University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Experiment 9 : Implementation of BGP Protocol

Submitted By:

Name: Muztoba Hasan Sinha

Roll No : 15

Name: Bholanath Das Niloy

Roll No : 22

Submitted On :

April 19, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

Contents

1 Introduction

We were tasked with implementing a network of routers to simulate the BGP protocol. This was to improve our understanding of the BGP protocols and further develop the practical understanding required to use this algorithm in distributed systems.

2 Theory

2.1 Overview

Border Gateway Protocol (BGP) is a standardized exterior gateway protocol designed to exchange routing and reachability information among autonomous systems (AS) on the Internet. BGP is classified as a path-vector routing protocol, and it makes routing decisions based on paths, network policies, or rule-sets configured by a network administrator.

BGP used for routing within an autonomous system is called Interior Border Gateway Protocol, Internal BGP (iBGP). In contrast, the Internet application of the protocol is called Exterior Border Gateway Protocol, External BGP (eBGP).

2.2 Operating Principles

BGP neighbors, called peers, are established by manual configuration among routers to create a TCP session on port 179. A BGP speaker sends 19-byte keep-alive messages every 30 seconds (protocol default value, tunable) to maintain the connection. Among routing protocols, BGP is unique in using TCP as its transport protocol.

When BGP runs between two peers in the same autonomous system (AS), it is referred to as Internal BGP (iBGP or Interior Border Gateway Protocol). When it runs between different autonomous systems, it is called External BGP (eBGP or Exterior Border Gateway Protocol). Routers on the boundary of one AS exchanging information with another AS are called border or edge routers or simply eBGP peers and are typically connected directly, while iBGP peers can be interconnected through other intermediate routers. Other deployment topologies are also possible, such as running eBGP peering inside a VPN tunnel, allowing two remote sites to exchange routing information in a secure and isolated manner.

The main difference between iBGP and eBGP peering is in the way routes that were received from one peer are typically propagated by default to other peers:

- New routes learned from an eBGP peer are re-advertised to all iBGP and eBGP peers.
- New routes learned from an iBGP peer are re-advertised to all eBGP peers only.

These route-propagation rules effectively require that all iBGP peers inside an AS are interconnected in a full mesh with iBGP sessions.

How routes are propagated can be controlled in detail via the route-maps mechanism. This mechanism consists of a set of rules. Each rule describes, for routes matching some given criteria, what action should be taken. The action could be to drop the route, or it could be to modify some attributes of the route before inserting it in the routing table.

During the peering handshake, when OPEN messages are exchanged, BGP speakers can negotiate optional capabilities of the session, including multiprotocol extensions and various recovery modes. If the multiprotocol extensions to BGP are negotiated at the time of creation, the BGP speaker can prefix the Network Layer Reachability Information (NLRI) it advertises with an address family prefix. These families include the IPv4 (default), IPv6, IPv4/IPv6 Virtual Private Networks and multicast BGP. Increasingly, BGP is used as a generalized signaling protocol to carry information about routes that may not be part of the global Internet, such as VPNs.

In order to make decisions in its operations with peers, a BGP peer uses a simple finite state machine (FSM) that consists of six states: **Idle**; **Connect**; **Active**; **OpenSent**; **OpenConfirm**; and **Established**. For each peer-to-peer session, a BGP implementation maintains a state variable that tracks which of these six states the session is in. The BGP defines the messages that each peer should exchange in order to change the session from one state to another.

The first state is the Idle state. In the Idle state, BGP initializes all resources, refuses all inbound BGP connection attempts and initiates a TCP connection to the peer. The second state is Connect. In the Connect state, the router waits for the TCP connection to complete and transitions to the OpenSent state if successful. If unsuccessful, it starts the ConnectRetry timer and transitions to the Active state upon expiration. In the Active state, the router resets the ConnectRetry timer to zero and returns to the Connect state. In the OpenSent state, the router sends an Open message and waits for one in return in order to transition to the OpenConfirm state. Keepalive messages are exchanged and, upon successful receipt, the router is placed into the Established state. In the Established state, the router can send and receive: Keepalive; Update; and Notification messages to and from its peer.

2.2.1 Idle State

1. Refuse all incoming BGP connections.
2. Start the initialization of event triggers.
3. Initiates a TCP connection with its configured BGP peer.

4. Listens for a TCP connection from its peer.
5. Changes its state to Connect.
6. If an error occurs at any state of the FSM process, the BGP session is terminated immediately and returned to the Idle state. Some of the reasons why a router does not progress from the Idle state are:
   (a) TCP port 179 is not open.
   (b) A random TCP port over 1023 is not open.
   (c) Peer address configured incorrectly on either router.
   (d) AS number configured incorrectly on either router.

2.2.2 Connect State

1. Waits for successful TCP negotiation with peer.
2. BGP does not spend much time in this state if the TCP session has been successfully established.
3. Sends Open message to peer and changes state to OpenSent.
4. If an error occurs, BGP moves to the Active state. Some reasons for the error are:
   (a) TCP port 179 is not open.
   (b) A random TCP port over 1023 is not open.
   (c) Peer address configured incorrectly on either router.
   (d) AS number configured incorrectly on either router.

2.2.3 Active State

1. If the router was unable to establish a successful TCP session, then it ends up in the Active state.
2. BGP FSM tries to restart another TCP session with the peer and, if successful, then it sends an
3. Open message to the peer.
4. If it is unsuccessful again, the FSM is reset to the Idle state.
5. Repeated failures may result in a router cycling between the Idle and Active states. Some of the reasons for this include:
   (a) TCP port 179 is not open.
   (b) A random TCP port over 1023 is not open.
   (c) BGP configuration error.
   (d) Network congestion.
   (e) Flapping network interface.

