UNIVERSITY OF AMSTERDAM

Visualizing and configuring network layers across on demand networks

Iwan Hoogendoorn - `Iwan.Hoogendoorn@os3.nl`
Fahimeh Alizadeh - `Fahimeh.Alizadeh@os3.nl`

April 2, 2013

Abstract

This paper presents an on-going research to develop the Inter-Cloud Architecture[1], and focusses on the visualization problems of different network-on-demand layers in multi-provider multi-domain heterogeneous cloud based networks. Cloud computing allows enterprise customers to make use of computing resources (hardware and software) that are delivered across the internet. This paper is an extension to a research that is currently performed by members of the SNE research group. In a nutshell the research performed by the SNE research group is about creating, managing and deleting (virtual) hosts (nodes) and offering networks as an application across different cloud providers. In extend to that this paper focuses on visualizing these different layers of networks (that are offered as an application) with the use of various webdevelopment tools and languages. The proposed visualization (3D Carousel visualization) method intends to provide a structured way of displaying each different network type, or protocol that is installed as an application between two or more virtual nodes spread across different cloud providers.

1 Introduction

One of the most important business goals for cloud providers is to deliver remote services through the local network or the Internet. The idea is to keep the clients computing as thin as possible.

One of the first cloud computing related milestones was the introduction of Saleforce.com in the year 1999. Salesforce.com was one of the first companies that offered the concept of delivering an enterprise class application through a simple website.[1]

Amazon Web Services (AWS) founded in 2002, provided an infrastructure that made it possible for companies to host their applications in the cloud. Amazon provided a suite of different types of cloud services like storage and computing power[2]. AWS is currently one of the largest commercial cloud providers with considerable amount of market share that offers services to make it possible for individuals or businesses to host their applications/services in the cloud. Besides AWS, there are various other companies offering the same kind of services.

Now that it is possible to move applications to one cloud provider there is also a possibility to use multiple cloud providers. When companies decide to distribute their application across multiple cloud providers the need may be there to make communication between these virtual nodes possible. Different cloud instances provide different interfaces. By having the number of different cloud providers growing and growing this will result in a higher number of incompatible interfaces for different cloud instances. With this happening it is necessary to make inter-cloud connections more feasible.

2 Problem statement and related research

Nowadays more and more companies move their applications and services to such a cloud provider that they now host on their local infrastructure. The reason for this could be for cost saving purposes, providing redundancy or maybe performance improvement.

Moving different aspects of the company (infrastructure, service, and platform) can be done in various ways for whatever reason. One could choose to move their applications and/or data partially or fully to one single provider. If the use of one single cloud provider is a problem for the reasons mentioned earlier, there is also a possibility to use two or more cloud providers. Based on cloud provider services and company policies, different choices can be made and each choice has its pros and cons. Members of *The SNE Research group* (Marc X. Makkes, Yuri Demchenko, Cees de Laat and Robert Meijer) of the University of Amsterdam introduced a new method on "Defining Inter-Cloud Architecture for Interoperability and Integration" [2].

In their research they created a controller based solution where one can cre-

[1]http://www.computerweekly.com/feature/A-history-of-cloud-computing
[2]http://www.computerweekly.com/feature/A-history-of-cloud-computing

ate, delete and connect virtual cloud instances across multiple cloud providers with the use of one single user interface[3]. The controller called "Sarastro"[4], basically receives and sends generic commands that supports a way of unified communication with all (different) participating cloud providers.
Another part of this ongoing research, are so called "NetApps" or "On Demand Networking"[5].

A good example of "On Demand Networking" is that we have two or multiple nodes that reside in two or multiple cloud providers. These nodes are connected with each other with the use of Layer 2 VPN technology. With "On Demand Networking" one can offer a specific network type (or network layer) as a specific software package (or service)[5]. Ultimately it is possible to run a separate IPv4 network on top or the current Layer 2 VPN, run IPv6 as a service or maybe MPLS over IPv4, or even offer routing as a service (on demand).

On the virtual nodes, Sarasto instantiates one or more internet routers and connects them with tunneling protocols via the Internet. The overlay network that is then created can be actively managed by Sarasto, for instance by changing its topology and connecting nodes or adding more overlaid networks. The overlay networks can have in many properties a strong resemblance to the Internet and this justifies to name them virtual internets. Sarastro is able to support control programs that continuously (dynamically) adapt virtual internets to maintain application specific properties. *The SNE Research group* presents two virtual internets that they generated. The first one (virtual internet) can be for example wrapped around the world and illustrates the use of their software and typical properties of virtual internets. The second virtual internet can demonstrates the concept of a control programs by showing how every network element can maintain at least two connections. Major use cases of virtual internets, are best effort routing over the Internet, private internets for closed user groups and run-time environments[4].

