M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Tanárki József

PREDICTIVE ANALYTICS OF IP/MPLS NETWORKS

KONZULENS

Dr. Huszák Árpád

BUDAPEST, 2017

Table of contents

Összefoglaló

A 2000-es évek végén megjelent és mára szinte mindenki által használt okostelefonok nem csak a társadalmi berendezkedésünket és az internetezési szokásainkat változtatták meg. Az „always online” életmód a felhasználókon kívül a szolgáltatói szektorokra is nagy hatást gyakorolt. Ma már nem csak eszközökre, hanem komplett szolgáltatásokra van igényünk, amelyek a szolgáltatói környezetben mindennél előbbre helyezték a felhasználói élményt.

Az internet eléréssel rendelkező eszközök száma folyamatosan nő és ez a trend úgy tűnik, hogy még jó néhány évig nem változik. Az IPv4 protokoll nyújtotta hálózati címzés határait már elértük, az IPv6 protokollt azonban globálisan még nem igazán használjuk. A megnövekedett ügyfélszámnak köszönhetően az Internet szolgáltatók hálózata is rendkívül gyors és komplex fejlődésen megy át, így a bekövetkező hibák lehetősége is jelentősen növekszik. Emberi erőforrásokkal a bekövetkező hibák az esetek többségében csak reaktív módon kezelhetők, így ha csak rövid időre is, de sokszor elkerülhetetlen, hogy a felhasználó ezeket ne vegye észre.

A dolgozatom célja olyan adatelemzési módszer alkalmazása a hálózatokra, amelyekkel gépi tanulási algoritmusok segítségével képesek lehetünk előre jelezni a hálózatban valószínűsíthetően kialakuló hibákat, eseményeket. A módszer a tradícionális hálózatmonitorozási protokollok adatait használja, így szinte bármilyen jelenlegi IP/MPLS, de akár teljesen más protokollokat használó hálózatokra is alkalmazható.

A fent megnevezett módszert egy egyszerűsített Internet szolgáltatói hálózati modellre alkalmaztam, amelyet virtuálisan, egy számítógép erőforrásait használva valósítottam meg. Dolgozatomban részletesen bemutatom a hálózat szimulációra használt eszközöket, alkalmazásokat és protokollokat, majd ismertetem a gépi tanulás alapjait, a legfontosabb gépi tanulási modelleket és algoritmusokat. Végül bemutatom, hogyan alkalmazhatóak különböző regressziós algoritmusok az IP cím kihasználtság prediktív analízisére.

Abstract

At the end of 2000s smartphones were introduced and so far they have been used by most of the people around the world. These devices have had radical effects on our social structure and the usage of the Internet. Being "always online" has changed the service providers from every aspect as well. Today, we are not just buying new smart equipments, we need complete services, which shifted the focus on the customer experience with the highest priority in a service provider environment.

The number of customers, who are reaching the internet, is increasing continuously and it seem that this trend won't change in the next couple of years. We have reached the limits of the IPv4 Addressing scheme, but IPv6 is still not used globally. Due to the increased number of customers, Internet Service Providers have to develop their networks in a really fast and that's why complex way. In this kind of environment the possibility of the occurring failures are so high, that these can be managed by humans mostly reactively, and there is always a chance of the customer experience degradation can happen, even for a short time period.

The aim of my dissertation is applying data analysis for networks, with which we can use machine learning algorithms to predict the possibility of failures or of the occurring events. The method uses data of traditional network management protocols and that's why it is suitable for any kind of IP/MPLS networks, or for networks using any other protocols.

I applied this method above for a simplified model of a typical Internet service provider's network, which was realized by me in a virtual environment using the resources of a personal computer. In my document I give detailed description of the equipments, applications and protocols used for network simulation, then I expound the basics of machine learning, the most important models and machine learning algorithms. In the end, I explain how different kinds of regression algorithms can be used for the prediction of IP address utilization.

1 Introduction

Network management and operation is quite a hot topic today. Network operators have the proper management systems, departments and processes to monitor and maintain todays' fast growing next generation networks. Network devices produce an enormous volume of data, which is filtered and visualized in the network operation centres.

However the growing of customer needs leads to fast implementation and in most of the cases, documentations cannot cover everything which was deployed. The entropy of the networks strictly grows, resulting complexity and unmanageability. We reached a point which cannot be handled by humans without proper machine-based support.

Networking engineers and experts are using self-developed scripts and tools to automate the hard and repetitive work, they have the deep knowledge of how the network behave, but this requires several years of experience in an Internet Service Provider network.

The overall assumption is that this complexity can be controlled by machine learning techniques and the industry has to focus on developing machine learning based networking solutions to increase the customer experience and satisfaction.

In the following chapters I go through the protocols which are necessary to build up an ISP network. I give a summarization of machine learning basics and algorithms. Then I show my basic ISP model which I implemented virtually and give a detailed description of how I refined the model step by step solving all the issues came up. The problems I met were based mostly on resource limitations, that is why I had to restart the network implementation process several times. Finally, I was able to establish a simple working model, which is still complex enough to show my results about applying the previously mentioned machine learning algorithms on a big amount of data produced by my network. With all this knowledge, I can predict the right values, which can describe the behaviour of the subscribers and the network also.

On the following overview sections I tried to focus on the most important key-points of the topics, because the literature seems to be endless, when we are talking about IP networks.

2 Overview of IP/MPLS Networks

2.1 Internet Protocol

IP is the network layer protocol in TCP/IP described in RFC791 [1]. TCP/IP means the Internet protocol suite, a conceptual model and a set of communication protocols used in interconnected systems of packet-switched computer networks.

IP is responsible for the logical addressing and information for routing packets throughout the network. Packets or datagrams are blocks of data, which are transported via the network from a source to a destination.

The sources and destinations are hosts (workstations, servers etc.) or routers and these are identified by 32-bit IPv4 or 128-bit IPv6 addresses. IP also provides fragmentation and reassembly of long datagrams for transporting over the network with small maximum transition units (MTU).

2.1.1 IP communication types

IP uses the following types of addressing methods for communication between the network entities:

Unicast addressing represents a *one-to-one* association between a source and a destination. Each address uniquely identifies the endpoints.

Broadcast addressing represents a *one-to-all* association in a Layer 2 segment (OSI-model), which means a network subnet.

Multicast addressing represents a *one-to-many-of-many* or *many-to-many-of-many* association. Packets are routed simultaneously in a single transmission to many recipients.

Anycast addressing represents a *one-to-one-of-many* association where packets are routed to any single member of a group of potential receivers. In this group all the members are identified by the same destination address.

Geocast addressing refers to the delivery of information to a group of destinations in a network identified by their geographical locations. This is a special form of multicast addressing used by some routing protocols for mobile ad-hoc networks. (IPv6 only)

2.1.2 IP address-assignment strategies

The simplest way of obtaining an IP address to an interface of a networking node is to configure it manually.

The IPv6 Neighbor Discovery (ND) protocol provides a much simpler feature, which can be used by the hosts to determine their full IPv6 addresses without the help of DHCP. This kind of addressing method is the Stateless Address Autoconfiguration (SLAAC) and described in RFC4862 [2].

Beside the static configuration, there is another method to assign IP address to an interface in a stateful way. Dynamic Host Configuration protocol (DHCP) is described in RFC2131 [3] for IPv4 and RFC3315 [4]. Using DHCP we can provide further functions or options to the hosts, a detailed description can be found in the following section about them.

2.2 Dynamic Host Configuration Protocol

DHCP provides configuration parameters to the hosts on the network. DHCP is not just a protocol, which is delivering host-specific configuration parameters from a DHCP server to a host, but it describes the mechanism of allocating addresses to the hosts.

DHCP uses a *client-server* model, where designated DHCP servers allocate network addresses and deliver configuration parameters to the hosts. These hosts are the clients in the model and are dynamically configured to receive the parameters from the server.

2.2.1 Address allocation mechanisms

There are three mechanisms that DHCP supports for IP address allocation. In *automatic allocation* DHCP assigns a permanent IP address to the client. The DHCP server can assign IP addresses for a limited period of time, this mechanism is the *dynamic allocation*. Sometimes it can happen, that a dynamically configured host needs the same address every time, when it tries to connect to the network. In this case a *manual allocation* is needed, and the network administrator can configure these addresses on the DHCP server to be associated by the special clients.

2.2.2 Client-server interaction – allocating a network address

The following DHCP messages are exchanged in the simplest case of typical client-server interaction.

1.) The client broadcasts a DHCPDISCOVER message on its local physical subnet. This message may include some options that suggest values for the IP address and lease duration. DHCP relay agents can pass the message towards the DHCP servers in another subnet.

2.) Each server may respond with a DHCPOFFER message. This message includes an offered available and unused network address and some other configuration parameters in DHCP options (e.g.: DNS, Default Gateway etc.)

3.) The client sends DHCPREQUEST messages back to the servers (if there is more than one). In this message the client can request the offered parameters from one server and can implicitly decline the offers from the others. It can confirm the correctness of previously allocated address after, or extend the the lease on a particular network address.

4.) DHCPACK message is sent back from the server with the client configuration parameters, with the committed network address.

It is worth to mention, that all of these messages above are broadcasted throughout the Layer 2 segment.

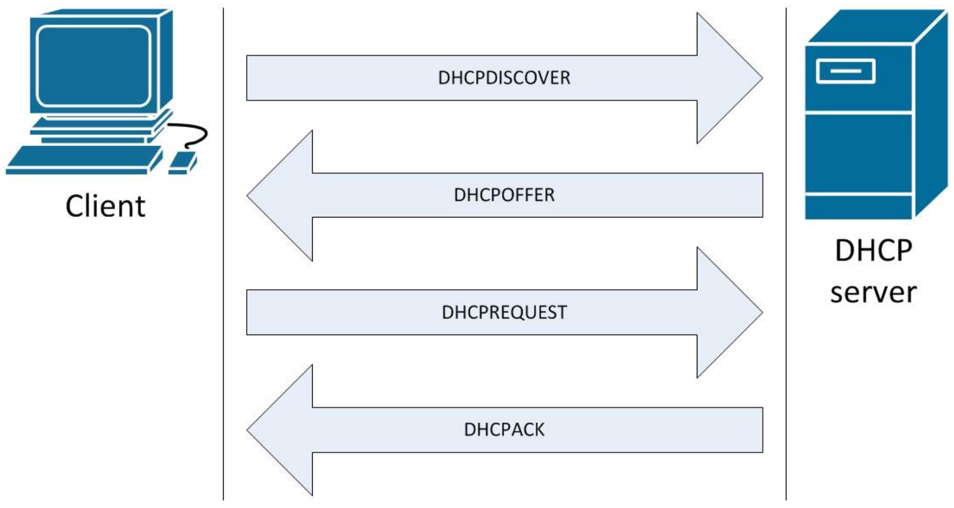


Figure 2.1.: Simple DHCP communication

2.3 Routing Protocols

In this section I briefly describe the most important protocols and techniques to deploy the proper routing-forwarding architecture used in today's IP/MPLS networks. Routing protocols can be categorized as distance-vector, link-state, hybrid or path-vector and as a hierarchical or flat.

2.3.1 Static Versus Dynamic Routing

Static routes are manually configured on a router. Due to this manual configuration they cannot react to network outages, unless the static route specifies the outbound interface. In this case, if this interface goes down, the static route is removed from the routing table of the router.

Routing protocols (RPs) use algorithms to dynamically determine the best route to a destination. When the network topology changes RP adjusts the routes automatically. Every RP calculates the best path toward to a destination with the usage of metrics. The following dynamic routing protocols exist: RIPv1, RIPv2, IGRP, EIGRP, OSPF, IS-IS, RIPng, OSPFv3, EIGRP for IPv6, BGP.

2.3.2 Interior Versus Exterior RPs

RPs can be divided into two categories: interior gateway protocols (IGP) and exterior gateway protocols (EGP). IGPs are used within an organisation's administrative domain. EGP is used to communicate with exterior domains, where routing information can be changed between the administrative domains. Today BGP (Border Gateway Protocol) is the de facto EGP.

2.3.3 Hierarchical Versus Flat RPs

Some RPs are defined to have a backbone network. This backbone network and the internetwork form two levels of hierarchy, which is sufficient to provide scalability. The backbone consists of only some devices, the so called backbone routers. These devices service and coordinate the routes and traffic towards the non-local internetwork routers. OSPF and IS-IS are hierarchical routing protocols.

Flat RPs cannot be organized in a hierarchical way. In a flat topology every router forms a peering relationship with the others and no router have a special role. Flat RPs are EIGRP, RIPv1 and RIPv2.

2.3.4 OSPF

OSPFv2 is defined in RFC2328 [5] and is used in IPv4 networks. OSPF is a link-state routing protocol that uses Dijkstra's shortest path first algorithm to calculate the paths towards the destinations. OSPFv3 is designed to support only IPv6, but now it can support IPv4 as well. The support for multiple address families is defined in RFC5838 [6].

OSPF was created to support large networks, where RIP (the first dynamic routing protocol) was not able to handle such a big number of peers. The introduction of OSPF improved the speed of convergence and it provides the use of variable-length subnet masks (VLSM). Path calculation using the SPF algorithm improved also.

In OSPF, each router sends link-state advertisements (LSAs) about itself and its connections to other networks in a networking area. Based on the received LSAs, each router is able to calculate the best routes to a destination by running the SPF algorithm. Each OSPF router has the identical database which is describing the topology within the area. OSPF uses multicast addresses to communicate between the routers and uses IP protocol 89.

In a typical Internet Service Provider (ISP) network OSPF is used as the IGP.