2.2.4 OpenSent State

1. BGP FSM listens for an Open message from its peer.
2. Once the message has been received, the router checks the validity of the Open message.
3. If there is an error it is because one of the fields in the Open message does not match between the peers, e.g., BGP version mismatch, the peering router expects a different My AS, etc. The router then sends a Notification message to the peer indicating why the error occurred.
4. If there is no error, a Keepalive message is sent, various timers are set and the state is changed to OpenConfirm.

2.2.5 OpenConfirm State

1. The peer is listening for a Keepalive message from its peer.
2. If a Keepalive message is received and no timer has expired before reception of the Keepalive, BGP transitions to the Established state.
3. If a timer expires before a Keepalive message is received, or if an error condition occurs, the router transitions back to the Idle state.

2.2.6 Established State

1. In this state, the peers send Update messages to exchange information about each route being advertised to the BGP peer.
2. If there is any error in the Update message then a Notification message is sent to the peer, and BGP transitions back to the Idle state.

2.3 Finite State Machine for the BGP Protocol

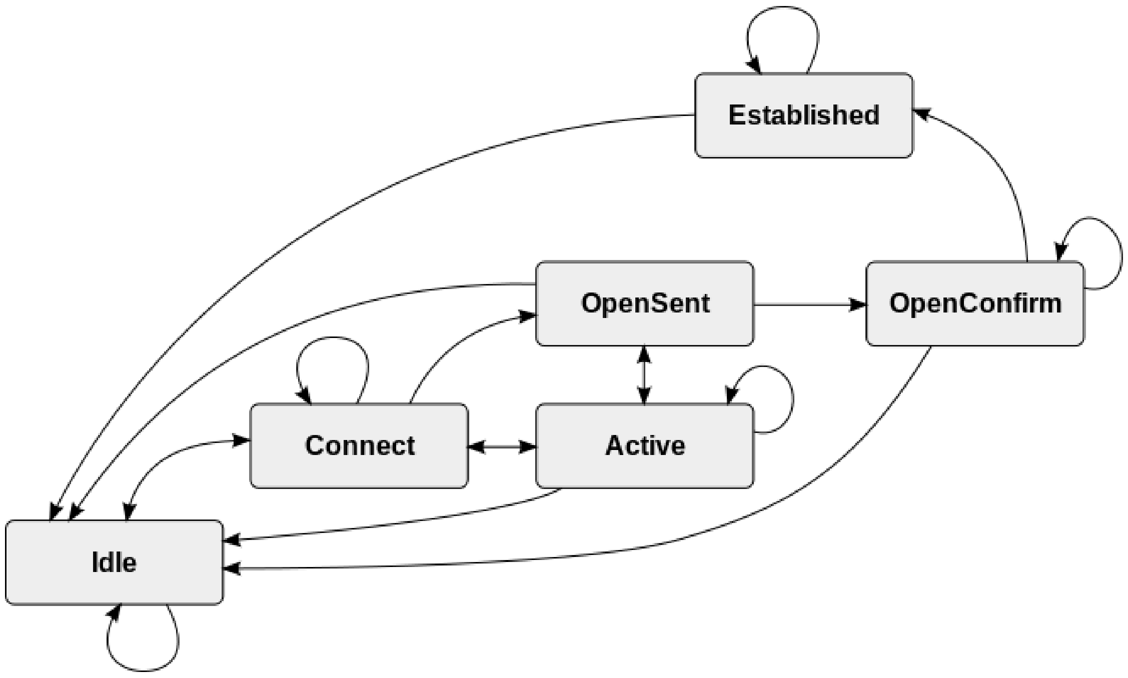


Figure 1: Finite State Machine for the BGP Protocol

2.4 BGP Message Types

BGP (Border Gateway Protocol) uses several types of messages to exchange routing information between BGP routers. The message types are:

1. **OPEN message**: The first message exchanged between BGP peers after they establish a TCP connection. The OPEN message contains information about the BGP capabilities of each peer, including the AS number, the BGP version number, and any optional capabilities.
2. **UPDATE message**: The most important message in BGP. It is used to advertise routing information between BGP peers. The UPDATE message contains information about a specific destination prefix, including the network address, subnet mask, and the next-hop address to reach the destination. The message may also include BGP attributes, such as the AS path, local preference, and community values, that provide additional information about the route.
3. **KEEPALIVE message**: A message sent periodically by each peer to inform the other peer that the BGP session is still active and that the router is still available to exchange routing information. If a peer does not receive a KEEPALIVE message within a certain time period, it will assume that the BGP session has failed and will terminate the connection.
4. **NOTIFICATION message**: A message used to signal an error condition to the peer. It may be sent for a variety of reasons, such as an invalid message format, an unrecognized attribute, or a problem with the BGP session. When a peer receives a NOTIFICATION message, it will terminate the BGP session and take appropriate action to resolve the problem.

Overall, these four BGP message types work together to establish and maintain a reliable exchange of routing information between BGP peers.

2.5 BGP Attributes

BGP (Border Gateway Protocol) uses attributes to describe the properties of a route, such as the path the route takes through the internet, the AS (Autonomous System) that originated the route, and the preference or metric of the route. BGP routers use these attributes to determine the best path to a particular destination.

