufacturing.

We are proposing a new improved version of the DSDV protocol[1]. DSDV is a distributed distance vector protocol that came to address the looping issues of the RIP[2] protocol. Essentially, in DSDV, each node maintains its routing table, which includes, for all reachable destinations, their length, next hop, and sequence number. Each node updates its routing table by receiving advertisements from its neighbours. Unfortunately, the DSDV update procedure leads to route fluctuation due to its criteria on electing attributes. This means that, in some situations, a node may change routes back and forth between different neighbours, even though there were no changes in the topology.

In this thesis we propose an extension to this protocol which we call “promise”. Its main novelty is that each node will not only elect its preferred routes, but will also keep other fallback routes (“promise” routes). A *promise* is a more recent route than the elected one, but with a worse metric (e.g., longer path length). The promise can thus be thought as a backup route which will be elected when there are changes in the topology.

Keywords: SDN; P4; Routing; Distance Vector; Promise

1. Introduction

Ideally, in a network, data packets are forwarded across optimal paths. The development of routing protocols to this end is quite challenging since this requirement must be guaranteed in conjunction with high-speed packet processing.

In this project, we leverage P4 and develop a protocol extension that overcomes the main drawback of the DSDV protocol[1]. DSDV[1] is a distributed distance vector protocol that came to address the poor looping properties of the RIP[2] protocol. In DSDV, each node keeps its routing table, with all known destinations, and their corresponding length, next hop, and sequence number. The sequence number is what prevents nodes from keeping outdated routes thus preventing the formation of routing loops. Routes are always preferred if their sequence number is more recent, with older routes being discarded. If two routes have the same sequence number, the one with the best metric is the preferred one. However, because of DSDV's criteria on deciding the elected route, some nodes may end up in a route fluctuation state: changing routes from route A to route B every time a new computation starts, even when there are no changes in the topology. For example, this may occur when the optimal path is not the first one to be announced to a node.

In order to overcome the DSDV issue of route fluctuation, we propose the introduction of the *promise route*. Each node, besides keeping its routing table with all the elected routes, will also keep another table with all promise routes. A *promise* is a route that is announced from a different neighbour than the elected one, and is more recent (higher sequence number). However, it has a worse metric (longer path length) than the elected route. This way, each node can keep as a secondary route this *promise* without changing its state whenever a new computation starts. The promise is elected, for instance, when the node realizes that the previously elected path got worse (e.g., longer length), or the link that connects to the elected route fails.

1.1. Main Contribution

The main contribution of this thesis is the proposal of an extension to the DSDV protocol: the promise. Our evaluation shows that the promise decreases route updates in the network, consequently improving its scalibility.

We implemented two versions in this thesis: one version includes all the logic in the Control Plane, and another in the Data Plane, with the Control Plane responsible to populate the match action tables only.

2. Related Work

This section describes research related to the subject of this thesis. Section 2.1 describes the concept of SDN. Section 2.2 introduces data plane programmability, and the language used to program the data plane, P4. Finally, section 2.3 discusses routing, by introducing the two main classes of routing protocols, the DSDV protocol[1], that is the subject of this thesis, and finally, some recent implementations of routing protocols on programmable hardware.

2.1. SDN: Software-Defined Networking

Conventional routers and switches run complex, distributed control software that is typically closed. Because such devices have their controller running in a distributed way, in order to configure and operate their networks, network administrators have to use different configuration interfaces that vary across vendors and even across different products from the same vendor. Thus, to define a new protocol or feature, a new hardware had to be fabricated to have this new functionality integrated. This industry was structured as a vertical market, resulting in a slow innovation process.

Software-Defined Networking (SDN) emerged as an innovative approach that changes the way operators run and configure networks enabling the programmability of a logically centralized controller. SDN offers an architecture that separates the control plane (routing decisions) from the data plane (forwarding decisions).

Control Plane The Control Plane runs centralized with a network-wide view. The centralized controller is the main responsible component for managing a set of switches dealing with all packet processing policies, determining the route packets should follow through the network. These routing policies are conveyed to the switch (data plane) through a southbound API (for example, OpenFlow[3]).

