BGP Simulator

Protocol Processing
Spring 2017

Group 8
Niko Hellgren

Contents

- Technical specification and limitations
- Program architecture
- BGP session initialization
- Routing engine
- Trust implementation

Technical specification

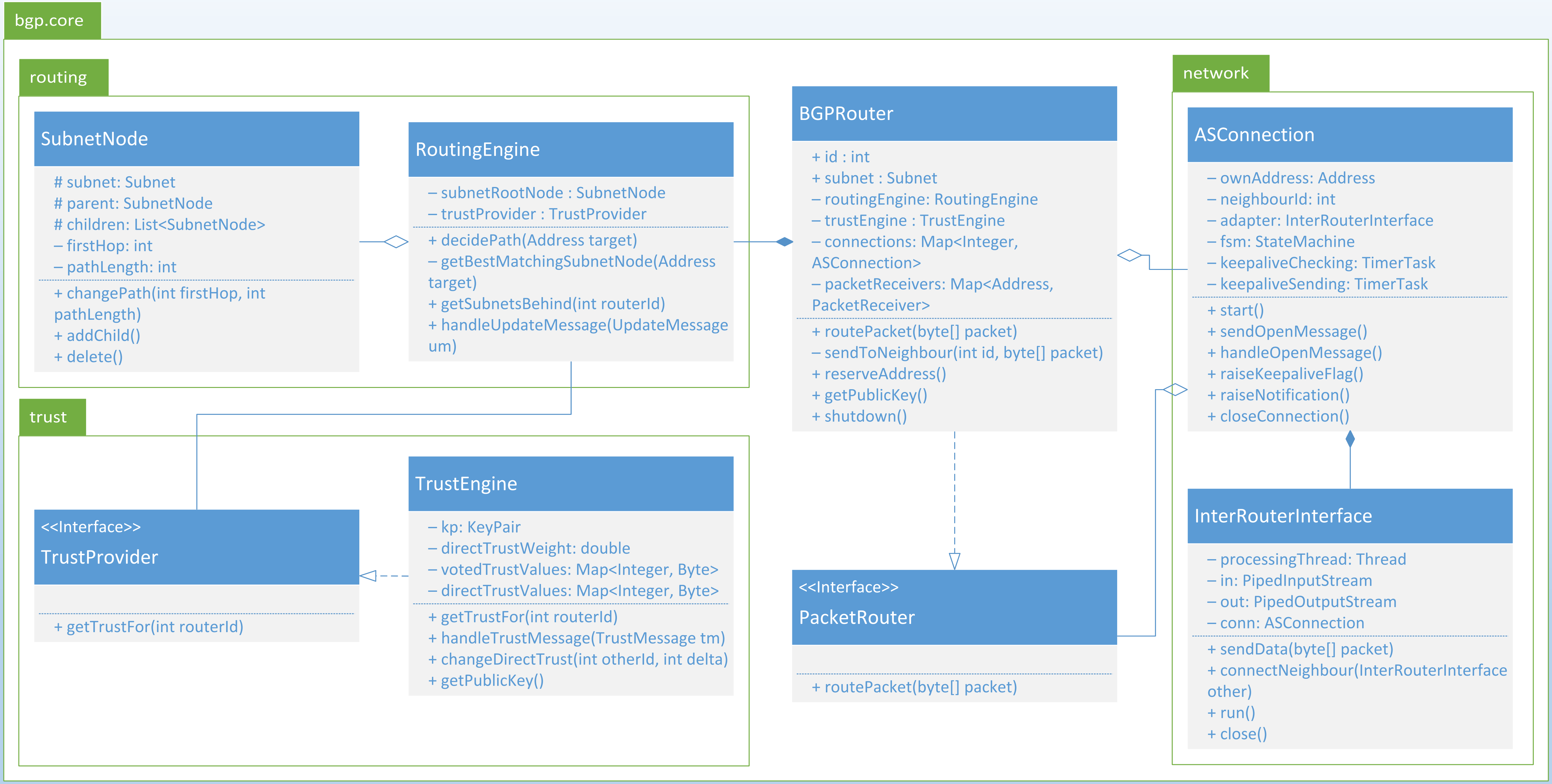
- Implemented in Java (built-in libraries + GraphStream graph plotting library for visualization purposes)
- Strongly threaded
- Inter-router data transmission (L1 & L2) using Java's `Stream` classes
- L3 implementation based on IPv4...
  - ...Although not using all the available fields
  - No packet fragmentation support
  - No different types of service
  - No differentiation between protocols