The BGP attributes are:

1. **AS_PATH**: Specifies the AS numbers that a route passes through to reach a destination. This is used to prevent loops in the network and to ensure that the best path is chosen based on the AS path length.
2. **NEXT_HOP**: Specifies the IP address of the next router in the path to the destination. This is used to forward packets to the correct next-hop router.
3. **LOCAL_PREF**: Specifies the local preference of the route, which is used to determine the best path within a single AS. This attribute is only used within an AS and is not advertised to other ASes.
4. **MED**: Specifies the metric or preference of the route, which is used to determine the best path between different ASes. This attribute is not always used, and it is only effective if the ASes have agreed to use it.
5. **ORIGIN**: Specifies the origin of the route, which can be one of three types: IGP (Interior Gateway Protocol), EGP (Exterior Gateway Protocol), or incomplete. This attribute is used to determine the trustworthiness of the route.
6. **COMMUNITY**: Specifies a community value that can be used to group routes together. This attribute is used for policy enforcement and routing control.
7. **MULTI_EXIT_DISC (MED)**: Specifies an alternate metric or preference of the route, which is used to influence traffic flow to a particular destination.
8. **AGGREGATOR**: Specifies the AS number and router ID of the BGP router that aggregated the route.
9. **CLUSTER_LIST**: Specifies a list of cluster IDs that are part of the cluster of BGP routers that advertised the route. This attribute is used to prevent routing loops when there are multiple BGP routers in the same AS.

These BGP attributes play a critical role in the path selection process, and they enable BGP routers to choose the best path to a particular destination.

3 Network Topology Details

3.1 Graph

3.1.1 Physical Network

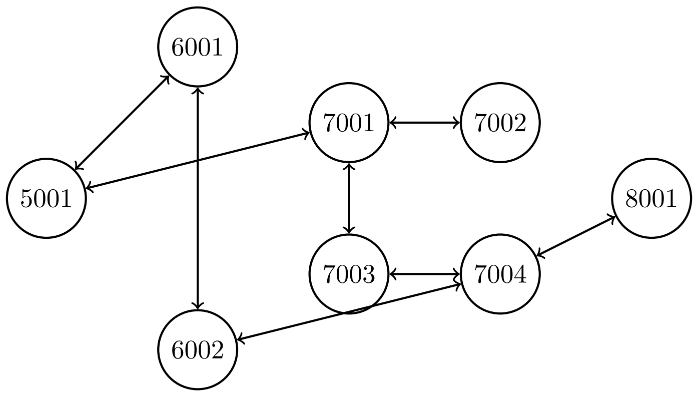


Figure 2: Physical Network

3.1.2 Virtual Network

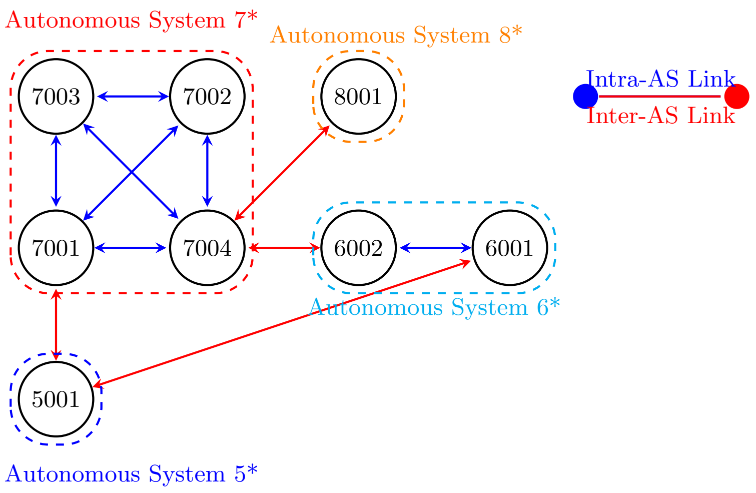


Figure 3: Virtual Network

3.2 AS-Paths

3.2.1 Autonomous System 5*

AS	**AS Paths to Reach It**
AS5*	-
AS6*	AS5*
AS7*	AS5*
AS8*	AS5*, AS7*

Table 1: AS paths to reach a device connected to AS5*

3.2.2 Autonomous System 6*

AS	**AS Paths to Reach It**
AS5*	AS6*
AS6*	-
AS7*	AS6*
AS8*	AS6*, AS7*

Table 2: AS paths to reach a device connected to AS6*

3.2.3 Autonomous System 7*

AS	**AS Paths to Reach It**
AS5*	AS7*
AS6*	AS7*
AS7*	-
AS8*	AS7*

Table 3: AS paths to reach a device connected to AS7*

3.2.4 Autonomous System 8*

AS	**AS Paths to Reach It**
AS5*	AS8*, AS7*
AS6*	AS8*, AS7*
AS7*	AS8*
AS8*	-

Table 4: AS paths to reach a device connected to AS8*

3.3 Forwarding Tables

3.3.1 Autonomous System 5*

Destination	Next Hop
AS6*	6001
AS7*	7001
AS8*	7001

Table 5: Forwarding Table for AS5*

3.3.2 Autonomous System 6*

Destination	Next Hop
AS5*	5001
AS7*	7004
AS8*	7004

Table 6: Forwarding Table for AS6*

3.3.3 Autonomous System 7*

Destination	Next Hop
AS5*	5001
AS6*	6002
AS8*	7004

Table 7: Forwarding Table for AS7*

3.3.4 Autonomous System 8*

Destination	Next Hop
AS5*	7004
AS6*	7004
AS7*	7004

Table 8: Forwarding Table for AS8*

3.4 Example of Policy Based Routing

Suppose we go back to our original graph once again, but now the `7001` router decides it will no longer forward any data packets coming from `AS5*` this will thus change the `AS-Paths`. The red indicates, it is doing policy based routing, and the green edge indicates that the link is basically useless here. Here `X` is a device connected to the `AS5*`.