2.3.5 Border Gateway Protocol

BGPv4 is described in RFC1771 [7]. BGP is an interdomain routing protocol and is used to exchange routing information (e.g.: network-reachability) between autonomous systems (AS). BGP is a path-vector protocol, where a path stands from the sequence of AS numbers (ASN). An ASN represents a whole AS and uniquely identifies the system. BGP node uses the TCP port 179 for communicating its peers. BGP also can be used inside large enterprise networks as internal BGP (iBGP) to form adjacency above the existing Layer 3 network. Using an IGP (e.g.: OSPF) it is easy to define BGP neighborships only on the edge of the network and let the IGP to handle the forwarding inside the network between the BGP peers.

With route redistribution BGP can learn all the connected routes, manually configured static routes and IGP routes and will be able to redistribute this information between the BGP peers.

Using route filtering and default route origination we can influence the view of the network of a BGP peer, reducing the resource intensity of path calculation and we can spare some bandwidth as well. BGP tables can be really big and in a well designed enterprise network it is not always necessary to distribute the full view of the Internet and Intranet between peers.

2.4 Multiprotocol Label Switching (MPLS)

2.4.1 Limitations of traditional IP routing and forwarding

In traditional IP packet forwarding the router analyzes the destination IP address in the network layer header of each packet. This happens independently at each hop in the network. Dynamic RPs or static configuration builds the database needed to analyze the destination IP addresses. That is what we call the routing table. Traditional IP routing is also called *hop-by-hop destination based unicast routing*.

Unfortunately Layer 2 switches do not have the capability to handle Layer 3 routing information or to influence the path selection by analyzing the network layer header information.

2.4.2 The MPLS architecture

RFC3031 [8] describes the MPLS architecture. MPLS combines the benefits of Layer 2 packet forwarding and Layer 3 routing. MPLS assigns labels to packets for transporting across a packet- or cell-based network. *Label swapping* is the forwarding technique, which means that the packet is *tagged* with a fixed-length label. This label tells the switching nodes, how to process and forward the data.

The MPLS architecture contains two separate components: the *data plane* and the *control plane*. Data plane is the so called forwarding component that uses a label-forwarding database. The control plane is responsible for creating the label-forwarding information (also called as *bindings*) between the interconnected switches. Figure 2.2 shows the basic architecture of IP routing with an MPLS capable node.

The MPLS capable router is similar to traditional IP routers, when it populates the IP routing table with IP routing protocols. This IP routing table is used to build up the IP forwarding cache or table, the so called *Forwarding Information Base* (FIB).

The MPLS node uses the IP routing table to determine the label binding exchange. In this case, the adjacent MPLS nodes exchange labels for individual subnets. These subnets are stored within the IP routing table. The label binding exchange is performed by the usage of the IETF-specified *Label Distribution Protocol* (LDP) or other vendor specific proprietary protocols (e.g.: Cisco TDP).

Based on the exchanged labels among MPLS nodes, the nodes build up the *Label Forwarding Table* and this table is used by the data plane to forward the labelled packets through the MPLS network.

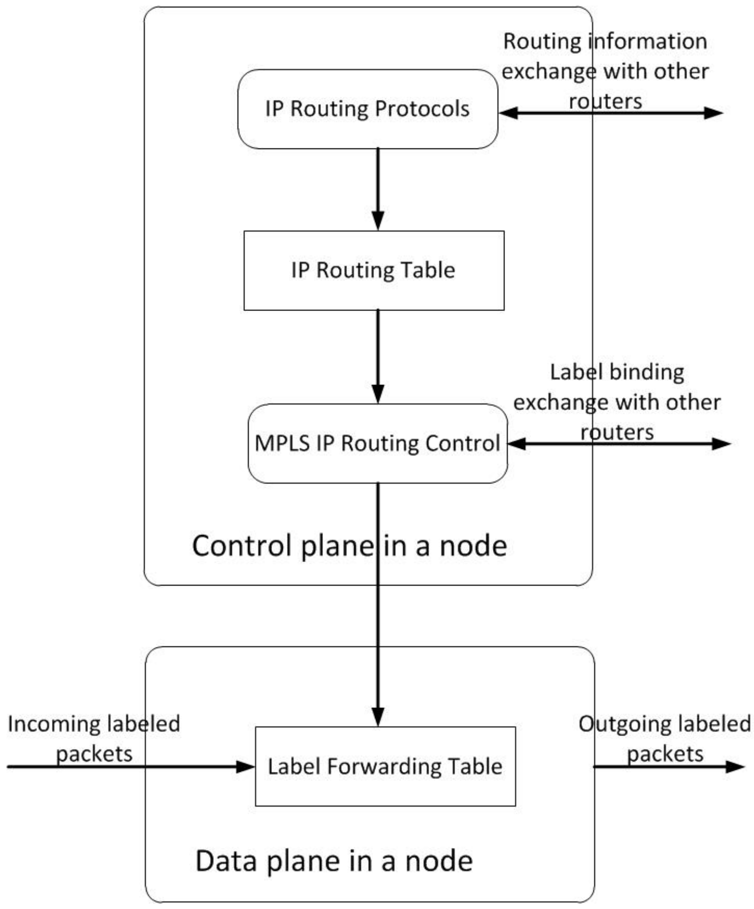


Figure 2.2.: MPLS node basic architecture

2.4.3 Label Switch Router

Any router or switch that performs label distribution and can forward packets based on labels is a *Label Switch Router* (LSR). Several different types of LSR exist. These nodes are differentiated by the functionalities they provide in an MPLS network (e.g.: Edge-LSR, ATM-LSR, ATM edge-LSR). The most important type to discuss is the

Edge-LSR. Edge-LSR can perform label imposition and deposition (*push* and *pop* action). Push action means prepending of labels (a label, or a stack of labels) to a packet in the ingress point. This ingress point is the edge of the MPLS network, that is where the Edge-LSR name is come from. On the egress point, where the packet leaves the MPLS network, the pop action is performed.

Each packet enters the MPLS network at an ingress LSR and leaves it at an egress LSR. This mechanism creates a *Label Switched Path.* Because LSRs are unidirectional, the direction of communication between the devices connected an MPLS network can be different.

2.5 Network Management – Operating IP/MPLS networks

After a proper network design and when the routers and switches are implemented and configured, the network "starts to live". To ensure that the network operates properly, the network operators need to gather operating statistics and in case of a failure or development, they need to manage the devices correctly. The gathered statistics can be information of bandwidth utilization, CPU and memory utilization of the devices, interface counters etc. Configuration changes can be made using a network management tool or simply via the *Command Line Interface* (CLI). For the proper network management ISO defines five type of processes. These processes also known as FCAPS:

- **Fault management:** detecting and correcting network fault problems
- **Configuration management:** scheduling, modifying and tracking configuration changes
- **Accounting management:** keeping track of subscriber circuits for billing services
- **Performance management:** measuring the effectiveness of the network at delivering packets
- **Security management:** tracking the authentication and authorization information

To support network management, we need to identify some elements to perform this task:

- **NMS:** Network Management System. A system runs the applications to manage and monitor the networking devices.
- **Network management protocols and standards:** These are used to collect and exchange necessary information between the NMS and the managed devices. Some examples: SNMP, MIB, Syslog, NetFlow, NETConf/YANG [9] etc.
- **Managed devices:** networking devices managed by the NMS
- **Management agents:** per protocol entities in the managed devices, like SNMP agents.

2.5.1 Simple Network Management Protocol (SNMP)

SNMP is an IP application layer protocol that has been the standard for exchanging management information between network devices. RFC1157 [10] describes SNMP, which gives a really simple solution and requires a really short code to implement, when vendors want to build SNMP agents on their products.

SNMP uses *User Datagram Protocol* (UDP), and therefore it does not require any connection oriented datagram connections that *Transmission Control Protocol* (TCP) could provide. Using UDP, the management traffic does not add a big overhead to the network.

SNMP components are:

- The managed devices
- The agent that runs on the managed device
- The NMS

A managed device can be any of the networking devices (routers, switches) or even servers. The devices collect and store the management information and can send the collected data to the NMS. To exchange SNMP information, SNMP community strings (password) can be used.

The SNMP agents gather the measured data by the devices and store them in SNMP format. They can respond to a manager request (with correct community) or can send SNMP traps if an event occurs. The NMS can *poll* the SNMP agents for data regularly and it is able to store, analyze and display the polled data.

Different privilege levels can be configured for each SNMP community. However SNMP allows “write” privileges as well, it is not really used by network operators, because SNMP does not provide any commitment about the successful write event. If the NMS can access to the CLI and have the proper scripts to run and check the configured parameters, configuring via SNMP can be possible. But if the NMS has CLI access, it is easier to configure the devices with scripts and then check the states with SNMP or with other CLI scripts. In the NETConf/YANG model this check is a built in basic function, that is why the industry is turning to change the SNMP based management to NETConf/YANG. There is another tool, which is Ansible [11]. Ansible is a powerful tool for automated CLI configuration.

2.5.2 Management Information Base (MIB)

A MIB is a collection of information stored on the local agent of the managed device. The organization of MIBs are hierarchical and can be accessed by the NMS. This hierarchical organization means a database, where objects are organized in a tree-like structure. Each object has a unique *object identifier* (OID), which represents the object in a dotted decimal format. RFC1213 [12] describes the TCP/IP MIBs. Some of these MIBs were used to collect data in my measurements as well.

2.5.3 Other Technologies for Network Management

NetFlow allows tracking of IP flows. IP flows are a set of specific IP packets passed through a router within a specific timeslot, which share the same source or destination address, port numbers, type of service or protocol number. NetFlow information is forwarded to a network data analyzer, or accounting and billing applications. The most used version is NetFlow Version 9 described in RFC3954 [13].

The *Syslog protocol* is described in RFC3164 [14]. Syslog can transmit event notification messages over the network. The network devices can generate and send syslog messages to an event server. Syslog operates over UDP similar to SNMP. Syslog messages are generated in many areas, mostly on a per protocol bases. A Syslog message can be generated in several severity levels from 0-7, where 0 means the highest severity. If a device generates a severity 0 message, it means an emergency event, which lead the system to an unusable state. Syslog is widely used in network troubleshooting and root cause analysis, because it describes the states of the system well, and the messages are organized on a detailed timescale.

3 Introduction to Machine Learning

The following section is based on the book of Sebastian Raschka titled Python Machine Learning [15]. During my search in literature, this book gave me the best overview to understand and use Machine Learning algorithms and I was quite happy when I found good examples for Regression Analysis and I quickly realized that how easy is to use these algorithms to examine the measured data of IP/MPLS networks.

Because ML and predictive analytics are so big topics, I try to concentrate for only regression in this section.

3.1 Machine Learning basics

The three different types of machine learning are: *supervised learning, unsupervised learning* and *reinforcement learning*. With supervised learning, we are able to make predictions about the future.

3.1.1 Supervised learning

With supervised learning the machines can learn a model from labelled *training data*, and this can give us the ability to make future predictions about the unseen data. The term *supervised* refers to a set of samples, where we already know the desired output.

There are two subcategories of supervised learning, *classification* and *regression*. In case of classification we can train a model using the supervised machine learning algorithm on certain properties of the mostly same kind of data (e.g.: e-mails), and if the samples are already marked with some *class labels*, the model can make decision about what type of class the data is belonging from further unseen data samples. Spam-filtering is working this way. *Regression* is the other subcategory, where the outcome signal is a continuous value.

3.1.2 Classification for predicting class labels

As I discuss it above, classification is used for predicting categorical class labels of new data samples or instances based on past observations. These class labels are discrete, unordered values and based on the *group membership* of the instances. *Binary classification* is about deciding on that an instance is a part of the group or not. These are two possible classes.

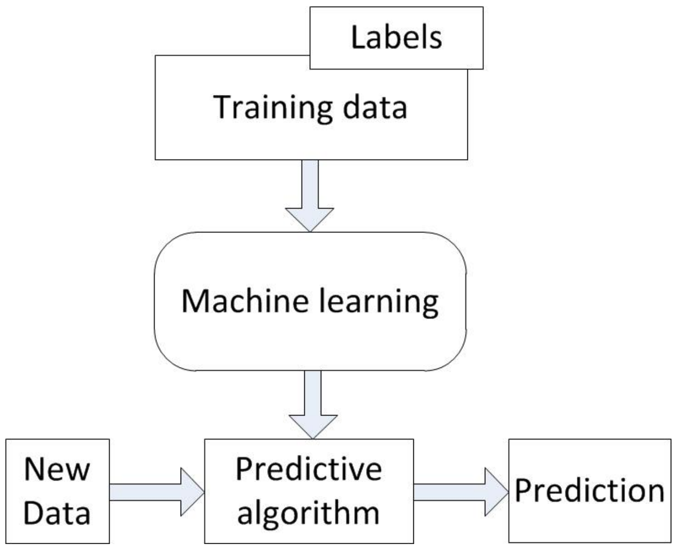


Figure 3.1.: Supervised learning

The class labels do not have to be binary, with *multi-class classification*, supervised learning can assign any labels to the new instances if they were presented in the training set.

3.1.3 Regression for predicting continuous outcomes

Regression analysis is the second type of supervised learning and is used to predict continuous outcomes. In this type of analysis, we have a given number *predictor* or explanatory variables and continuous response variable or the outcome. The task is to find the relationship between these variables and after that we are able to predict the outcome of the unseen input data.

3.1.4 Building machine learning systems

The figure below shows the right process for predictive modelling. Many machine learning algorithms require to transform the data of selected features to the same scale. This transform in most of the cases is ending on the range [0-1] or a standard normal distribution with zero mean and unit variance. If the selected features are highly correlated and redundant, dimensionality reduction may be required to run the ML algorithm faster.

To check, that our machine learning algorithm performs well not just on the training dataset, we often randomly divide the dataset into a separate training and test set. With the test set we can evaluate our final model, to see that it will perform well with unseen data.

