Designing a BGP-like protocol to provide QoS over SDN network

Ashirwad Pradhan 2019H1120043P, Shashank S 2019H1120054P

Abstract

The BGP protocol is a standardized EGP (Exterior Gateway Protocol) which is the main routing protocol on the internet today. This protocol is used by the Autonomous Systems to publish the set of IP prefixes they want to advertise to other Autonomous Systems. The protocol takes into account the rules-sets and policies set by the network administrator of the AS to configure routing decisions based on the paths that are advertised. Confederation of AS is a common practice where each AS in a confederation has iBGP peer with each other but on the internet, only one AS among the rest represents the confederation in order to exchange route information. This helps the AS to preserve the next hop, metrics, and preference information. While BGP is a well-known routing mechanism for Inter-AS communication it has its fair share of shortcomings. The pathfinding algorithm doesn't take into account the QoS of the path thereby producing routes that are sub-optimal for recent networking usage. The BGP cannot produce routing information based on bandwidth, latency or any other QoS related factors. Our approach suggests a method that will help to design an architecture over SDN which will take QoS into account when calculating the path for traffic which spans inter-AS.

Introduction

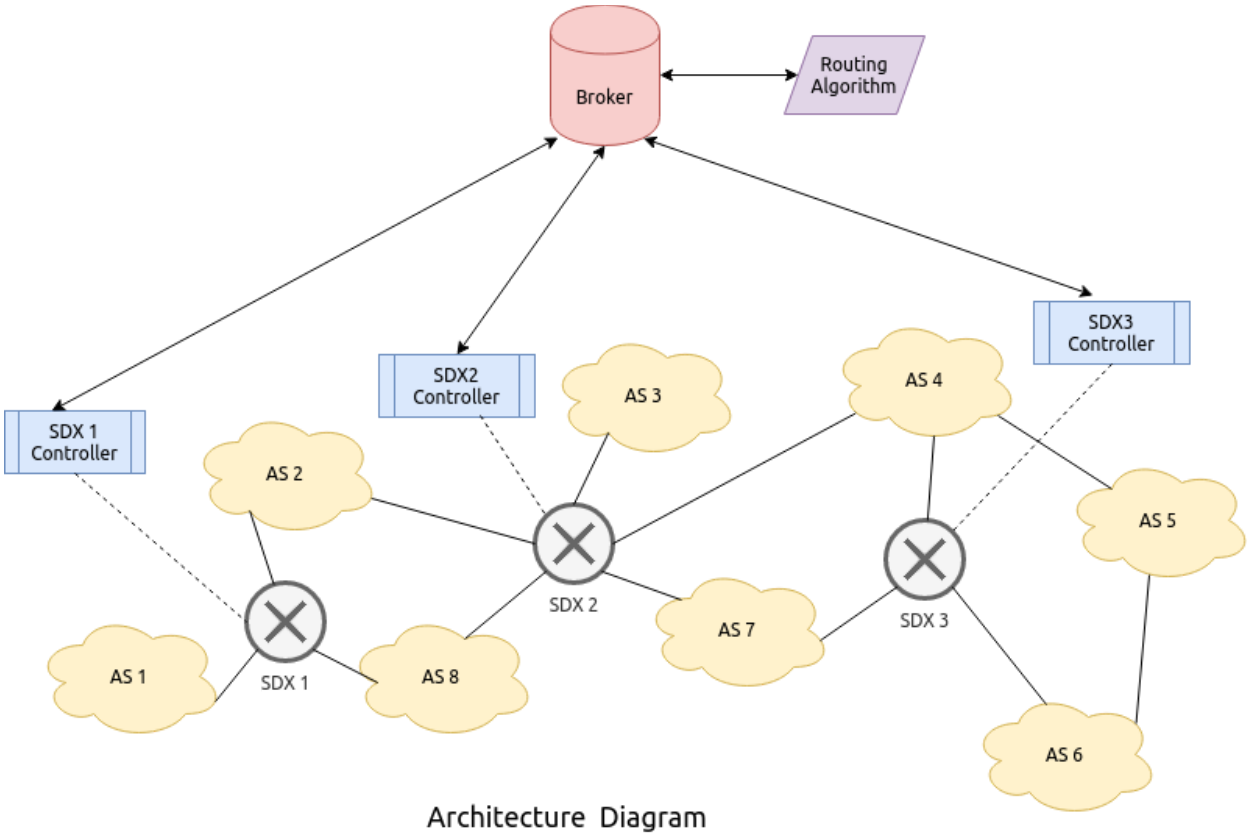
There has been some previous work involving QoS over SDN but none of the works explains the architecture as well as the routing algorithm which provides QoS and how it can be designed and implemented in a close to real world simulation.

