

Floodlight RESTful interface

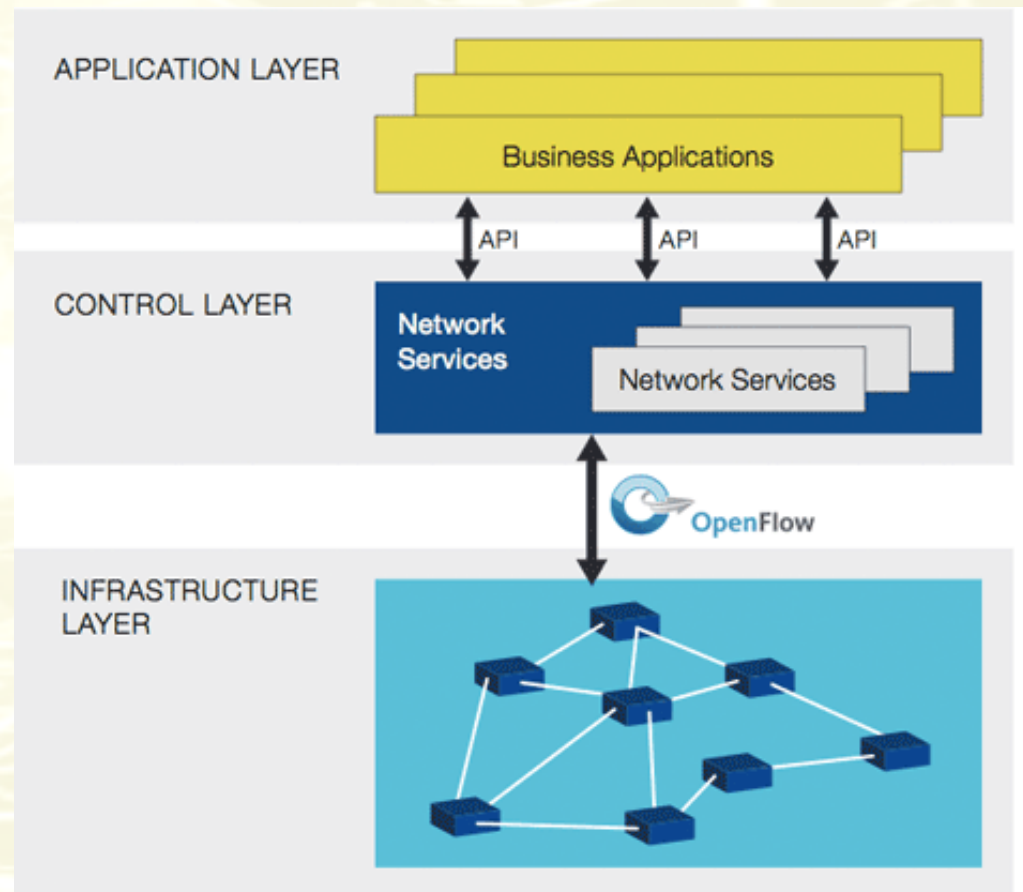
Carlo Vallati

Assistant Professor@ University of Pisa

c.vallati@iet.unipi.it

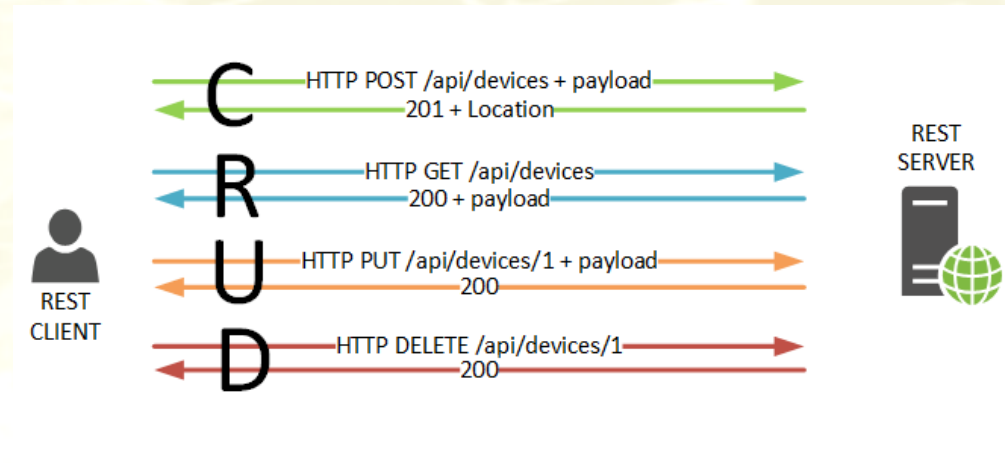
SDN Northbound interface

- Core advantage of Software Defined Networks over traditional deployments is their re-programmability
- Such changes are usually applied automatically by external applications and monitoring process to react to events (e.g. variations in the load)
- To this aim, a northbound interface is exposed by controllers