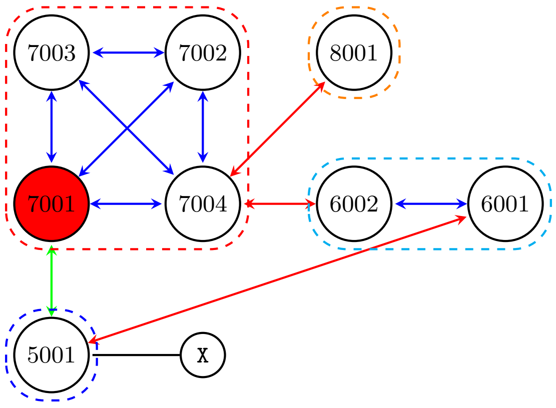


Figure 4: The `7001` router will not forward packets from `AS5*`

The updated the `AS-Paths` for a device connected to `AS5*` is as follows.

AS	**AS Paths to Reach It**
AS5*	-
AS6*	AS5*
AS7*	AS5*, AS6*
AS8*	AS5*, AS6*, AS7*

Table 9: Changed AS-Paths for a device connected to AS5*

This will also change the forwarding tables.

Destination	Next Hop
AS6*	6001
AS7*	6001
AS8*	6001

Table 10: Updated Forwarding Table for AS5*

3.5 Local Preferences

BGP attribute local preference is the second BGP attribute and it can be used to choose the exit path for an autonomous system. Here are the details:

1. Local preference is the second BGP attribute.
2. We can use local preference to choose the outbound external BGP path.
3. Local preference is sent to all internal BGP routers in your autonomous system.
4. Not exchanged between external BGP routers.
5. Local preference is a well-known and discretionary BGP attribute.
6. The default value is 100.
7. The path with the highest local preference is preferred

To demonstrate the local preference we have set the local preference of the AS6 to 150 in the node 7004 that way all of the nodes will get the propagated information and will pick an indirect path to the AS 5000 by disregarding the direct link to 5000. If we see this seemingly unoptimal behavior by the nodes we can conclude that our policy has worked.

3.5.1 Node 7001

Note how it disregards the direct link.

AS	**AS Paths to Reach It**
AS5*	AS6*, AS5*
AS6*	AS6*
AS7*	-
AS8*	AS8*

Table 11: AS paths from 7001 to all other prefixes

Destination	Next Hop	Local_Pref
AS5*	7004	150
AS6*	7004	150
AS8*	7004	100

Table 12: Forwarding Table for 7001

Note how the node has disregarded the link to 5000 through 7001. All other nodes show similar behaviour so we can conclude that the policy based routing has worked

4 Considerations for implementation

We have considered a few restrictions for implementation. Which are listed as follows

4.0.1 The graph is unweighted

The graph is unweighted because BGP weights are configured locally by policy according to each AS with the various attributes. Globally it works like a unweighted graph with AShops being the condition for routing.

4.0.2 The graph is not a multigraph

The graph is not a multigraph. That is two nodes are not directly connected by more than one edge/link. Since the graph is unweighted it wouldn't really make sense to add multiple direct edges between any two nodes.

4.0.3 The graph is undirected

The graph is undirected because the connections are duplex. Although we have planned for a possibility where it might not be, discussed in this section.

4.1 Modeling the network

We have maintained the following abstractions to implement BGP

4.1.1 Mapping port numbers to IP addresses

Since we are working in a local machine we have abstracted IP addresses to port numbers. To get a working BGP in real life we can simply just change the prefix logic and point it to the port 179. (since BGP runs on port 179). Here we are considering a prefix to all numbers greater than the thousandth place of the decimal number of the port. In real life it is the binary prefix of a number.

4.1.2 All nodes in the AS have information of all other nodes in the AS

This is done by the *iLinks* array in the code. This is to create BGP connections to all other nodes in the AS. Although it could be argued that only the border nodes could have full connectivity, The algorithm would still work. But we wanted to maintain homogeneity.

4.1.3 Disregarding intra AS routing

Our implementation completely disregards intra AS routing as intra AS routing completely depends on each ISP and it's routing algorithm choice which could be very different. In theory our algorithm would still work if we added different Intra ISP routing. But that'd be too hard to test.

4.1.4 Setting Local Preference

Local preference is set at each AS's border nodes and is propagated throughout. Also, it is not shared with the outside nodes because we can't force our policy to other ASes.

4.1.5 Disregarding all messages except UPDATE

To simplify the network we have chosen not to use the OPEN, KEEPALIVE and NOTIFICATION message. Since we don't have to handle error conditions. We assume (rightfully) that all nodes are up and are successfully sending the data to other nodes. Our confidence is backed by the TCP protocol and the fact that we are using the same machine to simulate all nodes.

4.1.6 BGP attributes handled

Our goal was to show that we can infact change intra AS routing based on policy. Now in the real world this change can be affected by a large number of attributes. We choose to focus on the following to implement our algorithm simply

1. **AS_PATH** This is a must for routing based on hops
2. **NEXT_HOP** This is again a must to fill the forwarding table
3. **LOCAL_PREF** This is the attribute we have chosen to show that we can indeed influence our shortest path with policy.

