ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Koshchii** Jméno: **Oleksandra** Osobní číslo: **452824**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Uživatelsky přívětivé ladění virtuálních počítačových sítí

Název bakalářské práce anglicky:

Pokyny pro vypracování:

Implement a tool that arranges and presents data gathered by the plotnetcfg[1] tool in an interactive graph form. The focus is on user experience with the typical user being a network administrator or a network developer. Research the needs of the typical users, gather their input, explore and analyze different approaches to the UI design, choose the most suitable approach and implement it.
Given the amount of data, it's important to not clutter the display with all the information available at once, yet all the data has to be easily and intuitively accessible in an ergonomic way. At any point in time, the presented data view has to be comprehensible and useful for the task at hand. For the typical user, there should be no need to search in documentation to find out how to get to a particular piece of information or how to get to a more generic view. Also, the data itself has to be presented in a well arranged, easily understood form.
Using data gathered by plotnetcfg and sosreport on three nodes of a typical OpenStack deployment, measure and evaluate the efficiency of understanding interconnections, packet flows and packet modifications between the network components by a typical user of the tool without and with the tool implemented.

Seznam doporučené literatury:

1] plotnetcfg json output format [online]. Available at: https://github.com/jbenc/plotnetcfg/blob/master/plotnetcfg-json.5
[2] D. Davis: The Essentials of Linux Network Administration, ActualTech Media, 2017
[3] Skydive Real-time network analyzer tutorial [online]. Available at: http://skydive.network/tutorials/first-steps-1.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Benc, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **31.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jiří Benc podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studentky

CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Bachelor's Thesis

User Experience in Virtual Computer Network Debugging

Oleksandra Koshchii
Software Engineering and Technologies

May 2019
https://github.com/sashkoboom/redhat_network_visualization
Supervisor: Ing. Jiri Benc

Acknowledgement

I would like to thank CTU FEL for being a great school for me, my friends and family for moral support throughout years of studying.

Declaration

I hereby declare that this thesis is my own original work and that all academic sources have been referenced according to the the Methodical instruction number 1/20 on Compliance with Ethical Principles of University Diplomas.

In Prague on 24. 05. 2019

....

Abstrakt / Abstract

V tomto dokumentu je posán proces tvorby uživatelsky-přivětivé vizualizace rozsáhlých virtuálních počítačových sítí. Jinými slovy, hlavním cílem bylo navrhnout, implementovat a otestovat program, který bude moci vykreslovat graf pomocí výstupu nástroje plotnetcfg[1]. Finální implementace musí moct vizualizovat počítačové interfaces, zařízení a data o nich, která jsou popsána v souborech ve formátu JSON odpovídajícím formátu generovanému pomocí plotnetcfg.

Klíčová slova: vizualizace; graf; uživatelský zážitek; ládění počítačových síti.

Překlad titulu: Uživatelský přivětivé ladění virtuálních počítačových sítí

This document describes the process of creating a user-friendly vizualization of large virtual computer networks. In other words, the main goal was to design a tool capable of rendering graph using output of the plotnetcfg tool[1]. The final program should be able to show network interfaces, devices and data about them described in a JSON file conforming to the format generated by plotnetcfg.

Keywords: visualisation; graph; user experience; computer network debugging.

Contents /

Tables /

Chapter 1
Introduction

The plotnetcfg[1] tool gathers network configuration data on a Linux system. Its main targets are nodes in a complex cloud infrastructure. [1]

A single such system typically contains large number (tens to hundreds) of virtual network devices, virtual bridges, failover solutions, tunnel overlays, etc. interconnected in various ways. To virtualize workloads, such systems often make use of containerization technologies which partition the networking interfaces into interconnected containers. This results in a convoluted topology. For developers working on and debugging cloud infrastructure, clear visualization of the topology and links between the virtual network interfaces is crucial. [4]

Thus, there is a need for a visualization tool capable of drawing graphs depicting interfaces and relations between them. To help with this effort, the plotnetcfg is capable of outputting all the gathered data in a JSON and DOT[5] format.

[1] `https://github.com/jbenc/plotnetcfg`

Chapter 2
Basic analysis

This chapter describes the initial analysis of the requirements for visualisation and examines the existing solutions of the addressed problem.

2.1 Requirements

Len Bass et al. state that a software architecture must be based on a set of well-specified requirements. [6] Because of that, based on the problem description presented in Chapter 1, a list of most general yet essential requirements for the final tool was gathered.

- **R1.** *Read input file in a JSON or in a DOT format.* It may assume it's well formed.
- **R2.** *Be platform independent.* and be usable on most computers without the need to install specific software (e.g. when debugging a problem at customer's site)
- **R3.** *Work offline.* To execute debugging in a wider range of conditions.
- **R4.** *Draw the network interfaces as nodes and draw the links and relations between the nodes as connecting lines.* This is the most comprehensible visual structure for network visualization. The network interfaces may be grouped into containers (a network interface must be in at most one container).
- **R5.** *Display metadata.* Network interfaces and containers have names; network interfaces may have a type, network addresses and states. Links may have a type and a tag.
- **R6.** *The data should be presented in a comprehensible arrangement.* E.g. by using different font styles and colors as well as shapes and colors of the container boxes, etc.
- **R7.** *Allow changing of placements of the elements.* The initial placement should be done in a way that the elements (network interfaces and containers) do not overlap, at least for the most common input data. The placement of the elements should be manually adjustable by the user, e.g. by dragging them with the mouse.

2.2 Possible solutions

There are three approaches to solve this task. First, it is known that an implemented solution already exists. [1] Second, it might be possible to find a visualization framework that can be configured according to the listed requirements. Third, another way is to design and implement the tool from scratch.

2.2.1 Graphviz [2]

Plotnetcfg already offers a tool that is able to visualize output, as well as to produce data in DOT format. The existing implementation uses Graphviz. It is capable to render a graph into a static PDF page containing information about virtual devices and showing the interconnections. It fulfills the following criteria:

- **R1.** *Read input file in a JSON or in a DOT format.* The program accepts DOT format.
- **R3.** *Work offline.* The program works offline.
- **R4.** *Draw the network interfaces as nodes and draw the links and relations between the nodes as connecting lines.* The program allows it as seen on Figure 2.1.

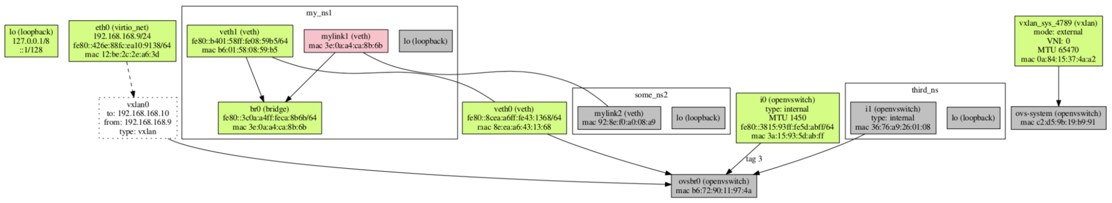


Figure 2.1. A sample visualization by Graphviz.

The resulting graphs, however, do not suit the other requirements.

- **R2.** *Be platform independent.* Graphviz software must be downloaded and installed from the official website[3] and does not provide an implementation for every platform.
- **R5.** *Display metadata.* The DOT language in general is too limited and is not suited for displaying of such complex relations. It also can't be used to display all metadata gathered from the system. It would not be practical anyway – the network configuration consists of too much data.
- **R6.** *The data should be presented in a comprehensible arrangement.* First, because of the network structure and Graphviz default behavior most networks are shown as a very wide graphs with a few or no hierarchy levels. Moreover, the result is stored in PDF format. Hence, a horizontal mouse dragging movement through a comparatively big sheet is required to view the graph in its entirety, which gives rise to frustration. Second, the boxes for interfaces are usually not well-arranged. In case of a complicated network structure the relations are very hard to follow, mostly because the lines are overlapping with the interface boxes.
- **R7.** *Allow changing of placements of the elements.* It is not interactive due to the PDF format limitations.

Therefore, this implementation does not meet the requirements R2, R5, R6, R7 for the visualization tool and, consequently, is not a suitable solution to the addressed problem

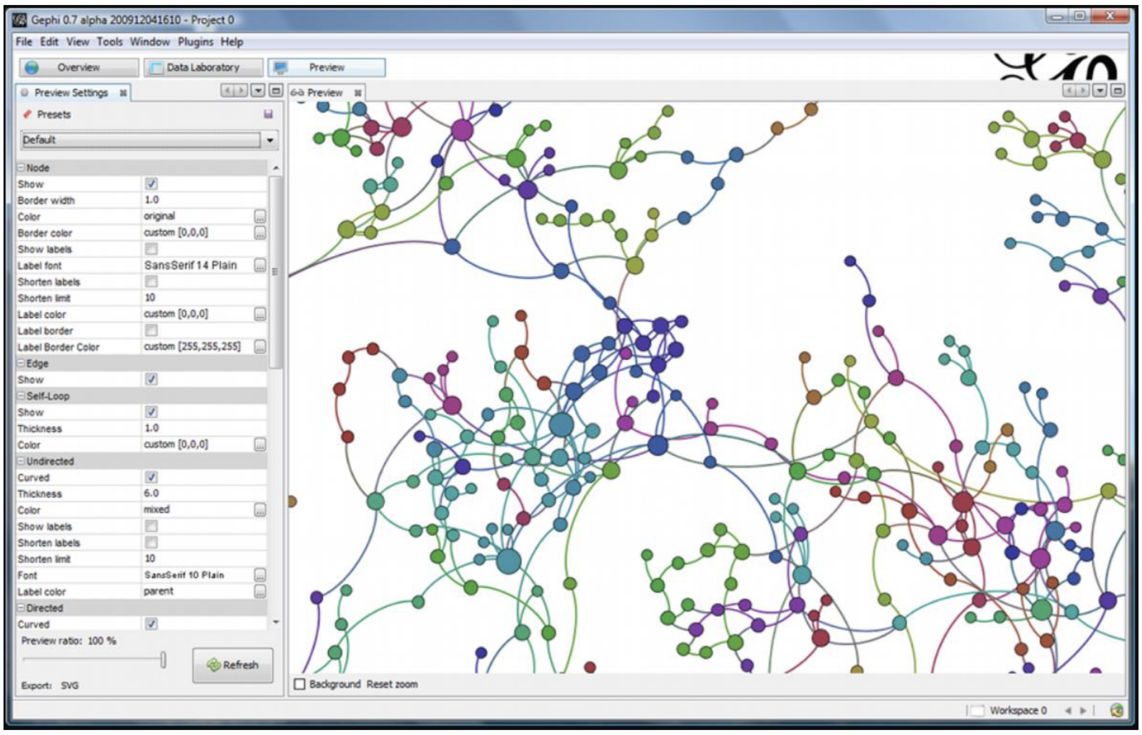
2.2.2 Gephi[3]

The closest possible solution to the problem found during the research is the use of Gephi. It is an open-source Java-based desktop visualization software for various kinds of graphs and networks.

- **R1.** *Read input file in a JSON or in a DOT format.* It is possible to implement a JSON-parsing interface for Gephi, as there are Java packages providing necessary functionality, for example: [1]
- **R3.** *Work offline.* As a desktop application, Gephi works offline.
- **R4.** *Draw the network interfaces as nodes and draw the links and relations between the nodes as connecting lines.* Gephi provides that possibility in its Graph API.

[1] `https://github.com/stleary/JSON-java`

- **R5.** *Display metadata.* Gephi implementation possess Attributes API, which is capable of defining node attributes as rows and columns.
- **R6.** *The data should be presented in a comprehensible arrangement.* Gephi allows that kind of configuration within its Graph API and Layout API.



Node	
Show	✓
Border width	1.0
Color	original
Border color	custom [0,0,0]
Show labels	
Label font	SansSerif 14 Plain
Shorten labels	
Shorten limit	10
Label color	custom [0,0,0]
Label border	
Label Border Color	custom [255,255,255]
Edge	
Show	✓
Self-Loop	
Show	✓
Thickness	1.0
Color	custom [0,0,0]
Undirected	
Curved	✓
Thickness	6.0
Color	mixed
Show labels	
Shorten labels	
Shorten limit	10
Font	SansSerif 10 Plain
Label color	parent
Directed	
Curved	✓

Preview ratio: 100 %

Export: SVG

Figure 2.2. A screenshot from Gephi. Source: [1]

As for the installation requirements, Gephi does not satisfy the requirement of simplicity in that matter.

