BGP Simulator Documentation

Group 8
Niko Hellgren, 505174

Protocol Processing, Spring 2017

Contents

0.1 Introduction

The simulator software is implemented in Java programming language using just the base libraries. For visualization purposes, a graph plotting library, GraphStream [1], was utilized. A high-level overview of the program structure is presented in 0.2. The code is available in `https://github.com/nipehe/BGP-simulator` for the time being.

The simulation has the following basic technical features and limitations:

- The simulation is strongly threaded, with each router and client working as its own thread and `Timer` threads utilized in testing and `KEEPALIVE` messages.
- Layer 1 and Layer 2 functionalities are simulated by Java's `PipedInputStream` and `PipedOutputStream` that are made to transit `byte[]` arrays between threads. The output streams are `synchronized` to make sure the sent packets do not mix up.
- Layer 3 functionality is implemented according to the IPv4 protocol. Although the constructed packets contain and transmit all the fields defined in [2], fields *Type of Service*, *Identification*, *Flags*, *Fragment Offset*, and *Protocol* are filled with default values in all cases, and they are not used for anything. This is partly enabled by the simulation not supporting packet fragmentation.
- A globally available static class provides the simulated routers and clients with DNS-like functionality, since implementing an actual DNS functionality is not essential in this simulation.
- To provide packets for the simulation, clients are attached to the router objects and they have functionality to send data to each other over the network.
- The simulated network uses BGP-4 messages to transmit information between routers.

- Since the simulation is focused on BGP, the idea of Autonomous Systems (AS) is abstracted into each AS being represented by one router, and the clients attached to this AS being connected straight to the router. This removes some features concerning, for example, the `NEXT_HOP` attribute, since the hops done are always to another AS/router.

- TCP functionality is not implemented, and BGP packets are transmitted as ordinary IP packet payloads. This removes the possibility to test or simulate security and stability issues caused by TCP, and simplifies the connection process, but streamlines the implementation due to the complicated specification and functionality of TCP [3].

- Routing information is transmitted using `UPDATE` messages after connection initialization and all routing table changes.

- Additional BGP message type, `TRUST` has been added for trust voting between routers. To avoid Man-in-the-Middle attacks (usually the router being voted on is on the transmission path of the vote), the trust votes are encrypted using 1024-bit RSA and signed using RSA with SHA-1. To avoid making key transmission overly complicated, a PKI-like functionality was made available as a globally available class, providing routers with other routers public keys. The trust implementation is discussed in more detail in Section 0.3.

- Behaviour in error situations has been implemented comprehensively, and both L1/L2 breakage, missing `KEEPALIVE` messages, and erroneous BGP messages cause the routers to drop the link and inform their neighbourhood of this. Possible issues effect the trust rate of the misbehaving router.

0.2 Code overview

This section discusses the code-level decisions and structure in the project. Smaller details are overlooked, and the focus is on main functioning parts of the program. A package-level visualization of the project can be found in Figure 1.

`bgp.core.SimulatorState` Global state to transmit information between parts of the simulation that is either not possible or relevant to transmit via the network. This contains registry of the routers and clients (with checks for duplicate ID's), DNS-like functionality that provides information about client's addresses for data transmission purposes, and PKI-like functionality that provides routers' public keys used in trust voting.

`bgp.core.BGPRouter` Objects from this class represent the routers that make up the network. Each router is given an ID (AS ID) and a specified subnet. Routers have a list of `ASConnection`s that represent the connections to the router's neighbours. Each router also has their own `RoutingEngine` and `TrustEngine`, responsible for routing table upkeep and neighbour trust calculations, respectively. The routers implement a DHCP-like behaviour, where

```
bgp
├── client ................................ Clients of simulator's routers
│   └── messages .................... Messages transmitted between clients
├── core ......................................... Router implementation
│   ├── messages ......................... BGP message implementations
│   │   ├── notificationexceptions. Exception classes used to notify about
│   │   │   situations that should raise a NOTIFICATION
│   │   └── pathattributes ....... Path attributes used in UPDATE messages
│   ├── network .......... Network adapter simulation, L1/L2 functionality
│   │   ├── fsm .... Finite state machine used by the inter-router connections
│   │   └── packet .. Interfaces defining behaviour for actors handling packets
│   ├── routing .................................... Routing table engine
│   └── trust ..................................... Inter-router trust table
├── tests ........... JUnit tests to ensure correct working of the program
├── ui ....................................... User interface components
└── utils...... Functionalities needed by various classes (e.g. IP addresses)
```

Figure 1: Package structure of the code

they provide clients with IP addresses. To avoid maintenance packets blocking the routing, there are two threads in each router: one for making packet routing decisions and one for processing BGP packets designated to current router. Despite this separation, BGP messages are transmitted via the same media as ordinary routed packets. `BGPRouter` also has functionalities that bind the routing and trust engine and various `ASConnection`s together.

0.3 Trust implementation

0.4 General notes from the project

- When Java interprets bytes as integers, they are shifted to range -128..127. Doing bitwise shifts also easily converts the values to a larger data size (e.g. 32-bit integers) instead of dropping the overflowing bits. This caused multiple errors in initial implementations of the bit-level manipulations, and adding bitmasking (e.g. `<value>&0xFF` to force a value to 8 bits) was necessary in most of the places to avoid issues caused by this. Lack of unsigned numbers also made value comparisons difficult at some points, since the interpretations of bytes easily flowed over to the negative numbers.

- Great built-in support for threads, timers and task executors in Java 8 made some otherwise difficult tasks (e.g. `KEEPALIVE` message sending and checking) really easy.

- Following object-oriented paradigm in development was both intuitive and helpful, due to the software being a simulator.

- Since the definition of BGP [4] only specifies the limitations and the requirements of the protocol, information concerning the practices that the BGP speaker should follow was hard to come by.

Bibliography

[1] GraphStream Team, "Graphstream - a dynamic graph library," Mar. 2017.

[2] J. Postel, "Internet Protocol," STD 5, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc791.txt`.

[3] J. Postel, "Transmission Control Protocol," STD 7, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[4] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, RFC Editor, January 2006. `http://www.rfc-editor.org/rfc/rfc4271.txt`.