5 Implementation of the Code Itself

5.1 Accessory scripts

Since we are using a large number of nodes, manually running them is quite cumbersome, that is why we have created two python scripts for quickly opening and closing the nodes in seperate terminal windows. These scripts assume that we are using a linux system and have the *alacritty* terminal installed. If we don't simply change the text 'alacritty' to whatever terminal emulator we have installed. (PS. We couldn't get a great solution for windows, we suspect due to it's environment variables)

5.1.1 opener.py

It has an array of all files to launch and opens them in seperate windows so that we can monitor each node independently.

5.1.2 closer.py

It uses psutil to identify all instances of alacritty and shuts them off.

5.2 Constants and state information in each node

Each node contains some information to help it route and to identify it as unique these are listed below.

5.2.1 ME

It simply contains the port number for the node. Ideally to implement BGP this would be simply replaced by the IP address.

5.2.2 localPrefTable

This contains all the local preference set by the network administrator.

5.2.3 PrefixTable

It contains the prefixes we know the route to and their routing information that is the next hop and the local preference of that route.

5.2.4 ASpaths

This records the ASpaths to all prefixes. Ideally we don't need such a map. But to fulfil the requirement of this assignment we need to show it.

5.2.5 ADPREF

Contains the Address prefixes of that nodes AS itself.

5.2.6 ASN

Contains the Autonomus System Number of that node.

5.3 Explanation of functions in the code

5.3.1 query

This handles the queries for each node. It simply takes a destination node port and extracts its prefix and searches for it in the same AS or the discovered prefixes. And prints the output accordingly

5.3.2 send

This function is used to send both eBGP and iBGP messages by simply swapping the link list and the origin. It constructs the message and then sends to the supplied by opening TCP connections to the ports.

5.3.3 sendPref

Some messages might contain a Local Preference. So this function sends the local pref to only the internal links just as above.

5.3.4 update

It simply updates the prefix table and ASpath given required parameters. This function simply exists for code cleanup.

5.3.5 recvt

Opens a server at the nodes defined port and listens for incoming connections. If it find a connection it opens a new thread an passes that thread the *receive* function and parameters to be independently handled.

5.3.6 receive

It receives message from the connection. Sees if the message is a loop and passes it on to be handled by *handlBGP* if the message is not looped.

5.4 Handling BGP

BGP functions are handled in the function *handleBGP*. It has a few steps

5.4.1 Local Prefrence additon

If the message is eBGP, it checks for the last hop AS and tries to see if any non default is set to it, if so it appends the local pref to the message and marks it for iBGP forwarding.

5.4.2 Updating

Updating follows a few conditions.

1. We simply add the prefix to the table if we haven't seen it before.
2. If our table already has the entry we check if the incoming local prefix is higher than the one in the table if so we update.
3. Failing that if the received path has less AS hops than the saved path we update.

5.4.3 Messaging

Messaging has a few conditions as well.

1. **If the incoming message is iBGP** We append our ASN to the path, update next hop as the current node and send it to all external links.
2. **If the incoming message is eBGP** This again has two conditions
   (a) *sending to external links* We append our ASN to the path, update next hop as current node and send it to all external links.
   (b) *sending to internal links* We set next hop as current node and see if there is a local preference added. If so we send with local preference else we send the message with just the next hop update.

5.5 The main loop

The program starts by opening a thread with the recvt function to handle messaging on separate threads. Then it waits for 5 seconds this is arbitrary to let all of the nodes to start. Then it advertises it's Address prefix to all external links. then it waits for the system to reach equilibrium then it accepts inputs to see if that address is reachable and how.

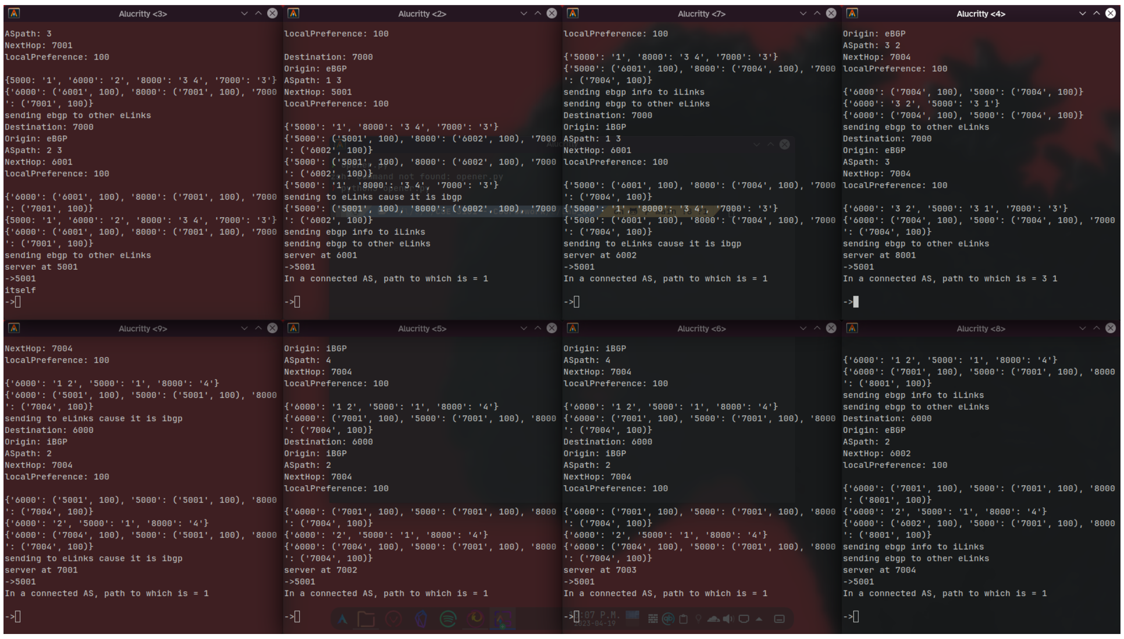
6 Extensions

We can simply extend the system to behave as a real BGP implementation by a few small steps, (not implemented due to time shortages)