- **R2.** *Be platform independent.* According to the installation guide, it's required to have a recent Java JRE installed. Gephi is compatible with Java 7 and 8 versions. Gephi is implemented on Windows, Linux and Mac OS platforms. Also, Gephi has issues with memory allocation and if too little memory is allocated, Gephi may stop running and lose the unsaved work. Moreover, for faster graph visualization, Gephi uses an OpenGL 3D engine, therefore, it demands a sufficient graphic card. If user's dedicated graphic card is manufactured before year 2010 or not installed at all, it is required to upgrade the hardware to run Gephi. [3]
- **R7.** *Allow changing of placements of the elements.* Even though it is possible for user to change the parameters of Gephi visualization on the run, Gephi does not allow manipulations with a specific node such as selecting it with a mouse and changing its position.

Therefore, Gephi is not a suitable solution to the addressed problem, because it does not meet some of the crucial requirements for the implementation.

2.2.3 Conclusion of the basic analysis

Since the existing solutions do not meet all the requirements, it is essential to develop a new solution that will suit the needs of the users.

[1] `https://gephi.org/screenshots/`

Chapter 3
Understanding virtual computer networks

The designed visualization is explanatory. The definition of an explanatory visualization by Noah Iliinsky and Julie Steele includes the fact that it tells a 'story', i.e. information that the visualization intends to convey from designer to user [7]. Without a basic understanding of data that are to be shown within the visualization it is impossible to imprint the required properties of said data into image in a clear and concise form.

Since the field is quite vast and complex, only it's basics were studied.

3.1 Elements and processes

The two most fundamental elements to review are network interfaces and namespaces. [8]

3.1.1 Network interfaces

Physical computers typically possess one or several network cards, also referred as network interfaces. They ensure communication within a computer network. It is possible to assign an IP address to it. Each interface has its unique name in the system, like "eth0", "vlan0" etc. Network interfaces may be physical as well as virtual. [8] These are major types of interfaces:

- bond

  In a complex server with several interconnected interfaces, it is possible that one of them might malfunction and cause the server to lose connection to the Internet. To avoid that, two interfaces (one of them being reserved) are connected to each other and referred to as bond. Despite being virtual, bond behaves like a typical network card. When bond receives a data package from outside, the operating system decides, where to send it next. When a data package is received by any of the interfaces connected to bond, bond receives it automatically as well. [9]
- bridge

  Bridge is another virtual interface that interconnects several other interfaces. When it gets a data package on one port, it automatically sends it on all other ports. The difference between a bond and a bridge is that a bond connects two places with two possible routes, while a bridge ensures that interfaces connected to it are able to "see each other" and communicate. [8]
- Open vSwitch

  Open vSwitch is a programmable variation of bridge. It has additional functionality besides sending packages, for example resending the package somewhere else or adding a VLAN tag to it. It creates a virtual interface connected to a switch. Within Open vSwitch, it is possible to program the direction of data flow. For these matters, such interfaces may have their own Open vSwitch rules listed similarly to routing table rules. [10]

3.1.2 Network namespaces

Big servers may contain several virtual machines. Each of them emulates presence of a hardware with it's own network card. Thus, virtual machine behaves as if there is a virtual interface within the server. Because maintenance of virtual machines consumes a lot of processing resources, the container technology is widely used instead.

Containers, also referred as namespaces, run inside one operating system. They behave as if they are isolated virtual machines that communicate through their own interfaces. They may have their own networking inside them with several interfaces. [11]

3.1.3 Tunnelling

Tunnelling works similarly to VPN (in fact, it is used in VPN development). It is a process, in which a secured logical connection is created between two endpoints by virtue of encapsulation. One network protocol is encapsulated into another. [12]

3.2 Major use cases

A typical use case is data loss. Within a chain of interfaces, data packages might get lost because one link could malfunction. To determine where it happened, network administrator must go through every interface and check if data flows through it. Thus, it is important to determine the chain quickly and to be able to determine where would data go next on crossroads. That could be learned from routing tables, Open vSwitch rules, etc.

Another use case is Open vSwitch development. The visualization might be useful to debug Open vSwitch activity efficiently. [4]

Specific tasks to be completed with the use of interactive graph visualization are listed in Appendix A.

3.3 Goals for the visualisation

While using this interactive graph visualization its typical user (network developer or network administrator) should be able to:

- rapidly determine state of the interface;
- see which interfaces belong to which namespaces;
- examine tunneled connections;
- view a chain of interfaces and determine the dataflow;
- look into routing tables;
- be able to see VLAN tag for any connection;

Chapter 4
User Research

Before heading into development, it is important to organize a primary user research. It helps to understand user's needs, motivations and behaviours through applying observation techniques, helding surveys etc. [13]

4.1 Designing user research

In this case, the methodology is to hold direct one-on-one interviews with the users to get their opinions on the possible design solutions presented as prototypes. As reported by Jeff Rubin, it is acceptable to ask the users about demonstrated prototypes straightforwardly. According to Rubin, questions can be global, like where to look for certain pieces of information within the layout, as well as about particular elements attributes, such as color. [13]

The design of the interactive graph visualization will be split into seven parts: graph arrangement, nodes color, arrows and lines, tunneling representation, initial information and additional information - one for each aspect of the visualization that holds data.

Because it is a visualization, it might be difficult for participants to figure out their preferences without viewing the examples, so each question about each aspect of the design should be supported with several illustrations.

Bigger number of examples of potential graphic solutions provided to users leads to more detailed opinions that can be collected from the participants. Jeff Rubins states that comparison tests with different prototypes matched against each other is a good way to avoid committing to one option that will turn out to be badly-designed in the future. [13]

4.1.1 Graph arrangement

According to Noah Iliinsky and Julie Steele, position is not the most efficient encoding method for data visualization. However it should be the first to be defined in a visualisation. [7] Therefore, the aspect of positioning will be discussed first.

Because interface nodes create hierarchical structures with parent nodes and child nodes, it is obvious to render them into hierarchical tree figures. However, in most of cases, all network data do not belong to one perfectly hierarchical tree. They rather form a set of independent structures that may or may not be connected in various ways. Thus, other options of graph arrangement were designed, too.

Figure 4.1 shows raw designs for overall graph structure.

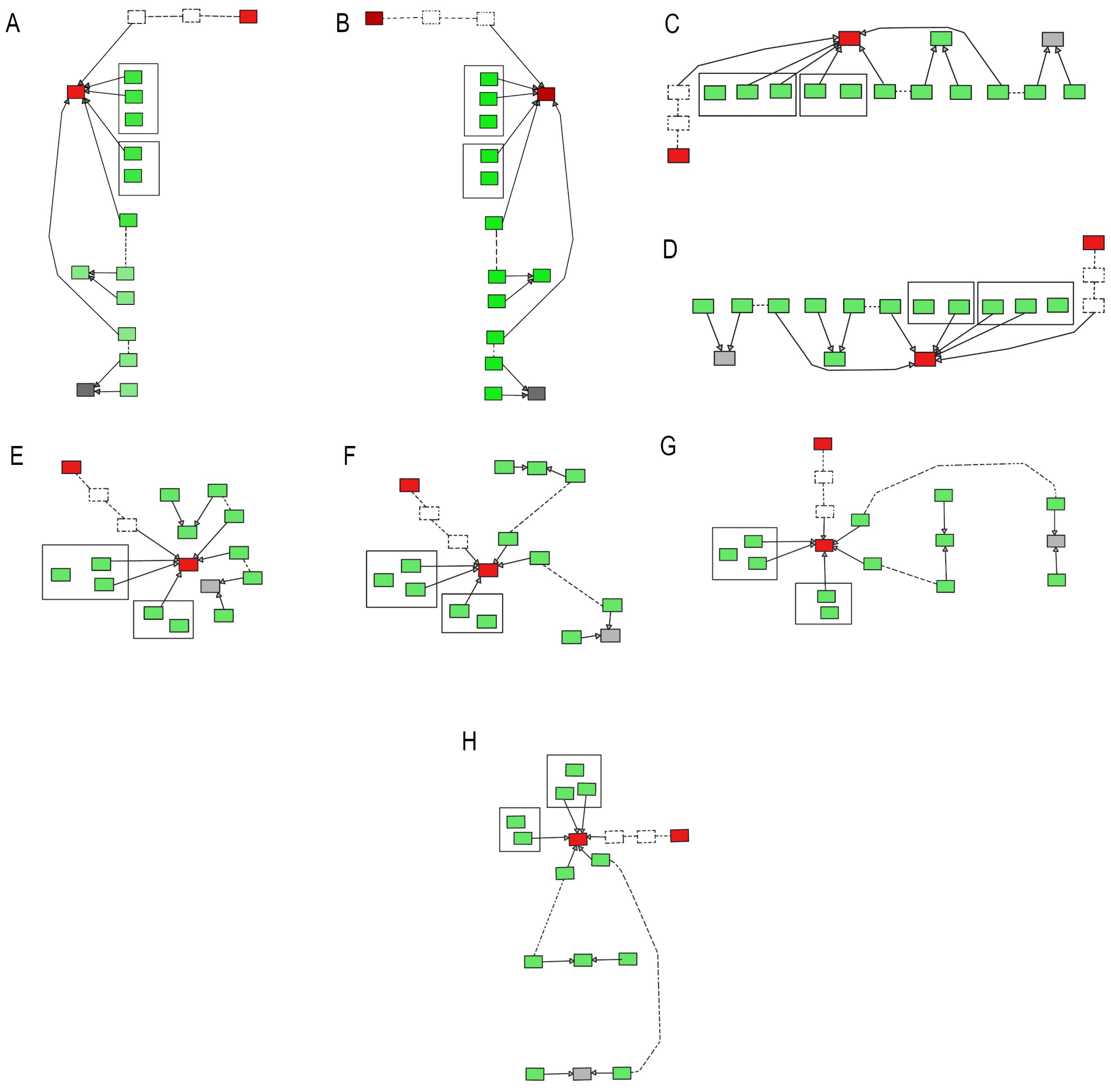


Figure 4.1. Designs for overall graph structure: hierarchical - left to right (A), hierarchical - right to left (B), hierarchical - top to bottom (C), hierarchical - bottom to top (D), clusters - roots in the middle (E), clusters - roots floating separately (F), clusters - roots in row (G), clusters - roots in column (H)

4.1.2 Interfaces color

Because interfaces are rendered as boxes, it is possible to not only to render some text about their corresponding interfaces on them, but also to add a visual color encoding in their appearance, for example by defining different fill or stroke attributes of the box that has some meaning.

As reported by Noah Iliinsky and Julie Steele [cite], color is not the best neither it is a universal tool for general visual data representation. These authors provide a number of reasons for that claim, such as different color interpretation in different cultures or accessibility limitations. However, color is a convenient property to label categorical data, such as states of the interfaces.

Plotnetcfg official documentation [1] defines the following states of an interface:

`down`*: the interface is administratively disabled.*

`up`*: the interface is up and operating.*

`up_no_link`*: the interface is up but has no link.*

`none`*: state cannot be determined or is not applicable to this kind of interface.*

Basing of these data, a set of options was designed. (Figure 4.2)

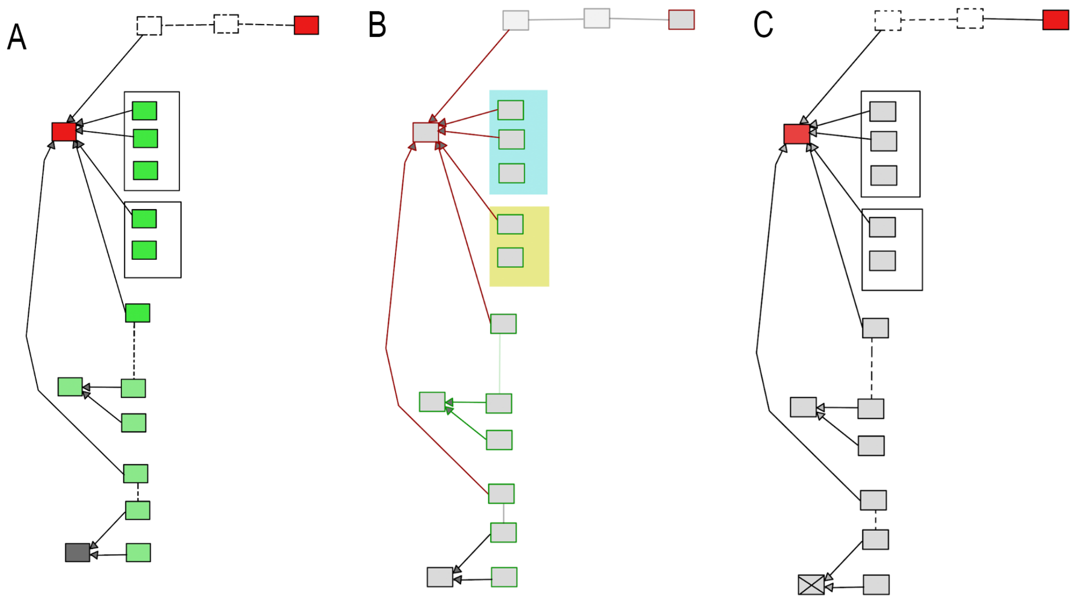


Figure 4.2. Designs for interface colors distribution: by status - fill as status: available - green, unavailable - red, down - gray (A); by status - stroke as status: available - green, unavailable - red, down - gray (B); by status - status: available - gray, unavailable - red, down - crossed (C)

4.1.3 Arrows type

Interconnections between interface nodes also provide information. For example, in the initial Graphviz implementation, each line between nodes had a “tag” property (rendered near the center of the line). It described, which tag is obtained by packets sent between interconnected interfaces.