Our solution focuses on the architecture based on Mininet and also provides a placeholder for any kind of routing algorithm which can be used in place. Mininet is a widely accepted solution to emulate large networks representative of real world networks. Mininet uses the network namespaces concept of Linux to create hosts which are having different network namespace thereby emulating as an individual network entity. To represent switches/routers we have used Open vSwitch in our implementation which are connected to the hosts described above. The switches are controlled by a SDN controller which is based on Ryu. The SDN controller uses OpenFlow 1.3 to communicate with the Open vSwitch. We have also used a publish-subscribe like system to take care of auctioning information from sellers and buyers. We have discussed more on this in the next section. To build a publish-subscribe system we have used Redis as our broker system. We have discussed how these above mentioned components can be used to realize a close to real world simulation which is actually implementable with very few changes. Our solution kindles more of an idea about a new perspective on how we can think a new BGP-like protocol on SDN could look like.

The Architecture of the System

The main idea behind our architecture is based on the idea of buying and selling. Let's take an analogy for example.

One fine day you wake up and you want to buy a car to fulfill your long awaited dream but the only thing is that you don't know which car to buy because there are 100s of models to choose from and other models you don't even know about. In a usual case, you have to visit all the dealers which are selling cars in your area and then brief them about your requirements and then they will suggest your car. But then you find out that the dealer doesn't have all the models and hence you move onto the next dealer to know about other models. In this way you will be frustrated at the end of the end and give up on your dream and it's definitely too much work. To make your situation easier, imagine an online site which takes your specific requirements as input let's say "Type, Top Speed, Engine HP, Cost" and then matches with the best possible car model which you can get given the cost you specified. This makes life so much easier.



Architecture Diagram

Compared with the above story: Here "you" are the Autonomous System who wants to subscribe to a path which provides best latency for a particular streaming service. The "car manufacturers" are the other AS which can provide the best latency path. The online site is the broker which matches the buyers and sellers requirements.

The AS who is a seller publishes a path which they want to sell. The AS sends this “publish/sell” message with the requirements to the SDX controller. The AS who wants to buy the path sends a “subscribe/buy” message to the SDX controller. All the SDX controller now sends all the above buy and sell messages to a broker. The broker basically subscribes to both the buyer as well as seller messages. After the broker gets all the messages it uses the routing algorithm to decide which seller sells to which buyer and a end to end global path is established. This path is now sent to all the SDX’s which install the paths using the flow modification message in their switch.

The architecture diagram represents the testbed that is implemented in order to experiment and check for the correctness of the routing algorithm. The routing algorithm is a placeholder that can be used to fit any custom-designed routing algorithm which works on the exposed predefined interface which is discussed later. These are the following components that are used to realize the above architecture.

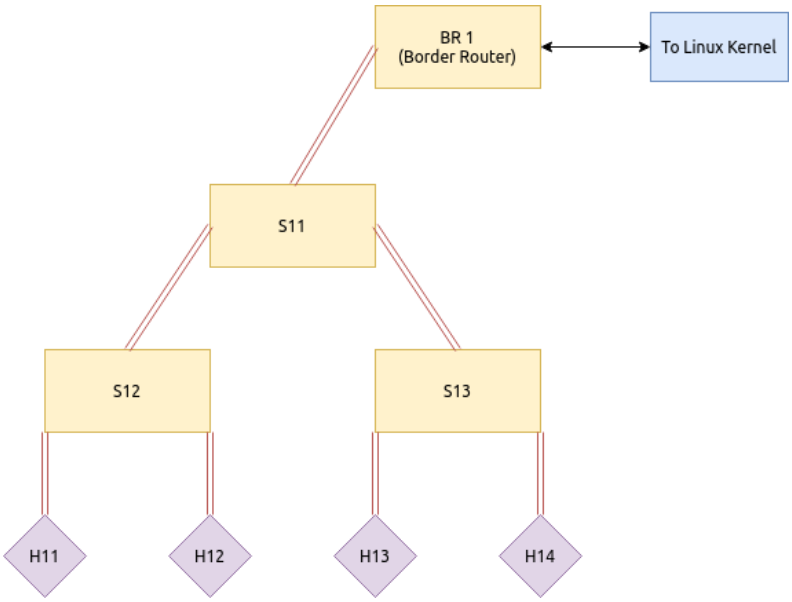
1. Autonomous System:

Each autonomous system is an Mininet instance which is used to emulate a network of hosts. The hosts are connected to each other with Open vSwitch. All the switches are controlled by a controller which is overlooked by the network administrator for that AS. Our implementation uses a custom controller based on Ryu. The controller can dictate the flows inside an individual switch to route traffic inside the AS.

- H11, H12, H13, H14 are all hosts inside the autonomous system. These are those systems which can provide any web service, streaming service.
- S11, S12, S13 are the AS internal switches. This takes care of the internal routing within an autonomous system.
- BR1 represents a border router that sends traffic to SDX on which it is connected to.

2. SDX Switch:

Each SDX switch is also a mininet instance with only one switch governed by one controller.
The SDX switch is the representation of an SDN based IXP which is an exchange point for two different autonomous systems. This allows two different AS to exchange traffic between one another. The SDX switch is governed by an SDX controller which is used to make flows based on rules defined by the broker in the publish-subscribe system. These flows in turn direct the traffic to the peer AS.



The SDX controller makes a request to the broker with the buy and sell messages it receives from it's peer autonomous systems. The SDX controller now sends the message to the broker which subscribes to both buy and sell messages. The broker now implements the routing algorithm by taking a list of all buy and sell messages as an input and produces a valid routing path. The routing path is sent to all the SDX's controllers so that it can be installed on the SDX switch. The default behaviour of the SDX switch without the routing algorithm in place is to flood the packets to all the ports.

3. How are different Mininet instances connected ?

Most of our research time went into figuring out this part. In the end we found a simple yet elegant solution.

There are many different solutions to achieve the above architecture. Every solution has its own problems to deal with. Our testbed consists of 8 AS's and 3 SDX's

- Each autonomous system and SDX can be represented with one VM entirely. In this case our testbed would need 11 VM's to work with. To connect them we can use GRE tunneling which is a well known method to connect two mininet instances running on different VM's.
  The problem with this approach is that it is too resource intensive. Each VM has its own set of processes to run and mininet on top of that. The main issue is that this setup is limited by the disk speed and usage. And moreover we only need 11 different mininet instances and not other services that are provided by the VM's. As the mininet instance is isolated in itself they are small containers within themselves with different namespaces. Therefore this idea was not the one we are going for.

- We can use 11 different physical machines and run Maxinet on top of that to simulate 8 AS's and 3 SDX's. This setup would be an overkill and moreover our concern here is to make a most efficient testbed and clearly this approach is not efficient enough for our problem at hand.
- **Our Solution:**

We created 11 different mininet instances on the same machine which is very resource efficient but how do we connect them. We could find no recorded documentation on how to do it for two mininet instances.

So we tried to connect two different Open vSwitch through Linux kernel routing.

Each border router is assigned with an IPv4 address corresponding to its mininet instance. And then we create a route between those assigned IPv4 addresses in the Linux Kernel Routing Table. To pass the packets from the Open vSwitch to the Linux Kernel we used the port (**local**) **OFPP.LOCAL** as our output port in the flow of the border router. The above can be achieved using the following commands.