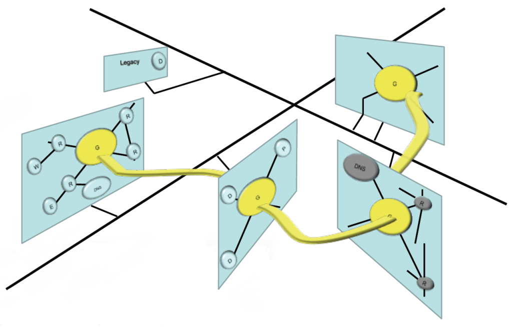

Figure 1: Multiple virtual internets connected with each other. Each virtual internet provides a collaboration platform for complex applications.[4]

With offering networking as a service, another layer of complexity is added. With having multiple layers of networking, there is also more to troubleshoot in case of failure. These current researches done by *The SNE Research group* only focuses on interconnecting these nodes between different cloud providers and offering networking on demand between these hosts.

2.1 Research question

With this research we try to answer the following question and try to build up on the research done by the *The SNE Research group*:

Can one provide network layering services in a visual organized way in order to display and debug (or troubleshoot) on a per-network-type layer method, and to what extend is this possible?

2.2 Main contributions of this paper

As mentioned before, this paper builds further on the work done by *The SNE Research group*, which is currently accepted, but is not published yet[3], that introduces the new approach and architecture for inter-cloud connection. The feasibility of the implementation of multi-layer view is examined. An implementation based on the Sarastro controller is developed. In order to visualize different on-demand networks layers, the data that is used for writing this paper is based on sample data and the work is simulated. Due to the fact that the base paper[3] is not published yet there is no possibility to integrate the work done in this paper or in Sarastro with the current work done.
Before we focus on the visualization aspect of these layers we will first describe how the cloud providers interconnects virtual nodes and how networking is provided on demand with the use of NetApps. Then we are going to look into various ways of representing this data in a highly structured way so that debugging and troubleshooting will be made less complex. Once a method is found we will build this in a simulation with the use of sample data in a way that this can be integrated in the current system built by *The SNE Research group.*

2.3 Outline

The next Section 3 gives a general introduction to cloud providers, cloud instances and inter-cloud connection. The explanation of the technical requirements and our implementation of this are described in Section 4. Section 5 focusses on the analysis of our proposed solutions to provide a multi-layer view. The conclusions that can be drawn based on the results of our research are given in Section 6. Finally in Section 7 some suggestions for further research on this topic are given.

3 Inter-cloud connection architecture

As mentioned before to host the companies data or application(s) in the cloud, one could choose either one cloud provider or several cloud providers. There are multiple reasons for companies to choose multiple cloud providers:

- **Single point of failure**: There is an idiom saying "Do not put all your eggs in one basket." which matches perfectly in this situation. If there is a risk that cloud providers are not able to provide the correct service there is always the opportunity by outsourcing some parts of the company to multiple cloud provider. The risk will be distributed among different providers. Although it increases the level of reliability for the company this is not always the appropriate choice. There may be a case that there is one specific cloud provider with better quality measurements which matches the company's requirements better, it may be needed to make a trade-off. While the risk of failure in one cloud provide is low enough and on the other hand the cloud provider satisfies the company requirements on its own (without creating instances on other providers), it would be a thoughtful decision to use one cloud provider instead of many.

- **Geographical locations**: One can decide to spread its services across multiple providers based on their locations. The idea behind this is to deliver the services with shorter duration to destinations that are far away. So by using multiple cloud providers it would be possible to cover the targeted area.
- **Different quality measurements**: The cloud providers are ranked based on their quality measurements in different aspects. So a company can choose to use one cloud provider that is rated as a lower-class cloud provider (for several aspects) and one that is rated as a higher-class cloud provider (for several aspects).

All the cloud providers that are selected, have their own proprietary way of accepting requests in order to manage (add, delete, change) virtual nodes. With the use of only one cloud vendor we only deal with one management interface, but as we have two or more providers we have multiple management interfaces. As mentioned earlier *The SNE Research group* created a middleware layer that acts as a controller (or translator) that can send and receive requests to different cloud providers to make this simpler and more manageable. The system is able to provide a network connection between the virtual nodes regardless the virtual nodes are situated in one single cloud provider, or separated and reside in different cloud providers.

3.1 Sarastro

Sarastro is the name of the middleware server that runs different type of services in order to unify the proprietary commands of the different cloud providers. The two most important services that are running on Sarastro are Ruby Fog, which is nothing more than a translator that translates commands to and from the different cloud providers and the JavaScript layer that communicates with the client as this is pictured in Figure 2 . The client will send data (in the form of .json files) towards Sarastro [4] and Sarastro will translate these into an action that needs to be performed towards the relevant cloud provider(s).
There are different type of .json files that are responsible for different type of actions. The relevant .json files are the ones where we add or delete a node, and the ones that connects nodes with each other (when the link command is applied).