1. If we keep ADPREF as an array then we can support arbitrary prefixes in the ASN just like in real life all we'd have to do is handle a small update operation when ASpath only contains self AS number.
2. We can not route outside traffic by not advertising iBGP to external links thus making parts of the graph directed and saving bandwidth costs.

7 Screenshots of working code

7.1 Working with all local preference set to 100



Alucritty <3>

```
ASpath: 3
NextHop: 7001
localPreference: 100

{5000: '1', '6000': '2', '8000': '3 4', '7000': '3'}
{'6000': ('6001', 100), '8000': ('7001', 100), '7000
': ('7001', 100)}
sending ebgp to other eLinks
Destination: 7000
Origin: eBGP
ASpath: 2 3
NextHop: 6001
localPreference: 100

{'6000': ('6001', 100), '8000': ('7001', 100), '7000
': ('7001', 100)}
{5000: '1', '6000': '2', '8000': '3 4', '7000': '3'}
{'6000': ('6001', 100), '8000': ('7001', 100), '7000
': ('7001', 100)}
sending ebgp to other eLinks
server at 5001
->5001
itself
->
```

Alucritty <2>

```
localPreference: 100

Destination: 7000
Origin: eBGP
ASpath: 1 3
NextHop: 5001
localPreference: 100

{'5000': '1', '8000': '3 4', '7000': '3'}
{'5000': ('5001', 100), '8000': ('6002', 100), '7000
': ('6002', 100)}
{'5000': ('5001', 100), '8000': ('6002', 100), '7000
': ('6002', 100)}
{'5000': '1', '8000': '3 4', '7000': '3'}
sending to eLinks cause it is ibgp
{'5000': ('5001', 100), '8000': ('6002', 100), '7000
': ('6002', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
server at 6001
->5001
In a connected AS, path to which is = 1

->
```

Alucritty <7>

```
localPreference: 100

{'5000': '1', '8000': '3 4', '7000': '3'}
{'5000': ('6001', 100), '8000': ('7004', 100), '7000
': ('7004', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
Destination: 7000
Origin: iBGP
ASpath: 1 3
NextHop: 6001
localPreference: 100

{'5000': ('6001', 100), '8000': ('7004', 100), '7000
': ('7004', 100)}
{'5000': '1', '8000': '3 4', '7000': '3'}
{'5000': ('6001', 100), '8000': ('7004', 100), '7000
': ('7004', 100)}
sending to eLinks cause it is ibgp
server at 6002
->5001
In a connected AS, path to which is = 1

->
```

Alucritty <4>

```
Origin: eBGP
ASpath: 3 2
NextHop: 7004
localPreference: 100

{'6000': ('7004', 100), '5000': ('7004', 100)}
{'6000': '3 2', '5000': '3 1'}
{'6000': ('7004', 100), '5000': ('7004', 100)}
sending ebgp to other eLinks
Destination: 7000
Origin: eBGP
ASpath: 3
NextHop: 7004
localPreference: 100

{'6000': '3 2', '5000': '3 1', '7000': '3'}
{'6000': ('7004', 100), '5000': ('7004', 100), '7000
': ('7004', 100)}
sending ebgp to other eLinks
server at 8001
->5001
In a connected AS, path to which is = 3 1

->
```

Alucritty <9>

```
NextHop: 7004
localPreference: 100

{'6000': '1 2', '5000': '1', '8000': '4'}
{'6000': ('5001', 100), '5000': ('5001', 100), '8000
': ('7004', 100)}
sending to eLinks cause it is ibgp
Destination: 6000
Origin: iBGP
ASpath: 2
NextHop: 7004
localPreference: 100

{'6000': ('5001', 100), '5000': ('5001', 100), '8000
': ('7004', 100)}
{'6000': '2', '5000': '1', '8000': '4'}
{'6000': ('7004', 100), '5000': ('5001', 100), '8000
': ('7004', 100)}
sending to eLinks cause it is ibgp
server at 7001
->5001
In a connected AS, path to which is = 1

->
```

Alucritty <5>

```
Origin: iBGP
ASpath: 4
NextHop: 7004
localPreference: 100

{'6000': '1 2', '5000': '1', '8000': '4'}
{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
Destination: 6000
Origin: iBGP
ASpath: 2
NextHop: 7004
localPreference: 100

{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
{'6000': '2', '5000': '1', '8000': '4'}
{'6000': ('7004', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
server at 7002
->5001
In a connected AS, path to which is = 1

->
```

Alucritty <6>

```
Origin: iBGP
ASpath: 4
NextHop: 7004
localPreference: 100

{'6000': '1 2', '5000': '1', '8000': '4'}
{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
Destination: 6000
Origin: iBGP
ASpath: 2
NextHop: 7004
localPreference: 100

{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
{'6000': '2', '5000': '1', '8000': '4'}
{'6000': ('7004', 100), '5000': ('7001', 100), '8000
': ('7004', 100)}
server at 7003
->5001
In a connected AS, path to which is = 1

->
```

