Modeling and Simulation of Routing Protocols in the Cloud

Dejan Spasov, Igor Jakimovski

Faculty of Computer Science and Engineering
Skopje, Macedonia

Abstract. Cloud computing is emerging computer paradigm where computing, storage and networking utilities are offered mainly to the business community. In this paper we use cloud computing in developing and simulating routing protocols. The benefit of simulating in the cloud is twofold: it provides hardware independence for the underlying test environment, in addition to better methods for monitoring the performance of the protocol.

Keywords: Cloud Computing, Network Simulation, Routing Protocols

1 Introduction

Cloud Computing is emerging method of computing where one can rent various services [1]. In other words it allows use of various services without the need of installing them on a local machine; the minimum requirement is only a computer connected on Internet and a web browser. Cloud computing services provided by Cloud Providers fall in one of the three categories: Interface as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud installations are classifiedin four categories: *Private* (provisioned for use by a single organization), *Community* (provisioned for exclusive use by a specific community of consumers), *Public* (provisioned for open use by general public) and *Hybrid* (composition of two or more distinct cloud infrastructures) clouds.

Routers are central devices in the contemporary networking world. Great demand is placed on them for providing shortest-path interconnections among various networks. In the majority of today's networks it is impossible for routers to find all the shortest paths without the help of the routing protocols. Though the method of finding the shortest-path has been settled more than 60 years ago, and the choice of the appropriate metric seems well elaborated, the convergence is still troublesome for the network administrators. One way to deal with the issue of slow convergence is to segment the network into smaller domains; but this approach shifts the administrative burden toward the humans.

Thus it is important for us to keep designing new and upgrading the existing routing protocols so that they can scale well with modern networking demands [2]. Cloud computing paradigm can offer several benefits in pursuing these efforts. The

A. Madevska Bogdanova, D. Gjorgjevikj (Editors): ICT Innovations 2014, Web Proceedings, ISSN 1857-7288
© ICT ACT –http://ictinnovations.org/2014, 2014

most obvious benefit is by providing scalable test environment that does not depend on the underlying hardware. In addition, synchronization and monitoring of the test nodes and network packets can be handled with plethora of existing tools.

In this paper we present a framework on how to build test network of routing nodes in a cloud environment. We use a private OpenNebula-base cloud [3] as IaaS that is administered by the Faculty of Computer Science and Engineering, on which we have Quagga routing nodes. We will elaborate our framework using a simple modification of the Routing Information Protocol.

2 Simulating Routing Protocols

Our network consists of three Debian virtual machines interconnected with three virtual networks (fig. 1) and one public access point for each machine. Virtual machines act as virtual routers (VRs) with the installation of the Quagga routing software [4].

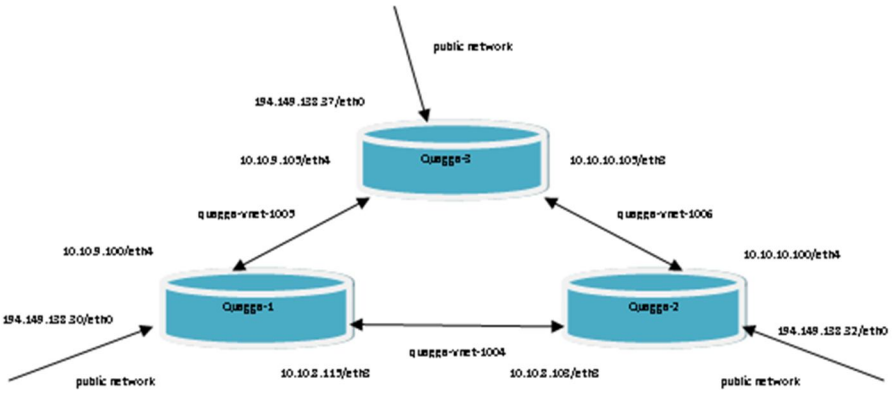

Figure 1. The topology of the test network

Quagga is open source routing software that implements most of today's routing protocols (RIP, OSPF and BGP). Routing protocols are configured via CLI known as VTY. Particular useful feature of Quagga command is their similarity with the standard commands found on Cisco's routers. In addition, the ability to simulate loopback interfaces can also be very useful in our simulations.

After initial configurations we verify network connectivity with successful pings among adjacent routers. Next we activate the RIP daemons on all VRs

```
/opt/quagga/sbin/zebra -d
/opt/quagga/sbin/ripd -d
```


© ICT ACT –http://ictinnovations.org/2014, 2014

and connect to all CLI in order to configure the Routing Information Protocol. We verify network connectivity of distant networks from the routing tables or with distant or extended pings.