Data Plane The Data Plane is responsible for forwarding each packet according to the policies received from the Control Plane, usually with extremely high performance requirements. In this layer, several tables are maintained to allow for lookup upon receiving a packet that executes the corresponding action in case of a match.

2.2. Data Plane Programmability

The adoption of the SDN paradigm started with Control Plane programmability, where, as stated before, the operator establishes the packet processing policies centrally. However, in the Data Plane, the forwarding pipeline was still restricted to match a fixed set of fields in the packet headers and to perform a fixed set of actions.

RMT switching chips[4] enabled programmability in the Data Plane. RMT was the first prototype of a programmable switch, allowing the Data Plane to be changed without modifying the hardware. With this hardware, the programmer can now define new header fields, new actions, and new ways to process packets. The main language to express this low level packet processing is P4.

2.2.1. P4

In 2014, a paper entitled "P4: Programming Protocol-Independent Packet Processors"[5] introduced the programming language P4 as a suggestion for how OpenFlow "should evolve in the future." In 2016, a revision to the P4 language was announced, culminating in the language specification for P4-16[6].

P4 is a language for describing how packets are processed by the Data Plane of a programmable switch. This language was motivated by the limitations of OpenFlow, which only allowed a limited set of header fields and actions, and by the advances in the field of reconfigurable switches[4]. The Data Plane is no longer fixed. It is defined by a P4 program. In this paper, Bosshart et al. defined three design goals for P4:

1. Reconfigurability in the field. Programmers should be able to change the way switches process packets once they are deployed.

2. Protocol independence. Switches should not be tied to any specific network protocol. Instead, it should be possible to implement and integrate new protocols' formats whenever desired.

3. Target independence. Programmers should not be tied to the specifics of the underlying hardware.

2.3. Routing

Now that we have a solid background on the architecture of programmable networks, which enable innovation on new protocols, we will be focusing on routing.

Whenever a data packet arrives at a switch, the switch has to look at the packet's header fields and determine which port is better to forward the packet to. This decision shall be reached according to the routing rules. Routing is the process by which forwarding tables are built (i.e., it is a Control Plane process). On the other hand, forwarding consists of looking up the received header parameters in the table and forwarding the packet to the corresponding port, a data plane process.

The primary goal of routing is to find out the optimal path between any two nodes. To achieve this goal, two operations on attributes are needed: election and extension. *Attributes* are the set of metrics that a given protocol may consider. Such metrics can be hop-count, capacity, available bandwidth, delay, and so forth. The *Election* operation consists in ranking two attributes and deciding which is the preferred one. Finally, "an extension operation composes two attributes into a third one, modeling how the attribute of a path is obtained from the attributes of concatenated sub-paths"[7]. For example, lets consider a node receiving an announced attribute containing a delay. The extension operation in this case consists in getting the maximum value of the received delay and the delay in the link that connects the node and the neighbour that announced this attribute. The operation to execute depends on the metric in use.

The main goal of routing protocols is to forward packets across the optimal path between two given nodes. The two main classes of routing protocols are distance-vector and link state.

Distance-Vector Distance-vector protocols have at their core the distributed Bellman Ford algorithm. It begins with the assumption that every node only knows how to reach its neighbours. Each node announces its subnet. Nodes extend the attributes advertised by their out-neighbors for each destination with the attribute link that connects them to their out-neighbor, resulting in candidate attributes. Then, an attribute is elected from among the candidates and advertised to the in-neighbors. In the end, each node ends up with a complete routing table, reaching convergence. It is important to point that each node only knows about the content of its routing table. We can differentiate two sub-classes of distance-vector protocols: *non-restarting* and *restarting*[7]. In *non-restarting* vectoring protocols, the destination only initiates one computation process. On the other hand, in *restarting* vectoring protocols, the destination repeatedly initiates independent computation processes where the older attributes are always discarded. DSDV[1] is a good example of a restarting distance-vector protocol. In this protocol, each routing table contains a destination and the number of hops to reach it. Also, each entry of the routing table has a sequence number attached. Nodes advertise their routing table periodically to all neighbors and advertise whenever a change in the network is detected (in that case, the sequence number is updated). Finally, routes with a more recent sequence number, compared to the node's stored information, are always preferred. If a node receives a route with an older sequence number, it discards it immediately. Moreover, if the route has an equal sequence number, the one with the smallest metric is used.