Technical specification

- BGP router data transmission using BGP-4
- For necessary but not essential features, a globally available state is present
  - DNS-like feature for resolving router addresses
  - PKI-like feature for distributing public keys used in trust voting
  - Static functionality to start the connecting process of two routers
- To simulate traffic in the network, clients with pinging ability attached to the routers

Technical limitations

- Emphasis of the simulation is in BGP
  - The notion of Autonomous System (AS) is absorbed into routers, and each AS consists of only one router and its connected users
- TCP is not implemented, BGP messages are transmitted as IP packet payload
  - Attacks and vulnerabilities based on TCP features are not present



bgp.core

routing

SubnetNode

subnet: Subnet
parent: SubnetNode
children: List<SubnetNode>
– firstHop: int
– pathLength: int

+ changePath(int firstHop, int pathLength)
+ addChild()
+ delete()

RoutingEngine

– subnetRootNode : SubnetNode
– trustProvider : TrustProvider

+ decidePath(Address target)
– getBestMatchingSubnetNode(Address target)
+ getSubnetsBehind(int routerId)
+ handleUpdateMessage(UpdateMessage um)

trust

<<Interface>>
TrustProvider

+ getTrustFor(int routerId)

TrustEngine

– kp: KeyPair
– directTrustWeight: double
– votedTrustValues: Map<Integer, Byte>
– directTrustValues: Map<Integer, Byte>

+ getTrustFor(int routerId)
+ handleTrustMessage(TrustMessage tm)
+ changeDirectTrust(int otherId, int delta)
+ getPublicKey()

BGPRouter

+ id : int
+ subnet : Subnet
– routingEngine: RoutingEngine
– trustEngine : TrustEngine
– connections: Map<Integer, ASConnection>
– packetReceivers: Map<Address, PacketReceiver>

+ routePacket(byte[] packet)
– sendToNeighbour(int id, byte[] packet)
+ reserveAddress()
+ getPublicKey()
+ shutdown()

<<Interface>>
PacketRouter

+ routePacket(byte[] packet)

network

ASConnection

– ownAddress: Address
– neighbourId: int
– adapter: InterRouterInterface
– fsm: StateMachine
– keepaliveChecking: TimerTask
– keepaliveSending: TimerTask

+ start()
+ sendOpenMessage()
+ handleOpenMessage()
+ raiseKeepaliveFlag()
+ raiseNotification()
+ closeConnection()

InterRouterInterface

– processingThread: Thread
– in: PipedInputStream
– out: PipedOutputStream
– conn: ASConnection

+ sendData(byte[] packet)
+ connectNeighbour(InterRouterInterface other)
+ run()
+ close()

Program architecture

- `bgp.simulation.SimulatorState`
  - Global state; contains router and client registry, DNS-like functionality to map addresses to routers, and PKI-functionality for trust voting
- `bgp.core.BGPRouter`
  - Base class for routers
  - Each instance is given a unique ID and a subnet
  - Provides a DHCP-like functionality to connected clients
  - Contains.
    - A list of `ASConnection` objects representing the connections to other routers
    - `RoutingEngine` responsible for routing decisions and routing table
    - `TrustEngine` responsible for calculating and voting on trust

Program architecture

- `bgp.core.ASConnection`
  - Represents a connection to another `BGPRouter`
  - Contains the finite state machine, and timers for checking and sending `KEEPALIVE` messages
  - Handles the connection initialization process
  - Generates new `OPEN`, `KEEPALIVE` and `NOTIFICATION` messages
- `bgp.core.network.InterRouterInterface`
  - Provides the L1/L2 functionalities for `ASConnection`
  - Uses Java's built-in `PipedInputStream` and `PipedOutputStream` classes
  - Transfers data as a byte stream, prepended by the length of the packet
  - Passes received packets to `BGPRouter` for routing