One of the most basic actions the end user can perform is creating or deleting one or multiple nodes (on a selected cloud provider). Another action is

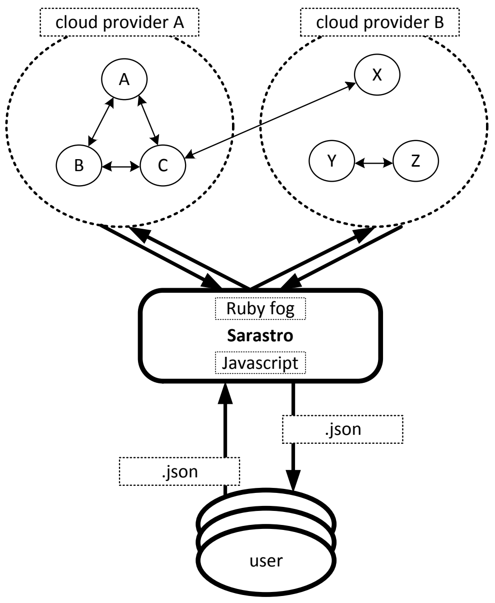

Figure 2: Illustration of communication of Sarastro

creating or deleting network connections between existing nodes (on the same or different cloud providers).

3.2 Network on demand

Other actions that can also be performed is to add "networking on demand". Creating a network connection between two virtual nodes regardless if they are in the same or in different cloud providers is now done by default with the use of Layer 2 Tunneling (L2TP). On top of this created network, the possibility will be there to build a network application that runs on these virtual nodes as this is illustrated in Figure 3.

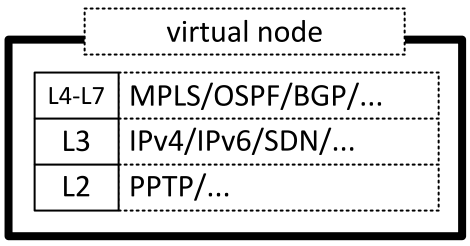

Figure 3: Illustration of different layers on virtual node

4 Experimental Methods

The intention of the research performed by *The SNE Research group* (Building a cloud infrastructure across multiple clouds) was to offer this in a structured and organized way, with the ability to achieve this with a minimum of administrative tasks. With this in mind, our goal is to apply the same simplicity to visualize these different types of networks.

One of the extra possibilities by doing this, is that this can also be applied for troubleshooting purposes. It does not make sense to build machines in the cloud and connect them with just a few mouse clicks and when there is a failure somewhere and one of the network layers is not working properly, it is still needed to go through all the layers of administration that it we tried to avoid earlier, by simplifying this.

With the visualization technique used, we are only focusing on visualizing the different network layers. The troubleshooting mechanisms are not included.

4.1 Technical considerations

Based on previous sections, it becomes clear that the on-demand network has enough layers of complexity. With having only three or four nodes to deal with, the network is still maintainable and troubleshooting is done fairly easy. When the number of nodes grows, troubleshooting may become a harder.
When troubleshooting a complex network it is harder to look at a network diagram which contains all the information and make a decision about the causes of a problem. What helps here is to separate different layers of data using nice presentation facilities. The following shows a list of browser technologies that where used to implement the multilayer view of the network. In order to keep the application stable it is recommended to use latest stable version of tools available.

- HTML5 Canvas[3]: Canvas is an element tag in HTML5 which provides

[3]http://www.html5canvastutorials.com/tutorials/html5-canvas-element/

a lot of features to draw and render graphs on the fly. The difference between this tag and other html tags is that canvas will be rendered with Javascript. Javascript programming with this element will be easier and the output will also be a welldesigned graphical result.

- CSS3[4]: This is the latest version of CSS and introduces new features in locating objects on the screen. Using a webkit-enabled browser (an open source web browser layout engine letting web browser to render special features), perspective views and also 3D transforms are possible.

- D3.js[5]: This stands for Data-Driven Documents. It is a JavaScript library which makes it possible to use data-structured files (such as json files) as input and draw well-designed diagrams. In this research it is used to draw network diagrams to show the cloud instances as nodes and their layer 2 VPN connection (or any other connection defined in layers) as links. There were some other libraries providing network diagrams but among them, D3 is the best option in terms of satisfying appearance preferences, high load speed and well-designed event triggered actions.

- Ajax[6]: This is a web-development technology which introduces asynchronous server-client data retrieving and processing. It means in case of an event being triggered, the whole screen will not be refreshed but specific parts of the screen containing the target element will be changed.