Link State The starting assumption for link-state routing is pretty similar to the one from distance-vector, every node knows the state of its neighbors and the cost of the link to reach them. The idea behind link state is that each node will forward all the information it knows to all nodes in the network (instead of just its neighbors like distance-vector). This means that every node will have enough information to have a complete vision of the network topology. The process that makes sure that the link-state information gets to every node is reliable flooding. The messages exchanged between all nodes are called link-state packets (LSP). Once a node receives the LSP from every node, it can construct a complete map for the network's topology. Then, it typically runs the Dijkstra algorithm to find all shortest paths. OSPF is one of the most widely used link-state protocols. Besides the essential characteristics of a link-state, OSPF also has some more features, such as authentication of routing messages, additional hierarchy, and load balancing.

2.3.1. DSDV Protocol

As stated before, DSDV[1] is a distributed distance vector protocol. It was developed to overcome the looping issues that the RIP protocol[2] has when running on dynamic topologies that constantly suffer changes. The main contribution to address this problem is the use of the sequence number, which makes each node being able to label a route as updated or outdated.

In DSDV, we have each node keeping its routing table which lists all the reachable destinations and their corresponding lengths, next hops, and sequence numbers. The way that the network converges is by having each node receiving advertisements from their neighbours and electing the preferred advertised routes. The metric used to evaluate a preferred path is the length, which is the same as hop count. So, a computation starts having a node advertising its subnet to its neighbours with a value length of one, which means that this node is one hop away from its neighbours. As the following nodes elect the preferred path to this node and advertise it to their neighbours, the length is continuously incremented at every new hop.

The criteria used in the DSDV protocol is as follows:

- First of all, the routes with more recent sequence number are always elected.

- If the sequence numbers are the same between multiple routes, the one with the lowest length is preferred.

This protocol has some drawbacks. For some topologies, the fact that a node blindly elects a route just because it has a more recent sequence number, may lead to route fluctuation. That is, for different reasons, the preferred route may not be the first one to be announced at some nodes. As a consequence, whenever a new computation starts, these nodes change routes back and forth, changing their states frequently. As a result, a flood of broadcast messages is caused, increasing the chance for packet reordering during transmission of data packets. This limitation is the main motivation to our thesis.

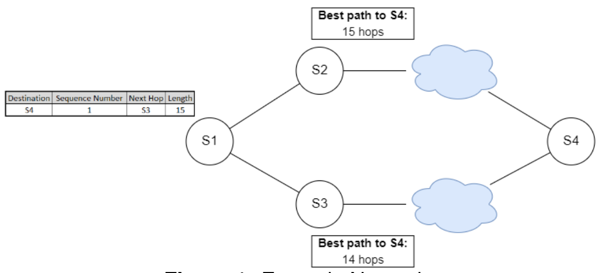
3. Design and Implementation

This chapter will describe the details of the Promise Extension of the DSDV Protocol we propose in this thesis. After presenting the key idea in section3.1 and describe the development environment in section 3.2, we give an overview of the design in section 3.3.

3.1. The Promise Extension: key idea

As mentioned before, the DSDV protocol has one limitation: route fluctuation. The fact that a node will always prefer a most recent route, or a route that has the same sequence number but a better metric, may cause changing the same routes back and forth every time a new computation is started.

In this chapter, we present a new solution to this problem by introducing the use of the *promise*. The idea is that instead of just electing the optimal path, we will also elect a *promise*. A *promise* can be thought of as a spare route, which will be elected in case there are some changes in the network that affect the optimal paths. For example, suppose there was a failure in the port that the elected path was announced from. In that case, the node could immediately elect the promise, never losing reachability to the announced destination. A *promise* has a more recent sequence number than the elected path but a worse metric. It also must come from a different neighbour than the elected one.



Destination	Sequence Number	Next Hop	Length
S4	1	S3	15

Figure 1: Example Network.

Let us consider the network from Figure 1. We assume between *S4* and *S2* we have fourteen nodes, and between *S3* and *S4*, we have thirteen. Node *S4* advertises its network to its neighbours for the second time (observe that the sequence number in *S1* routing table is 1). As is clear, the best path from node *S1* to reach *S4* is to send data packets to node *S3*, which has a length of fifteen (smaller than routing via *S2* with a hop length of 16). This information is kept in *S1* routing table.