For border router br1,

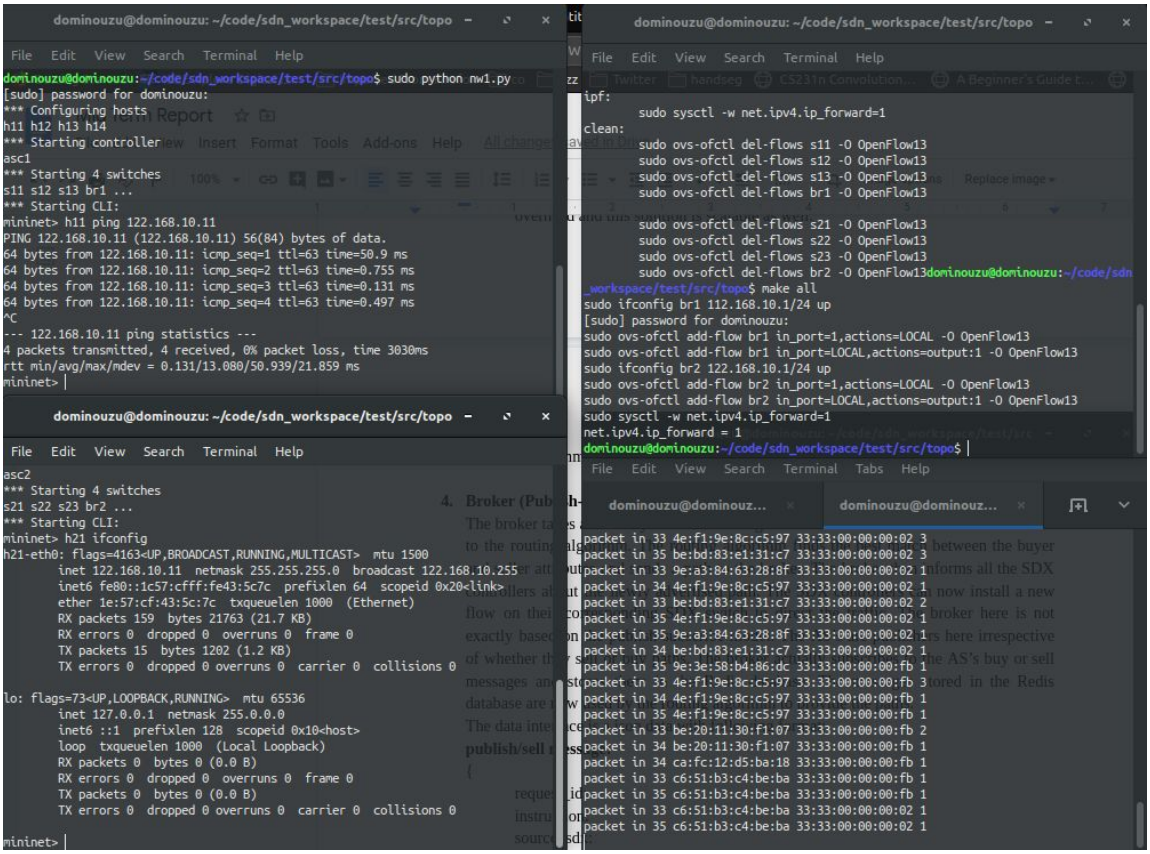
```
sudo ovs-ofctl add-flow br1 in_port=1,actions=LOCAL -O OpenFlow13
sudo ovs-ofctl add-flow br1 in_port=LOCAL,actions=output:1 -O OpenFlow13
```

For SDX switch,

```
sudo ovs-vsctl add-flow sdx in_port=LOCAL,actions=LOCAL -O OpenFlow13
```

So what we are essentially doing in the above steps is that we are passing the packets coming to the border router from port 1 to Linux Kernel, the Linux Kernel now routes the packet to SDX as we had already set up the route in place. The SDX now again sends all the packets to Linux kernel as the default behaviour of the switch is to flood the packets it receives. The routing algorithm when in place can dictate how SDX manages the packets by installing flows in it.

We have managed to realize a system with 11 mininet instances with minimal overhead and this solution is scalable as well.



4. Broker (Publish-Subscribe System):

The broker takes all the buy and send messages from the SDX controller and then sends it to the routing algorithm. The routing algorithm finds the best match between the buyer and seller attributes and sends a path to the broker. The broker then informs all the SDX controllers about the newly advertised path. The SDX controllers can now install a new flow on their corresponding SDX switch to direct the traffic. The broker here is not exactly based on the publish-subscribe model. The AS's are publishers here irrespective of whether they sell or buy paths. The broker actually subscribes to the AS's buy or sell messages and stores them in the Redis database. The messages stored in the Redis database are now used by the routing algorithm to provide the paths.