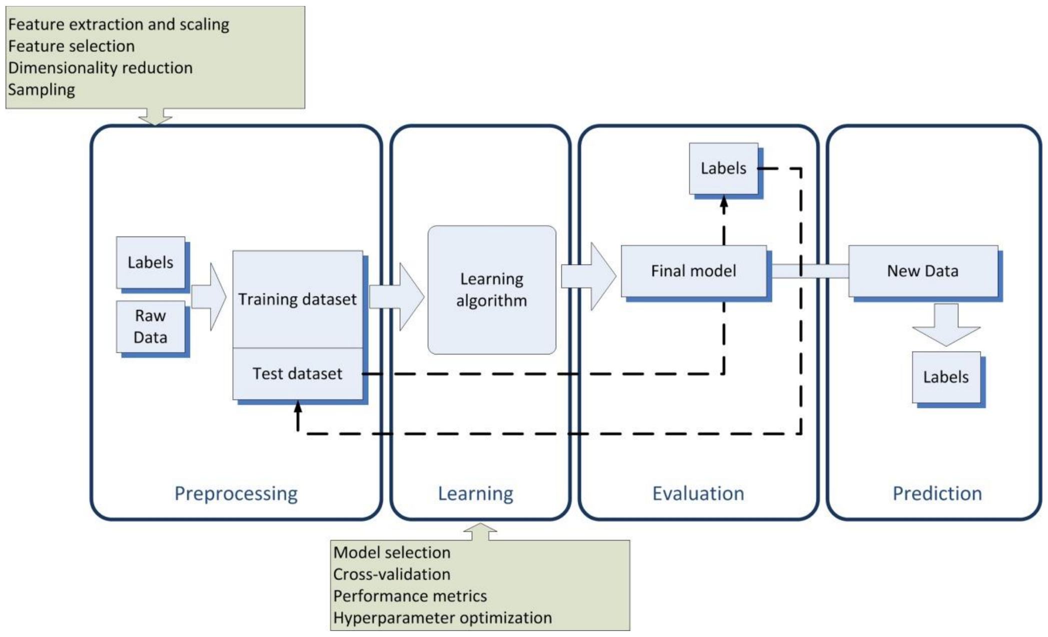


Figure 3.2.: Workflow diagram of predictive modelling

3.1.5 Using Python for machine learning

Python is one of the most popular programming languages for data science. A large number of libraries were developed by its community. In my implementation I used the *NumPy*, *SciPy*, *scikit-learn* and *matplotlib* libraries.

3.2 The Perceptron, the Adaline, and Machine Learning algorithms based on them

3.2.1 Artificial neurons and the early history of machine learning

The first concept about understanding of how the biological brain works was published by Warren McCullock and Walter Pitts in 1943 [16]. They discussed the simplified brain-cell in this publication, the so-called *McCullock-Pitts neuron.*

A few years later Frank Rosenblatt published his first concept of the so-called *perceptron*, based on the MCP neuron model [16]. With this perceptron rule, Rosenblatt proposed an algorithm, that would automatically learn the optimal weight coefficients, then these are multiplied with the input features a decision can be made, whether a neuron fires or not. If we talk about supervised learning, such an algorithm can be used to predict if a sample belongs to one class or the other.

If we are thinking about this problem as a binary classification task we can refer to the classes as 1 (as a positive class) and -1 (as a negative class) in the simplest case. We can then define an *activation function $\Phi(z)$*, which takes care of certain input values x and a corresponding weight vector w. Here z is the so-called net input ($z=w_1x_1 + \ldots + w_mx_m$):

$$\mathrm{w} = \begin{bmatrix} \mathrm{w}_1 \\ \vdots \\ \mathrm{w}_m \end{bmatrix}, \ \mathrm{x} = \begin{bmatrix} \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_m \end{bmatrix} \tag{1}$$

If the activation of a particular sample $x^{(i)}$, that is, the output of $\Phi(z)$, is greater than the defined threshold θ, we predict a class 1 and class -1, otherwise, in the perceptron algorithm, the activation function is a simple unit step function, which is also called the *Heaviside step function*:

$$z = \sum_{j=0}^{m} \boldsymbol{x}_j \boldsymbol{w}_j = \boldsymbol{w}^T \boldsymbol{x} \text{ ,and } \Phi(z) = \begin{cases} 1, & z \geq \theta \\ -1, & otherwise \end{cases} \tag{2}$$

The whole idea behind the neuron and perceptron models is to use a reductionist approach, how a single neuron in the brain works. The steps of the perceptron rule are summarized below:

1. Initialize the weights to 0 or small random numbers
2. For each training example $x^{(i)}$ perform the following steps:
    a. Compute the output value $\hat{y}$.
    b. Update the weights.

Here, the output value is the class label predicted by the unit step function that we defined earlier. The simultaneous update of each weight w_j in the weight vector $\boldsymbol{w}$ can be written as:

$$\mathrm{w_j} \coloneqq \mathrm{w_j} + \Delta \mathrm{w_j} \tag{3}$$

The value of Δw_j, which is used to update the weight w_j, is calculated by the perceptron learning rule:

$$\Delta \mathrm{w_j} = \eta(\mathrm{y}^{(i)} - \hat{\mathrm{y}}^{(i)})\mathrm{x}_j^{(i)} \tag{4}$$

Where η is the learning rate (a constant between 0.0 and 1.0), $y^{(i)}$ is the true class label of the *i*th training sample and $\hat{y}^{(i)}$ is the predicted class label. It is important to note, that all weights in the weight vector are being updated simultaneously, which means that we do not recompute the $\hat{y}^{(i)}$ before all of the weights Δw_j were updated. For a two dimensional datased our equations would be:

$$\Delta w_0 = \eta\left(y^{(i)} - \text{output}^{(i)}\right) \tag{5}$$

$$\Delta w_1 = \eta\left(y^{(i)} - \text{output}^{(i)}\right)x_1^{(i)} \tag{6}$$

$$\Delta w_2 = \eta\left(y^{(i)} - \text{output}^{(i)}\right)x_2^{(i)} \tag{7}$$

It is also important to mention that the convergence of the perceptron is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small. If the two classes cannot be separated by a linear decision boundary, we can set a maximum number of passes over the training dataset (that is what we call *epochs*) and/or a threshold for the number of tolerated misclassifications – the perceptron would never stop updating the weights otherwise.

Let me just summarize, how a perceptron works.

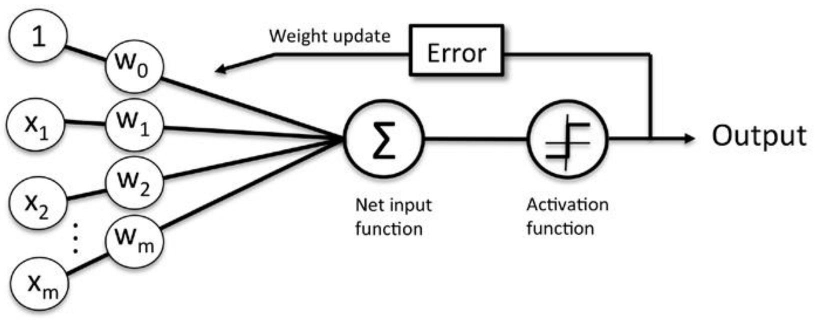


Figure 3.3.: The perceptron model (from the book of Raschka – Python Machine Learning)

The perceptron receives the inputs of a sample *x*, then combines them with the weights ***w*** to compute the net input. The net input is then passed to the activation function (in this case this is a unit step function), which generates the binary output -1 or +1 – the predicted class label of the sample. During the learning phase, this output is used to calculate the error of the prediction and update the weights.

3.2.2 Adaptive linear neurons

To move forward in understanding machine learning, I have to briefly describe the ADAptive Linear Neuron (Adaline). Adaline algorithm is really interesting, because it demonstrates well the key concept of defining and minimizing cost functions. This topic is really important to understand how advanced machine learning algorithms work for classification, *logistic regression* and *support vector machines*, as well for *regression models*.

The key difference between the Adaline rule (also known as *Widrow-Hoff rule*) and the perceptron is that the weights are updated base on a linear activation function rather than a unit step function like a perceptron. In Adaline, this linear activation function *Φ(z)* is simply the identity function of the net input so that $\Phi(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

The linear activation function is used for learning the weights, and a *quantizer*, which is similar to the unit step function is used to predict the class labels.

We can clearly see the difference to the perceptron, that know we use the continuous valued output from the linear activation function to compute the model error and update the weights.

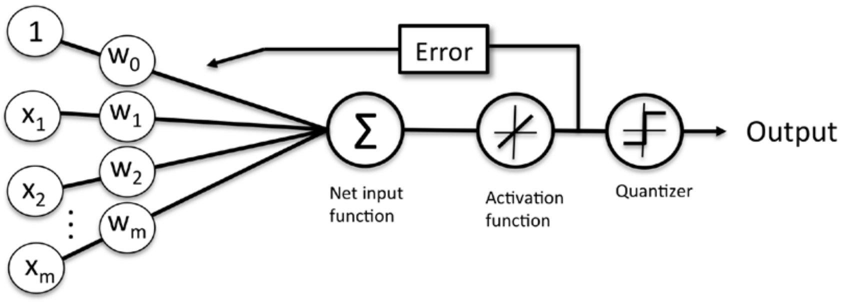


Figure 3.4.: The Adaline model (from the book Raschka – Python Machine Learning)

3.2.3 Minimizing cost functions with gradient descent

The key ingredient of supervised machine learning algorithms is to define an objective function, which can be optimized during the learning process. This function is often a *cost function* that we want to minimize. In case of Adaline, we can define the cost function *J* to learn the weights as the **Sum of Squared Errors** (**SSE**) between the calculated outcome and the true class label.

$$J(w) = \frac{1}{2}\sum_i \left(y^{(i)} - \Phi(z^{(i)})\right)^2 \qquad (8)$$

The term ½ is just added to make it easier to derive the gradient. Since this linear activation function is continuous, the cost function becomes differentiable. Another advantage of this cost function is that is convex, so that we can use a simple, but powerful optimization algorithm, which is called *gradient descent* to find the weights to minimize our cost function.

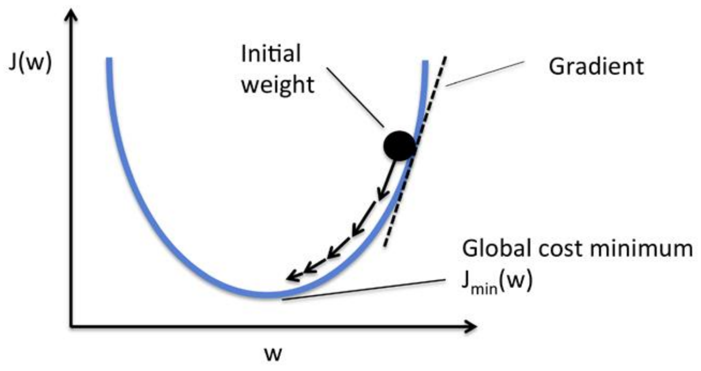


Figure 3.5.: Gradient descent (from the book of Raschka – Python Maschine Learning)

3.2.4 A simple linear regression model

A simple (*univariate*) linear regression can be used to find relationship between a single feature (explanatory variable *x*) and a continuous valued *response* (target variable *y*). The equation of the model is defined below:

$$y = w_0 + w_1 x \qquad (13)$$

In the equation $\boldsymbol{w_0}$ represents the weight, the y axis intercepts and $\boldsymbol{w_1}$ is the coefficient of the explanatory variable. The aim of the model is to learn the weights of the linear equation to describe the relationship between the explanatory variable and the target variable. This relationship can be used to predict the responses of new explanatory variables. The new explanatory variables shall not be included in the training dataset.

As we can see from the figure below, linear regression is used to find the best-fitting straight line through the sample points. This line is also known as the regression line and the vertical lines to the sample points are the so-called offsets or residuals, in other words the errors of the prediction.

3.2.5 The ordinary least squares linear regression model

However we described above the linear regression model, but we did not mention how we understand the term of a “best-fitting” line. To find the best solution, we have to introduce the *Ordinary Least Squares* (OLS) method to estimate the parameters of the regression line that minimizes the sum of squared vertical distances.

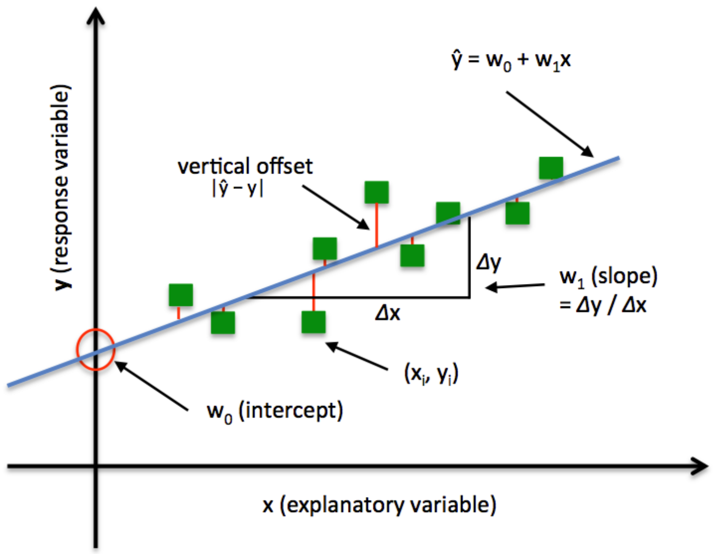


Figure 3.6.: Linear regression model (from the book of Raschka – Python Machine Learning)

Regarding the Adaline model, the artificial neuron uses a linear activation function and we defined *J(w)* as a cost function earlier. This cost function has to be minimized to learn the weights via the optimization algorithms, such as the *Gradient Descent* (GD) and *Stochastic Gradient Descent* (SGD). The cost function of adaline is the *Sum of Squared Errors* (SSE). This is identical to the OLS cost function that we defined earlier.

$$J(w) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{14}$$

In the equation $\hat{y}$ is the predicted value, in this case $\hat{y} = w^Tx$. The OLS linear regression is very similar to the Adaline, just without the unit step function. So without the unit step function or quantizer, the output of the model will be a continuous number instead of the class labels -1 or 1.