- JQuery[7]: This is another library of JavaScript which facilitates accessing the DOM and also makes the event-triggered action development more structured and object-oriented.

In order to implement the application, it is better to divide it into two separate phases: input data providing and data presentation.

4.2 Input data providing

As described before, Sarastro (as a middleware) initiates the inter-cloud connection. The front-end (which is *The SNE Research group* web application and the application created in this research as part of it) has nothing to do with running commands or directly connecting to cloud instances. The only thing that it has to do is send the proper requests to Sarastro and get their results. Sarastro provides the results in json files which are well-structured. So the input data for extracting different layer information is brought by Sarastro. The following shows the example .json file for a link between node1 and node2:

[4]http://www.css3.info/
[5]http://d3js.org/
[6]https://en.wikipedia.org/wiki/Ajax_(programming)
[7]http://jquery.com/

```
[{
        "job":"create_link",
        "requestid":3,
        "input":["email","1",{"src":"2","dst":"1","vid":"1"},1],
        "status":"done",
        "output":null,
        "start":"2013-03-19 21:34:02 +0100",
        "end":"2013-03-19 21:34:02 +0100"
}]
```

As it is shown in the code block, there are fields defined in .json file containing values in a structured way. For example the status of the link is shown in "status" field saying "done". Using Javascript libraries, it is possible to load the .json files and convert the information to proper objects. For example based on the "status" field, the color of the link on the screen can be decided.
Because the research is not published yet, it was not possible to work directly on that but the command set used to communicate with Sarastro and the .json files created by Sarastro as result are given which make the simulation of Sarastro's behavior in response to different incoming commands possible.
In order to provide the input data for the application and since the structure of the .json files are according to the sent commands, it is required to be aware of the commands sent to Sarastro and the output file received from it. Then the output .json files can be analyzed and used in the best way to display network layers.

4.3 Visualization

In this part, the required .json files are provided by Sarastro and it is needed to present them in proper way and also make multilayer view possible.

To visualize these different network layers and its corresponding information, it is decided that a very clear way of doing this is with the use of a 3D carousel. 3D carousel is well known in some other products but lacks on the on-demand network visualization topic. Each layer within the carousel will represent the IP addresses, the labels or the area where the routing protocol OSFP is configured. Each layer is defined in one different "div" HTML tag which has the same "nodes" set like any other layer. The only thing that may differ is the "links" set and also the labels and the information put for each node. Layers are in transparent mode which can be collected together and show the whole network in one sight. The reason to represent the layers in transparent mode is nevertheless the information of nodes and links are separated and classified in different layers but one can see all the layers together and not lose the whole sight of the network. The presentation can be narrowed to only one layer by one click on desired layer. It will rotate smoothly and all other layers will be invisible while the reviewer works on this layer.

4.3.1 Multiple layers

There is a variety of network types that can be offered as an "application" or "service". Based on the amount of information on each node and link and also requirements of web application the number of layers can vary. As a start point we visualize 3 layers such as "layer 2 vpn", "IPv4" and "IPv6" by using 3 nodes and 3 links. Figure 4 represents the nodes that all situated in different cloud providers which are connected together one by one. In order to visualize this information sample IP addresses are used which is shown in appendix A.1.

Layer 2 VPN This is a representation of the entire network as it shows all the nodes and links regardless of any specific condition. The labels for the nodes are name or server-id fields filled in output .json file from Sarastro.

IPv4 In this layer the labels on the nodes show the IPv4 addresses and the links are represented only if two nodes connected together have IPv4 addresses.

IPv6 The same happens for this layer also. IPv6 are shown as labels and the view of the links is dependent to the existence of IPv6 address for two nodes connected to the links.

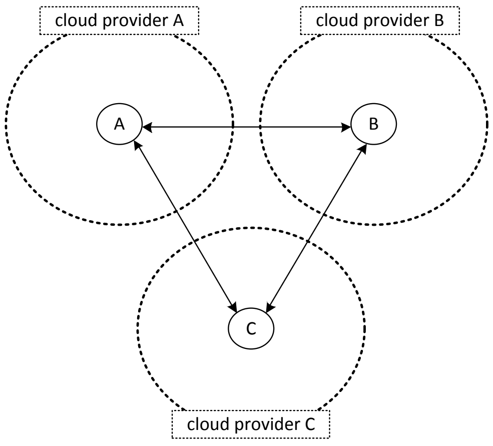

Figure 4: Illustration of virtual nodes across different cloud provider

4.3.2 Position of nodes

The nodes of the network can be located in layers based on different strategies. It can be decided according to the user's requirements and preferences.

Semi random positioning By default the D3.js library places the nodes on the screen in a way that seems random to the viewer but it is based on calculations of the best fitted position. At first the nodes are thrown randomly on the screen but there are algorithms and parameters that can be modified to optimize the placement of the nodes like weight-based forces. In this research there are different values used to fine-tune the locations such that the labeled information is clear and it is easy to realize that which nodes are connected.