Routing Information Protocol (RIP) is one of the oldest and still alive routing protocols. Its development began in the late '70s from the Xerox's XNS protocol. The first document that describes RIP was published in 1988 [5], however recent RFC extensions that were proposed to support IPv6 [6] and cryptographic authentications [7] secured its future existence.

RIP metric is an integer between 1 and 15, with 16 being reserved for infinity. The way the costs for traversing networks are associated is not specified in the standard, but due to the limit of 15, the cost is usually 1. This is the well-known *hop-count* metric used by RIP.

RIP packets are encapsulated in UDP segments before being sent over IP network. RIP configured routers send and receive RIP packets on port 520. RIP packet format is given on figure 2 [8]:

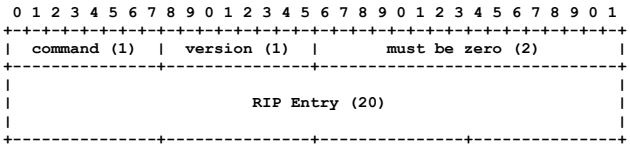

Figure 2. RIP packet format.

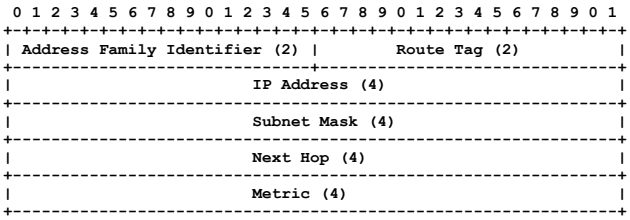

Figure 3. RIP entry.

We can notice that RIP packets are aligned on the 32 bit boundaries. Version field (fig. 2) helps to distinguish between RIP version 1 and RIP version 2 packets. The command field defines two types of messages:

1. *Request* – from a neighbor router to send all or part of the routing table
2. *Response* – from the neighbor router with all or part of the routing table.

Each RIP packet (fig. 2) can carry information for up to 25 routes. Parameters requested or sent back for one route are carried with one RIP entry (fig.3). In order to be able to monitor in real-time packet flow through the network, we have to install Wireshark. Then we have to install Xming server and use X window system (fig. 4).

© ICT ACT –http://ictinnovations.org/2014, 2014

3 Developing Routing Protocols

Next, we will demonstrate the procedure for simulating new routing protocols. We want to modify Quagga' s RIP implementation so that routing packets can carry more than one metric for a route (fig. 5). Additional fields in the routing packets can be used for various reasons, for example, to compare different metrics or to put time stamps in the packets. RIP uses hop-count to decide the best route, though better metrics exist. On the other hand hop-count cannot be removed from the RTE updates because mechanisms for routing loop detection (like split horizon) depend on it.

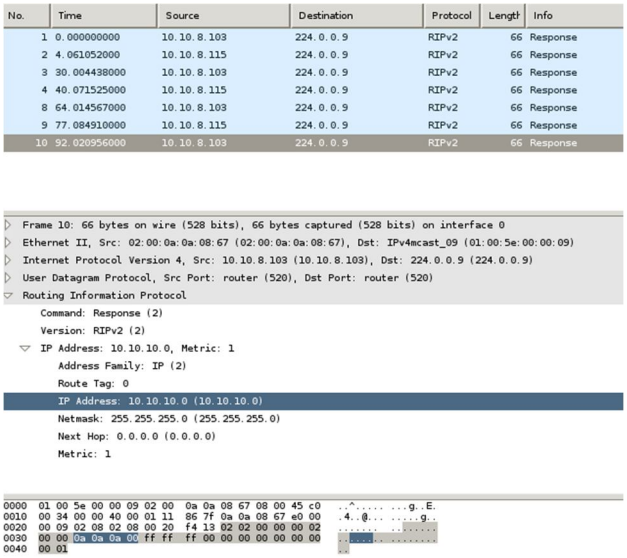

Figure 4. RIP entry captured by Wireshark.

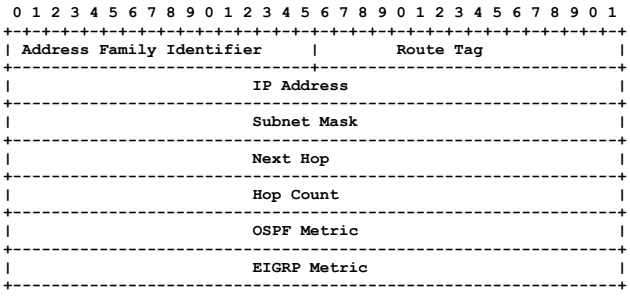

Figure 5. Modified RIP entry.