Suppose *S1* receives first this second announcement from node *S2*. In DSDV, we would have *S1* electing this route. Thus, when the optimal route arrives (through node *S3*), it would elect again this better path. This behaviour would occur in every single computation if the network conditions remained the same. On the other hand, with our Promise Extension, node *S1* would treat the first announcement that comes from *S2* as a promise: it is more recent than the path in the forwarding table but has worse metric. Once the announcement from *S3* arrives, node *S1* realizes that this is still the best route to reach *S4*.

So, in this example, we can see that the Promise Extension enabled *S1* not to change the state of the next hop. In DSDV, node *S1* would first elect the route that was announced by node *S2*, and then, would change again to the optimal route announced by node *S4*. Besides avoiding these route fluctuations, the promise can be used immediately in case the connection to S3 fails, thus avoiding S4 from being unreachable. The only update was to keep the promise as backup.

3.2. Development Environment

This section will describe all tools used to implement the Promise Extension to DSDV.

3.2.1. P4Runtime

P4Runtime[8] is an open source API developed to enable the Control Plane software to control the Data Plane. An important aspect of this tool is that it is possible to control **any** Data Plane, from fixed-function or programmable switch ASIC to software switches running in a virtualized environment.

Regardless of what protocols or features the Data Plane is running, the framework of the P4Runtime remains unchanged, meaning that a wide variety of controllers can use this API. When programming the Data Plane by adding new protocols and features to the P4 switch, the P4Runtime API automatically updates, leaving no changes in the Control Plane.

This framework may be used in remote controllers and local controllers. Since our protocol is distributed, we will have one local control plane managing every P4 switch.

3.2.2. Behavioral Model (BMv2)

The Behavioral Model[9] is the refered P4 software switch. There are two versions of the *Simple Switch* that run different Control Plane interfaces:

- **simple_switch**
- **simple_switch_grpc**

We use the *simple_switch_grpc*, which is the one that is compatible with the P4Runtime controller. P4C[10] is the compiler we use to compile P4 programs to this switch.

3.2.3. Mininet

Mininet[11] is a network emulator designed to run on Linux. It can be configured via a CLI or with a Python API. The developer is free to customize its network and design the topology. We can create hosts, links, assign IPs to the interfaces, and define link bandwitdh and delays to emulate any network.

Mininet is a powerful tool for testing and evaluating network protocols as ours. We can simulate link failures and visualize how this action affects the network by checking the reachability to every node, for instance. We can create BMv2 switches programmed with P4 and emulate them in a virtualized network.

3.3. Promise Design

The Promise Protocol is a distributed distance vector protocol based on DSDV. Each switch has its local P4Runtime control plane that applies all the policy rules to the Data Plane. This means that the network nodes do not have a complete view of the network, and there is no centralized controller orchestrating these nodes. The only information they keep is the hop length and the next hop to reach to each destination. In addition, we also maintain the *promise*.

When a new computation starts, we have each switch[1] announcing its subnet to its neighbours, as Figure 2 suggests. Then, switches that receive probes from their neighbours will evaluate whether the probe is to elect or not, according to their policy. If they elect it, they will announce it to their neighbours again (except the neighbour that sent the newly elected probe). Finally, the protocol converges when we have connectivity among every node in the network.

Every forwarding rule must be populated in the match-action table. Since it is only possible to populate the match-action table from the Control Plane, we use the controller when we intend to add or update a new entry in the match-action table.

Every probe will always be firstly processed in the Data Plane. If the switch elects a new probe it sends it to the Control Plane to be further processed and populated in the match-action table.

In addition, we maintain another table with the promise, a backup route used only when the primary fails.