As for the arrows, initial visualization had them, but connections did not have hierarchical direction, as for arrow from A to B meant that A is a parent of B. Instead, direction of the arrow represented connection direction, i.e arrow from B to A meant that B is plugged into A (whilst B being child of A). Despite all of that, when we discuss a real computer network, all the connections are usually bidirectional, which means that data within one connection between virtual devices flows in both ways. Therefore, arrows are not only pointless as an element of information but may also cause confusion.

Because of that, several arrow placement designs were created. (Figure 4.3)

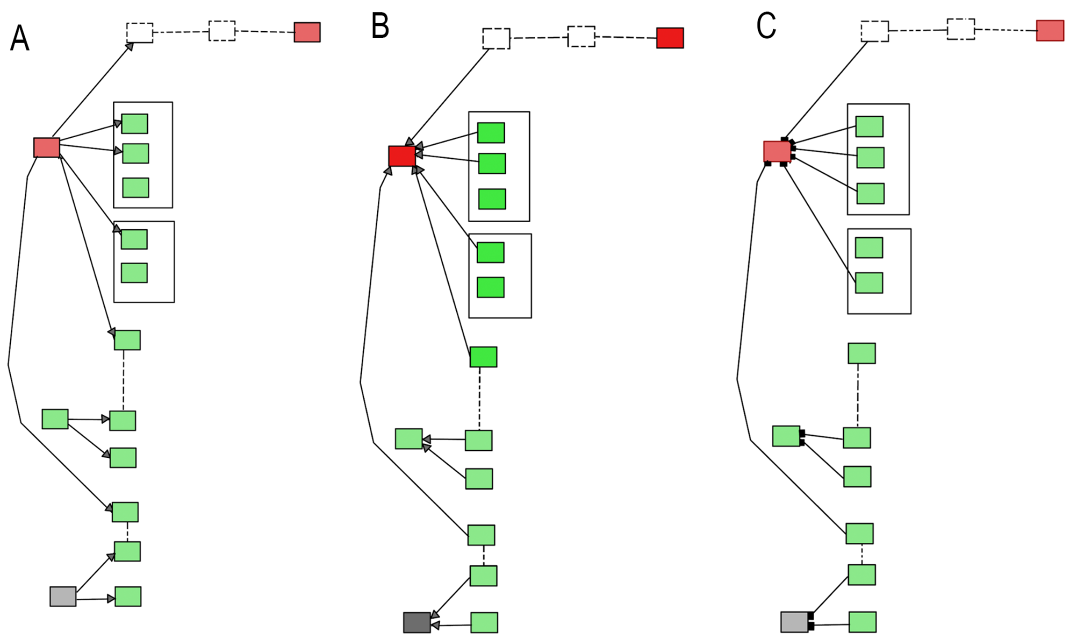


Figure 4.3. Designs for arrows shape and placement options: hierarchical - from parent to child node (A); by the direction of connection (B); metaphor of connection (C)

4.1.4 Lines color

Lines color can also encode information, such as states. Alternatively, their color may indicate, from which namespace is the line coming. These and other options are listed below. (Figure 4.4)

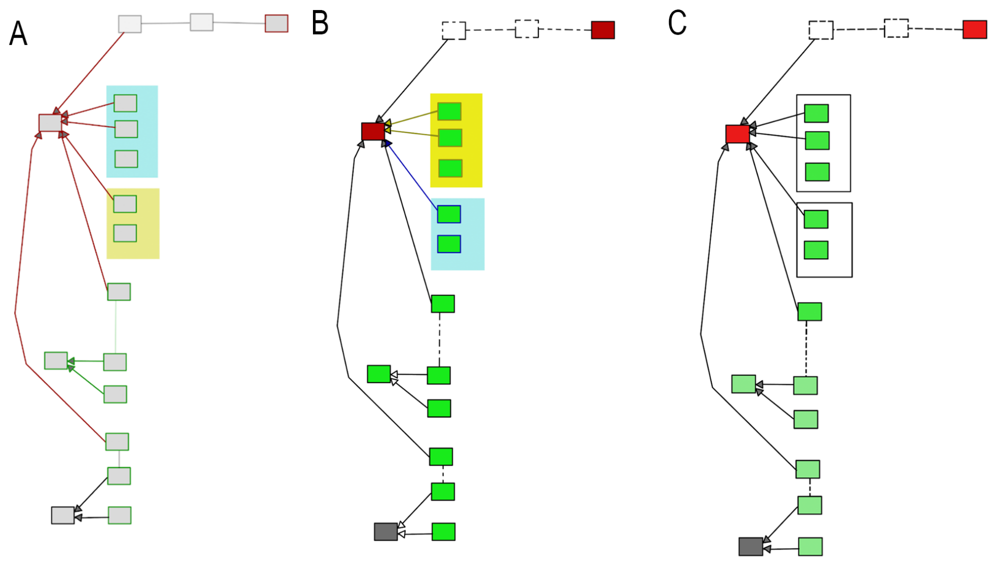


Figure 4.4. Designs of lines color: by status - stroke as status of the next node (A); by namespace - stroke by the namespace color (B); all black (C).

4.1.5 Tunneling

Tunneling is another property to interfaces and their interconnections that must be encoded. The initial visualization encoded tunneled devices and connections with dashed stroke as for “being less visible”. That is so because, in real systems, tunneling provides a way to “shadow” one network operation from another, as mentioned in Chapter 3. This approach conforms to the usability heuristics defined by Jakob Nielsen, “Match between system and the real world” [14]. The other variant suggests higher opacity of tunneled interfaces which serves the same purpose. (Figure 4.5)

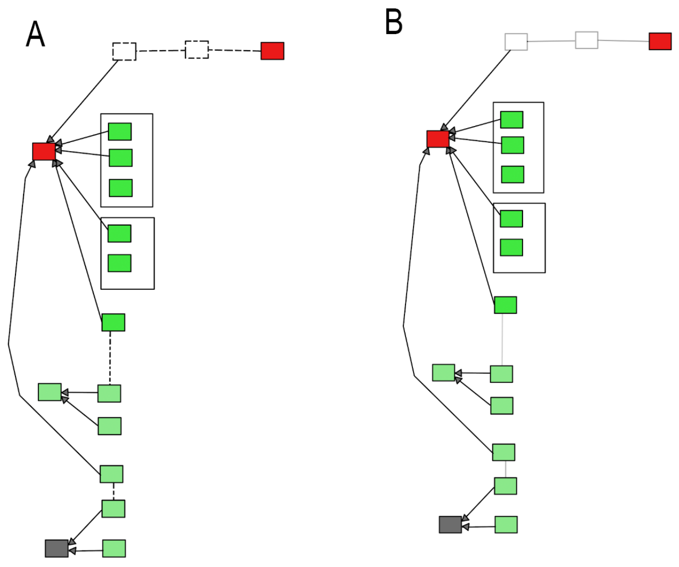


Figure 4.5. Designs for tunneling encoding: dotted elements (A); lower opacity (B).

4.1.6 Initial information

As each interface is represented by a box, it is important, for one thing, to distinguish them by some text on them, and, for another, to be able to read information about the devices that is most crucial and useful efficiently. It means that each interface and namespace box need to have a set of fields that is always visible. Also, there should not be too much information on every small box which may confuse the user.

A complete list of possible fields being rendered on every node by default was gathered. As for the interfaces:

- interface name
- driver
- IPv4 addresses
- IPv6 addresses
- MAC addresses
- MTU
- tunneling parameters
- bond mode
- firmware info
- XTP loaded

as for the namespaces:

- namespace name
- amount of routing tables
- amount of IP table rules
- indication : IP table rules loaded

The lists were shown to each participant. They were asked to tell which fields they would rather see on the corresponding boxes. Their feedback is listed in Table 4.1

4.1.7 Additional Information

For the time of this thesis development, the plotnetcfg output contained only sets of routing tables for namespaces. These tables should be placed in convenient locations. For one hand, they should be fully-visible to be studied by users efficiently. For another, they should not drastically overlap the rendered graph, since most of use cases require to study the routing table whilst tracing the nodes connections, as described in Chapter 3. Several designs were created using wireframes made with Balsamiq Mockups [1]. (Figure 4.6)

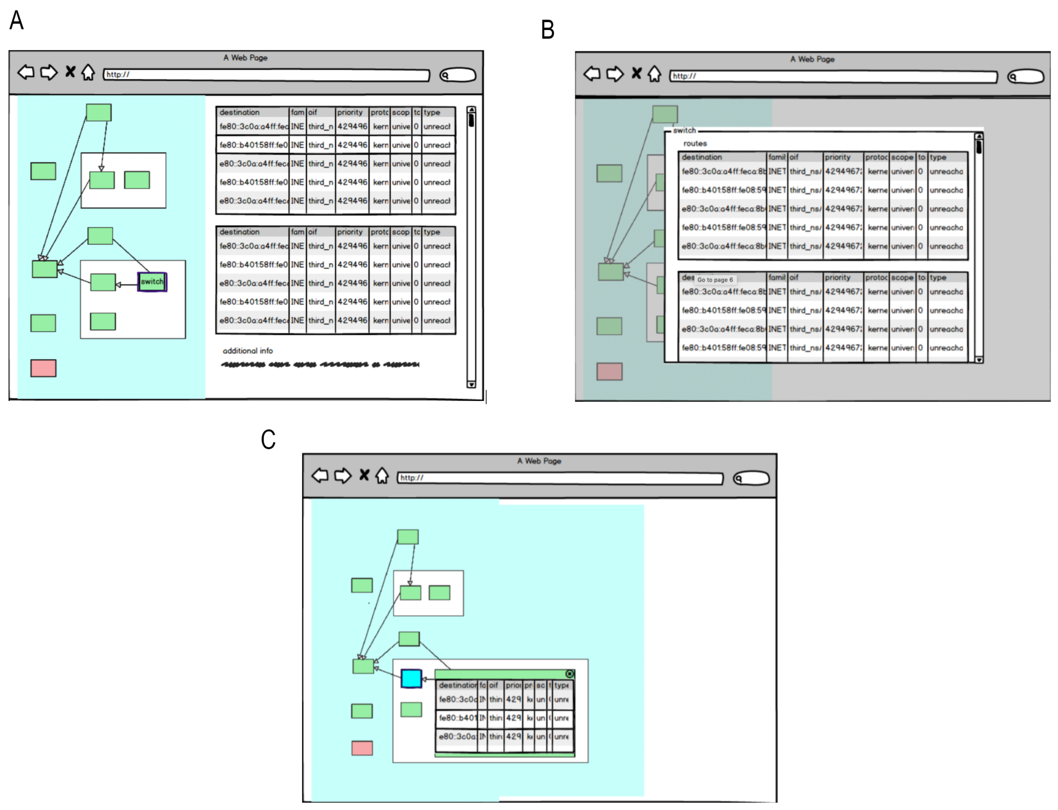


Figure 4.6. Designs for additional information display: info panel on the side (A); modal window (B); shrinking node (C). Wireframes were made with Balsamiq Mockups.

[1] https://balsamiq.com/wireframes/

4.2 Results of the user research

Three direct one-on-one interviews were held. In this section, their results will be presented with inputs provided by each surveyed interviewee (referred as Participant 1-3).