After untiring the Quagga routing software, we implement changes in the `ripd.c` and `ripd.h` files.

© ICT ACT –http://ictinnovations.org/2014, 2014

In `ripd.h` we have to extend the size of the RIP entries (known as RTE) to 28 bytes

```
#define RIP_RTE_SIZE       20
```

and to limit the number of RTE in a RIP packet to 19

```
#define RIP_MAX_RTE        25
```

so that the maximum RIP packet size remains smaller than 512 bytes. Next we extend the structure holds RTEs `struct rte` and the structure that holds routing table entries `struct rip_info`

```
/* RIP routing table entry which belong to rip_packet. */
struct rte
  struct in_addr prefix;  /* Prefix of rip route. */
  struct in_addr mask;    /* Netmask of rip route. */
  struct in_addr nexthop; /* Next hop of rip route. */
  u_int32_t metric;       /* Metric value of rip route. */
  u_int32_t OSPFmetric;
  u_int32_t EIGRPmetric;

/* RIP route information. */
struct rip_info
  /* RIP nexthop. */
  struct in_addr nexthop;
  struct in_addr from;

  /* Which interface does this route come from. */
  unsigned int ifindex;

  u_int32_t metric;
  u_int32_t OSPFmetric;
  u_int32_t EIGRPmetric;
```

Next we insert test values in the new fields

```
  rinfo-> OSPFmetric=0xffffffff;
  rinfo-> EIGRPmetric=0x0f0f0f0f;

  /* Write RTE to the stream. */
  num = rip_write_rte (num, s, p, version, rinfo);
```

and write modified routing table entries to the output stream

```
static int rip_write_rte (…)
  if (version == RIPv1)
      ... ... ... ... ... ... ...
      stream_putl (s, rinfo->metric_out);
      stream_putl (s, rinfo-> OSPFmetric);
```


© ICT ACT –http://ictinnovations.org/2014, 2014

```
        stream_putl (s, rinfo->metric_ EIGRPmetric);
    else
        ... ... ... ... ... ... ...
        stream_putl (s, rinfo->metric_out);
        stream_putl (s, rinfo-> OSPFmetric);
        stream_putl (s, rinfo->metric_ EIGRPmetric);
```

On the other hand, the code that receives packets does not need to be modified, except few changes of constants. Structure `struct rip_packet` contain as substructure the structure `struct rte`, thus we can access new fields from the instances of `rip_packet` that receive incoming packets.

4 Monitoring the Changes with Wireshark

One of the most important aspects in designing new protocol is real-time analysis of the protocol. For known protocols, this can be done with packet analyzers, but the problem is that packet analyzers cannot recognize new protocol, thus they will display only raw data in hexadecimal format. Figure 1 shows how one of the most famous packet sniffers – Wireshark [9] reacts to the modifications of the RIP protocol. It our case, Wireshark recognizes the changes as malformed RIP packets. We have chosen Wireshark in our framework for real-time packet analysis due to its open source.

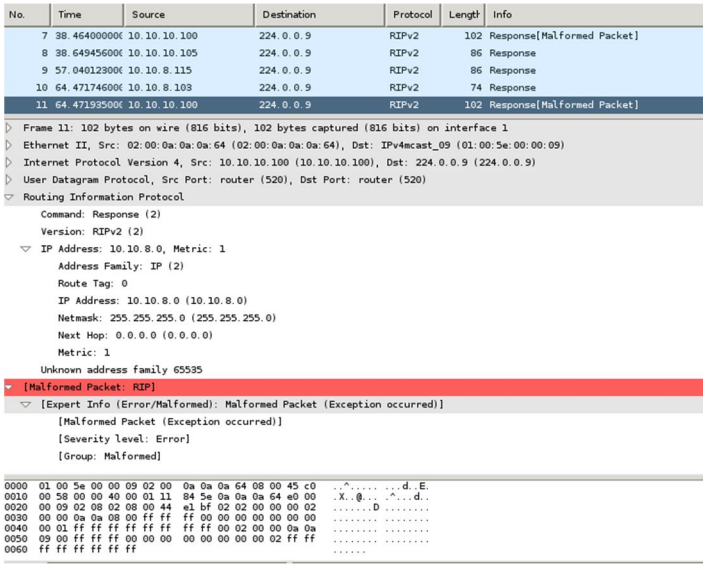

Figure 6. Response from Wireshark on the modified RIP protocol

© ICT ACT –http://ictinnovations.org/2014, 2014

After untiring the Wireshark 1.8.6 tar ball, we have to open the file packet-rip.c located in the folder `wireshark-1.8.6/epan/dissectors`. To accommodate the new packets, we have to increase the packet length from 20 bytes to 28 with

```
#define RIP_ENTRY_LENGTH 28
```