Physical simulation positioning Each cloud instance belongs to one cloud provider which is located in different zones. Sarastro can provide latitude and longitude of each zone which helps find their exact location. In case that the user is concerned about the geographical placement of the cloud instances, it would be more preferable to put nodes on the screen regarding to their physical locations. This can be done be retrieving the list of zones for each cloud provider and also each zone's exact place. The next step would be scaling real location values into screen size dependent values.
The source code is provided in appendix A.2.

5 Evaluation

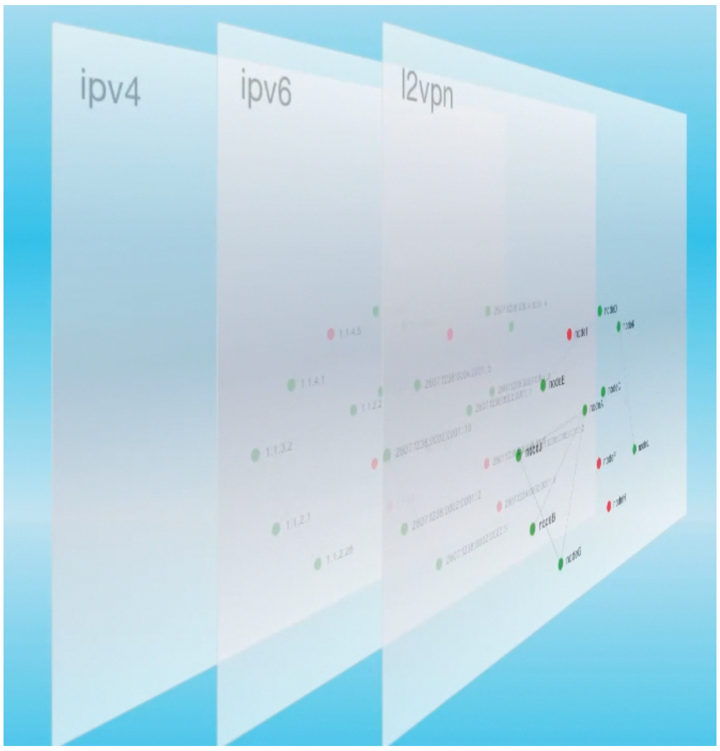

Figure 5: Multilayer view in the web application

As it is shown in figure 5, the layers are placed together with a difference value in X-Axis. The nodes are colored as green and red which shows the status of the node based on the Sarastro results. Red color means that the node is just created and it is in its initialization phase. It does not work properly because all the commands required to be run are not finished yet. On the other hand green color means that the cloud instance is in its working status and all the commands are properly operated.

Because of the lack of time all the features that can be possibly provided, are not implemented in the application. The following describes some characteristics of the proposed application:

Synchronous multilayer view While the user is working with multilayer view, it is best to provide the real time refreshment of the presented data. Because in case that one node goes from state ”created” (red color) to ”done” (green color), the user expects to see the new information set for that node in different layers like IPv4 address, IPv6 address and etc. This can be done using Ajax which sends asynchronous requests to Sarastro getting .json files and refresh the nodes and links that are changed. This can be an event-triggered action which is triggered using timers. The current version of Sarastro provides the set of commands that are enough for this feature. The following code block shows an example command to get links:

```
jQuery.get('/netapp/links', {vid : "1" }, function(data)
```

One sight view of the entire network Although it is valuable to see different layers it is always needed to combine all the information of layers and show the whole network in one sight. This will give the impression of the structure of the whole network to the user and helps the user finds desired areas to investigate more.

Modification of each layer In the current application introduced by *The SNE Research group*, it is already possible to change the network configuration only with a few clicks. Another feature for this project is modification of each layer. For example the user sees the IPv6 layer and notices that one node does not have the IPv6 address. It should be possible for user with a few mouse clicks and make a connection between two nodes which results in resolving IPv6 addresses for both of them. Because of the unavailability of Sarastro, the integration phase was not possible so this has to be done later. Sarastro should be able to create links between nodes based on specific layers.

5.1 Technical limitations

The main goal to achieve with this project is to separate information layers and present them in such a way that eases the understanding of the network. In terms of scalability, the bottleneck of the application has to be analyzed.

The number of nodes In case of large companies which contains a high number of departments with high number of services, the number of nodes that is needed to complete its infrastructure, will be high as well. D3.js works properly with 5000 total number of nodes[8] in one diagram. Although in case of some large companies even this number is not enough but also the presentation of this amount of nodes in one layer will not show a lot of information. The

[8]http://stackoverflow.com/questions/7730806/visualization-dead-slow-when-using-a-large-dataset

diagram will be so dense and the links and paths will not be distinguishable. One solution could be fisheye distortion[9] which magnifies part of the diagram based on a triggered-event. The other solution could be to separate the nodes in categories and in case of the user's request, the desired category will be zoomed in and show the nodes inside.