Participants were asked to vote for one or multiple options for each aspect of graph visualisation discussed in previous section. The final scores were modelled with Google Docs. [1]

4.2.1 Graph arrangement

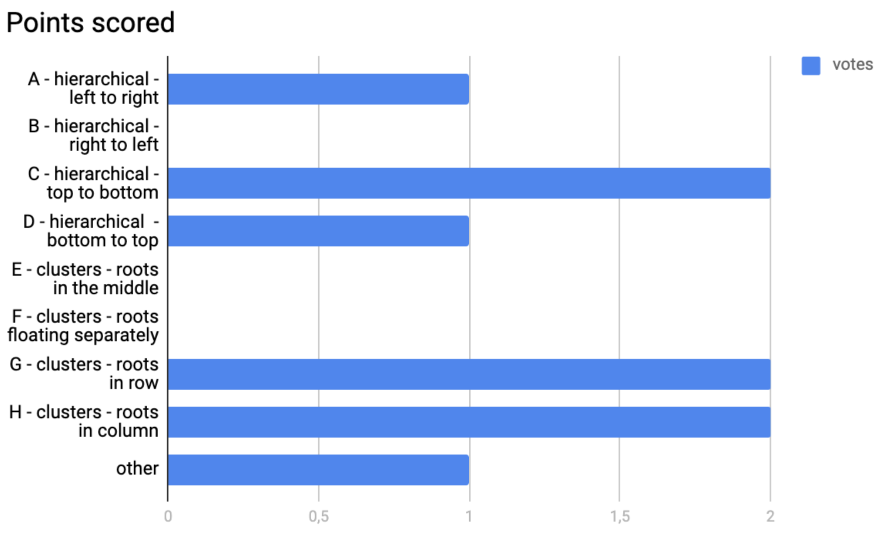


Figure 4.7. Score for graph arrangement

Overall participants liked the concept of graph arrangement to be hierarchical. The most popular variant was "top-to-bottom", two of three participants put it as their prior preference. Although hierarchical variants received as many voices in total as different types of arrangement (the former got 4 against the 4 for the latter), participants noted that they preferred hierarchical arrangement over the alternatives.

However, some participants pointed out that too wide or too tall graph would be hard to scroll through and to understand. This is one of the main flaws of the initial implementation, which used this exact type of arrangement. Nevertheless, some of the participants suggested to make the graph structure scalable through interactive design. One of them proposed hiding big amounts of children when zooming out.

[1] `https://docs.google.com`

4.2.2 Interfaces color

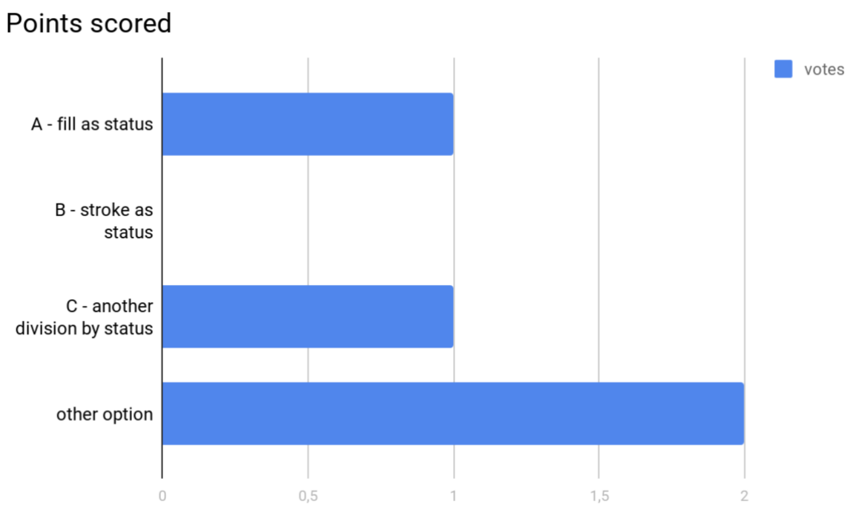


Figure 4.8. Score for interfaces color

Interface color question caused a lot of discussion. All of the participants had different views on this topic. (Figure 4.8)

The first participant pointed out that visual division by color would be inaccessible for color-blind users, which is a solid argument. Their preference was the optionC C, which contained only red and gray.

The second participant initially liked the "green - red - gray" variant but agreed it was inaccessible after I mentioned the remark from the first participant. Therefore, they disliked all of the choices, but suggested "green - green crossed - gray" color scheme for available, unavailable and shut down interfaces respectively.

The third participant disliked the idea of crossing interfaces that are unavailable, as the crossing lines will overlap the useful information written on the box of the interface node. Instead of that, they made a suggestion to make the division not by filling of the nodes, but by the wider stroke. The available interfaces would be green, unavailable - green with yellow stroke, shut down - green with red stroke. They also had the idea of making a legend which would give users the means for understanding the relations of colors with conveyed information.

4.2.3 Arrows type

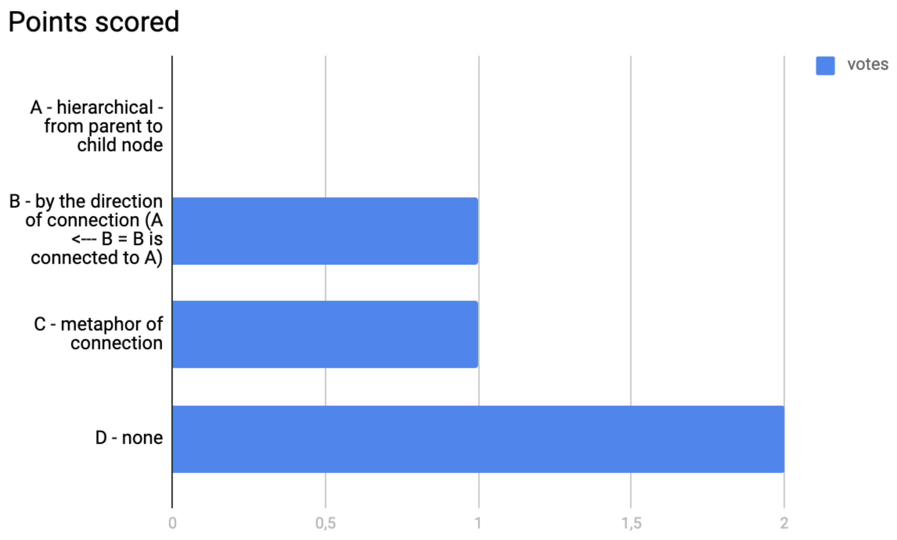


Figure 4.9. Score for arrows type

Regarding the arrows, participants mostly didn't think this aspect was necessary to render at all. According to Participant 2, as far as graph is arranged hierarchically already, there is no need to add arrows.

4.2.4 Lines color

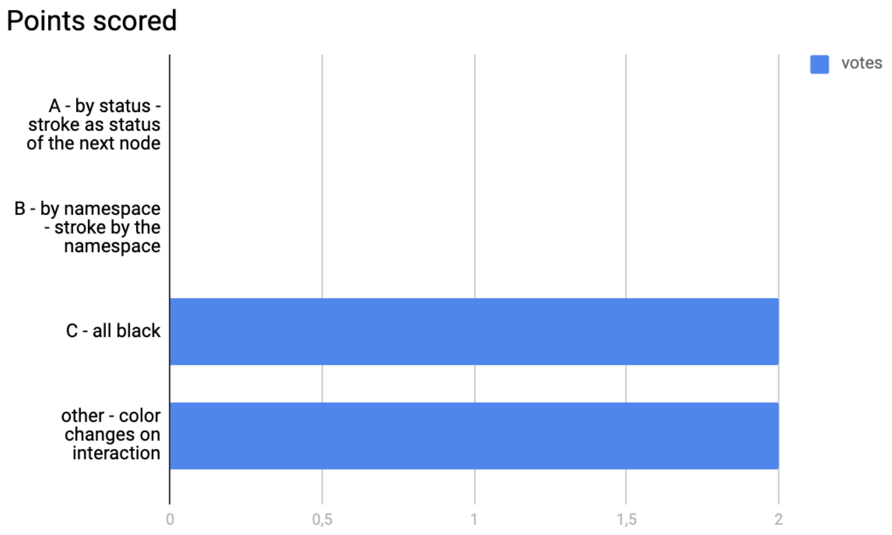


Figure 4.10. Score for lines color

All of the participants had an opinion that, with large amount of interfaces, the lines color would not really contribute to better understanding of the overall structure. Moreover, they heavily critiqued the B variant, with assigning lines the color similar to the namespace color they're heading to. According to the participants, if there would be thousands of namespaces, it would mean thousand of colours for lines, so they wouldn't be really distinguishable. Therefore, all of the participants prefered "all back" variant. (Figure 4.10)

But, in addition to that, participants supported the idea of lines color changing with user interaction. Two of them considered highlighting the lines when mouse is over them. The third participant came up with a whole interactive model. According to them, the lines would be all black by default, when clicking on line - line would have color of the namespace; when mouse is over a namespace, all the lines coming from a namespace would also highlight; when when mouse is over a node, lines coming from the node would highlight.

Thus, as reported by the participants, for this case, the best solution is to develop an interactive model.

4.2.5 Tunneling

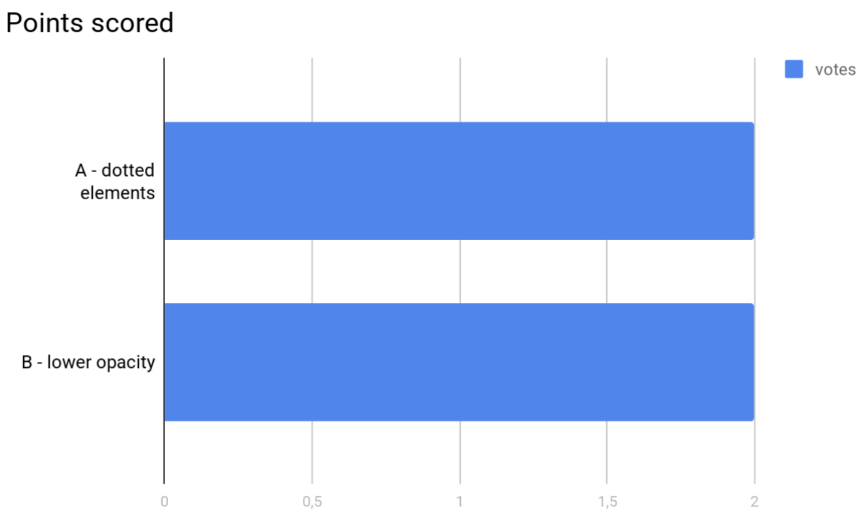


Figure 4.11. Score for tunneling

In case of tunneling there wasn't neither a real discussion, nor strong opinions from the participants. (Figure 4.11)They either preferred the dotted variant or liked them both. They also noted that the tunneling could be represented by a different type of arrow, leaving nodes representation the same as other nodes.

4.2.6 Initial information

Participants' feedback is listed in Table 4.1 with following answers:

- ++ - this field must be there
- + - this field should be there
- ? - this field should be there, but the participant hesitated (the reason may be listed in remarks)

The remarks:

A) Participant 3: agree, but show it only if interface is virtual
B) Participant 3: agree, but 1) addresses are too long to fit in the boxes; 2) there may be more than 1 address, which would cause even more visual complications
C) Participant 3: hide MTU to a secondary information panel

Participant	1	2	3	remarks
INTERFACES				
interface name	++	++	+	
driver	++	+	?	A
IPv4 addresses	?	+	?	B
IPv6 addresses	?	+	?	B
MAC addresses	+	++	+	
MTU	+	+	?	C
tunneling parameters				
bond mode				
firmware info				
XTP loaded	+			
NAMESPACES				
namespace name	+	+	++	
amount of routing tables	+			
amount of IP table rules	+			
indication : IP table rules loaded	+			

Table 4.1. Feedback for options of initial information to be written on interface boxes

4.2.7 Additional Information

In this case, participants either liked the idea of having a panel on the side, or suggested to make a floating panel near the node that was selected.

The third participant made up an interactive model as follows: when mouse is over a node, a small floating panel with additional info appears near it; when user clicks on the node, a bigger panel with more information appears. Another participant thought of buttons inside every interface, so the additional info could be revealed after clicking.

It is noteworthy that all of the participants came up with the same idea of dividing additional info about nodes into expandable sections. Moreover, they all thought about it independently, therefore, it is intuitive to organise it this way.

Chapter 5
Design

In this chapter is about creation and ideas behind the final design of the interactive graph visualization from both graphical and technological perspectives.

5.1 Visual interactive design

Based on the data and opinions from users described in the previous chapter and on additional research an interactive design was created to visualise all previously mentioned aspects of the graphs. Each subsection will be divided into two parts. The first will describe the final visual decision. The second will be discussing possible problems faced by the users as a result of chosen decision as well as by possible solutions. The issues described in the second part have occurred either during the design development or during the implementation.

5.1.1 Graph arrangement

Overall graph structure was decided to be hierarchical: top to bottom.

Noah Iliinsky and Julie Steele state that items located on top of the page are treated as more important than items on the bottom. People mostly read from the top down, so top elements will be read first. [7] This view is proved by the result of user research, in which all three participants stated that this is the most intuitive option. Thus, the overall structure was designed to be a vertical hierarchy with roots on top.