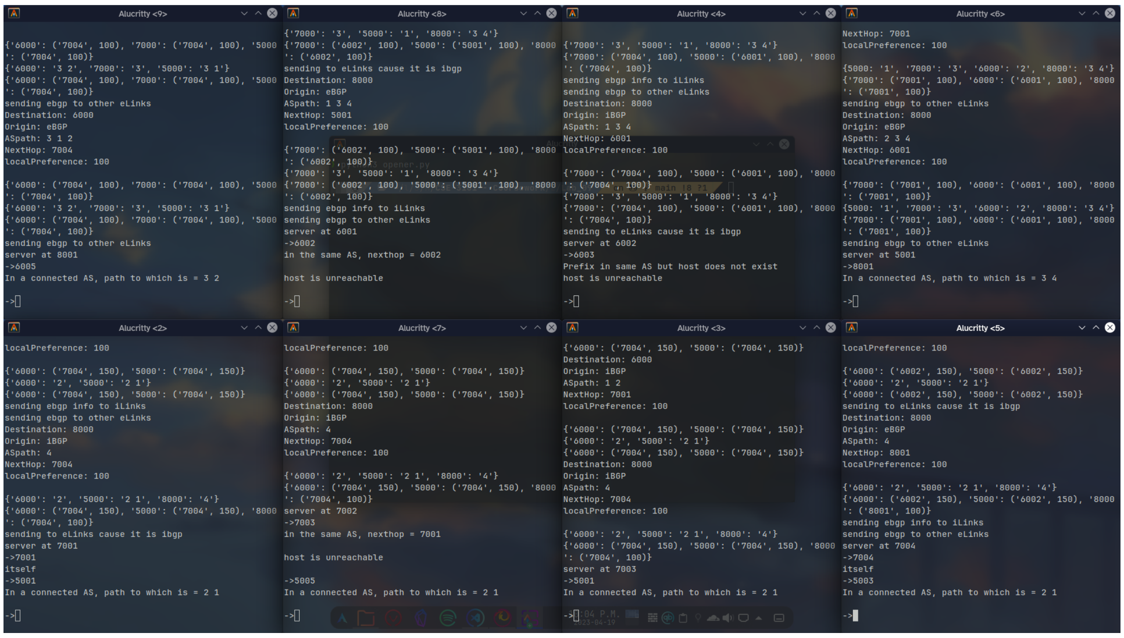
Alucritty <8>

```

{'6000': '1 2', '5000': '1', '8000': '4'}
{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('8001', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
Destination: 6000
Origin: eBGP
ASpath: 2
NextHop: 6002
localPreference: 100

{'6000': ('7001', 100), '5000': ('7001', 100), '8000
': ('8001', 100)}
{'6000': '2', '5000': '1', '8000': '4'}
{'6000': ('6002', 100), '5000': ('7001', 100), '8000
': ('8001', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
server at 7004
->5001
In a connected AS, path to which is = 1

->
```

Figure 5: All nodes pick the fewest hop path

Notice how all the nodes in the AS 7, all of them in the lower half of the screen, (here marked as 3) directly jump to the node 5001 because there is a direct link. All others can be shown to have optimal paths too.

7.2 Different Queries and working when we set the local pref of AS6 to 150 in AS7



Alucritty <9>

```

{'6000': ('7004', 100), '7000': ('7004', 100), '5000
': ('7004', 100)}
{'6000': '3 2', '7000': '3', '5000': '3 1'}
{'6000': ('7004', 100), '7000': ('7004', 100), '5000
': ('7004', 100)}
sending ebgp to other eLinks
Destination: 6000
Origin: eBGP
ASpath: 3 1 2
NextHop: 7004
localPreference: 100

{'6000': ('7004', 100), '7000': ('7004', 100), '5000
': ('7004', 100)}
{'6000': '3 2', '7000': '3', '5000': '3 1'}
{'6000': ('7004', 100), '7000': ('7004', 100), '5000
': ('7004', 100)}
sending ebgp to other eLinks
server at 8001
->6005
In a connected AS, path to which is = 3 2

->
```

Alucritty <8>

```
{'7000': '3', '5000': '1', '8000': '3 4'}
{'7000': ('6002', 100), '5000': ('5001', 100), '8000
': ('6002', 100)}
sending to eLinks cause it is ibgp
Destination: 8000
Origin: eBGP
ASpath: 1 3 4
NextHop: 5001
localPreference: 100

{'7000': ('6002', 100), '5000': ('5001', 100), '8000
': ('6002', 100)}
{'7000': '3', '5000': '1', '8000': '3 4'}
{'7000': ('6002', 100), '5000': ('5001', 100), '8000
': ('6002', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
server at 6001
->6002
in the same AS, nexthop = 6002

host is unreachable

->
```

Alucritty <4>

```

{'7000': '3', '5000': '1', '8000': '3 4'}
{'7000': ('7004', 100), '5000': ('6001', 100), '8000
': ('7004', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
Destination: 8000
Origin: iBGP
ASpath: 1 3 4
NextHop: 6001
localPreference: 100

{'7000': ('7004', 100), '5000': ('6001', 100), '8000
': ('7004', 100)}
{'7000': '3', '5000': '1', '8000': '3 4'}
{'7000': ('7004', 100), '5000': ('6001', 100), '8000
': ('7004', 100)}
sending to eLinks cause it is ibgp
server at 6002
->6003
Prefix in same AS but host does not exist
host is unreachable

->
```