The number of layers There is always limitation on the number of elements that can be used in one page. In this project, layers are shown in terms of "div" element. Based on the screen size and the elements size, the number of elements in one page can be fine-tuned but still it does not fade the limitation of number of layers. There are some solutions that can be taken into account based on the situation: using scroll to show all the layers, set priorities for layers and not show all of them in multilayer view, decrease the size of layers in order to fit more layers in one page and much more. There are already some tests done on the maximum number of "div" elements in some well-known browsers[10]. Interestingly the limitation of "div" tags in HTML files is a number in the range $[10^5, 5 * 10^5]$. In this project, the number of layers will not be an issue.

6 Conclusion

The main question was if it was possible to provide network layering services in a visual organized way in order to display and debug (or troubleshoot) on a per-network-type layer method and to what extend this was possible.

By visualizing the different network layers with the use of a 3D carrousel we were able to provide the information on a per network per protocol basis. Although there may be better ways of structuring and visualizing this type of information, we were able to do this in a proper fashion and prove that the different network layers can be visualized in a structured manner and that therefor troubleshooting can be made easier.

The user is able to look at each (layered) provided network on demand (or application) individually, or is able to stack several layers on top of each other to get a better overview of the network.

When we look at current Network Management Systems (NMS), they are aimed to provide the user an overview of the network, up to device and interface levels. The drawback here is that these kind of NMS'es are normally bound to a single administrative domain, making it harder to apply this between nodes and networks between different cloud providers. Another drawback is that debugging on system level per protocol is hardly seen.

With the system we developed we are able to achieve the shortcomings mentioned above.

[9]http://bost.ocks.org/mike/fisheye/
[10]http://stackoverflow.com/questions/3486239/maximum-number-of-divs-allowed-in-web-page

7 Suggestions for Further Research

While making it possible to configure, and visually display network related information on different layers we noticed that this method can also be used to display information about virtual cloud nodes itself. In our research we only offer 2 extra layers on top of the default ”L2VPN” layer.

This method can also be used to display other type of networks like Software Defined Networking (SDN), or maybe even information about other network routing protocols like RIP for example. It could also be interesting to integrate information about the link utilization and visually make real time routing decisions and to have this displayed in concept before an actual change is made or decisions are taken. Now we only focused on the visualization aspect of the different network layers that are offered on demand that can lead to better and structured troubleshooting” and ”debugging” aspect. This visualization method along with other possible routing information techniques might be worth to research.

References

[1] Y. Demchenko, M.X. Makkes, R. Strijkers, and C. de Laat. Cloud computing technology and science (cloudcom), 2012 ieee 4th international conference on. pages 666–674, December 2012.

[2] Y. Demchenko, C. Ngo, M. Makkes, R. Strijkers, and C. de Laat. Defining inter-cloud architecture for interoperability and integration. *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 174–180, 2012.

[3] Yuri Demchenko Rudolf Strijkers Cees de Laat Robert Meijer Marc X. Makkes, Canh Ngo. Defining intercloud federation framework for multi-provider cloud services integration. *CLOUD COMPUTING 2013, The Third International Conference on Cloud Computing, GRIDs, and Virtualization (Accepted Paper)*, 2013.

[4] Jan Sipke van der Veen Yuri Demchenko Cees de Laat Marc X.Makkes, Rudolf Strijkers, Jan Sipke van der Veen Yuri Demchenko Cees de Laat Robert Meijer Makkes, Rudolf Strijkers, and Robert Meijer. Sarastro: Virtual internets in the cloud. *[Online]* `http://staff.science.uva.nl/~delaat/sc/sc12/2012-11-08-UvA-Virt_Internets.pdf`, August 2012.

[5] C. de Laat R. Strijkers, M. Cristea and R. Meijer. Application framework for programmable network control. *Advances in Network-Embedded Management and Applications*, pages 37–52, 2011.