Each visualization is expected to contain several hierarchical structures. This must be handled by the design. Regarding visualisation of separate hierarchies which are not by any means connected, it is important to mention the notion of semantic distance. This term refers to how close or distant two units of language are in terms of their meaning. [15]. In the present thesis semantic distance will be used only for the task of locating fully independent graphs apart from each other.

Issue:

- long horizontal image

  Because the overall structure is chosen to be vertically hierarchical and, because in most cases network structure is consisted of many small independent structures, the final visualizations are expected to be horizontally long. This would not only cause complications in moving through the whole plane, but also in finding a particular interface node of namespace node.

  Possible solutions :

  - to implement a way to perform fast zoom in and out
  - to implement horizontal scrollbars
  - “bunch of children”: if node has a huge amount of childless children, hide them into one node or cloud representing bunch of nodes which can be expanded
  - to add search: when user enters an interface name in search field, the interface is highlighted

- many nodes in a single tree are on one level
  Possible solution: in case of bigger amount of nodes on one hierarchical level, group them into a few rows (ensure there is enough padding between levels for visible hierarchy)

5.1.2 Nodes color

Interfaces color. A color is assigned to each state:

down - red. *up* - gray, *up no link* - white, *none* - yellow.

This distribution is based on the discussion with surveyed users.

Issue: visual categorical division must be accessible - i.e. not depend on color only.

Color accessibility is important, as it lets users with visual impairments to understand the data as efficiently as non-visually-impaired users. To rely solely on color is unethical. [16].

Possible solution : To decrease dependency on color, it was decided to use patterns.

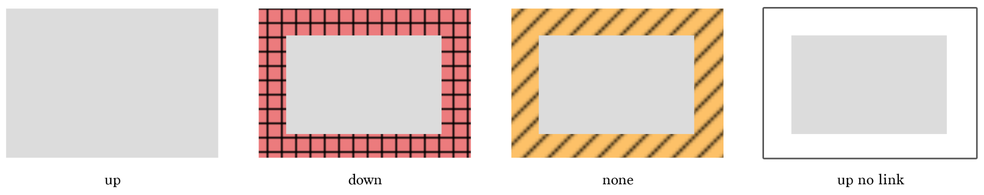


Figure 5.1. Raw design of interface state representation. Gray rectangles in the center will contain text about nodes.

For yellow and red states, stripes and crossed pattern were added respectively. Yellow and stripes correspond to their traditional meaning of caution, red and crosses were chosen to represent prohibition or unavailability.

This design was tested for the most common types of color-blindness with use of Adobe Photoshop CC Proof Setup function for color-blindness [17] https://helpx.adobe.com/photoshop/usi colors.html. Results are shown on Figures 5.2, 5.3

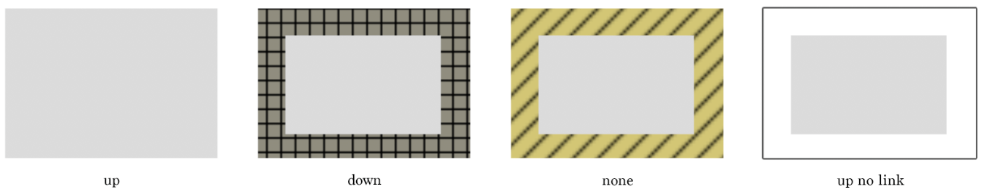


Figure 5.2. Appearance of interface node design for protanopia color-blindness type.

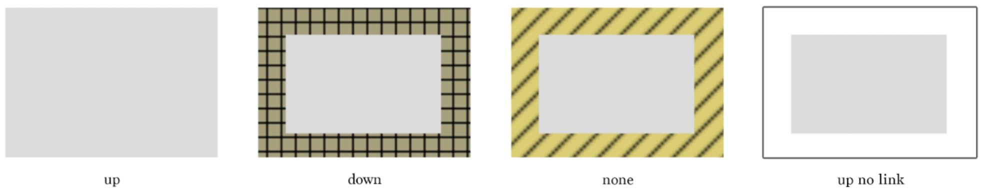


Figure 5.3. Appearance of interface node design for deuteranopia color-blindness type.

Namespaces color. In the case of present design task, there is no significant reason to visually divide namespaces into categories, thus they were decided to be in a single color.

5.1.3 Arrows

After user research, it was decided not to render arrows at all. Connections between nodes are bidirectional and data flow goes both ways, therefore arrows would not really contribute to understanding the network structure. Moreover, the arrows might bring harm to the overall simplicity and could cause uncertainty in data interpretation by user, as their presence does not have a meaningful reason. It is also worth mentioning that arrow direction traditionally represent hierarchy, which should be rendered in this visualization. Despite that, the graph would be already hierarchically structured.

Issue: lines are going through nodes they do not belong to (Figure 5.4)

It is difficult to implement an algorithm which would produce a complex graph without nodes being crossed by the lines that do not belong to said nodes. This problem is illustrated in Fig. N5. Moreover, the visualization must give users the ability to move any interface node, which adds complexity to the problem of node rearrangement.

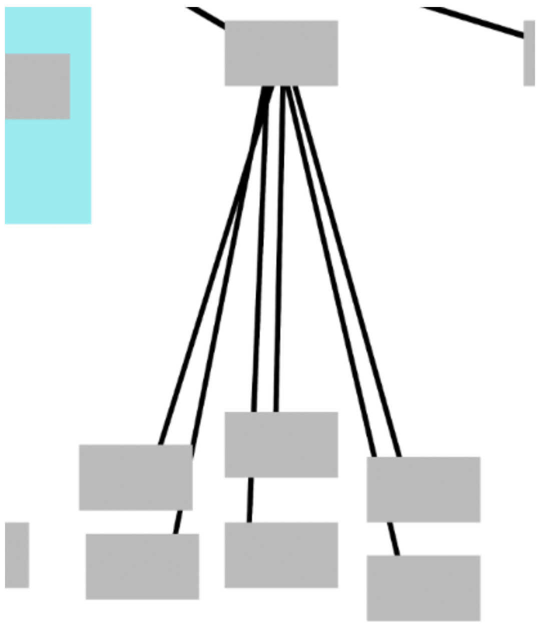


Figure 5.4. Illustration of the problem with lines going through nodes they do not relate to.

Possible solution:

Hierarchical parents and children would get round marks at the end of the line. (Figure 5.5) Because marks are similar on both sides, they do not represent direction. Also, marks resemble real-world hardware plugs, which brings Compatibility with Reality property to the visualization, which is one of the Nielsen heuristics for usability inspection. [14]

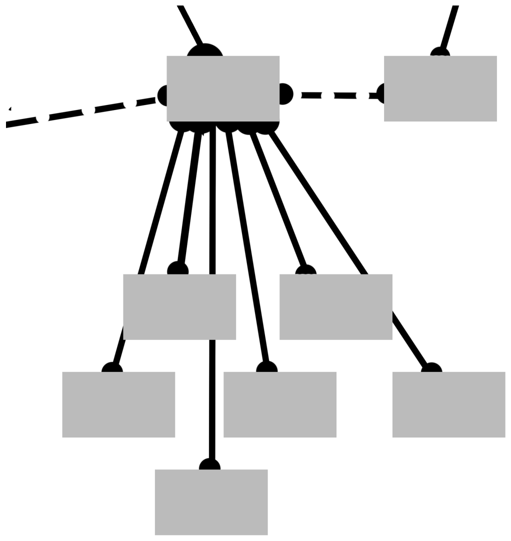


Figure 5.5. Raw design for marks at the end of the lines.

5.1.4 Lines color and interaction

In static condition, i.e. with no user input, lines between interface nodes would be black. User research have shown that the less confusing option is to change lines color only to highlight some nodes upon user interaction. Thus, an interaction model for mouse movements and clicks was developed as follows:

- initial state
  - action: no action
  - line color: black
  - illustration: Figure 5.5
- mouse over line
  - action: mouse over line L between interface nodes I1, I2
  - line color: L, N1 and N2 are highlighted
  - illustration: Figure 5.6

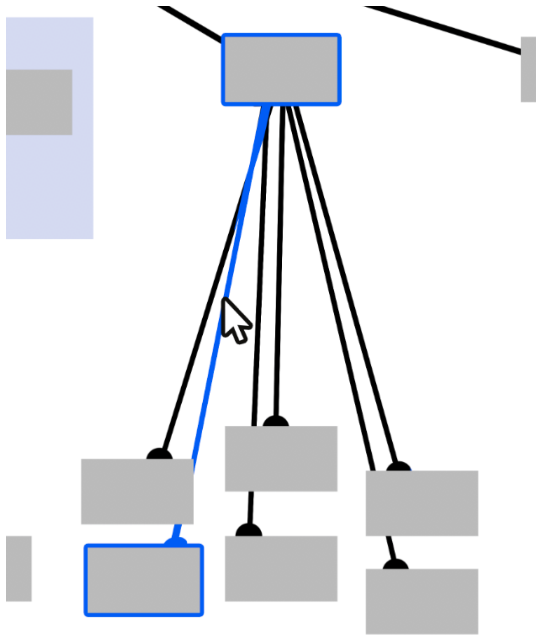


Figure 5.6. Illustartion for `mouse over` interaction with lines

- mouse over namespace rectangle
  - action: mouse is over namespace node N
  - line color: all lines coming to/from N are highlighted
  - illustration: Figure 5.7

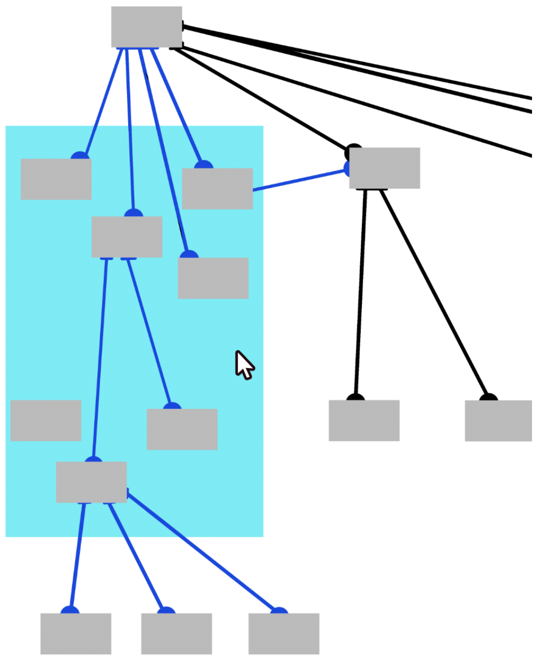


Figure 5.7. Illustartion for `mouse over` interaction with namespaces

- mouse over interface rectangle
  - action: mouse is over interface node I

- line color: all lines coming to/from I are highlighted, as well as all interfaces I is connected to
- illustration: Figure 5.8

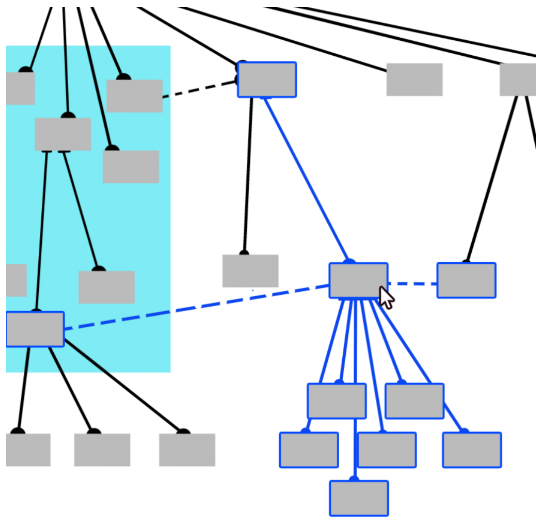


Figure 5.8. Illustartion for `mouse over` interaction with interfaces

Issue: highlighting with color is not enough for bigger graphs.

Rendered graphs are in most cases expected to be huge, with big amount of nodes displayed. Firstly, even with smart hierarchical structure, some connections between nodes may end up far beyond the screen. Secondly, even with highlighted lines it might be hard to concentrate on two specific nodes with a complex detailed structure around them.