3.2.6 Evaluating the performance of linear regression models

If we plot the residuals (the differences between actual and predicted values), for the training and test data and we cannot find any pattern, it means that the errors are randomly distributed and our model works as we expected.

Another useful quantitative measure of a model's performance is the so-called *Mean Squared Error*, which is the average value of the SSE cost function that we minimize to fit the linear regression model as I described earlier. The MSE is useful for comparing different regression models or for tuning their parameters via a grid search and cross-validation:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{15}$$

If we calculate the MSE in a model for the training data and the test data, and we see that the value of MSE on the training set is lower that the MSE on the test set, it can be an indicator, that our model overfits the training data.

Another and more useful approach is to report the coefficient of determination (*R^2*), which means the standardized version of the MSE, for better interpretability of the model performance. In other words, *R^2* is the fraction of response variance that is captured by the model.

$$\mathrm{R}^2 = 1 - \frac{\mathrm{SSE}}{\mathrm{SST}} \tag{16}$$

SSE here is the sum of squared errors and *SST* is the total sum of squares:

$$\mathrm{SST} = \sum_{i=1}^{n}\left(y^{(i)} - \mu_y\right)^2 \tag{17}$$

We can say as well, that SST is the variance of the response and *R^2* is just the rescaled version of the MSE:

$$\mathrm{R}^2 = 1 - \frac{\mathrm{MSE}}{\mathrm{Var(y)}} \tag{18}$$

For the dataset *R^2* is bounded between 0 and one, but for the test set it can be negative as well. If the model fits perfectly the data *$R^2 = 1$, and MSE = 0.*

3.2.7 Polynomial regression

If we add polynomial terms to the simple linear regression model, or in other words we turn the line into a curve, our equation looks like the following:

$$y = w^0 + w^1x + w^2x^2 + \dots + w^dx^d \tag{19}$$

In the equation *d* denotes the degree of the polynomial. Although we can use polynomial regression to model a nonlinear relationship, it is still considered as a multiple linear regression model, because of the linear regression coefficients *w*.

3.2.8 Decision tree regression

A decision tree is grown by iteratively splitting its nodes until the leaves are pure or a stopping criterion is satisfied. If we are using the decision trees for classification, we define the *entropy* as a measure of impurity to determine, which feature split maximizes the *Information Gain* (IG). IG is defined below for a binary split:

$$\mathrm{IG}(\mathrm{D_p}, \mathrm{x}) = \mathrm{I}(\mathrm{D_p}) - \frac{\mathrm{N_{left}}}{\mathrm{N_p}} \mathrm{I}(\mathrm{D_{left}}) - \frac{\mathrm{N_{right}}}{\mathrm{N_p}} \mathrm{I}(\mathrm{D_{right}}) \tag{20}$$

Here, x is the feature to perform the split, N_p is the number of samples in the parent node, N_{left} and N_{right} are the number of samples in the child nodes, I is the impurity function, D_p is the dataset of the parent node, and D_{left} and D_{right} are the datasets of the child nodes. The aim of the model is to find the feature split that maximizes the information gain.

3.2.9 Random forests

A random forest can be considered as an *ensemble* of decision trees. The ensemble learning idea is came from combining *weak learners* to build a more robust model, a *strong learner*, that has a better generalization error and is less susceptible to overfitting.

The random forest algorithm can be summarized in four steps:

1. Draw a random *bootstrap* sample of size *n* – randomly choose *n* samples from the training set with replacement.
2. Grow a decision tree from the bootstrap sample. At each node:
    1. Randomly select *d* features without replacement
    2. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
3. Repeat the main steps from 1 to 2 *k* times.
4. Aggregate the prediction by each tree to assign the class label by *majority vote*.

Majority voting simply means that we select the class label that has been predicted by the majority of classifiers and received more than 50% of the votes. Strictly speaking, majority vote refers to binary class settings only.

4 The simplified ISP network model

Generally, it is not an easy job to build up a simplified model of an ISP network. The aim of modelling a service provider architecture is to collect similar data to the live environment. I built up different scenarios and I had to simplify it every time when I started to measure the network, because of the hardware resources. During the simplification I tried to focus on the most important protocols and elements of the network and highlight the crucial building blocks.

4.1 Planning, used tools and resources to build up the network

4.1.1 Hardware resources

The whole network runs in a virtual environment, which is using only one physical machine, so I had to take care of my configuration and hardware resources. I used a simple PC, with a second generation Intel Core i5 processor (4 cores, 3.4GHz processor base frequency), 8 GB 1333MHz DDR3 RAM and a 7200 rpm hard disk drive.

For measurement purposes I used my notebook, but from the viewpoint of resources, in practice any kind of computer can do the job, so it is not necessary to list all the HW details of this machine. Of course, in a real environment we have to provide the right hardware configuration, but for a basic SNMP polling, which was the only measurement feature I had to provide, the statements above can be true.

To connect the virtual network with the measuring machine I had to involve my home *Local Area Network* (LAN) in the topology. I tried to simulate a so-called Out of Band network (OOB), which is mostly used by ISPs for management and measurement purposes. Because it has to be a physically separated independent network, my LAN at home could match this criteria. It seems to be a good idea, to dedicate another machine for the measurement eliminating lots of resource dependent failures as well, as I did with my notebook.

4.1.2 Operation systems, applications, tools

On the PC a 64-bit Windows 10 operation system provides the ability to virtualize the resources of the hardware. The measuring laptop runs the 32-bit Windows 7 OS that does not have server capabilities, that is why we have to install some applications to support the functions of a real management server. At least we need the capability of a SNMP server. We can simply solve this problem by installing the Net-SNMP package on our system and we can run all the commands from the command prompt. To collect the data from the network I used only this method, however I tried different types of measurement applications and I built up a whole measurement system using Cacti [18]. Cacti is a complete frontend to RRDTool [19], which is the Open Source industry standard, high performance data logging and graphing system for time series data. RRDtool can be easily integrated in shell scripts, perl, python and many other programming language based applications. Cacti stores all the collected information in a MySQL database. The frontend is completely PHP driven and from a browser it is really easy to manage the whole system. Cacti supports SNMP to collect the data, but it support scripts as well to run them via CLI and store the results of the scripts.

In my virtual network topology I used a DHCP server, which was realized by using VirtualBox [20] to run a Windows Server 2008 R2 OS image. VirtualBox is one of the most known virtualization tool and it is really easy to use. We can use VB to divide and allocate our computer's resources to install a new OS via VB as a real machine once and after that, we are able to use it virtually any time we want.

The other virtualization application I used is GNS3 [21]. The *Graphical Network Simulator* is a network software emulator that is released in 2008. It allows the combination of virtual and real devices to simulate complex networks. It uses *Dynamips* emulation software to simulate a Cisco or Juniper OS. Furthermore GNS3 supports VMware, VirtualBox, IOU (IOS on UNIX) and Qemu virtualization. This means, that we can use any of the virtualization tools above and connect them in GNS3 to simulate closely anything we want to work with, but we always need to take care of the resources.

For emulating the routers I used several Cisco image files. The following list shows the different used versions of Cisco IOS and also represents the modelled router platform in GNS3:

- Cisco IOS Software, 7200 Software (C7200-ADVENTERPRISEK9-M), Version 15.2(4)S4, RELEASE SOFTWARE (fc2)
- Cisco IOS Software, 3700 Software (C3725-ADVENTERPRISEK9-M), Version 12.4(15)T14, RELEASE SOFTWARE (fc2)

- Cisco IOS Software, 2600 Software (C2691-ADVIPSERVICESK9-M), Version 12.4(25a), RELEASE SOFTWARE (fc2)

For running the SNMP polling from my notebook, I wrote a simple Windows batch file using just one loop to poll all the necessary MIB OIDs from all the SNMP agents in the network. Running this batch file continuously provides the simulation of real network measurement by storing the information in different maps and files.

The goal of using as many different image files (from different *vendors*), applications and tools as possible is the following: on one hand to highlight the issues of building a standardized network in *multi-vendor* and *multi-OS environment,* but on the other hand to prove, that my solution for the analysis can be used for any kind of data exported from any kind of system.

4.1.3 Simplification, Issues and Lessons Learned

4.1.3.1 Accessing the DHCP-MIB in a DHCP server

One of the most important examples of the issues mentioned above was that I had to implement a DHCP server in four different ways until I found the right method to grant access to the exact DHCP-MIB [22] OIDs. For the first time I used a simple Cisco c2691 router image and configured the router as a DHCP server. However it seemed obvious for me, that if the DHCP server configuration is supported by a router IOS, then the DHCP-MIB should be implemented. Unfortunately when I run an *snmpwalk* command from the SNMP server, which is used for querying all data from a given point of the OID-tree, I was unable to find any DHCP-MIB related information in the router.

After investigating this issue I found that the supported MIBs shows a dependency to the correspondent licensed version of the router OS, but I was still convinced that a router image, which is supporting advanced enterprise features (this is what the “ADVENTERPRISEK” notation is used for) should contain the DHCP-MIB.

I thought simultaneously that this problem could be a GNS3 limitation also, so I borrowed a real Cisco c871 router with the ADVENTERPRISEK licensed version of IOS, connected it to my home LAN, and configured it as the DHCP server. At least I had a supposition that to a real device I can add somehow the needed SNMP MIBs. In case of this simple router and the version of IOS, finally I failed.

But I still had the chance to succeed with IOS XE, which is the newer implementation of IOS supporting the needs of a *Next Generation Network* (IOS XR also). I found the clear documentation how to add MIBs to the router, so I downloaded a CSR1000v image, which is the virtualized router image of Cisco running IOS XE, deployed it in VirtualBox and started the investigation. After one or two hours of trying to add a simple MIB I thought I had to stop for a bit and search for other examples.

Finally I used a Windows Server 2008 R2 image in VirtualBox. It was easy to configure the DHCP pools using the GUI, but after running the snmpwalk command, I still was unable to access to the DHCP-MIB. Some minutes of Google searching guided me to modify the system registry database and manually add the necessary lines to use the DHCP-MIB.

4.1.3.2 Connecting GNS3 and physical devices to the LAN

It is possible in GNS3 to provide virtualized bridge interface to the Ethernet connection of the computer. When I planned the network on paper, my first scenario was that I will connect my virtual *border router* to internet through my home LAN and propagate a default route towards the other devices. In this case I should have deployed the whole OOB network and the measurement server as well virtually, which caused some problems with in my hardware resource utilization. I describe this issue in the further section.

So I had resource problems and I decided that I move the role of the measurement server to another physical machine. In this case the OOB network and the exit towards the Internet would use the same Layer 2 network, in other words the LAN. To avoid any possible loop in my network I introduced simple Layer 3 traffic separation using *Virtual Routing- and Forwarding-tables* (VRF). VRFs can be used to deploy IP- or L3VPNs which means, that a router can use more than one routing-table simultaneously. In this case, the traffic can be separated in Layer 3 level, even the IP addresses can overlap until there is a need for communication between the VRFs [23]. So I planned to introduce a L3VPN for management purposes using *VRF-Lite* [24]. But L3 separation needs L2 separation as well if we use the same medium (in this case the Ethernet connection of the PC) to forward IP packets. This kind of separation can be implemented using *Virtual Local Area Networks* (VLANs) [25]. Unfortunately the virtual Ethernet bridge provided by GNS3 does not support VLANs, so the complete traffic separation cannot be performed, and in this way the routes of the management VRF can be seen in the *global* routing table. This is a kind of the so-called *Route-leaking*, which sometimes can be necessary in a real environment to provide access from one VRF into another, but we still need to be careful to keep the control, which IP addresses can be reached from the logically divided domains to avoid loops.

Finally I decided to leave the idea of the full Internet access behind, and use a simple router to simulate the Internet with some *loopback* interfaces configured with public addresses and to form eBGP peering connection to propagate the modelled-Internet routes towards the ISP network as a traditional ISP-to-ISP connection looks alike.

A final consideration had been made using Windows OS in the network. I had to enable the important ports on the built in Windows *Firewall* to use SNMP and ICMP echoes.

4.1.3.3 Simplification of the architecture

As I described above I planned a much bigger ISP network with a virtualized OOB network. Furthermore I wanted to present how dynamic RPs can provide redundancy (I also had Machine Learning usecases for this scenario), so I deployed redundant *core network routers*, redundant *Route Reflectors* and full-mesh connections within the *edge routers* and core routers. The core routers are the so-called *Provider routers (P routers).* From functionality point of view, these are simple MPLS LSRs. The edge routers are *Provider-Edge routers (PE routers)* or MPLS edge-LSRs. If we use iBGP inside our network as a dynamic routing protocol, we need to introduce a full-mesh neighborship between the nodes or implement Route Reflectors (RRs). If we are using the RR scenario, we have to define the adjacency relations only between an iBGP router and the RRs. In this way adding a new router into the network does not require further configurations of the already existing nodes. The iBGP neighborship between the RRs and all the routers provides that every router in the topology can see the others advertised routes, but in this kind of topology RRs are responsible to propagate the routing advertisements within the routers.

In a service provider network only the PE routers speak iBGP with an IGP (e.g.: OSPF or IS-IS) and MPLS configuration. On the P routers only the IGP and the MPLS run.

I was able to configure such a big and redundant network on my PC in GNS3 with many *Virtual PC* hosts representing the customers and the network worked well. However, during the measurement, because of the continuously changing behaviour of the VPCs and SNMP polling, I encountered some delivery errors in the measured data. It means, that my PC was unable to handle the huge amount of packets generated by the communication of the routers, the simulated changing behaviour of VPCs and the continuously polled SNMP data.