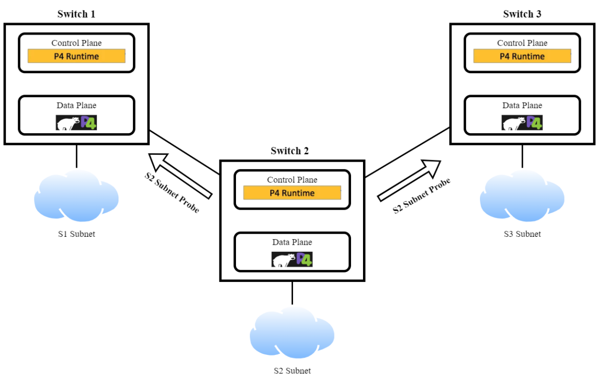


Figure 2: High-level Promise Protocol design.

4. Evaluation

In this chapter we present the evaluation of the Promise Protocol. It is organized as follows: Section 4.1 states the questions we aim to answer in our evaluation; The following, Section 4.2 describes the methodology and setup. Finally, in section 4.3, we present and discuss the results.

4.1. Objectives

We aim to answer the following questions:

- Does the Promise Extension improve the stability of the DSDV protocol?
- What is the performance gain of moving part of the decision logic to the Data Plane?

As mentioned before, the Promise Protocol uses the DSDV protocol as baseline to search for optimal paths. So the difference to the DSDV Protocol is just the use of the promise. As such, we will evaluate four implementations:

- Promise Protocol with all its logic implemented in the Control Plane.
- Promise Protocol with all the decision logic implemented in the Data Plane.
- DSDV Protocol with all its logic implemented in the Control Plane.

[1]We could also call it router as the P4 switch is performing both routing and forwarding.

- DSDV Protocol with all the decision logic implemented in the Data Plane.

In summary, our tests aim to achieve the following goals:

- Check connectivity within the network, to make sure the protocol is behaving correctly.
- Observe a smaller convergence time when the decision logic is entirely implemented in the Data Plane.
- Show that the Promise Protocol reduces route instability by decreasing unnecessary state changes.
- Show that the Promise Protocol is fault tolerant.

4.2. Methodology and experimental setup

We evaluated our protocol in real network topologies. For this purpose we averaged the *Topology Zoo*[12], a source of real network topologies. We selected three networks with different sizes:

- Abilene Network [13], this is the smallest network, with eleven nodes:

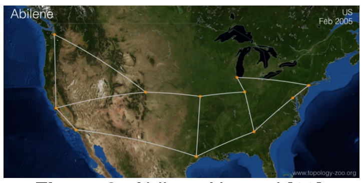


Figure 3: Abilene Network[13].

- Bell South Network [14], a fifty one node network:

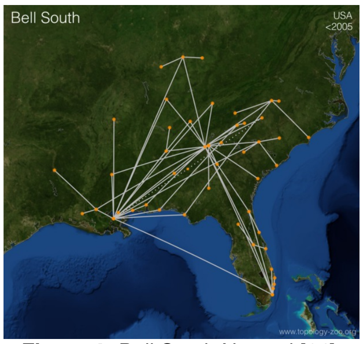


Figure 4: Bell South Network[14].

- GTS_CE Network. This is the largest network with one hundred and forty nine nodes:

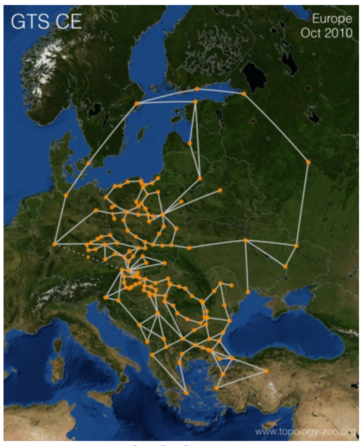


Figure 5: GTS_CE Network[15].

We considered realistic link delays based on empirical data [16].

To have statistical confidence in our results, we run our test, for each topology a thousand times. In each trial, we retrieve the time to converge, we send a random link down to emulate link failure, and retrieve the convergence time after the link failure.

To run the GTSCE Network, we had to create a Virtual Machine on a server to be able to have a better computational power than a personal computer has. Our Virtual Machine has 8 cores and a RAM with 20GB.

4.3. Results

In a networking environment, performance is everything. Therefore it is important, for a routing protocol, to assure that it converges in the smallest time possible.

In this section we present and discuss results on the stability of the Promise Protocol, in terms of routing changes, and performance, in terms of convergence time.