A Appendix

A.1 Sample data - simulation

CLOUD PROVIDER	*LOCAL HOSTNAME*	*LOCAL ADDRESS*	*REMOTE HOSTNAME*	*REMOTE ADDRESS*
CLOUD_PROV_A	NODE_A	1.1.2.1/30	NODE_B	1.1.2.2/30
CLOUD_PROV_B	NODE_B	1.1.3.1/30	NODE_C	1.1.3.2/30
CLOUD_PROV_C	NODE_C	1.1.4.1/30	NODE_A	1.1.4.2/30
CLOUD_PROV_A	NODE_A	2607:f238:0002:0001::/64	NODE_B	2607:f238:0002:0002::/64
CLOUD_PROV_B	NODE_B	2607:f238:0003:0001::/64	NODE_C	2607:f238:0003:0002::/64
CLOUD_PROV_C	NODE_C	2607:f238:0004:0001::/64	NODE_A	2607:f238:0004:0002::/64
CLOUD_PROV_A	NODE_A	LABEL_21	NODE_B	LABEL_22
CLOUD_PROV_B	NODE_B	LABEL_31	NODE_C	LABEL_32
CLOUD_PROV_C	NODE_C	LABEL_41	NODE_A	LABEL_42

A.2 Source code

main.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;␣charset=utf-8
    " />
<title>Network Visualization</title>
<link rel="stylesheet" href="css/main.css" media="screen" type="
    text/css"/>
<link rel="stylesheet" href="css/3d.css" media="screen" type="
    text/css"/>
</head>
<body>
<script src="js/jquery.min.js"></script>
<script src="js/jquery.transit.min.js"></script>
<script src="js/d3.v3.min.js"></script>
<script src="js/diagram.js">
</script>
<div class="hero-unit" ><h1 id="logo-text">Network Visualization
    Application</h1></div>
<div id="navigation" class="well" style="float:left;">
<ul class="menu_main">
<li class="first"><a href="#" onclick="show_all()">Show network</
    a></li>
<li class="first"><a href="#" onClick="show_3d()">Multilayer view
    </a></li>
</ul>
</div>
<div id="experiment" style="display:none;␣float:left;margin-left
    :80px;">
<div id="cube">
<div id="layer_ipv4" onclick="show_me(this)" class="face␣ipv4"></
    div>
<div id="layer_ipv6" class="face␣ipv6" onclick="show_me(this)"></
    div>
<div id="layer_l2vpn" class="face␣l2vpn" onclick="show_me(this)">
    </div>
```

```
</div>
</div>
<style>
.node {

  stroke-width: 1.5px;
}
.link {
  stroke: #999;
  stroke-opacity: .6;
}
</style>
</body>
</html>
```

main.css

```
body {
	font-family: "Helvetica Neue",Helvetica,Arial,sans-serif;
	font-size: 13px;
	line-height: 18px;
	color: #333;

}
div {
	display: block;
}
.container{
	margin-top: 2em;
	margin-bottom: 2em;
}
.well {
	min-height: 20px;
	padding: 19px;
	margin-bottom: 20px;
	background-color: #f5f5f5;
	border: 1px solid #eee;
	border: 1px solid rgba(0,0,0,0.05);
	-webkit-border-radius: 4px;
	-moz-border-radius: 4px;
	border-radius: 4px;
	-webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,0.05);
	-moz-box-shadow: inset 0 1px 1px rgba(0,0,0,0.05);
	box-shadow: inset 0 1px 1px rgba(0,0,0,0.05);
}
#navigation .well {
	padding-top: 0;
	padding-bottom: 0;
	margin-bottom: 0;
	box-shadow: inset 0 1px 1px rgba(0,0,0,0.05);
}
```

```
ul, menu, dir {
       display: block;
       list-style-type: disc;
       -webkit-margin-before: 1em;
       -webkit-margin-after: 1em;
       -webkit-margin-start: 0px;
       -webkit-margin-end: 0px;
       -webkit-padding-start: 40px;
}
ul {
list-style: disc;
}
li {
       display: list-item;
       text-align: -webkit-match-parent;
}
#navigation ul {
       padding: .5em 0;
}
#navigation li {
       list-style-type: none;
       padding: .5em 0;
}
#navigation ul.menu_main {
       margin: 0;
}
#navigation li:before {
       content: " ";
       color: #ccc;
}
a, a:visited {
       background: inherit;
       color: #79A325;
       text-decoration: none;
}
.row:before, .row:after {
display: table;
content: "";
}
.node text {
  pointer-events: none;
  font: 10px sans-serif;
}
```

diagram.js

```
var nodes=new Array();
var links=new Array();
var graphnodes=new Array();
var placements= {};
function network(){
```

```
load_nodes();