I had to rethink my scenario and my expectations about the measurement as well. My aim was to produce data as similar as possible to the real networks. So I ended up with a new final scenario described in the following sections.

4.2 Describing the implementation of the ISP network

4.2.1 The Final Topology

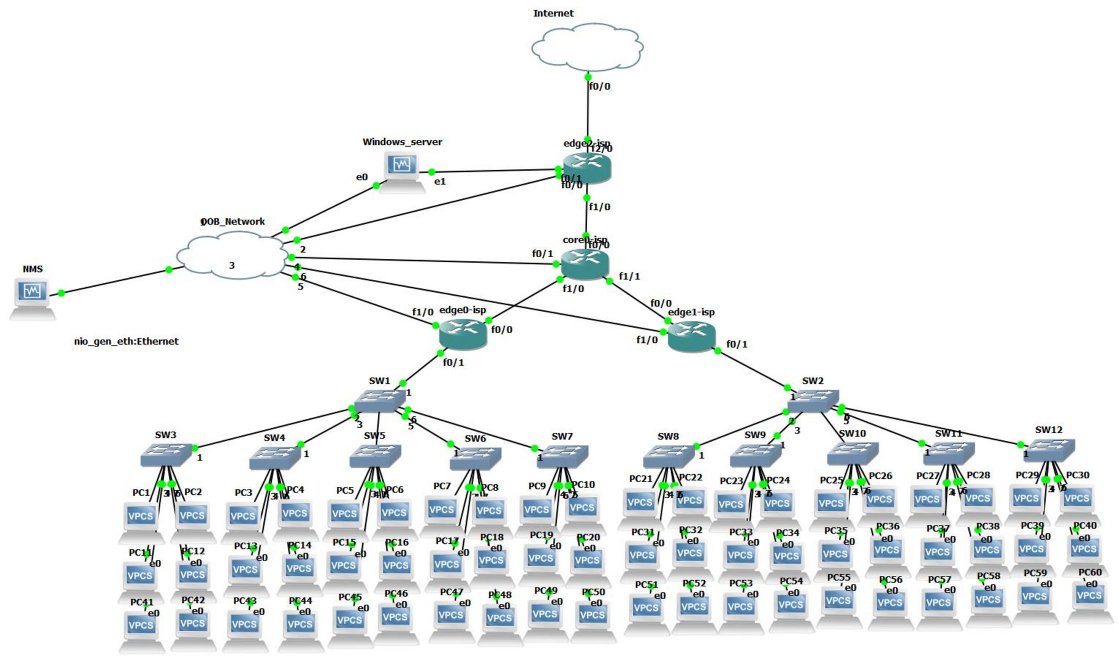


Figure 4.1.: The simplified model of an ISP network

On the figure above we can see the final simplified topology of the implemented network. It has three PE routers and one P router as the parts of the ISP's MPLS IP core network. One Router symbolises the Internet that is why I changed the symbol of the router to an Internet cloud. The OOB cloud represents an independent Layer2 network that has separated connections to all the manageable devices. The NMS is connected to this network as well. We have a separated DHCP server connected to the edge2-isp router and several VPCs representing the end-users.

As I expounded in the previous section, I had to give up the redundancy of the network and with my measurements I shifted the focus to the end-users' behaviour and the IP address pool utilization. Because this type of communication is only a client-server type of communication, we will see later, that the collected data from the core network routers cannot describe exactly what is happening on the end of the network, unless we are machines, which are able to process every packet sent through the network.

4.2.2 Configuring the IP/MPLS Core with the DHCP server

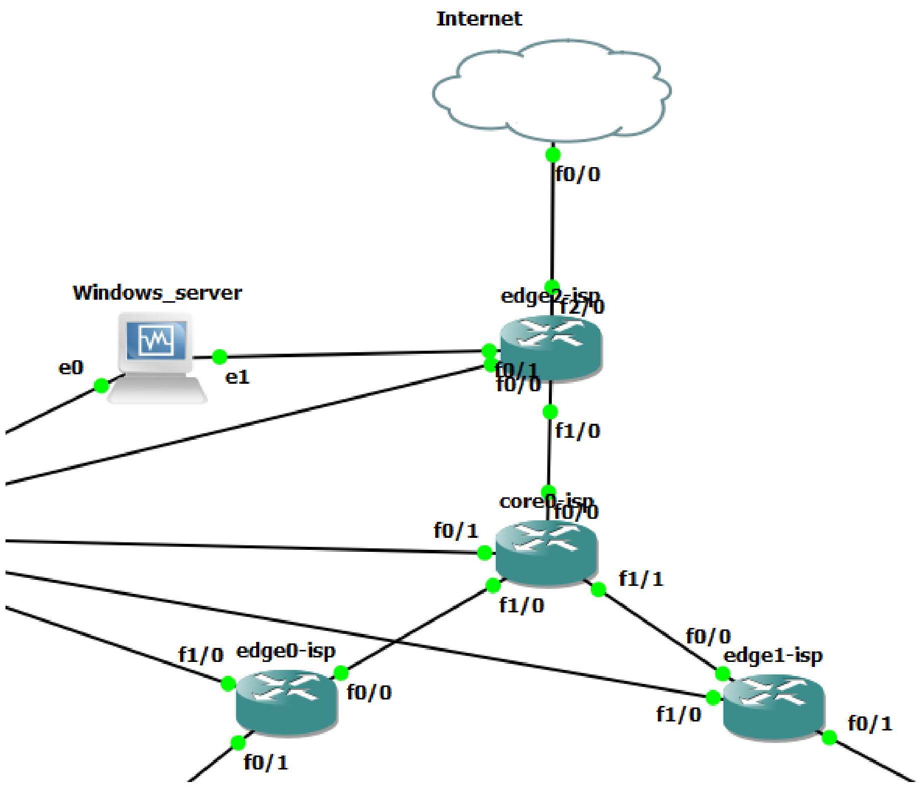


Figure 4.2.: The IP/MPLS Core

In this section I describe the configuration of the IP/MPLS Core network, the router representing the Internet and the DHCP server.

4.2.2.1 ASN & IP addressing scheme

Before we start to implement the network we need to plan the IP addressing very careful. It is very important to allocate the right subnet sizes with a margin which can be used if the network grows and new nodes need new addresses. For the sake of simplicity

I used IPv4 addresses only. All the used addresses are private or allocated for documentation purposes only. It means that these addresses can never be used on the Internet.

For using the right addresses, domain names, ASNs in my documentation, I followed the rules described in these RFCs:

- Public IPv4 – RFC5737 [26]
- Private IPv4 – RFC1918 [27]
- Domain names – RFC2606 [28]
- IPv6 – RFC3849 [29]
- ASN – RFC5398 [30]
- Private ASN – RFC6996 [31]

The planned addressing scheme can be found below.

Public IPv4	Description
192.0.2.0/24	Infrastructure
198.51.100.0/24	Customer Pool
203.0.113.0/24	BGP
192.168.1.0/24	Management

Table 4.1.: Allocating pools for the ISP network

Device	Interface	Neighbor_Device	Neighbor_Interface	IPv4 Subnet
core0-isp	FastEthernet0/0	edge2-isp	FastEthernet0/1	192.0.2.0/31
core0-isp	FastEthernet1/0	edge0-isp	FastEthernet0/0	192.0.2.2/31
core0-isp	FastEthernet1/1	edge1-isp	FastEthernet0/0	192.0.2.4/31
edge2-isp	FastEthernet1/0	DHCP_server	Ethernet1	198.51.100.64/30
edge0-isp	FastEthernet0/1	Customers		198.51.100.30/27
edge1-isp	FastEthernet0/1	Customers		198.51.100.63/27
edge2-isp	FastEthernet2/0	Internet	FastEthernet0/0	203.0.113.8/31

Table 4.2.: Interface addressing scheme

Device	Interface	IPv4 Address
core0-isp	Loopback0	203.0.113.0
edge0-isp	Loopback0	203.0.113.1
edge1-isp	Loopback0	203.0.113.2
edge2-isp	Loopback0	203.0.113.3

Table 4.3.: Loopback addresses

ASNumbers	ASN
ISP	64496
Other ISP	64497

Table 4.4.: AS Numbers

Device	Interface	IPv4 Address
NMS	eth0	192.168.1.67/24
DHCP_server	eth0	192.168.1.68/24
edge2-isp	fa0/0	192.168.1.73/24
core0-isp	fa0/1	192.168.1.70/24
edge0-isp	fa1/0	192.168.1.71/24
edge1-isp	fa1/0	192.168.1.72/24

Table 4.5.: OOB network addressing

4.2.2.2 PE router configuration

I added comments to the configurations (see the Appendix) to describe the functions clearly. The edge2-isp router is used as a border router towards the Internet. It also has connection to the DHCP server. The edge0-isp and edge1-isp routers are representing a branch office or a part of a town, where the subscriber connections are terminated with the help of aggregation switches.

On the figure below we can see the edge router terminating subscriber connections.

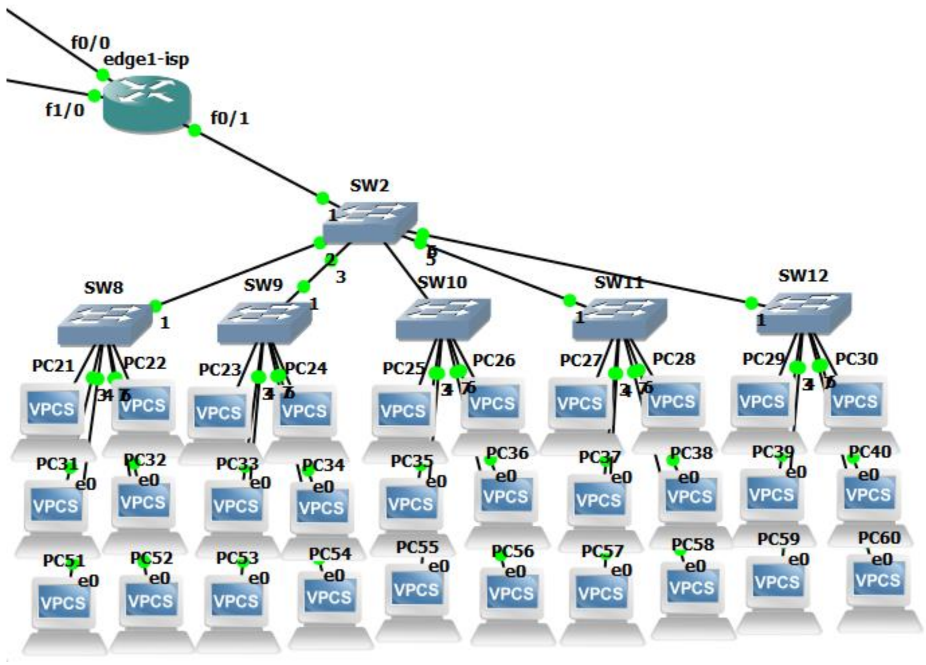


Figure 4.3.: The edge router terminates the subscriber connections

4.2.2.3 P router configuration

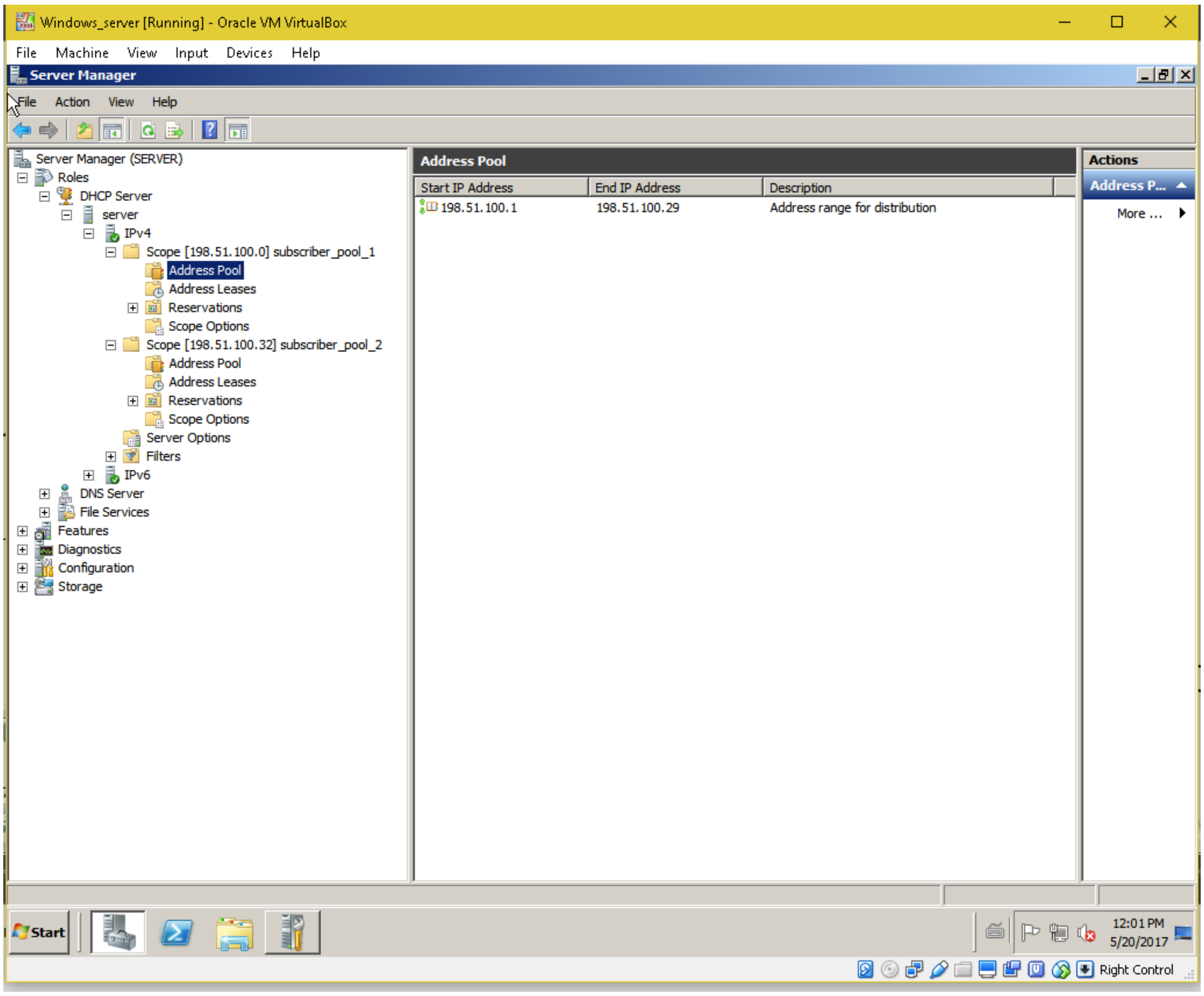
The core0-isp router is the P router of the network. This router runs only the OSPF process and MPLS and the biggest advantage of using a core router in the network, that it has direct connections to PE Routers and with connections it can provide multiple path to the same destination with MPLS. Machines used for P router function provide the highest bandwidth with high port density and designed for forwarding the packets as fast as possible to not introduce further delays into the network. That is why only OSPF runs here as IGP, because it can provide the fast convergence which is needed for the MPLS network.