4.3.1. Stability

In this section we ask if the Promise Protocol improves the stability of the network compared with the DSDV protocol. To this goal, we measure how much each protocol change their routes states. We thus count the number of changed states, that is, the total number of times that every switch in the network had to change its forwarding table.

We compare the Promise Protocol with the baseline DSDV protocol.

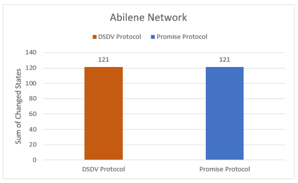


Figure 6: Comparison of the routing stability between the Promise Protocol and the DSDV baseline in Abilene Network.

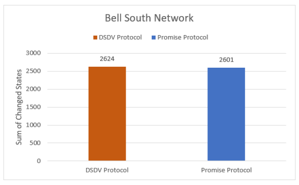


Figure 7: Comparison of the routing stability between the Promise Protocol and the DSDV baseline in Bell South Network.

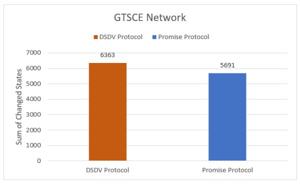


Figure 8: Comparison of the routing stability between the Promise Protocol and the DSDV baseline in GTSCE Network.

Figures 6-8 show the results on the number of changes without link failures. We can observe that the Promise Protocol is more stable. However, the difference between the baseline and the Promise Protocol is not significant for small networks. For smaller networks, the probes do not traverse many nodes, so there will not be too much delay. For that reason the optimal path is commonly the first path to be announced to the nodes for most of the times. For larger networks, like GTSCE, we can see that the difference is much higher and protocol stability becomes more relevant.

The second test includes failing a random link, and check the convergence time again. In Figures 9-11 we present, for each network, the number of changed states, after link failure.

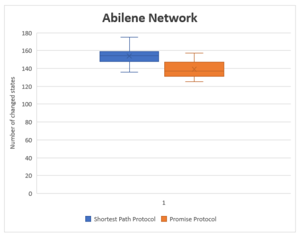


Figure 9: Comparison of the routing stability after one link failure in Abilene Network.

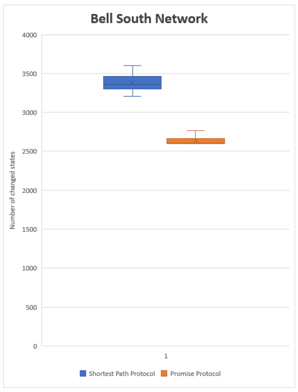


Figure 10: Comparison of the routing stability after one link failure in Bell South Network.

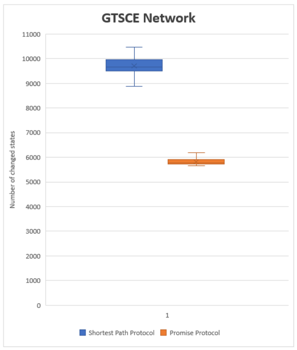


Figure 11: Comparison of the routing stability after one link failure in GTSCE Network.

In the Abilene network we observe a reduction of around 10% of the number of changed states when using the Promise Extension. For the Bell South network we got approximately a 22% reduction, and finally for the GTSCE network we got a 40% reduction.

4.3.2. Impact on merging the decision logic into the Data Plane

In this section, our main goal is to understand the performance gain of moving packet processing to the Data Plane.

As mentioned before, the Control Plane versions need to forward every probe to the Control Plane to be further processed. On the other side, the Data Plane versions have their decision logic offloaded to the Data Plane, which means that the switch is able to act on the received probes without accessing the Control Plane. Thus, the Data Plane versions will only forward the elected probes to the Control Plane, if there are changes to be done on the forwarding table.

In figures 12-17 we present the results as a box plot showing the median and the first and third quartiles.

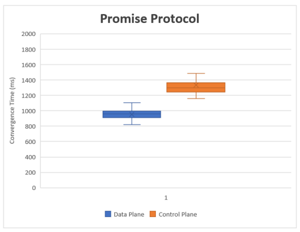


Figure 12: Convergence time of both Promise Protocol's versions in Abilene Network.