}

function load_nodes(){
d3.json("nodes.json", function(error, graphnodes) {
var index;
for (index = 0; index < graphnodes.length; ++index) {
        var id=graphnodes[index].input[3];
        placements[id]=index;
        nodes[index]=new Object();
var group=0;
        switch(graphnodes[index].status){
                case "done":
                        group="green";
                        break;
                case "created":
                        group="red";
                        break;
                default:
                        group="blue";
}
        nodes[index].group=group;
}

        load_links("ipv4",graphnodes);
        load_links("ipv6",graphnodes);
        load_links("l2vpn",graphnodes);
});
}


function load_links(layer,graphnodes){
d3.json("links.json", function(error, graphlinks) {
links[layer]=new Array();
var index;
var link_set=false;
for (index = 0; index < graphlinks.length; ++index) {
        switch (layer)
        {
                case "ipv4":
                        link_set = false;
                        if(graphnodes[placements[graphlinks[index].
                            input[2].src]].output.public_ip_address
                            != undefined && graphnodes[placements[
                            graphlinks[index].input[2].dst]].output.
                            public_ip_address != undefined)
                                {
                                        link_set = true;
                                }
                break;
                case "ipv6":
                        link_set = false;
                        if(graphnodes[placements[graphlinks[index].
                            input[2].src]].output.ipv6_address !=
                            undefined && graphnodes[placements[
                            graphlinks[index].input[2].dst]].output.
                            ipv6_address != undefined)
                                {
                                        link_set = true;
                                }
                break;
                case "l2vpn":
                        link_set = true;
                break;
                case "mpls":
                        link_set = false;
                break;
                case "ospf":
                        link_set = false;
                break;
        }
        if (link_set == true)
                {
                links[layer][index]=new Object();
                links[layer][index].source=placements[graphlinks[
                    index].input[2].src];
                links[layer][index].target=placements[graphlinks[
                    index].input[2].dst];
                links[layer][index].value=parseInt(graphlinks[index
                    ].input[3]);
                }
}
        set_nodes_title(layer,graphnodes);
        draw_network(layer,nodes,links[layer]);
});


}

function set_nodes_title(layer, graphnodes)
{
        var index;
        for (index = 0; index < graphnodes.length; ++index) {
                if (layer == "ipv4") {
                        nodes[index].name=graphnodes[index].output.
                            public_ip_address;
                }
                if (layer == "ipv6") {
                        nodes[index].name=graphnodes[index].output.
                            ipv6_address;
                }
                if (layer == "l2vpn") {
                        nodes[index].name=graphnodes[index].input
                            [2].name;
                }
        }
}


function draw_network(layer,nodes,links){
var base="#layer_"+layer;
var width = 600,
    height = 500;

var color = d3.scale.category20();

var force = d3.layout.force()
    .charge(-140)
    .linkDistance(130)
    .gravity(0.01)
    .friction(0.1)
    .size([width, height]);
d3.select(base).html("<div␣class='layer_title'>"+layer+"</div>");
var svg = d3.select(base).append("svg")
    .attr("width", width)
    .attr("height", height);
force.nodes(nodes);

  force.links(links);

  force.start();

  var link = svg.selectAll(".link")
      .data(links)
    .enter().append("line")
      .attr("class", "link");

  var node = svg.selectAll(".node")
      .data(nodes)
    .enter().append("g")
        .attr("class", "node")
        .call(force.drag);

  node.append(/*"image"*/"circle")
        .attr("r", 5)
        .style("fill", function(d) { return d.group; });

  node.append("text")
```

```
      .attr("dx", 12)
      .attr("dy", ".35em")
      .text(function(d) { return d.name });

  force.on("tick", function() {
       force.resume();
    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node.attr("transform", function(d) { return "translate(" + d.x
         + "," + d.y + ")"; });
  });

}




/** 3D **/

var one_layer= false;
$(document).ready(function (){
network()
show_all();
$("#experiment").show();
});


function show_me(layer){
if(!one_layer){
                      $("#cube␣>␣.face").each(function (key,value)
                          {
                              var x=key*100;
                              var state="rotateY(-1deg)␣translateZ
                                  (100px)"
                              $(this).css("WebkitTransform",state)
                                  ;
                              $(this).css("transform",state);
                              $(this).css("MozTransform",state);
                              $(this).css("OTransform",state);
                              $(this).css("msTransform",state);
                              $(this).css("visibility",'hidden');
                              $(this).children().css("visibility",
                                  'hidden');

                      });
                      $(layer).css("visibility",'visible');
```

```
            $(layer).children().css("visibility",'
              visible');
            one_layer=true;
}

}

function show_3d(){
    if(one_layer){
            $(".layers_title").css("visibility",'hidden'
              );
            $(".layer_title").css("visibility",'visible'
              );
            $("#cube␣>␣.face").each(function (key,value)
              {
                $(this).css("visibility",'visible');
                $(this).children().css("visibility",
                  'visible');
                var x=key*100;
                var state="translate("+x+"px)␣
                  rotateY(45deg)␣translateZ(100px)"
                $(this).css("WebkitTransform",state)
                  ;
                $(this).css("transform",state);
                $(this).css("MozTransform",state);
                $(this).css("OTransform",state);
                $(this).css("msTransform",state);
            });
        one_layer=false;
    }
}

function show_all(){
    one_layer=false;
    show_me("#cube␣>␣.face␣:last");

}
```