情報ネットワーク学演習 II
第5回レポート課題

所属：大阪大学 大学院情報科学研究科 情報ネットワーク専攻
提出者: 33E16019 満越貴志
電子メールアドレス: t-mangoe@ist.osaka-u.ac.jp

提出年月日：平成 28 年 11 月 7 日

課題内容

ルータのコマンドラインインターフェース（CLI）を作成する。具体的には、以下のコマンドを実装する。

1. ルーティングテーブルの表示
2. ルーティングテーブルエントリの追加と削除
3. ルータのインタフェース一覧の表示

1　ルーティングテーブルの表示

ルーティングテーブルの表示を行うメソッドとして、show_routing_table というメソッドを SimpleRouter クラスに実装した。以下が、そのメソッドのコードである。

```
def show_routing_table(dpid)
  puts "-- Routing table --"
  @routing_table.db.each do |hash|
    hash.each do |prefix,address|
      print IPv4Address.new(prefix)
      puts " / " + address.to_s
    end
  end
  puts "-- end --"
end
```

このメソッドでは、インスタンス変数として保持している@routing_table からエントリを取り出し、宛先の prefix と出力先の address を出力する。ルーティングテーブルのエントリでは、prefix は integer として保存されているため、そのまま出力すると IP アドレスとしては表現されない。そのため、prefix を表示させる際には IPv4Address のオブジェクトを作成し、IP アドレスとして分かりやすい標記で表示されるようにしている。

RoutingTable クラスにおいて、エントリを保存するデータベースはインスタンス変数として宣言されており、外部から直接アクセスすることができない。しかし、ルーティングテーブルの内容を表示させるには、そのデータベースのインスタンス変数にアクセスする必要がある。そのた

め、RoutingTable クラスにおいて、データベースのインスタンス変数を取得するためのゲッターメソッドを、以下のように定義した。

```
  def db
    @db
  end
```

このゲッターメソッドを利用することにより、

```
@routing_table.db.each
```

のように、RoutingTable のエントリを格納しているデータベースのインスタンス変数にアクセスすることができる。

CLI のコマンドの実装は、simple-router-t-mangoe/bin/のディレクトリの中に simple-router というファイルを作成し、そこに記述を行った。ルーティングテーブルを表示させるコマンドは”show_table”という名前で、以下のように実装した。

```
  desc 'Show routing table'
  arg_name 'dpid'
  command :show_table do |c|
    c.desc 'Location to find socket files'
    c.flag [:S, :socket_dir], default_value: Trema::DEFAULT_SOCKET_DIR

    c.action do |_global_options, options, args|
      dpid = args[0].hex
      Trema.trema_process('SimpleRouter', options[:socket_dir]).controller.
        show_routing_table(dpid)
    end
  end
```

2　ルーティングテーブルエントリの追加と削除

2.1　エントリの追加

ルーティングテーブルにエントリを追加するメソッドとして、add_routing_table というメソッドを SimpleRouter クラスに実装した。実装内容は以下のようになっている。

```
  def add_routing_table(dpid,prefix,mask_length,next_address)
    args = {netmask_length: mask_length.to_i,
            destination: prefix,
            next_hop: next_address}
    @routing_table.add(args)
  end
```

エントリを追加する実質的な処理の部分は、RoutingTable クラスに既存の add メソッドを利用した。add_routing_table メソッドでは、引数で渡された情報をハッシュにまとめ、RoutingTable クラスの add メソッドを呼び出す処理を行う。

エントリを追加する CLI のコマンドとして”add_table”を定義し、以下のように実装を行った。

```
  desc 'Add entry to routing table'
  arg_name 'dpid prefix netmask-length next'
  command :add_table do |c|
    c.desc 'Location to find socket files'
    c.flag [:S, :socket_dir], default_value: Trema::DEFAULT_SOCKET_DIR

    c.action do |_global_options, options, args|
      dpid = args[0].hex
```

```
    prefix = args[1].to_s
    mask_length = args[2].to_s
    next_address = args[3].to_s
    Trema.trema_process('SimpleRouter', options[:socket_dir]).controller.
      add_routing_table(dpid,prefix,mask_length,next_address)
  end
end
```

このコマンドではルータの ID に加えて、prefix とネットマスク長、ルーティング先のアドレスの 3 つの情報を引数として指定する。これらの情報は文字列に変換され、add_routing_table メソッドに渡される。

2.2 エントリの削除

ルーティングテーブルからエントリを削除するメソッドとして、SimpleRouter クラスに remove_routing_table メソッドを実装した。実装内容は以下のようになっている。

```
def remove_routing_table(dpid,prefix,mask_length)
  args = {netmask_length: mask_length.to_i,
          destination: prefix}
  @routing_table.remove(args)
end
```

このメソッドでは、prefix とネットマスク長を引数として受け取り、その情報にマッチするエントリをルーティングテーブルから削除する。エントリを実際に削除する処理については RoutingTable クラスにメソッドが存在していなかったため、以下のように remove メソッドを RoutingTable クラスに実装した。

```
def remove(options)
  netmask_length = options.fetch(:netmask_length)
  prefix = IPv4Address.new(options.fetch(:destination)).mask(netmask_length)
  @db[netmask_length].delete(prefix.to_i)
end
```

この remove メソッドを SimpleRouter クラスの remove_routing_table メソッドから呼び出すことで、ルーティングテーブルからエントリが削除されることとなる。

エントリを削除する CLI のコマンドとして ”rm_table ”を定義し、以下のように実装を行った。

```
desc 'Remove entry from routing table'
arg_name 'dpid prefix netmask-length'
command :rm_table do |c|
  c.desc 'Location to find socket files'
  c.flag [:S, :socket_dir], default_value: Trema::DEFAULT_SOCKET_DIR

  c.action do |_global_options, options, args|
    dpid = args[0].hex
    prefix = args[1]
    mask_length = args[2]
    Trema.trema_process('SimpleRouter', options[:socket_dir]).controller.
      remove_routing_table(dpid,prefix,mask_length)
  end
end
```

このコマンドでは引数として、ルータの ID と prefix、ネットマスク長を受け取る。

3　インターフェース一覧の表示

ルータのインターフェース一覧を表示させるメソッドとして、SimpleRouter クラスに show_interface メソッドを実装した。実装内容は以下のようになっている。

```
def show_interface(dpid)
  puts "--Router's interface--"
  Interface.each do |interface|
    print "port_number : "
    puts interface.port_number
    print "mac_address : "
    puts interface.mac_address
    print "ip_address : "
    puts interface.ip_address.value
    print "netmask_length : "
    puts interface.netmask_length
    puts ""
  end
  puts "--end--"
end
```

この show_interface メソッドでは、ルータの各インターフェースに対して情報を表示させている。表示する内容の説明を print 文で表示し、情報の内容を puts 文で表示している。こうすることで、情報の説明と内容が一文で表示される。各インターフェースについて、情報を全て出力した後、空行を出力するようにしている。これは、それぞれのインターフェースの情報群を空行で区切り、情報を分かりやすくするためである。

インターフェースの一覧を表示する CLI のコマンドとして ”show_interface ”を定義し、以下のように実装を行った。

```
desc 'Show router interface'
arg_name 'dpid'
command :show_interface do |c|
  c.desc 'Location to find socket files'
  c.flag [:S, :socket_dir], default_value: Trema::DEFAULT_SOCKET_DIR

  c.action do |_global_options, options, args|
    dpid = args[0].hex
    Trema.trema_process('SimpleRouter', options[:socket_dir]).controller.
      show_interface(dpid)
  end
end
```