4.2.2.4 Internet router configuration

The router, represents the Internet, has some public IP addresses configured on several Loopback interfaces. It has an eBGP connection to the edge2-isp router and advertises the so-called *Internet addresses* towards the ISP network.

4.2.2.5 DHCP server configuration

Configuring a DHCP server on Windows server platform is really easy, because we can use the GUI to configure the DHCP pools. Figure 4.4. shows the configured DHCP pools.



Start IP Address	End IP Address	Description
198.51.100.1	198.51.100.29	Address range for distribution

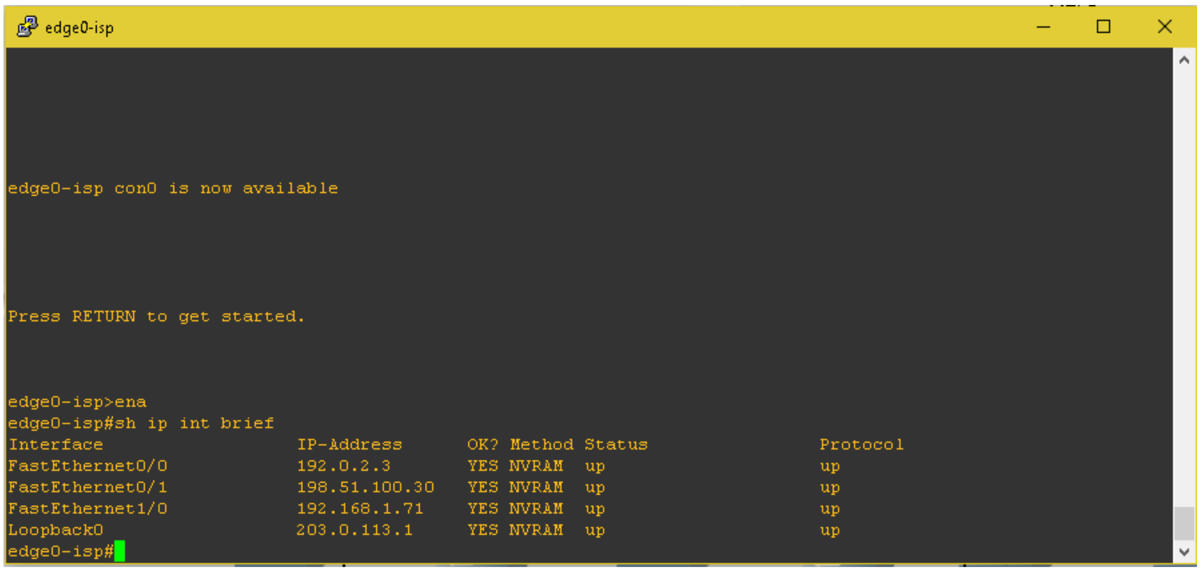
Figure 4.4.: DHCP server role in Windows

4.2.3 Verifying the network configuration

After we build up a network the network administrators have to make sure that the network works properly before releasing it into production state.

4.2.3.1 Checking interface states

First of all we need to check that all the configured interfaces are up and operating from both physical and logical viewpoint correctly. We must to do it on every network element. An example can be seen below for checking the interface states:

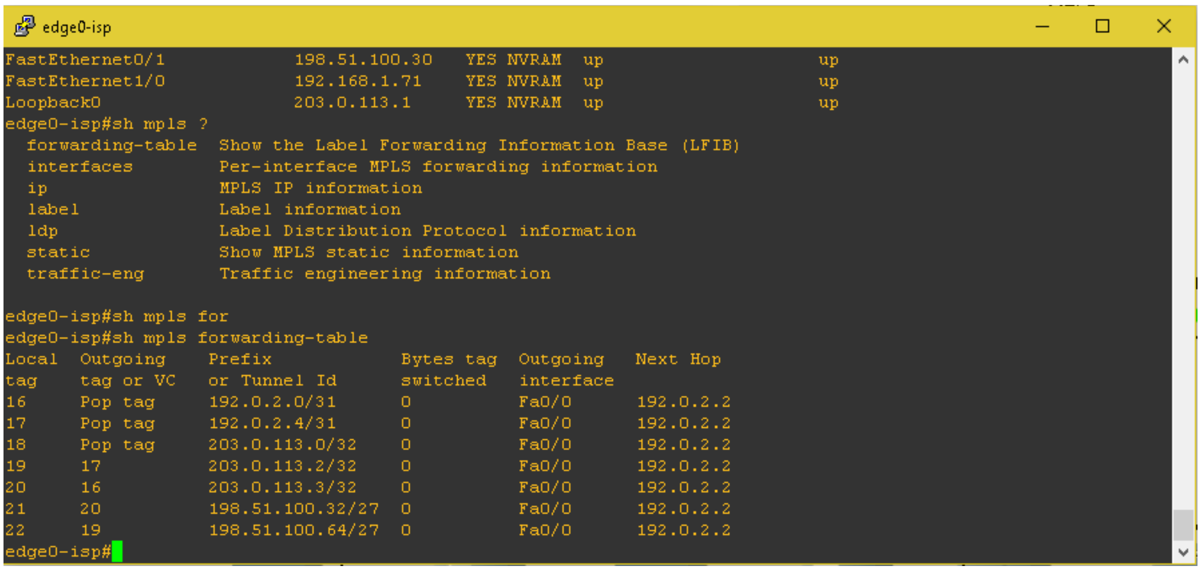


```
edge0-isp con0 is now available



Press RETURN to get started.


edge0-isp>ena
edge0-isp#sh ip int brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.0.2.3       YES NVRAM  up                    up
FastEthernet0/1        198.51.100.30   YES NVRAM  up                    up
FastEthernet1/0        192.168.1.71    YES NVRAM  up                    up
Loopback0              203.0.113.1     YES NVRAM  up                    up
edge0-isp#
```

Figure 4.5.: Verifying networking interfaces

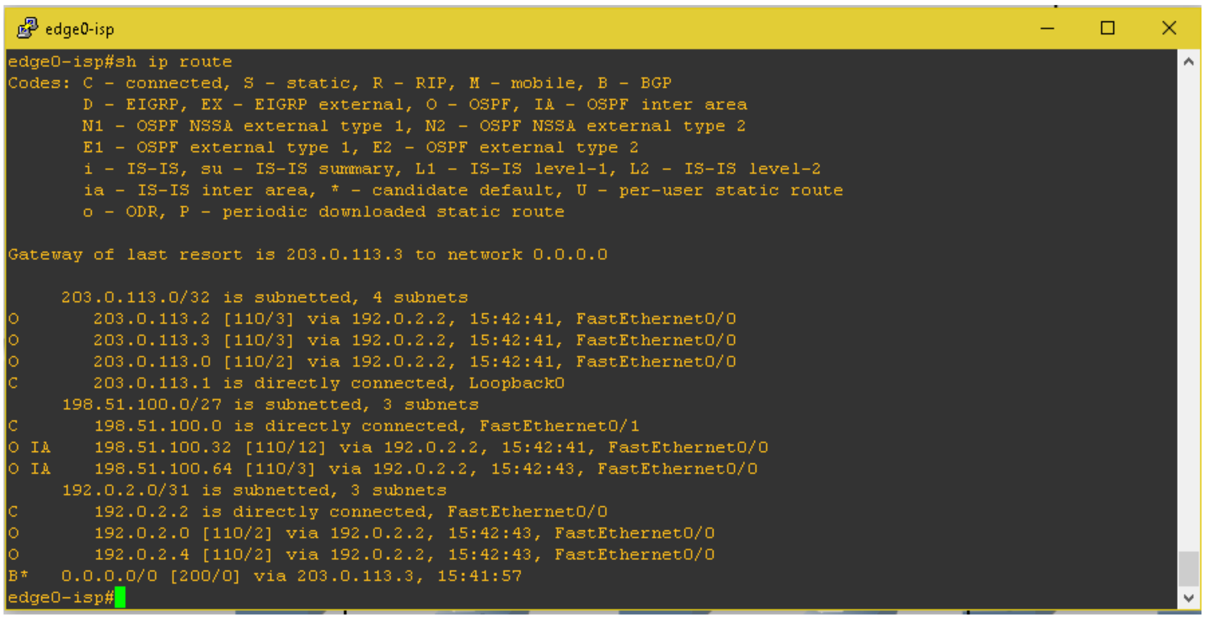
4.2.3.2 Verifying MPLS and Routing tables

If we know that all the interfaces are connected, we need to check that the configured protocols are also working in the expected way.



```
FastEthernet0/1            198.51.100.30   YES NVRAM  up                    up
FastEthernet1/0            192.168.1.71    YES NVRAM  up                    up
Loopback0                  203.0.113.1     YES NVRAM  up                    up
edge0-isp#sh mpls ?
  forwarding-table  Show the Label Forwarding Information Base (LFIB)
  interfaces        Per-interface MPLS forwarding information
  ip                MPLS IP information
  label             Label information
  ldp               Label Distribution Protocol information
  static            Show MPLS static information
  traffic-eng       Traffic engineering information

edge0-isp#sh mpls for
edge0-isp#sh mpls forwarding-table
Local  Outgoing    Prefix            Bytes tag  Outgoing   Next Hop
tag    tag or VC   or Tunnel Id      switched   interface
16     Pop tag     192.0.2.0/31      0          Fa0/0      192.0.2.2
17     Pop tag     192.0.2.4/31      0          Fa0/0      192.0.2.2
18     Pop tag     203.0.113.0/32    0          Fa0/0      192.0.2.2
19     17          203.0.113.2/32    0          Fa0/0      192.0.2.2
20     16          203.0.113.3/32    0          Fa0/0      192.0.2.2
21     20          198.51.100.32/27  0          Fa0/0      192.0.2.2
22     19          198.51.100.64/27  0          Fa0/0      192.0.2.2
edge0-isp#
```

Figure 4.6.: Verifying the MPLS forwarding-table

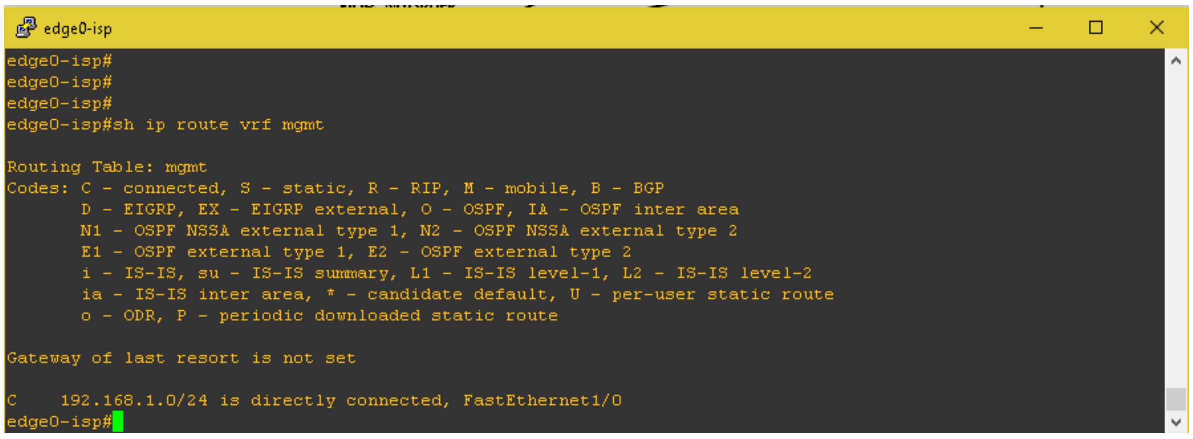


```
edge0-isp#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 203.0.113.3 to network 0.0.0.0

     203.0.113.0/32 is subnetted, 4 subnets
O       203.0.113.2 [110/3] via 192.0.2.2, 15:42:41, FastEthernet0/0
O       203.0.113.3 [110/3] via 192.0.2.2, 15:42:41, FastEthernet0/0
O       203.0.113.0 [110/2] via 192.0.2.2, 15:42:41, FastEthernet0/0
C       203.0.113.1 is directly connected, Loopback0
     198.51.100.0/27 is subnetted, 3 subnets
C       198.51.100.0 is directly connected, FastEthernet0/1
O IA    198.51.100.32 [110/12] via 192.0.2.2, 15:42:41, FastEthernet0/0
O IA    198.51.100.64 [110/3] via 192.0.2.2, 15:42:43, FastEthernet0/0
     192.0.2.0/31 is subnetted, 3 subnets
C       192.0.2.2 is directly connected, FastEthernet0/0
O       192.0.2.0 [110/2] via 192.0.2.2, 15:42:43, FastEthernet0/0
O       192.0.2.4 [110/2] via 192.0.2.2, 15:42:43, FastEthernet0/0
B*   0.0.0.0/0 [200/0] via 203.0.113.3, 15:41:57
edge0-isp#
```

Figure 4.7.: Verifying the routing table

As we can see above, the edge0-isp router works properly, it knows the routes propagated by OSPF, it sees the connected interfaces and it has a default route towards the Internet propagated by BGP.