RESTful APIs

- Controller defines the APIs of the northbound interface using the RESTful paradigm
- Each controller exposes a set of resources that can be used by applications to retrieve network information or issue configuration changes or trigger operations



| Resource | POST create | GET read | PUT update | DELETE delete |
|------------|------------------|-------------|-------------------------------------|------------------|
| /dogs | create a new dog | list dogs | bulk update dogs | delete all dogs |
| /dogs/1234 | error | show Bo | if exists update Bo if not error | delete Bo |

Floodlight RESTful API – STEP 1

- Each Floodlight module can expose a RESTful interface for external applications. REST interface for the *loadbalancer*.
- First step, add the `IRestApiService` among the dependences and retrieve a reference to it

```
protected IRestApiService restApiService; // Reference to the Rest API service

...
@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l = new ArrayList<Class<?
        extends IFloodlightService>>();

    ...
    // Add among the dependences the RestApi service
    l.add(IRestApiService.class);
    return l;
}

...
@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    ...
    // Retrieve a pointer to the rest api service
    restApiService = context.getServiceImpl(IRestApiService.class);
}
```

Floodlight RESTful API – STEP 2

- Second step, create a class representing the RESTful interface

```
public class LoadBalancerWebRoutable implements RestletRoutable {  
    /**  
     * Create the Restlet router and bind to the proper resources.  
     */  
    @Override  
    public Restlet getRestlet(Context context) {  
        Router router = new Router(context);  
  
        // Add some pre-defined REST resources available in the floodlight framework  
  
        // This resource will show the some summary stats on the controller  
        router.attach("/controller/summary/json", ControllerSummaryResource.class);  
  
        // This resource will show the list of modules loaded in the controller  
        router.attach("/module/loaded/json", LoadedModuleLoaderResource.class);  
  
        // This resource will show the list of switches connected to the controller  
        router.attach("/controller/switches/json", ControllerSwitchesResource.class);  
  
        return router;  
    }  
    @Override  
    public String basePath() {  
        // The root path for the resources  
        return "/lb";  
    }  
}
```


Floodlight RESTful API – STEP 3

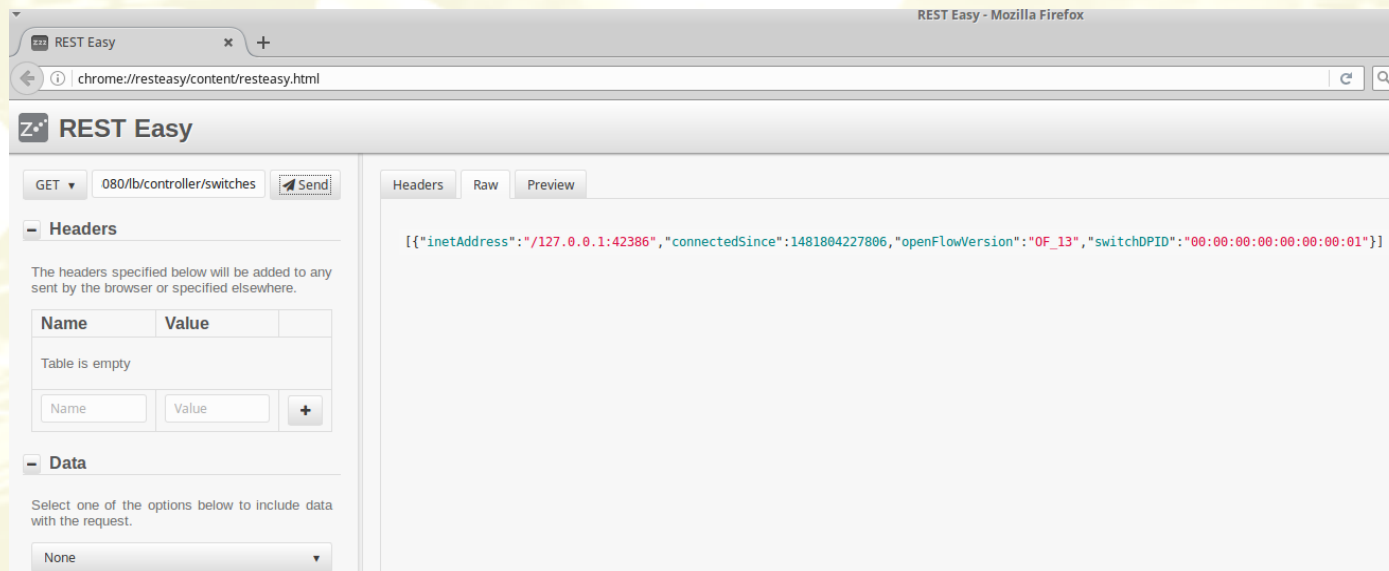
- Third step, add the REST interface at startup

```
@Override
public void startUp(FloodlightModuleContext context) throws FloodlightModuleException
{
    ...

    // Add as REST interface
    restApiService.addRestletRoutable(new LoadBalancerWebRoutable());
}
```

Floodlight RESTful API – TEST

- The module is now ready to expose some pre-defined floodlight resources, e.g. to show general statistics on the SDN network
- The REST interface can be tested through one of the REST add-ons for browsers, for example “REST Easy” for Firefox
- The REST interface is exposed by floodlight on the port
- Test the resource <http://127.0.0.1:8080/lb/controller/switches/json> to get a list of the switches connected