The data interface is a json data with following format:

publish/sell message:

{

request_id:

instruction:

source_sdx:

destination_sdx:

pathelet_id:

bandwidth:
latency:
cost:
lifetime:

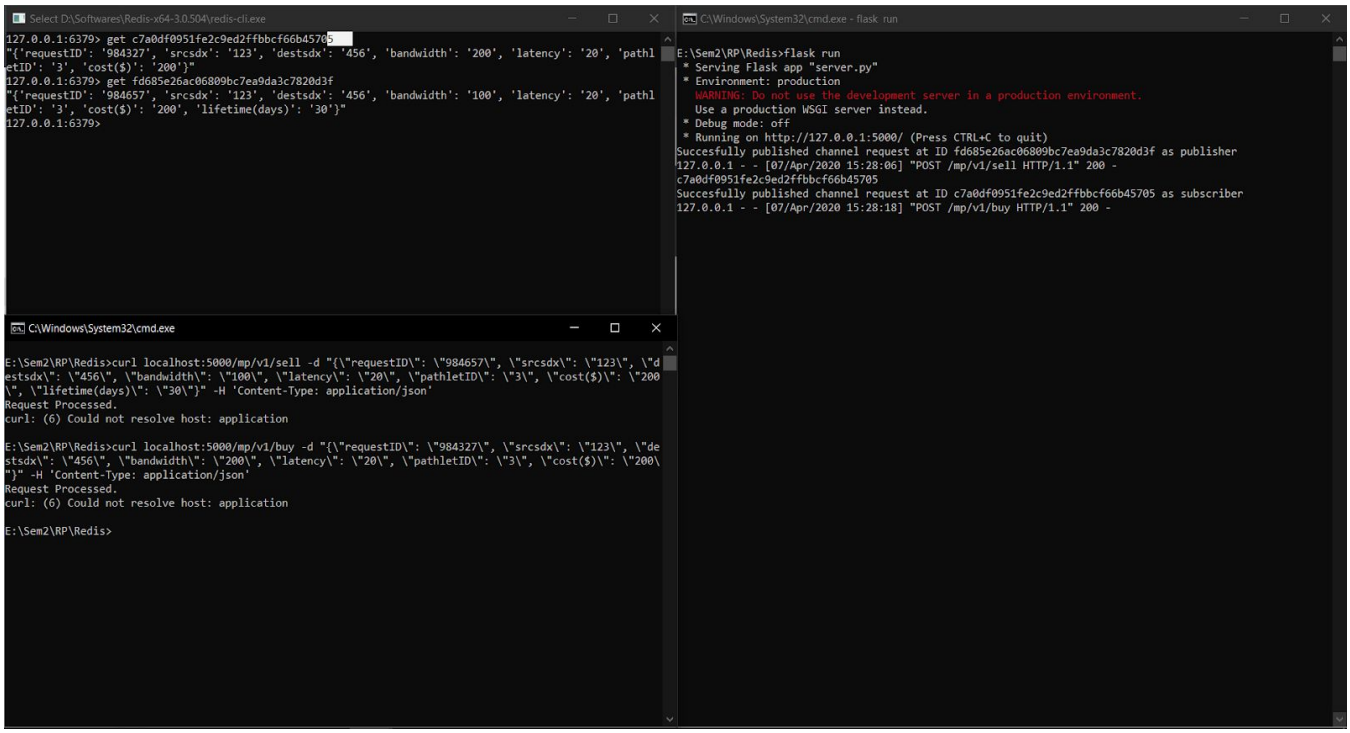
}

subscribe/sell message:

{

request_id:
source_ip_prefix:
dest_ip_prefix:
bid_amount:
bandwidth:
latency:

}



Conclusion

The above architecture helps us to realize a close enough real world emulation of inter AS communication. This architecture can be used to test a EGP routing algorithm which can take QoS parameters and come up with a path which cannot be provided by BGP.