We also need to check that the routing tables for the configured VRFs are built up:



```
edge0-isp
edge0-isp#
edge0-isp#
edge0-isp#
edge0-isp#sh ip route vrf mgmt

Routing Table: mgmt
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet1/0
edge0-isp#
```

Figure 4.8.: Verifying VRFs

4.2.4 End-to-end testing

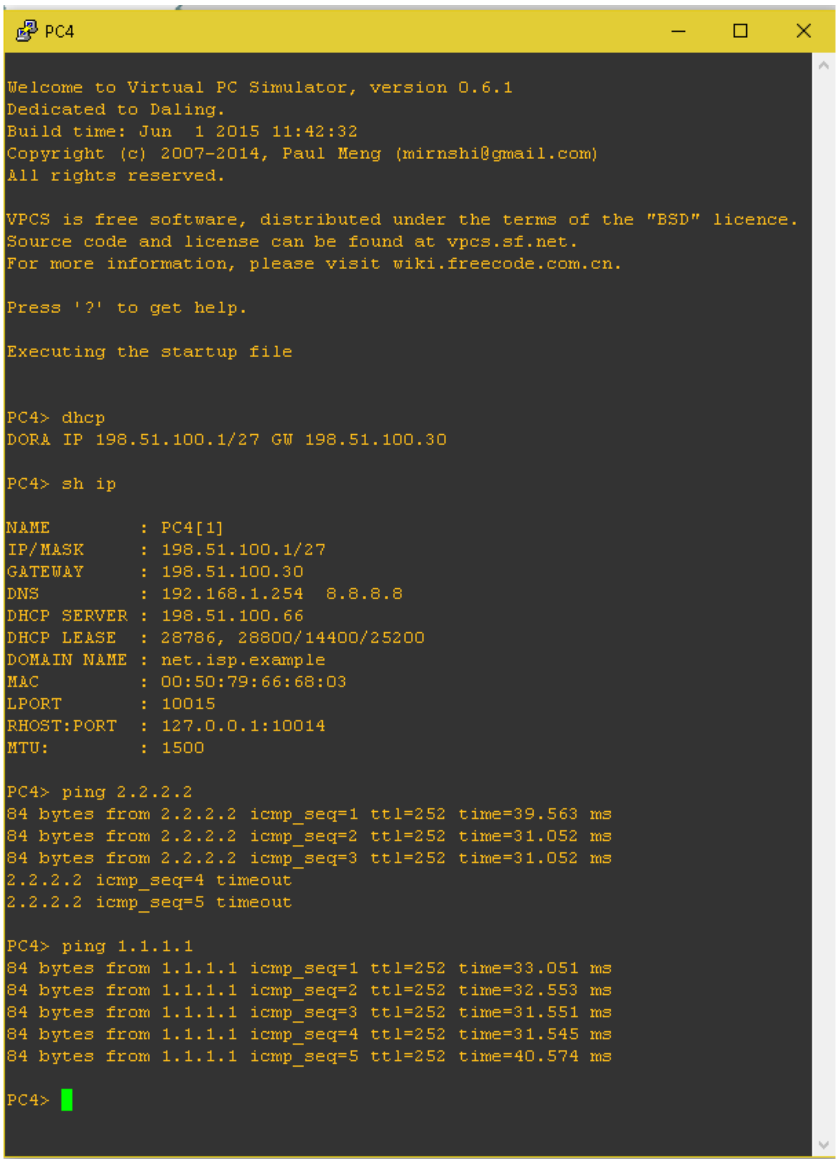
The final step of implementation is to build up test cases and see that everything works well in the network. In my test case I checked, that the hosts can reach the DHCP server, can ask for an IP address and can reach the Internet. The other important thing is to try to do some measurements to make sure, that the NMS has the right connectivity to the devices.

4.2.4.1 Testing the user connectivity

From both separated network I randomly chose a VPC and tested the network. As we can see on the figure below, the VPC was able to ask for a new IP Address with DHCP. *DORA* means, that the DHCP Discovery was sent out, the Offer has arrived, a Request was sent out, and the Ack has arrived. We can see that the host get a new IP address. It is necessary to check, that all the options are also configured automatically, such as the default gateway, DNS server and so on.

If the automated configuration is done, we can start to test the connectivity towards the Internet. I randomly chose an IP address and started to ping it. As we can see not all of the ICMP echoes were successful. This can happen in a production environment also, so to make sure, that the host is working in a proper way, let us try to reach some other sites. Now we see that it works fine.

It is also necessary to check the DHCP server. Windows server provides easy export of the DHCP data.



```
PC4

Welcome to Virtual PC Simulator, version 0.6.1
Dedicated to Daling.
Build time: Jun  1 2015 11:42:32
Copyright (c) 2007-2014, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.

Press '?' to get help.

Executing the startup file


PC4> dhcp
DORA IP 198.51.100.1/27 GW 198.51.100.30

PC4> sh ip

NAME        : PC4[1]
IP/MASK     : 198.51.100.1/27
GATEWAY     : 198.51.100.30
DNS         : 192.168.1.254  8.8.8.8
DHCP SERVER : 198.51.100.66
DHCP LEASE  : 28786, 28800/14400/25200
DOMAIN NAME : net.isp.example
MAC         : 00:50:79:66:68:03
LPORT       : 10015
RHOST:PORT  : 127.0.0.1:10014
MTU:        : 1500

PC4> ping 2.2.2.2
84 bytes from 2.2.2.2 icmp_seq=1 ttl=252 time=39.563 ms
84 bytes from 2.2.2.2 icmp_seq=2 ttl=252 time=31.052 ms
84 bytes from 2.2.2.2 icmp_seq=3 ttl=252 time=31.052 ms
2.2.2.2 icmp_seq=4 timeout
2.2.2.2 icmp_seq=5 timeout

PC4> ping 1.1.1.1
84 bytes from 1.1.1.1 icmp_seq=1 ttl=252 time=33.051 ms
84 bytes from 1.1.1.1 icmp_seq=2 ttl=252 time=32.553 ms
84 bytes from 1.1.1.1 icmp_seq=3 ttl=252 time=31.551 ms
84 bytes from 1.1.1.1 icmp_seq=4 ttl=252 time=31.545 ms
84 bytes from 1.1.1.1 icmp_seq=5 ttl=252 time=40.574 ms

PC4>
```

Figure 4.9.: Testing user connectivity

Client IP Address	Name	Lease Expiration	Type	Unique ID	Desc	Network Access	Protection	Probation Expiration	Filter Profile
198.51.100.1	PC41.net.isp.example	5/11/2017 8:38:10 PM	DHCP	5079666803		Full Access	N/A	None	
198.51.100.33	PC281.net.isp.example	5/11/2017 8:48:30 PM	DHCP	507966681b		Full Access	N/A	None	

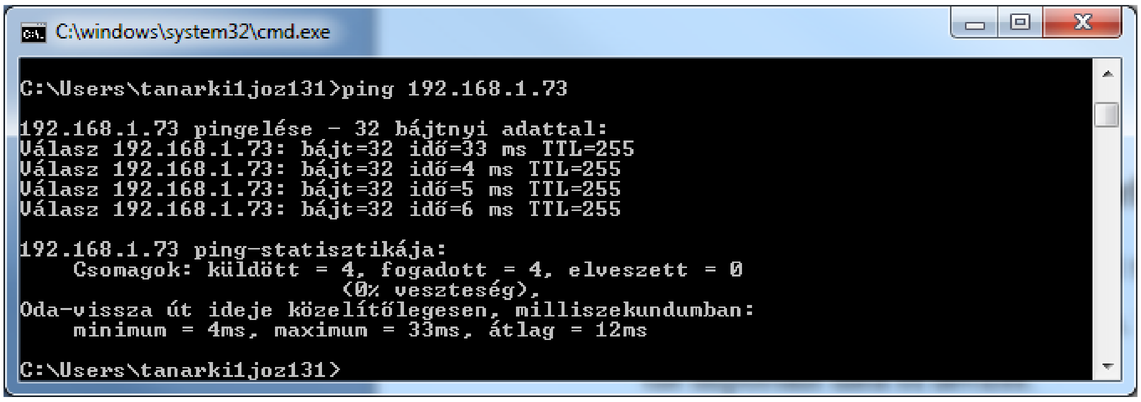
Table 4.6.: DHCP lease export

4.3 Measuring the Network

As I described earlier, I used the OOB network to connect all the management interfaces of the networking elements. NMS is the system that is using SNMPv2 to pool the important data of devices.

4.3.1 Testing the NMS

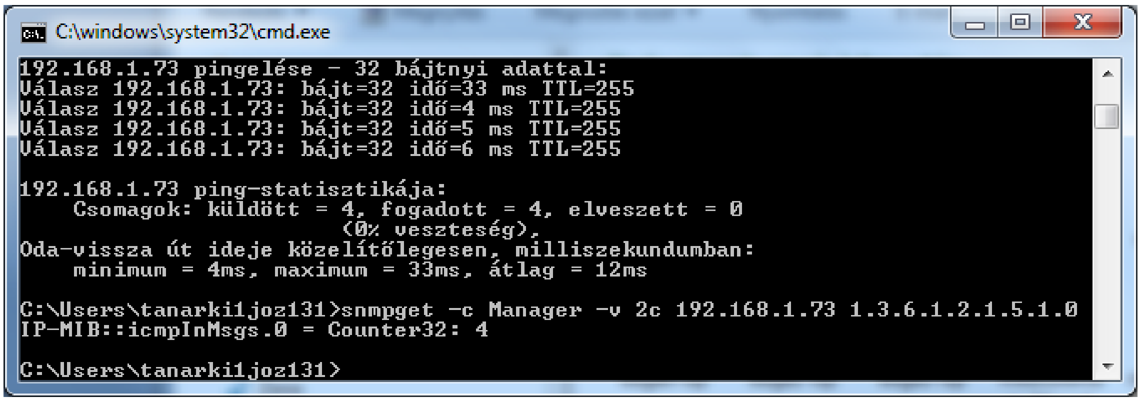
First of all we need to be sure that the communication works between the NMS and networking devices. I used ping to verify these connections. Unfortunately the language used on the NMS is Hungarian, but we can see the ping worked well:



```
C:\Users\tanarki1joz131>ping 192.168.1.73

192.168.1.73 pingelése - 32 bájtnyi adattal:
Válasz 192.168.1.73: bájt=32 idő=33 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=4 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=5 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=6 ms TTL=255

192.168.1.73 ping-statisztikája:
    Csomagok: küldött = 4, fogadott = 4, elveszett = 0
                    (0% veszteség),
Oda-vissza út ideje közelítőlegesen, milliszekundumban:
    minimum = 4ms, maximum = 33ms, átlag = 12ms

C:\Users\tanarki1joz131>
```

Figure 4.10.: Testing connectivity

After this action I tried to poll a single SNMP parameter to verify that SNMP works also. I polled the counter of ICMPInMsgs OID, to see that after the first ping the system counts the data well and it is accessible:



```
192.168.1.73 pingelése - 32 bájtnyi adattal:
Válasz 192.168.1.73: bájt=32 idő=33 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=4 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=5 ms TTL=255
Válasz 192.168.1.73: bájt=32 idő=6 ms TTL=255

192.168.1.73 ping-statisztikája:
    Csomagok: küldött = 4, fogadott = 4, elveszett = 0
                    (0% veszteség),
Oda-vissza út ideje közelítőlegesen, milliszekundumban:
    minimum = 4ms, maximum = 33ms, átlag = 12ms

C:\Users\tanarki1joz131>snmpget -c Manager -v 2c 192.168.1.73 1.3.6.1.2.1.5.1.0
IP-MIB::icmpInMsgs.0 = Counter32: 4

C:\Users\tanarki1joz131>
```

Figure 4.11.: SNMP polling

Fortunately everything works fine, so the network is ready for production and measurement.

4.3.2 Concepts of simulation and measurement

To produce very similar data to the live network environment my concept regarding the simulation was, that I should try to simulate the subscriber behaviour on the VPCs.

I introduced a regular subscriber behaviour model:

1. The subscriber turns on his/her computer which tries to connect to the Internet. The computer by default asks for an IP address via DHCP
2. The subscriber then starts to use the service and generate different types of traffics. He/she connects to different websites for example.
3. The subscriber leaves the computer
    a) the computer is then turned off
    b) the computer stays online

In a very simple way the VPCs use DHCP to ask for an IP address, and can give it back when they disconnect the network. For traffic generation I used the ICMP echoes with different packet size and recurrence (bigger, smaller, or the combination of the with sweep ping).

The measurement took 72 minutes. I tried to scale down 3 days to 72 minutes to ensure, that I can specify some *busy hours* when most of the subscribers are connected and using the service in a very intensive way.