Alucritty <6>

```
NextHop: 7001
localPreference: 100

{5000: '1', '7000': '3', '6000': '2', '8000': '3 4'}
{'7000': ('7001', 100), '6000': ('6001', 100), '8000
': ('7001', 100)}
sending ebgp to other eLinks
Destination: 8000
Origin: eBGP
ASpath: 2 3 4
NextHop: 6001
localPreference: 100

{'7000': ('7001', 100), '6000': ('6001', 100), '8000
': ('7001', 100)}
{5000: '1', '7000': '3', '6000': '2', '8000': '3 4'}
{'7000': ('7001', 100), '6000': ('6001', 100), '8000
': ('7001', 100)}
sending ebgp to other eLinks
server at 5001
->8001
In a connected AS, path to which is = 3 4

->
```

Alucritty <2>

```
localPreference: 100

{'6000': ('7004', 150), '5000': ('7004', 150)}
{'6000': '2', '5000': '2 1'}
{'6000': ('7004', 150), '5000': ('7004', 150)}
sending ebgp info to iLinks
sending ebgp to other eLinks
Destination: 8000
Origin: iBGP
ASpath: 4
NextHop: 7004
localPreference: 100

{'6000': '2', '5000': '2 1', '8000': '4'}
{'6000': ('7004', 150), '5000': ('7004', 150), '8000
': ('7004', 100)}
sending to eLinks cause it is ibgp
server at 7001
->7001
itself
->5001
In a connected AS, path to which is = 2 1

->
```

Alucritty <7>

```
localPreference: 100

{'6000': ('7004', 150), '5000': ('7004', 150)}
{'6000': '2', '5000': '2 1'}
{'6000': ('7004', 150), '5000': ('7004', 150)}
Destination: 8000
Origin: iBGP
ASpath: 4
NextHop: 7004
localPreference: 100

{'6000': '2', '5000': '2 1', '8000': '4'}
{'6000': ('7004', 150), '5000': ('7004', 150), '8000
': ('7004', 100)}
server at 7002
->7003
in the same AS, nexthop = 7001

host is unreachable

->5005
In a connected AS, path to which is = 2 1

->
```

Alucritty <3>

```
{'6000': ('7004', 150), '5000': ('7004', 150)}
Destination: 6000
Origin: iBGP
ASpath: 1 2
NextHop: 7001
localPreference: 100

{'6000': ('7004', 150), '5000': ('7004', 150)}
{'6000': '2', '5000': '2 1'}
{'6000': ('7004', 150), '5000': ('7004', 150)}
Destination: 8000
Origin: iBGP
ASpath: 4
NextHop: 7004
localPreference: 100

{'6000': '2', '5000': '2 1', '8000': '4'}
{'6000': ('7004', 150), '5000': ('7004', 150), '8000
': ('7004', 100)}
server at 7003
->5001
In a connected AS, path to which is = 2 1

->
```

Alucritty <5>

```
localPreference: 100

{'6000': ('6002', 150), '5000': ('6002', 150)}
{'6000': '2', '5000': '2 1'}
{'6000': ('6002', 150), '5000': ('6002', 150)}
sending to eLinks cause it is ibgp
Destination: 8000
Origin: eBGP
ASpath: 4
NextHop: 8001
localPreference: 100

{'6000': '2', '5000': '2 1', '8000': '4'}
{'6000': ('6002', 150), '5000': ('6002', 150), '8000
': ('8001', 100)}
sending ebgp info to iLinks
sending ebgp to other eLinks
server at 7004
->7004
itself
->5003
In a connected AS, path to which is = 2 1

->
```

Figure 6: Different queries and changed results when we add a policy

Notice how all nodes in AS7(in the bottom row) now take the longer route through AS6. Several queries have been shown which are explained below.

1. **Querying itself** returns that itself is queried
2. **Querying another node in the AS** Returns the next hop and the fact that it is in the same AS
3. **Querying same prefix as AS but node does not exist** We are simply reported this fact
4. **Querying an unreachable AS** That is the prefix is not in the prefix table. It simply reports

5. **Querying a node in another AS** We simply report its ASpath
6. **Querying an unreachable node in a reachable AS** Remember that our nodes don't have global information about other ASes therefore it simply tells us the route to the AS.

References

[1] Shieldsquare captcha. https://networklessons.com/bgp/bgp-as-path-filter-example. [Online; accessed 2023-04-19].

[2] What is BGP? https://www.cloudflare.com/learning/security/glossary/what-is-bgp/. [Online; accessed 2023-04-19].

[3] Aneek. Local preference - BGP. https://community.cisco.com/t5/routing/local-preference-bgp/td-p/4132585, aug 8 2020. [Online; accessed 2023-04-19].

[4] Contributors to Wikimedia projects. Border gateway protocol. https://en.wikipedia.org/wiki/Border_Gateway_Protocol, apr 18 2023. [Online; accessed 2023-04-19].

[5] Rene Molenaar. Bgp local preference attribute. https://networklessons.com/bgp/how-to-configure-bgp-local-preference-attribute, feb 22 2013. [Online; accessed 2023-04-19].

[6] Noction. Understanding the AS path and AS path prepending. *Noction*, oct 19 2015. [Online; accessed 2023-04-19].

[7] Noction. What you should know about BGP's LOCAL_PREF. *Noction*, oct 7 2015. [Online; accessed 2023-04-19].