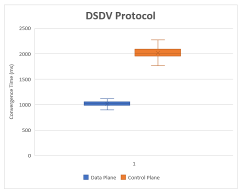


Figure 13: Convergence time of both DSDV Protocol's versions in Abilene Network.

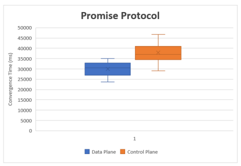


Figure 14: Convergence time of both Promise Protocol's versions in Bell South Network.

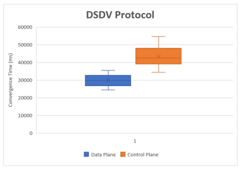


Figure 15: Convergence time of both DSDV Protocol's versions in Bell South Network.

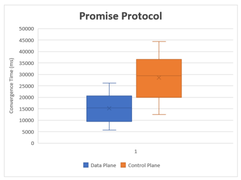


Figure 16: Convergence time of both Promise Protocol's versions in GTSCE Network.

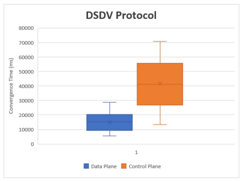


Figure 17: Convergence time of both DSDV Protocol's versions in GTSCE Network.

The main conclusion is that by offloading part of the protocol computation to the switch data plane we clearly improve convergence time. In addition, if the target is a hardware switch, we also save CPU cycles.

Also note that our evaluations were made on a software switch running in the CPU of a server. If it were made to run on real hardware (e.g., Intel Tofino[17]), the performance and scalability gains would be orders of magnitude higher. However, it is not clear whether we would need to take additional adaptations to the protocol to fit its capabilities to a real hardware pipeline.

4.4. Summary

With these tests, we have shown that the Promise Extension solves the issue of the DSDV Protocol: route fluctuation. As Figures 9-11 suggest, the Promise Protocol is more scalable than the DSDV Protocol. We keep observing a bigger reduction on the changed states as we use larger networks.

We can also conclude that processing packets in the Data Plane reduces the convergence time, no matter what the size of the network.

5. Conclusions

5.1. Summary

Data Plane programmability has brought us the freedom to innovate and create new routing protocols. Thanks to these advances, we are able to create new protocols that run in high rate in real networks.

In this paper, we have shown that the use of the *promise* makes network protocols more stable, improving the scalibility of large networks. Protocols like DSDV, that can cause route fluctuation, can thus benefit with the use of the Promise Extension. Since the nodes will not change routes so often, we will notice a decrease on broadcast traffic flooding through the network, leaving more free bandwidth to actually use it to send user data. Also, it can prevent packet reordering, so the users will get better network experience.

5.2. Future Work

For future research, we plan to integrate the Promise Extension on DSDV and on other distributed vectoring protocols. We also plan to evaluate their solution in real hardware.

References

[1] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, Oct. 1994. [Online]. Available: https://doi.org/10.1145/190809.190336

[2] G. Malkin, "Rip version 2," 1998.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rex-

ford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[4] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2534169.2486011

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[6] P4-16 language specification. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf

[7] J. L. Sobrinho, "Fundamental differences among vectoring routing protocols on non-isotonic metrics," *IEEE Networking Letters*, vol. 1, no. 3, 2019.

[8] P4runtime specification. [Online]. Available: https://p4.org/p4-spec/p4runtime/v1.0.0/P4Runtime-Spec.pdf

[9] Behavioral model (bmv2) library. [Online]. Available: https://github.com/p4lang/behavioral-model

[10] P4 compiler. [Online]. Available: https://github.com/p4lang/p4c

[11] Mininet. [Online]. Available: http://mininet.org/

[12] Topology zoo. [Online]. Available: http://www.topology-zoo.org/index.html

[13] Abilene network. [Online]. Available: http://www.topology-zoo.org/files/Abilene.gml

[14] Bell south network. [Online]. Available: http://www.topology-zoo.org/files/Bellsouth.gml

[15] Gtsce network. [Online]. Available: http://www.topology-zoo.org/files/GtsCe.gml

[16] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement based analysis, modeling, and synthesis of the internet delay space," 2006. [Online]. Available: https://doi.org/10.1145/1177080.1177091

[17] Barefoot tofino: world's fastest p4-programmable ethernet switch asics. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/