Next, along with the old `hf_rip_metric` dissector that handles route's metric, we have to define two more global dissectors that handle the two new fields in the route's update

```
static int hf_rip_metric1 = -1;
static int hf_rip_metric2 = -1;
```

In order to improve the display of dissecting information, for example, in the detailed information section in the tree view of the packet information, we have to modify the `hf_register_info hf[]` structure in `proto_register_rip`. In particular, we have to register additional information associated with the new OSPF and EIGRP fields with the following code:

```
{&hf_rip_metric1,
    {"OSPF Metric", "rip.metric1", FT_UINT32, BASE_DEC,
     NULL, 0, "OSPF metric for this route", HFILL}},

{&hf_rip_metric2,
    {"EIGRP Metric", "rip.metric2", FT_UINT32, BASE_DEC,
     NULL, 0, "EIGRP metric for this route", HFILL}},
```

This code should be positioned after the `&hf_rip_metric handler`. The function that dissects regular RIP packets is `dissect_ip_rip_vektor(…)`. Packet data is kept in a buffer known as TVB (Testy Virtual Buffer). The first step is to extract the new data fields from the TVB along with the old metric:

```
guint32 metric, OSPFmetric, EIGRPmetric;

metric = tvb_get_ntohl(tvb, offset+16);
OSPFmetric = tvb_get_ntohl(tvb, offset+20);
EIGRPmetric = tvb_get_ntohl(tvb, offset+24);
```

The following function

```
ti = proto_tree_add_text(tree, tvb, offset,
  RIP_ENTRY_LENGTH, "IP Address: %s, Metric: %u",
   ip_to_str(tvb_get_ptr(tvb, offset+4, 4)), metric);
```

adds labels to the compressed information field (red rectangle on fig. 4), and it should be updated appropriately with the information that we want to display.

© ICT ACT –http://ictinnovations.org/2014, 2014

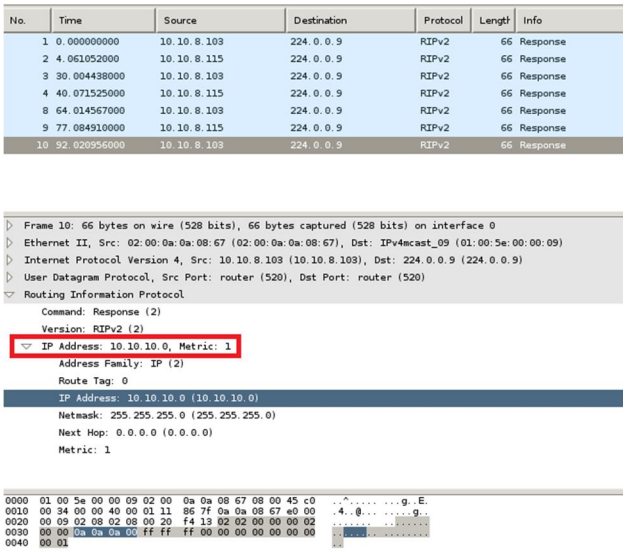

Figure 7. The Compressed Information Field

Finally, in order to display the new fields in the detailed information part of the Wireshark GUI, we have to type in the following updates:

```
proto_tree_add_uint(rip_vektor_tree, hf_rip_metric, tvb,
                        offset+16, 4, metric);

proto_tree_add_uint(rip_vektor_tree, hf_rip_metric1, tvb,
                        offset+20, 4, OSPFmetric);

proto_tree_add_uint(rip_vektor_tree, hf_rip_metric2, tvb,
                        offset+24, 4, EIGRPmetric);
```

After recompiling Wireshark with `make` and `make install` and allowing X11 forwarding, we obtain Wireshark analyzer that properly displays new protocol fields (fig. 5).