Program architecture

- `bgp.core.routing.RoutingEngine`
  - Handles `UPDATE` messages, decides the routing paths and keeps up the routing table
- `bgp.core.trust.TrustEngine`
  - Calculates trust values for neighbours, constructs and responds to `TRUST` messages
- `bgp.client.BGPClient`
  - Represents the clients attached to the network for testing purposes
  - Connected to a single `BGPRouter`, receives an IP address from it
  - Capable of responding to ping requests

Program architecture

- `bgp.utils.Address`
  - `Address` and its subclass `Subnet` represent the IP addresses and subnets, can be converted to byte or long representation
- `bgp.utils.PacketEngine`
  - Functionalities regarding the construction, modification and validation of IPv4 packets

BGP session initialization

1. Create a new `ASConnection`
2. Connect the `InterRouterInterfaces`' streams
3. Set the FSM to state `CONNECT`
4. Send an `OPEN` message, retry until a `KEEPALIVE` is received or 10 retries have been done
5. Wait (independently of the previous steps) for `OPEN` message. Once received, start sending and checking `KEEPALIVE` messages
6. After `KEEPALIVE` is received, change state to `ESTABLISHED`
7. Send own routing table information to the other party, start using the connection for packet routing

Routing engine – Decision process

```
Validate the received UPDATE message
FOR EACH withdrawn route
    IF exactly matching subnet in routing table is not found
        Do nothing
    ELSE IF the first hop on the UPDATE message matches the first hop of
    current best path
        Remove the routing information for the subnet
        Store the subnet to be sent to other neighbours
    ELSE
        Store the information about own best path to revoked subnet to be sent to the revoking
        peer after processing
```

Routing engine – Decision process

```
FOR EACH NLRI row
    IF no exact subnet match is found from routing table
        Create a new subnet node to routing table with sending router as first hop
    ELSE IF local preference values of existing and new routes’ first hops differ
        Store the preferred route to routing table
    ELSE IF trust-scaled path lengths of existing and new routes’ differ
        Store the shorter route to routing table
    IF selected path was changed
        Store new routing information to be sent to other neighbours
```

Routing engine – Decision process

```
Modify the UPDATE message:
    Add own ID to AS_PATH
    Remove revoked routes that caused no changes to routing table
    Remove NLRI entries that caused no changes to routing table
FOR EACH neighbour not in the AS_PATH
    Set NEXT_HOP to the address of the interface used to communicate with neighbour
    Send the UPDATE message to neighbour
IF withdrawn routes that did not modify routing table exist
    Send UPDATE message containing NLRI for these subnets to the revoking neighbour
```

Trust implementation

- Each router holds two trust values for each neighbour: inherent trust and voted trust
- Stored as values in range -128..127 for easier transmission, scaled to 0..1 when used for path cost calculations
- *Normalized routing criterion with trust rate* calculated similarly to He's solution[1]: $scaled\ cost = \frac{initial\ cost}{trust\ rate}$
- Trust rate is the weighted average of inherent and voted trust
- Voted trust is the average of received votes on the neighbour
- Inherent trust is specified by operator and neighbour behaviour

[1] L. He, "A novel scheme on building a trusted IP routing infrastructure", 2006

Trust implementation – Voting

- Voting is done via specific `TRUST` messages
  - As no BGP session to the voter is established, the messages are not BGP messages per se, but should be considered as ordinary UDP datagrams
- Messages consist of a flag specifying if they are vote requests or responses, voted ID and target ID. If the vote is a response, the message also contains the vote (encrypted using 1024-bit RSA), and a signature. Public keys are provided by the global state.

Trust implementation – Voting

- Voting is triggered by `UPDATE` messages with more than one host in the `AS_PATH`
  - A vote request is sent to the second-order neighbour in the `AS_PATH` list and a token is stored to ensure only one vote per request is accepted
  - The second-order neighbour responds by taking the trust value, padding the single-byte vote with random bits to avoid brute-force attacks, encrypts the vote using requester's public key, signs the encrypted vote with own private key and sends the response back
  - The original requester checks that the token has not yet been used, validates the signature and decrypts the payload
  - Extracted trust value is then calculated to the averaged voted trust