Floodlight RESTful API – Custom res

- In order to add custom resources to your module additional class must be defined
- New resources can be defined only through external classes as this one for example that simply returns a custom JSON message:

```
/**
 * Test resource
 */

public class TestResource extends ServerResource {

    @Get("json")
    public Map<String, Object> Test() {

        Map<String, Object> info = new HashMap<String, Object>();

        info.put("name", "value");

        return info;

    }
}
```


Floodlight RESTful API – Custom res

- As external class, such custom resources can not access to internal module information, neither trigger internal operations
- In order to export functionalities and data to other classes the module must define and implement a public interface available to other module as follow:

```
// Service interface for the module  
// This interface will be use to interact with other modules  
// Export here all the methods of the class that are likely used by other modules  
  
public interface ILoadBalancerREST extends IFloodlightService {  
    // Method exposed by the module  
    // Method to retrieve the info on the real servers of the pool  
    public Map<String, Object> getServersInfo();  
    // Method to set the hardtimeout for load balancing  
    public void setHardTimeout(int newValue);  
}
```

Floodlight RESTful API – Custom res

- The module must include the interface through the following steps:
 1. Add the implementation of the interface:

```
public class LoadBalancerREST implements  
    IOFMessageListener, IFloodlightModule, ILoadBalancerREST {
```

```
@Override  
public Collection<Class<? extends IFloodlightService>> getModuleServices() {  
    Collection<Class<? extends IFloodlightService>> l =  
        new ArrayList<Class<? extends IFloodlightService>>();  
    l.add(ILoadBalancerREST.class);  
    return l;  
}  
  
@Override  
public Map<Class<? extends IFloodlightService>, IFloodlightService> getServiceImpls()  
{  
    Map<Class<? extends IFloodlightService>, IFloodlightService> m =  
        new HashMap<Class<? extends IFloodlightService>, IFloodlightService>();  
    m.put(ILoadBalancerREST.class, this);  
    return m;  
}
```

Floodlight RESTful API – Custom res

- The module must implement the methods of the interface:

```
@Override
public Map<String, Object> getServersInfo(){
    Map<String, Object> info = new HashMap<String, Object>();
    for( int i = 0; i < SERVERS_MAC.length; i++){
        info.put( SERVERS_IP[i] , SERVERS_MAC[i]);
    }
    return info;
}

@Override
public void setHardTimeout(int newValue){
    HARD_TIMEOUT = (short) newValue;
    System.out.println("Timeout value changed to " + newValue + "\n");
}
```

Floodlight RESTful API – Custom res

- For each REST resource its implementation can invoke the methods of the public interface, e.g. a resource to retrieve the list of the servers in the pool

```
// Implementation of a resource to retrieve the list of the servers in the pool
public class GetServerInfo extends ServerResource {
    @Get("json")
    public Map<String, Object> Test() {
        ILoadBalancerREST lb =
            (ILoadBalancerREST)getContext().getAttributes()
                .get(ILoadBalancerREST.class.getCanonicalName());
        return lb.getServersInfo();
    }
}
```

Floodlight RESTful API – Custom res

- Resource to set the value for load balancing

```
public class ChangePeriod extends ServerResource {
    @Post
    public String store(String fmJson) {

        // Check if the payload is provided
        if(fmJson == null){
            return new String("No attributes");
        }

        // Parse the JSON input
        ObjectMapper mapper = new ObjectMapper();
        try {
            JsonNode root = mapper.readTree(fmJson);
            // Get the field hardtimeout
            int newValue = Integer.parseInt(root.get("hardtimeout").asText());
            ILoadBalancerREST lb = (ILoadBalancerREST)
                getContext().getAttributes()
                    .get(ILoadBalancerREST.class.getCanonicalName());
            lb.setHardTimeout(newValue);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return new String("OK");
    }
}
```




Floodlight RESTful API – TEST

- Test the resource <http://127.0.0.1:8080/lb/timeout/json>
- The argument must be specified in JSON format in the POST payload as follow:

REST Easy

POST ▼

<http://127.0.0.1:8080/lb/timeout/json>

Send

+ Headers

- Data

Select one of the options below to include data with the request.

Custom ▼

Enter the data and its corresponding MIME type below.

application/json

```
{"hardtimeout": "30"}
```

+ Authentication

Headers Raw Preview

200 OK

| Name | Value |
|----------------------------------|-------------------------------------|
| Content-Length | 2 |
| Content-Type | application/json; charset=UTF-8 |
| Date | Sat, 17 Dec 2016 11:36:03 GMT |
| Accept-Ranges | bytes |
| Server | Restlet-Framework/2.3.1 |
| Vary | Accept-Charset, Accept-Encoding, Ac |
| Access-Control-Expose-Headers | Authorization, Link |
| Access-Control-Allow-Credentials | true |
| Access-Control-Allow-Methods | GET,POST,PUT,DELETE |
| Access-Control-Allow-Origin | * |
| Access-Control-Allow-Headers | * |
| Connection | keep-alive |

References

- *Floodlight REST tutorial:*
<https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+add+a+REST+API+to+a+Module>