Possible solution: lower opacity of elements that are not highlighted (Figure 5.9) Make highlight and opacity decrease last when user makes a click, return to initial state after the second click

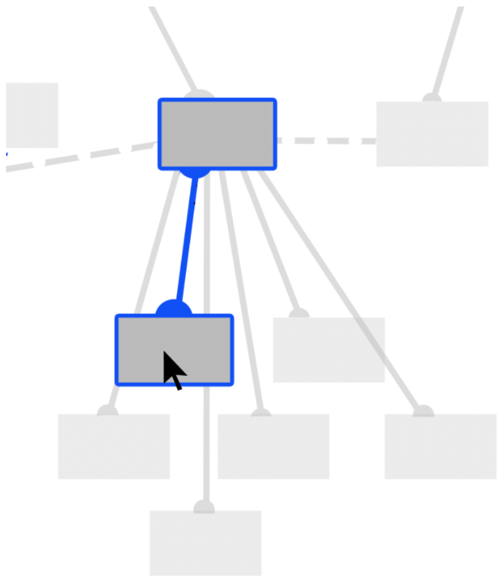


Figure 5.9. Overall opacity decreases when specific elements are highlighted

5.1.5 Tunneling

After the user research and further analysis, it was decided to use the initial design of tunneled interfaces and connections based on the Graphviz implementation where the stroke between the tunneled nodes is dashed. The rest of the interface design will remain the same as for their non-tunneled contreparts (Figure 5.10).

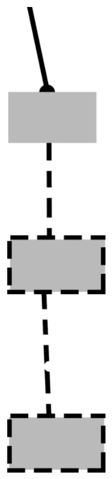


Figure 5.10. Raw design of tunneling representation.

5.1.6 Initial info

User research has shown that the following fields are best to be shown in following scale configurations

Static info on interfaces

- normal scale

  text on the box: interface name

  argumentation:

  - interfaces are rather small
  - many information on each of hundreds of visible boxes is confusing

  illustartion: Figure 5.11

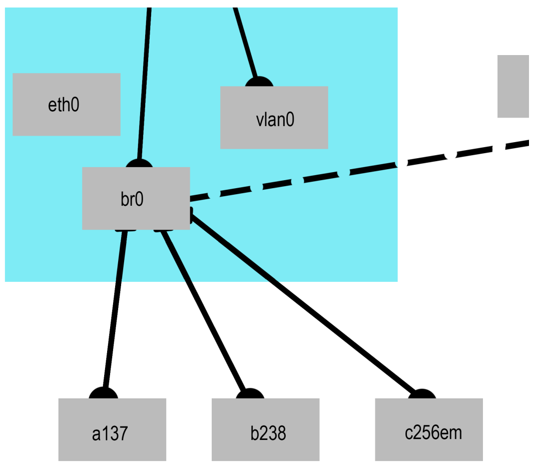


Figure 5.11. When graph is in normal scale, there are interface names on interface boxes.

- graph is scaled down

  text on the box: none (interfaces are only filled with their status color and pattern)

  argumentation:

  - it's easier to find a red or yellow interface fast
  - any text on small boxes is unreadable

  illustartion: Figure 5.12

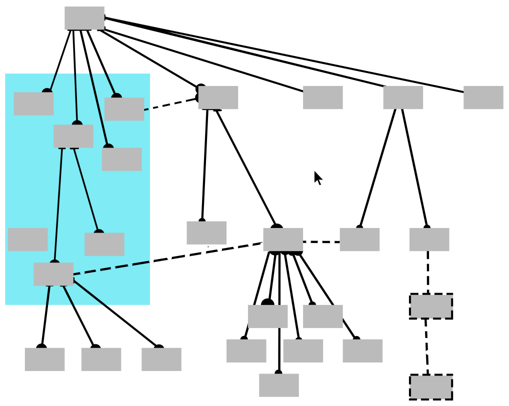


Figure 5.12. When graph is scaled down, there is no info on interface boxes.

- graph is scaled up

  text on the box:

  - interface name
  - MAC address
  - MTU

  argumentation:

  - more space for text
  - the most important info according to users

  illustartion: Figure 5.13

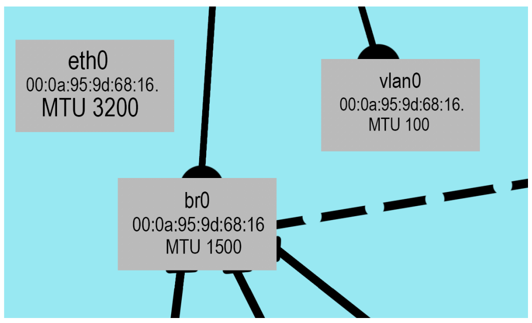


Figure 5.13. When graph is scaled up, there is a set of fields displayed on the interface boxes.

Static info on namespaces

- any scale

  static text in the corner:

  - namespace name

- count of routing tables loaded

  argumentation:

- fast way to get number of routing tables at any scale
- information in the corner would not overlap anything

5.1.7 Additional Information

As before, the design described below is based on the interviews with the users. Additional info would appear after following actions:

- mouse click on interface

  consequence: a floating panel near the interface appears with:

  - interface name
  - interface state
  - driver
  - IPv6, IPv4 addresses
  - MAC address
  - MTU

  (panel is dismissed by clicking on x button)
- mouse click on namespace

  consequence: a floating panel appears with a set of routing tables

5.1.8 Color palette

Aesthetics contribute a lot to visual appeal of any webpage. Users not only find visually attractive websites to be more usable, but also tend to be more tolerant to usability issues on such webpages [18]. This phenomenon is described in great detail in the book "Emotional Design: Why We Love (or Hate) Everyday Things" by Don Norman [19].

Therefore, because color palette is one of the few fully predictable aspects of every visualization, it should be designed as visually pleasing.

The color palette for this visualization was generated by `COLOR SUPPLY` [1] application as shown in Figure 5.14.

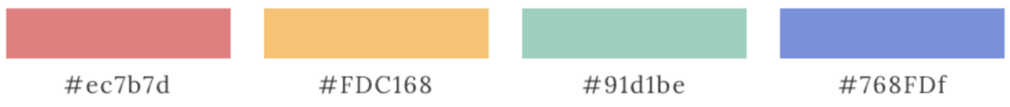


Figure 5.14. Color palette generated with COLOR SUPPLY tool.

While red and yellow would be used as primary colors for states as will be described in section , green and blue would be used for namespace colors and highlights respectively. Another color to be used is gray, also assigned to an interface state. (Figure 5.15)

Figure 5.15. Gray color for the color palette

[1] `https://colorsupplyyy.com/app`

5.2 Technology

The solution must run on any platform and have minimal installation requirements. As web browsers are ubiquitous, a suitable option is to develop the program with HTML and a scripting language. Thus, the tool is developed as a browser application running on JavaScript (the latest version of the language at the time is ES6/7). Also, JavaScript possesses native APIs to read and parse JSON from file, which is useful for our purposes. The graph structure would be represented in SVG format, which is easily created and manipulated by JavaScript. Though implementation of a SVG manipulation library from scratch would be too complicated and time-consuming. Therefore, for these purposes, it was decided to use D3[1] framework.

5.2.1 D3.js

Data-Driven Documents or D3.js is a JavaScript library for data visualization which uses web standards. D3 provides:

- a data-driven approach to the visualization;
- an interface to create and manipulate SVG elements;
- a framework for variable user interaction, such as dragging and dropping specific elements, panning and zooming the SVG container;
- a framework for programming graphs with a possibility of programming node behavior, i.e. node arrangement would be dictated by the predefined rules.[20]

5.2.2 D3 Force

`D3 Force` module contains a simplified implementation of a velocity Verlet[21] numerical integrator which simulates physical forces on particles. It is practically useful for generating interactive graphs representing network and hierarchy relations. [22]

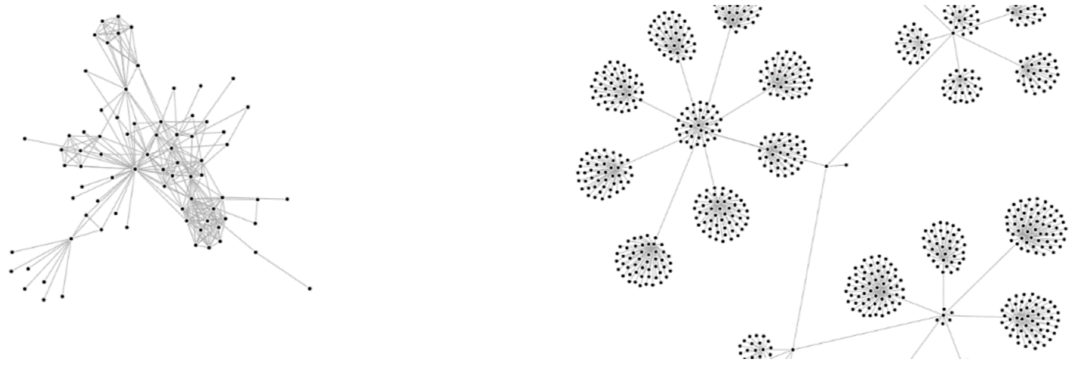


Figure 5.16. Examples of D3 Force visualization. Sources: [2], [3]

The framework allows to create a so called "force simulation" and define the forces for its calculations of node positions. The force itself is a function which modifies nodes' positions or velocities; it accepts a D3 node structure as an argument and recalculates its geometrical "x", "y" values according to its rules. In force simulation, calculations are held within interactions. After each interaction each nodes position is recalculated for every force to approach the optimal position.

[1] https://d3js.org
[2] http://bl.ocks.org/mbostock/ad70335eeef6d167bc36fd3c04378048
[3] https://bl.ocks.org/mbostock/95aa92e2f4e8345aaa55a4a94d41ce37

D3 offers a range of its default configurable forces, such as collision force (a force that prohibits node overlapping by defining unpassable circles around them), force-X and force-Y (a node is approaching a specified "x" or "y" value), centering force (a node is approaching a specified "x" and "y" values) etc. and also allows to program custom force functions.[20]

5.3 Architecture

1) Collect JSON data from the user.
2) From these data, separate the information about namespaces, interfaces and interfaces' relations into arrays of three different data structures, which would be suitable for further D3 manipulations.
3) Calculate default positions for nodes representing interfaces and rectangles representing namespaces.
4) Render a force-directed graph with D3.

Because the solution depends heavily on D3, which is used at the end of the pipeline, the order of design and implementation was reversed relative to the rendering process.

5.3.1 Render

The rendering interface must be able to:

- accept sets of: objects representing interfaces, objects representing links between interfaces, objects representing namespaces (nodes and links objects are demanded to be defined by the D3)
- render them as D3 force-directed graph similar to Figure5.17

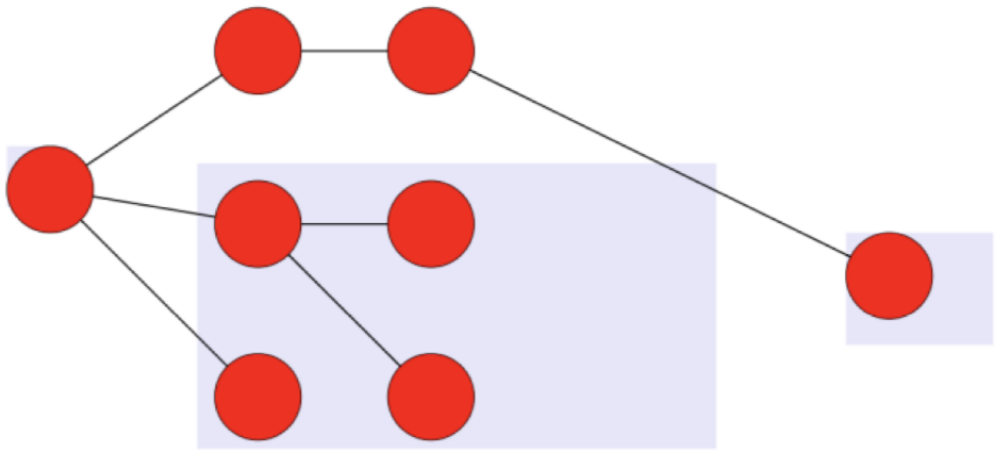


Figure 5.17. Graph generated by the initial implementation of rendering interface as an example of a supposed force-directed graph structure

Considering the typical network topology, the graph must have a tree-like structure (a child node may have one or many parent nodes and vice versa).

To fulfill the requirement of understandable node arrangement, it was decided to place nodes horizontally into corresponding hierarchical levels, i.e. the parent nodes would always be on the left and the child nodes would be on the right. The desired alignment can be managed by D3's force type "forceX" which makes node move towards the given "x", as if it was magnetically attracted.

To avoid overlapping nodes, the use of D3s' default collision force mentioned before is a reasonable option.

As for the namespace rectangles, they would have a static predefined position and the interface nodes would not be allowed to leave the area of a namespace they belong to. It would be provided by a custom force which would control whether supposed future position is situated inside the predefined rectangle (if it doesn't, the node will be left on the brink of the rectangle).

5.3.2 Handling data for rendering

The general plotnetcfg JSON output structure is an object, containing a set of namespace objects, which contain a set of interface objects providing most of the metadata. As stated before, on this step, the data from JSON must be put into objects which will be suitable for the rendering interface. There must be separate classes for interfaces and namespaces which would represent the graphical elements and contain the data. These classes are revealed on the Figure 5.18. Objects in the final implementation contain more fields and methods than listed in the Figure 5.18, but weren't mentioned because of being irrelevant to the current discussion.