© ICT ACT –http://ictinnovations.org/2014, 2014

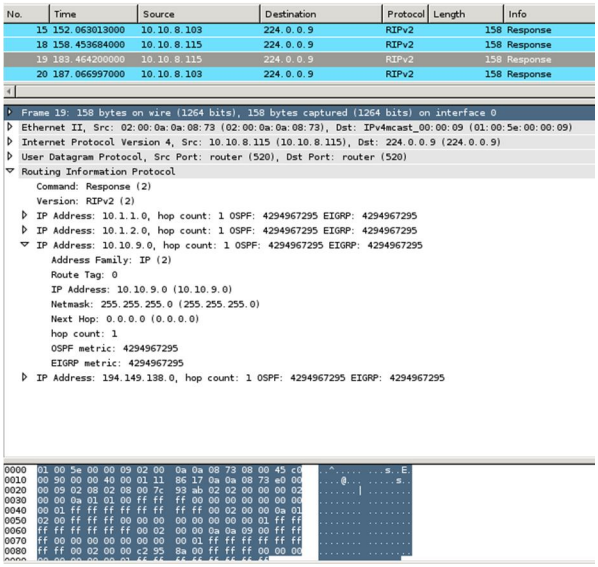

Figure 8. Proper display of the new protocol fields

5 Conclusion and Future Goals

We have demonstrated a procedure for testing new routing protocols. At early stages of development, testing in the cloud offers the benefit of hardware independence thus saving time with hardware related issues.

However, we believe that cloud environment not only offers time saving, but also additional features not easily available in real environments. A routing protocol is characterized with its speed of convergence, percentage of redundant traffic through the network and CPU usage on the routers. The speed of convergence can be measured with the NTP protocol and the redundant traffic can be measured with Wireshark. In particular, the most interesting feature of cloud computing is that CPU usage over the entire network can easily be monitored with the Sunstone administrator's interface (fig. 9); thus gaining deeper insight into routing protocol's behavior.

In addition to obvious advantages in designing new network protocols, simulating computer networks with cloud computing has advantages in education. For example, lab works in computer network classes can be based on simulations in the cloud. Thus we can easily dissect protocols and students will be more involved in interacting with network protocols. On the other hand, cloud environment gives the advantage for online coursework and removes the constraint that students will have to in the same physical location with the network equipment.

© ICT ACT –http://ictinnovations.org/2014, 2014

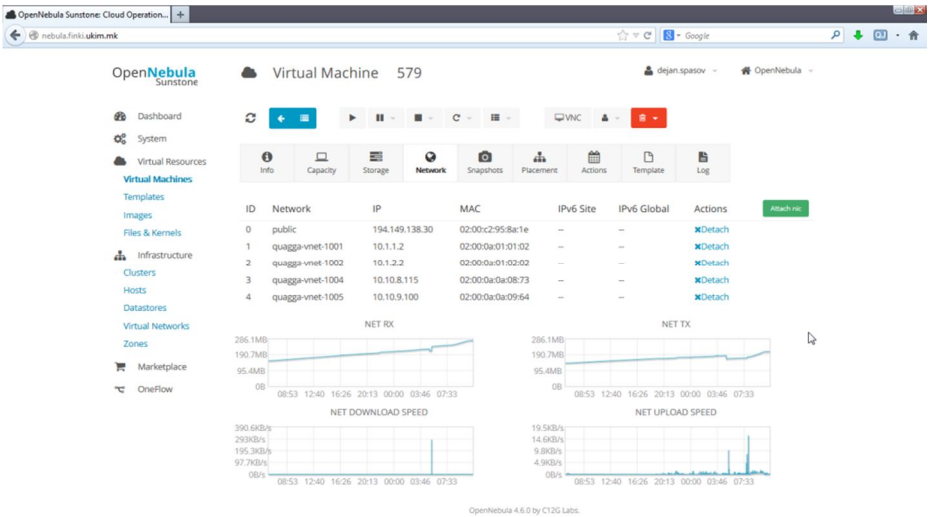

Figure 9. Sunstone administrators interface

6 Acknowledgement

This work was partially financed by the XaaSinCloud project at Faculty of Computer Science and Engineering at the Ss. Cyril and Methodius University, Skopje, Macedonia.

References

1. https://community.emc.com/community/support/blog/2013/07/15/5-cloud-computing-trends-for-2013
2. Cisco Networking Academy: CCNP Advanced Routing Protocols.
3. www.opennebula.org
4. http://www.nongnu.org/quagga
5. Hedrick C.: Routing Information Protocol. RFC 1088, IETF standard, June 1988.
6. Malkin, G., Minnear, R.: RIPng for IPv6. RFC 2080, IETF standard, January, 1997.
7. Atkinson, R., Fanto, M.: RIPv2 Cryptographic Authentication. RFC 4822, IETF standard, February, 2007.
8. Malkin G.: RIP version 2. RFC 2453, IETF standard, November 1998.
9. Wireshark Developer's Guide. wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html

© ICT ACT –http://ictinnovations.org/2014, 2014