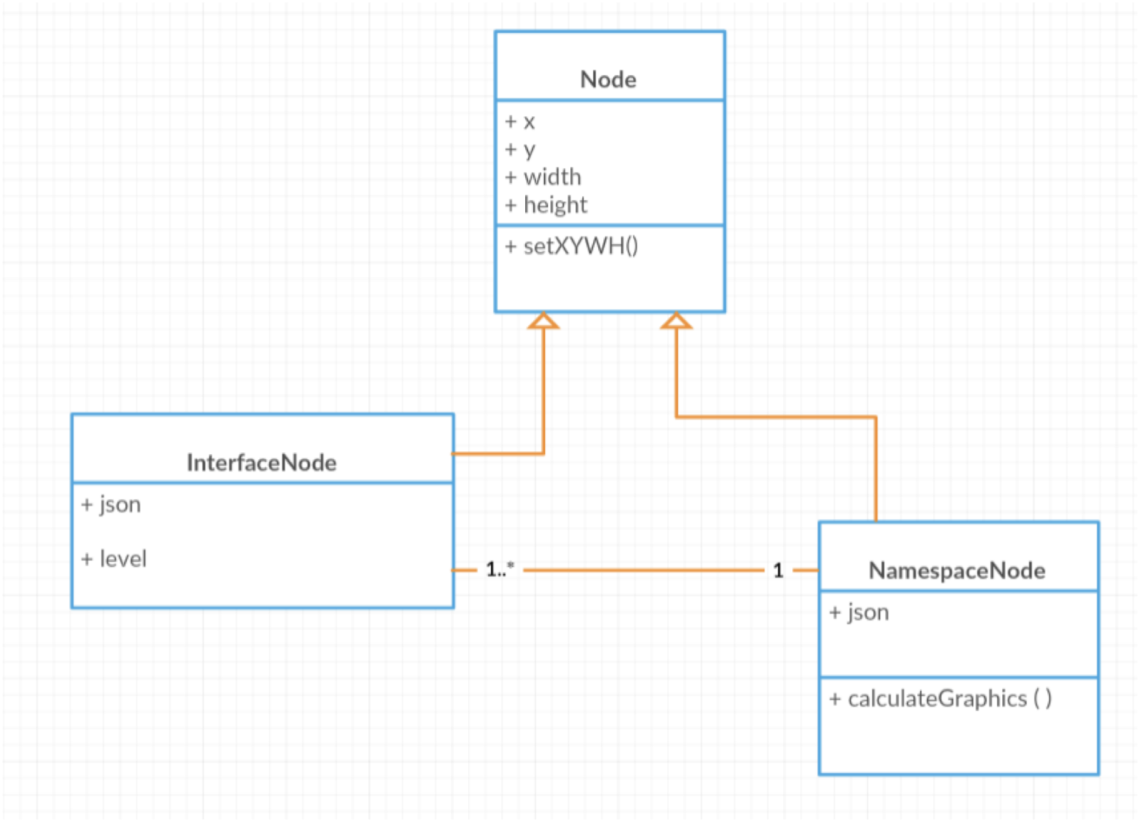


Figure 5.18. Class diagram for objects that will be used by D3

The Node class contains the data about geometrical position and size. The InterfaceNode class extends it by "json" field which is meant to hold all the metadata. Class NamespaceNode contains metadata of the namespace in "json" field and a reference to a set of IntefaceNodes included in the namespace.

Because a JSON input of a specific format is parsed, it is expected that there would be no empty namespaces as well as interfaces without namespaces containing them, therefore the logical relation between NamespaceNode class and InterfaceNode class is strictly “1 to 1...*”.

5.3.3 Getting data from the user

Because plotnetcfg output is generated into a file on user’s computer, acquisition of their data is realised through typical Select file... button powered by Javascript native File API [1].

[1] `https://developer.mozilla.org/en-US/docs/Web/API/File`

Chapter 6 Implementation

The final program is a web page with a file selection input. After the user chooses a JSON file from their computer, its contents are parsed and rendered into an interactive graph visualization (Figure 6.1)

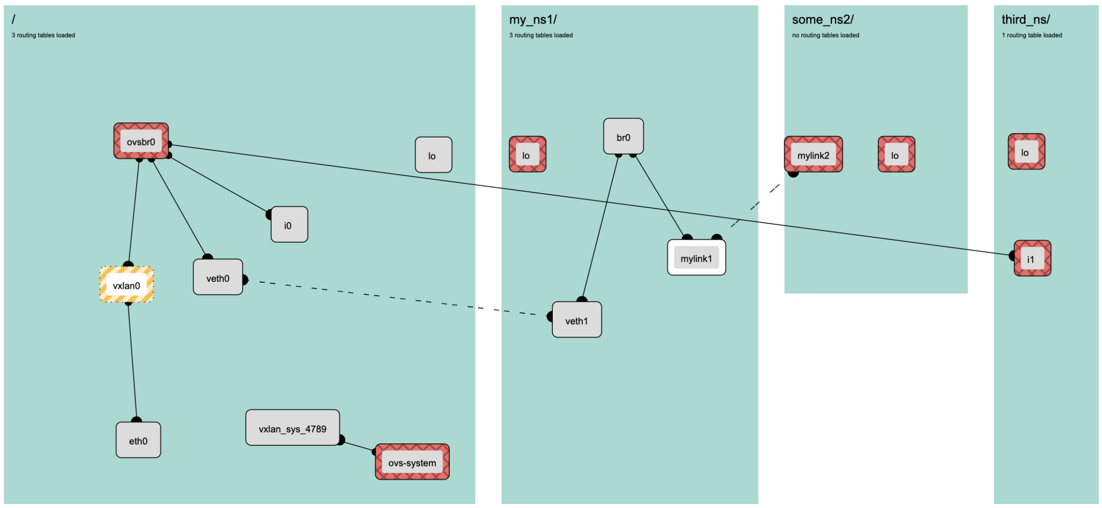


Figure 6.1. The interactive graph visualization.

The resulting force-directed graph:

- contains nodes for interfaces, which are draggable, but could not leave the area of their namespace or overlap over each other;
- displays interfaces' metadata and routing tables of namespaces inside the floating windows;
- is pannable; the panning executes when user drags the plane by the space outside interface nodes;
- is zoomable; the graph scales in response to mouse wheel scrolling or touchpad scale gestures.

The installation process is described in Appendix D.

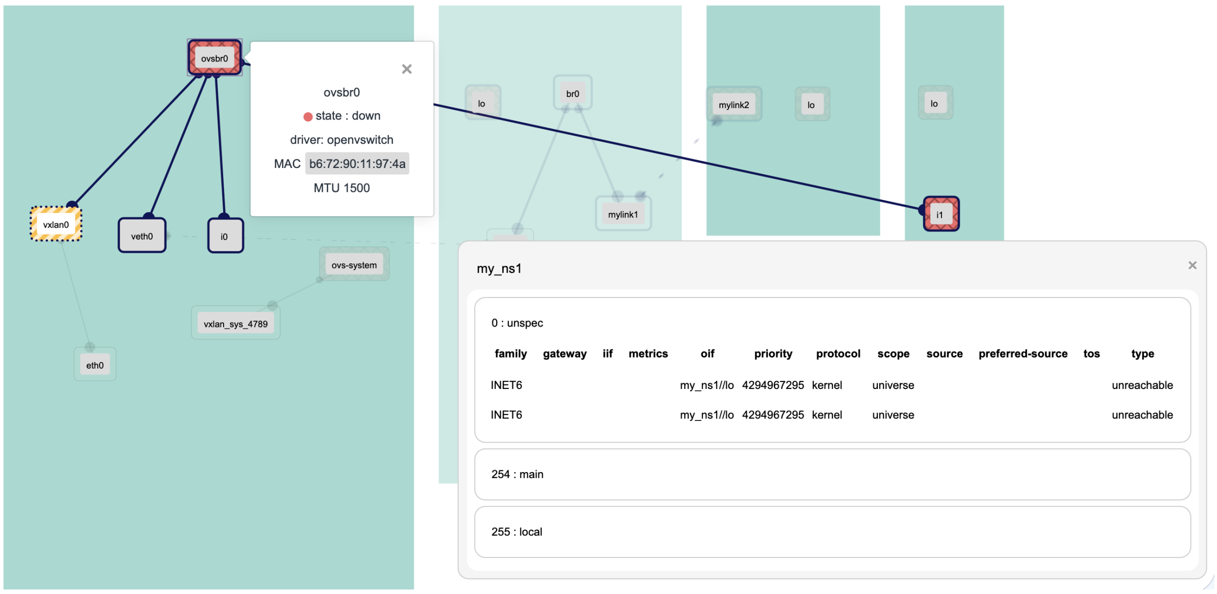
6.1 Analysis of the implementation

As for the requirements stated in the Chapter 2, the final implementation suits all of the following:

- **R1.** *Read input file in a JSON or in a DOT format.* It parses JSON provided by the user.
- **R2.** *Be platform independent and work on most computers without a need of installation of specific software* As a web-page, it is able to run on any of the modern browsers, which exist for most of the platforms. It was tested in Google Chrome,

Opera, Microsoft Edge and Safari. The installation process is simple as well and does not require any additional software to be installed besides any modern browser.

- **R3.** *Work offline.* The program does not communicate with any external sources, so it does not require Internet connection in runtime.
- **R4.** *Draw the network interfaces as nodes and draw the links and relations between them as lines connecting them.* It does, as seen on Figure 6.1.
- **R5.** *Display metadata.* Metadata are displayed inside floating windows (Figure 6.2).
- **R6.** *The data should be presented in a comprehensible arrangement.* The graph structure is implemented according to the design described in Chapter 5, which is based on talks with user mentioned in Chapter 4. It will also be tested from this perspective in Chapter 7.
- **R7.** *Allow changing of placements of the elements.* It is implemented as well. User interaction possibilities are listed above.



ovsbr0

● state : down

driver: openvswitch

MAC b6:72:90:11:97:4a

MTU 1500

my_ns1

0 : unspec

family	gateway	iif	metrics	oif	priority	protocol	scope	source	preferred-source	tos	type
INET6				my_ns1//lo	4294967295	kernel	universe				unreachable
INET6				my_ns1//lo	4294967295	kernel	universe				unreachable

254 : main

255 : local

Figure 6.2. Example of metadata displayed

Thence, the final implementation provides all the necessary functionality. However, it is not possible to assume its usability. Thus, additional testing with real users must be held.

Chapter 7
User testing

This chapter describes the testing process of the final implementation by the users.

7.1 Testing design

A quantitative testing was held, comparing two programs for debugging virtual computer networks. The first one is sosreport [1], and the second is interactive graph visualization designed in the present work. Sosreport is a program that gets configuration information from the computer it is running on. Its output is a set of files to be analyzed by the user. Plotnetcfg works in a similar way. [23]

7.1.1 Target group

Target group are network administrators or network developers. They are expected to have previous experience with sosreport.

7.1.2 Methodology

The programs were evaluated by participants using System Usability Scale. It is a commonly used and reliable method of measuring usability created by John Brooke. Participants were filling in questionnaires containing ten statements each with five response points representing a scale from 'strongly disagree' to 'strongly agree'. The responses are scored and their numerical values are used for further statistical analysis. [24]

In addition to that, every user was asked to share their own opinion on the implemented program. Riccardo Mazza states that even though controlled experiments are helpful in estimation of efficiency and usability, they are less appropriate when it comes to discovering problems that can show up during the observations and for getting information on user's preferences, impressions, and attitudes. According to him, the best way to determine if a certain visual representation is useful for a certain target group is by asking participants explicitly. [25]

7.1.3 System Usability Scale Questionnaires

A test questionnaire consists of 10 statements about participants' experience with evaluated program. Each claim can be answered with one of five options from `strongly disagree` to `strongly agree`. Those are " strongly disagree ", " I don't know ", " I slightly agree ", " I strongly agree".

The statements are listed below.

- Q1 I would like to use this program frequently.
- Q2 I find this program unnecessarily complex.
- Q3 I think this program is easy to use.

[1] `https://github.com/sosreport/sos`

- Q4 I think I would need the support of a technical person to be able to use this program properly.
- Q5 I think the various functions in this program are well integrated.
- Q6 I think there is too much inconsistency in this program.
- Q7 I would imagine that most people would learn to use this program very quickly.
- Q8 I found this program very cumbersome to use.
- Q9 I feel very confident using this program.
- Q10 I needed to learn a lot of things before I could get going with this program. [24]

The actual questionnaires were distributed using Google Forms [1].
Questionnaire for Sosreport: [2]
Questionnaire for interactive graph visualization: [3]

7.2 Testing

During the testing participants were asked to complete two similar debugging tasks using each of the discussed programs and then fill the questionnaires related to each program. They also could gave additional voluntary feedback in form of open text.

The tasks were designed by this work's supervisor Jiri Benc and are enclosed in Appendix A.

7.3 Measuring the results

Every answer from both questionnaires was evaluated according to the Table 7.1. Table 7.2 shows the answers by each participant as well as a final test scores for each of the programs. One of the participants didn't fill the form for sosreport because of time limitations.

Answer	score for odd statements	score for even statements
I strongly disagree	1	5
I slightly disagree	2	4
I don't know	3	3
I slightly agree	4	2
I strongly agree	5	1

Table 7.1. Evaluation for statements from testing questionnaires.

7.3.1 Statistical analysis

Null hypothesis: *"In terms of debugging virtual computer network and understanding it's structure, it is true that sosreport has the same usability as interactive graph tool developed by Koshchii Oleksandra".*

The final score for data mentioned above was evaluated using a paired sample t-test [26] with statistical significance threshold $p \leq 0.05$.

[1] `https://www.google.com/forms/about/`
[2] `https://docs.google.com/forms/d/e/1FAIpQLSfT3QRps9636yCDJ6p4l03rtpZIzB_icRRqMwE9pfLWqqxPTQ/viewform?usp=sf_link`
[3] `https://docs.google.com/forms/d/e/1FAIpQLScKYtyY6POmDthp3TB0qkxzAefhdofQIDoQGNvQD2d4oWoXxw/viewform?usp=sf_link`

Participant	1	2	3	4	5	6	7	8	9	10	sum
sosreport											
1	2	5	2	5	1	3	1	2	4	2	27
2	5	3	3	3	3	5	3	3	5	1	34
3	2	2	2	2	2	5	1	1	2	1	20
4	3	4	2	5	2	2	5	2	4	5	34
interactive graph visualisation											
1	4	4	4	5	4	5	4	5	4	3	42
2	3	5	4	5	4	5	5	2	4	5	42
3	5	5	5	5	4	5	5	5	5	4	48
4	5	4	5	5	5	4	5	4	5	5	47
5	5	5	2	5	2	5	5	2	2	4	37

Table 7.2. The final score distribution.

Independent variables are two selected programs.

Dependent variables are the participants' responses to the questionnaires' statements.

The value of p is 0.0330.

The result is significant because $p < 0.05$. (where p is the probability of the null hypothesis value)

The calculation was performed using the GraphPad program [1]. Full GraphPad output is enclosed in Appendix B.

Therefore, statistical analysis supports the following claim:

"In terms of debugging virtual computer network and understanding it's structure, it is not true that sosreport has the same usability as interactive graph tool developed by Koshchii Oleksandra".

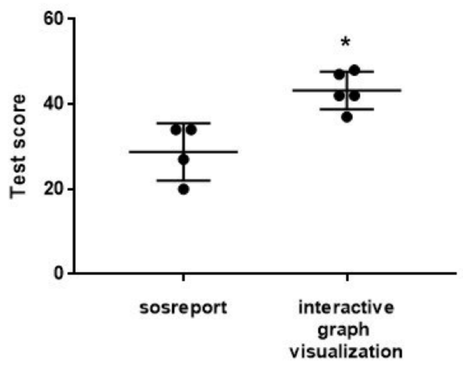


Figure 7.1. Scatter dot plot for test scores

Since average score of graph visualization is higher than that of sosreport (86,4 for the former and 62,4 for the latter), and, also, as seen on Figure 7.1, it is possible to claim that:

"In terms of debugging virtual computer network and understanding it's structure, interactive graph tool developed by Koshchii Oleksandra is more usable than sosreport".

7.3.2 Other feedback

Participants mostly had similar problems and remarks regarding their experience with the interactive graph visualisation. Overall, according to them, their feelings about

[1] `https://www.graphpad.com/`

the program were positive. One of the Participants stated that, with interactive graph visualisation, they finished their task two times faster than with sosreport.

This is a complete list of issues faced by users during the testing. Each statement is followed by a number of users who pointed it out.

- "search" field - when user entered an IP address, the interface wasn't found, unless they added a netmask to it (5/5)
- program lacked horizontal scroll bar (5/5)
- Open vSwitch rules were not implemented yet (5/5)
- when users tried to build a path for data flow, only a part of it could be highlighted; it would be better, if there was a possibility to expand highlighted selection (4/5)
- program used a lot of CPU performance during rendering (3/5)
- "search" field - users expected to be able to search with regular expressions they are used to (2/5)
- the window with routing tables had fixed position and couldn't be folded - sometimes it overlapped important content (2/5)
- as an alternative to horizontal scroll bar, users expected to use a "teleport" function, which would pan the whole graph from one endpoint to another after user would click on a circular end mark (2/5)
- there was no distinctive visual difference between bridges, bonds a Open vSwitches, which could benefit to overall understanding of network structure (2/5)
- if namespaces had different order, in some cases, graph could have had cleaner structure and be more comprehensible (1/5)

Further analysis of these issues is beyond the scope of this thesis.

7.4 Conclusions of the testing

Considering the results of statistical analysis mentioned in Section 7.3.1 and users' feedback, it is possible to claim that the developed program succeeds as a tool for debugging virtual computer networks.

Chapter 8
Conclusion

This document described the process of design, implementation and testing of the interactive graph visualization depicting connections and structure of virtual computer networks.

It started with a basic analysis and discussion of other available solutions and approaches to network visualisation, which led to a necessity to develop and implement a new solution. A user research was conducted, focusing on what visual and interactive structure the users would prefer to see in the developing visualization. Based of this data, an interactive visual design was created and a description of used technology and assembled architecture is provided in this thesis. The final implementation was analyzed according to initial requirements and tested by users.

The final user testing compared the interactive graph tool with a program that is commonly used in the field. The testing had proven that in terms of debugging virtual computer network and understanding it's structure the interactive graph tool is more usable than its counterpart.

To summarise, it is safe to claim that even though the final implementation has some issues and can be improved, it succeeds as a tool for virtual computer network debugging and fulfills all given requirements.

References

[1] Jiří Benc. *plotnetcfg, Tool to visualize network config.* `https://github.com/jbenc/plotnetcfg`. visited on 2019-5-16.

[2] *Graphviz - Graph Visualization Software.* `https://www.graphviz.org`. visited on 2019-5-16.

[3] *Gephi, The Open Graph Viz Platform.* `https://gephi.org/`. visited on 2019-5-16.

[4] M. Tim Jones. *Virtual networking in Linux.* `https://help.ubnt.com/hc/en-us/articles/115005778547-Intro-to-Networking-Virtual-Private-Networks-Tunneling`. visited on 2019-5-21.

[5] *The DOT Language.* `https://www.graphviz.org/doc/info/lang.html`. visited on 2019-5-20.

[6] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice.* 2 edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0-201-19930-0.

[7] Noah Iliinsky Julie Steele. *Designing Data Visualizations*. O'Reilly Media, Inc., 2011. ISBN 9781449314774.

[8] Hangbin Liu. Red Hat Developer. *Introduction to Linux interfaces for virtual networking*. 2018. `https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking`. visited 2019-5-21.

[9] A Linux Foundation Collaborative Project. *Bonding*. 2016. `http://docs.openvswitch.org/en/latest/topics/bonding/`. visited 2019-5-21.

[10] A Linux Foundation Collaborative Project. *What Is Open vSwitch?* 2016. `http://docs.openvswitch.org/en/latest/intro/what-is-ovs/`. visited 2019-5-21.

[11] Ciro da Silva da Costa. *Using network namespaces and a virtual switch to isolate servers*. 2018. `https://ops.tips/blog/using-network-namespaces-and-bridge-to-isolate-servers/`. visited 2019-5-21.

[12] Inc. Ubiquiti Networks. *Intro to Networking - Virtual Private Networks and Tunneling.* `https://www.ibm.com/developerworks/linux/library/l-virtual-networking/`. visited on 2019-5-21.

[13] J. Rubin. *Handbook of Usability Testing*. Wiley, 1994. ISBN 0471594032. `https://dl.acm.org/citation.cfm?id=561768`.

[14] J. Nielsen. *Enhancing the explanatory power of usability heuristics*. Boston, MA: Proceeding, CHI '94 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1994. ISBN 0-89791-650-6. `https://dl.acm.org/citation.cfm?id=191729`.

[15] Graeme Hirst Saif Mohammad. *Measuring Semantic Distance using Distributional Profiles of Concepts.* `http://saifmohammad.com/WebDocs/Measuring-Semantic-Distance.pdf`. visited on 2019-5-16.

[16] Justin Rey Reyna. *Here's What You Need to Know About Color Accessibility in Product Design.* `https://uxplanet.org/heres-what-you-need-to-know-about-color-accessibility-in-product-design-aecbd0c30628`. visited on 2019-5-16.

[17] *Proofing colors.* Adobe Photoshop User Guide `https://helpx.adobe.com/photoshop/using/proofing-colors.html`.

[18] The Interaction Design Foundation. *What is Aesthetics?* `https://www.interaction-design.org/literature/topics/aesthetics`. visited on 2019-5-16.

[19] Don Norman. *Emotional Design: Why We Love (or Hate) Everyday Things.* Basic Books; 1 edition, 2005. ISBN 978-0465051366. `https://www.nngroup.com/books/emotional-design/`.

[20] Scott Murray. *Interactive Data Visualization for the Web, 2nd Edition.* O'Reilly Media, 2017. ISBN 9781491921296. `https://www.oreilly.com/library/view/interactive-data-visualization/9781491921296`.

[21] Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Phys. Rev.*. 1967, 159 DOI 10.1103/PhysRev.159.98.

[22] *d3-force.* `https://github.com/d3/d3-force`. visited on 2019-5-16.

[23] Bryn Reeves. *SoS.* `https://sos.readthedocs.io/en/latest/index.html`. visited on 2019-5-19.

[24] John Brooke. *SUS-A quick and dirty usability scale (in "Usability Evaluation in Industry", PW Jordan, B Thomas, I McLelland, BA Weerdmeester (eds)).* London: Taylor and Francis, 1996. ISBN 978-0748404605.

[25] Riccardo Mazza. *Introduction to Information Visualization.* Springer Publishing Company, Incorporated , 2009 . ISBN ISBN 878-1-84800-219-7. `https://dl.acm.org/citation.cfm?id=1529936`.

[26] M. Navara. *Pravděpodobnost a matematická statistika: Skriptum FEL ČVUT,.* Praha: 2007. `http://cmp.felk.cvut.cz/~navara/pms/`.

Appendix A
Tasks

These tasks were developed by this thesis's supervisor. They were used during the final usability testing mentioned in Chapter 7.

- Task 1:
  In the `vm-night` container, a `ping 192.168.147.183` command is executed. Through what chain of interfaces does the traffic flow?
  In the same `vm-night` container, a `ping 198.145.29.83` command is executed. Through what chain of interfaces does the traffic flow now?
- Task 2:
  In the `vm-party` container, a `ping 192.168.100.133` command is executed. Through what chain of interfaces does the traffic flow?
  In the same `vm-night` container, a `ping 195.113.144.230` command is executed. Through what chain of interfaces does the traffic flow now?

Appendix B
GraphPad Output

```
Table Analyzed Data 1
   Column B interactive graph visualization vs. Column A sosreport
   Paired t test
   P value 0.0330
   P value summary *
   Significantly different (P ¡ 0.05)? Yes
   One- or two-tailed P value? Two-tailed
   t, df t=3.754 df=3
   Number of pairs 4
   How big is the difference?
   Mean of differences 16
   SD of differences 8.524
   SEM of differences 4.262
   95% confidence interval 2.436 to 29.56
   R squared (partial eta squared) 0.8245
   How effective was the pairing?
   Correlation coefficient (r) -0.4078
   P value (one tailed) 0.2961
   P value summary ns
   Was the pairing significantly effective? No
```

Appendix C
Abbreviations

Here is a list of abbreviations used in this thesis.

JSON	JavaScript Object Notation
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SVG	Scalable Vector Graphics
ES	ECMAScript
D3	Data Driven Documents
VLAN	Virtual Local Area Network

Appendix D
Installation and running

These steps must be executed to install and run the latest version of the tool.

1) Clone or download the repository: `https://github.com/sashkoboom/redhat_network_visualization`
2) Go to `redhat_network_visualization/run` and open "index.html" in your browser.
3) To launch the algorithm, push "Select file..." button and select one of the .json files located in `redhat_network_visualization/samples`. This folder contains some examples of the plotnetcfg JSON output.