4.3.3 The measured data

On the DHCP server I monitored the following SNMP OIDs (the names are representing the exact type of message that is why further explanation is not needed):

- parDHCPTotalNoOfAcks
- parDHCPTotalNoOfDeclines
- parDHCPTotalNoOfDiscovers
- parDHCPTotalNoOfNacks
- parDHCPTotalNoOfOffers
- parDHCPTotalNoOfReleases

- parDHCPTotalNoOfRequests
- subnet_1_NoAddFree
- subnet_1_NoAddInUse
- subnet_1_NoPendingOffers
- subnet_2_NoAddFree
- subnet_2_NoAddInUse
- subnet_2_NoPendingOffers

On the routers I monitored each and every interface (except the mgmt interface) asking for the following OIDs:

- ifHCInBroadcastPkts
- ifHCInMulticastPkts
- ifHCInOctets
- ifHCInUcastPkts
- ifHCOutBroadcastPkts
- ifHCOutMulticastPkts
- ifHCOutOctets
- ifHCOutUcastPkts

I used a simple Windows batch file to poll these parameters. Every polled different type of data is stored in a different file in a hierarchical map structure.

After the simulation was completed I created an Excel document, in which every measurement data was imported. Later I was able to export the results into a CSV (*comma-separated values)* file, which is easy to import into a python panda database.

5 Using ML algorithms to predict continuous values in the ISP network

In this final section I describe, how ML algorithms can handle the previously measured data. From python programming viewpoint I try to explain only the most important things, if the Reader have further questions about the implemented code, he/she can find the way to the proper documentation through the Bibliography.

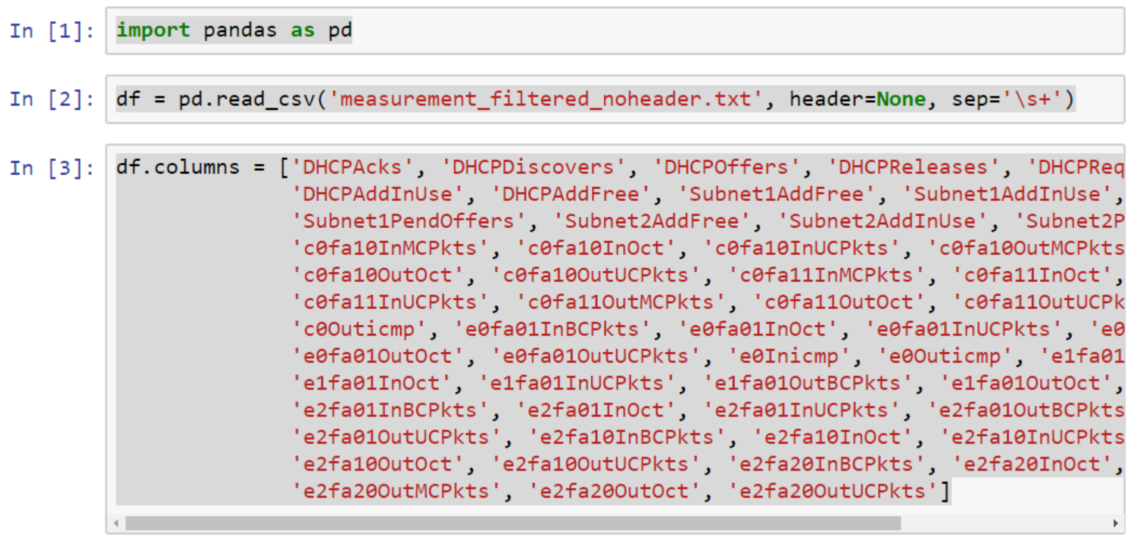
I used the cropped parts of the Jupyter notebook to show the implemented code instead of copying it into the document and I did it with reason. An iPython notebook has unbeatable built in visualization features, which makes the understanding really easy and the programming process fast and fun.

5.1 Exploring and visualizing the measured dataset

Because of resource limitation I filtered out some unnecessary columns from my measurement results. First of all I eliminated data which can be paired. For example if a router interface is connected to another router's interface in a point-to-point relation, it makes sense to examine the measured data only on one end of the connection. I also eliminated the columns which have only zero values.

I also added some columns where I calculated the sum of the DHCP parameters per sample.

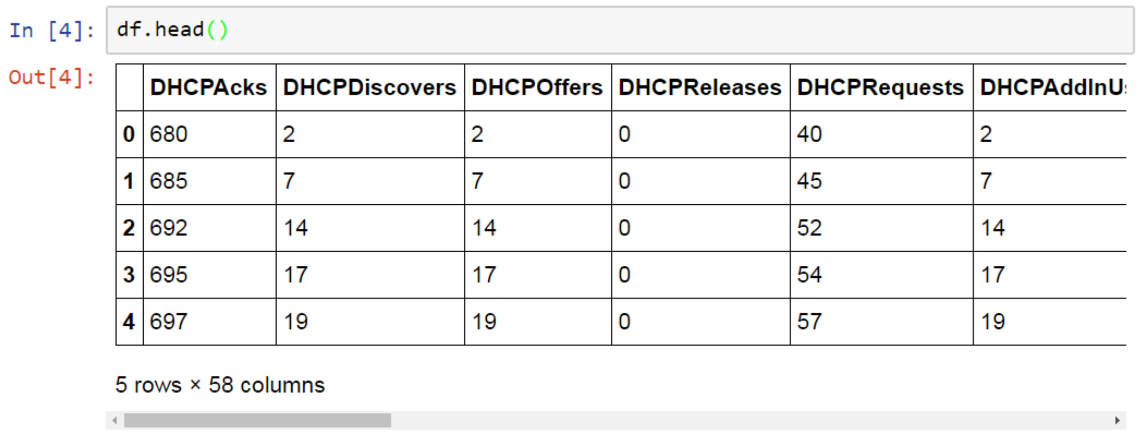
Then I imported the filtered dataset into a panda *DataFrame* and renamed the columns using shorter names:



```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('measurement_filtered_noheader.txt', header=None, sep='\s+')

In [3]: df.columns = ['DHCPAcks', 'DHCPDiscovers', 'DHCPOffers', 'DHCPReleases', 'DHCPReq
                      'DHCPAddInUse', 'DHCPAddFree', 'Subnet1AddFree', 'Subnet1AddInUse',
                      'Subnet1PendOffers', 'Subnet2AddFree', 'Subnet2AddInUse', 'Subnet2P
                      'c0fa10InMCPkts', 'c0fa10InOct', 'c0fa10InUCPkts', 'c0fa10OutMCPkts
                      'c0fa10OutOct', 'c0fa10OutUCPkts', 'c0fa11InMCPkts', 'c0fa11InOct',
                      'c0fa11InUCPkts', 'c0fa11OutMCPkts', 'c0fa11OutOct', 'c0fa11OutUCPk
                      'c0Outicmp', 'e0fa01InBCPkts', 'e0fa01InOct', 'e0fa01InUCPkts', 'e0
                      'e0fa01OutOct', 'e0fa01OutUCPkts', 'e0Inicmp', 'e0Outicmp', 'e1fa01
                      'e1fa01InOct', 'e1fa01InUCPkts', 'e1fa01OutBCPkts', 'e1fa01OutOct',
                      'e2fa01InBCPkts', 'e2fa01InOct', 'e2fa01InUCPkts', 'e2fa01OutBCPkts
                      'e2fa01OutUCPkts', 'e2fa10InBCPkts', 'e2fa10InOct', 'e2fa10InUCPkts
                      'e2fa10OutOct', 'e2fa10OutUCPkts', 'e2fa20InBCPkts', 'e2fa20InOct',
                      'e2fa20OutMCPkts', 'e2fa20OutOct', 'e2fa20OutUCPkts']
```

Figure 5.1.: Importing the network measurement dataset

It is crucial to check whether the import was successful or not. So let me show, that we successfully created the dataset:



```
In [4]: df.head()
Out[4]:
```

	DHCPAcks	DHCPDiscovers	DHCPOffers	DHCPReleases	DHCPRequests	DHCPAddInU
0	680	2	2	0	40	2
1	685	7	7	0	45	7
2	692	14	14	0	52	14
3	695	17	17	0	54	17
4	697	19	19	0	57	19

5 rows × 58 columns

Figure 5.2.: Showing the first five rows of the dataset

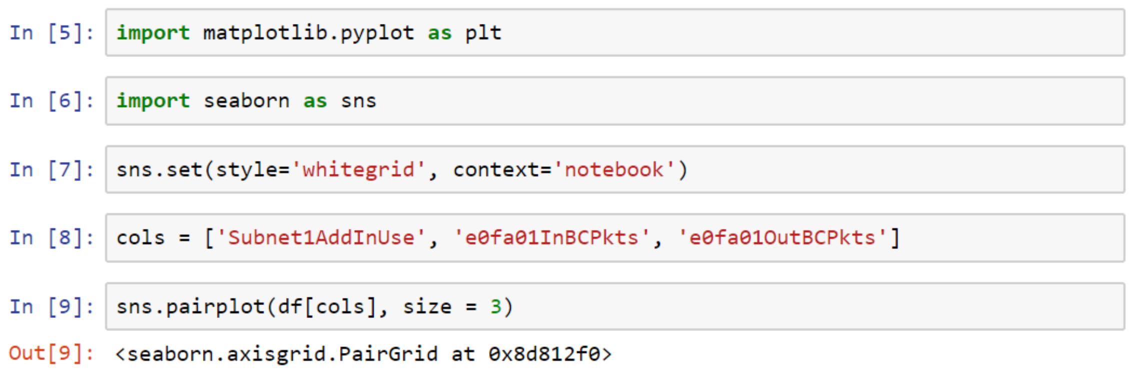
5.1.1 Visualizing the important characteristics of the dataset

Exploratory Data Analysis (EDA) is an important and recommended first before we start training our ML model. I used some simple techniques from the graphical EDA toolbox, which is really useful to visually detect the correlation between features, to see the distribution of the data and so on.

A *scatterplot matrix* allows us to visualize the pair-wise correlations between the different features of the dataset in one place. For this we will use the *pairplot* function of the *seaborn* library. Seaborn is a Python library for drawing statistical plots based on matplotlib.

On the following picture I plotted only 3x3 scatterplot matrix. I made lots of experiments by plotting different features. As you can see above, we have 58 features in our dataset. To plot a 58x58 scatterplot matrix requires so much resources, that I had to shrink the scope. So I had some presumptions which data features should be visualized.

As I described earlier, DHCP uses Broadcast messages for communication. So it makes sense in our scenario to visualize the IP address usage related to the router interface measurements, which the DHCP server is connected to.



```
In [5]:  import matplotlib.pyplot as plt

In [6]:  import seaborn as sns

In [7]:  sns.set(style='whitegrid', context='notebook')

In [8]:  cols = ['Subnet1AddInUse', 'e0fa01InBCPkts', 'e0fa01OutBCPkts']

In [9]:  sns.pairplot(df[cols], size = 3)
Out[9]:  <seaborn.axisgrid.PairGrid at 0x8d812f0>
```

Figure 5.3.: Shrinking the scope to create a small scatterplot matrix

After I realized how easy to see the connections between the BC packets and the IP address usage I plotted the graph for the different subnets and for the total numbers of IP Addresses as well as I show it on the following pages.

```
In [10]: plt.show()
```

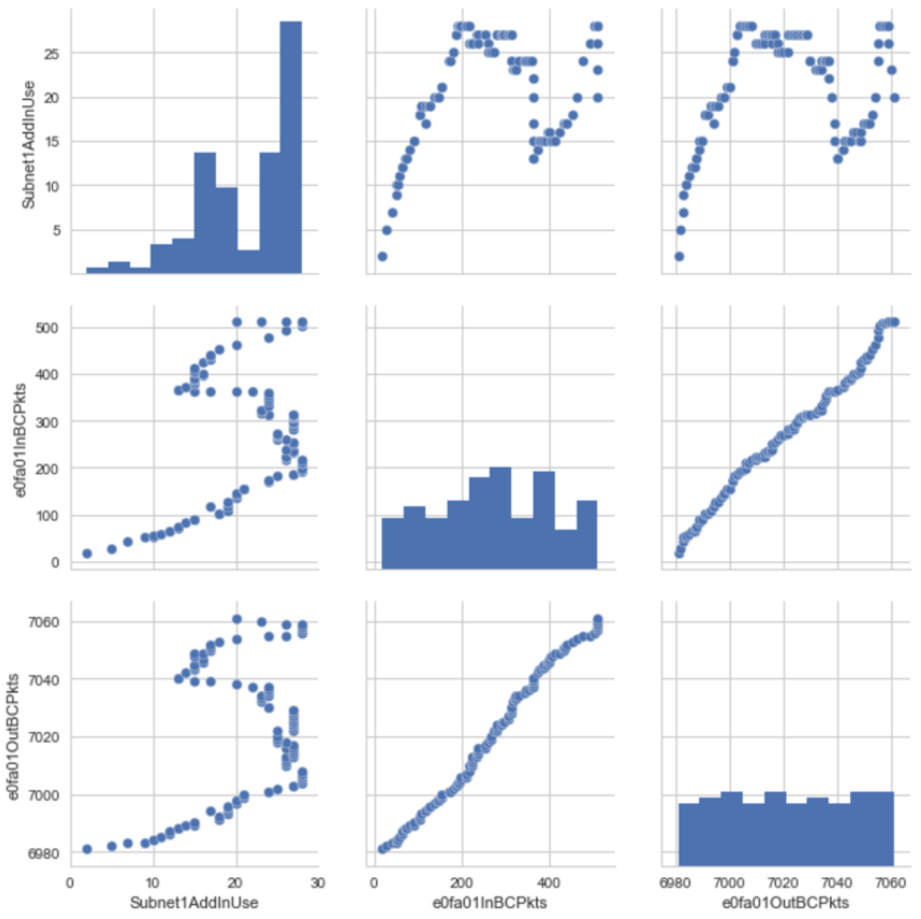


Figure 5.3.: Relations between the first DHCP pool usage and the BC packets

We can see a nonlinear relation on the upper-right two plots. The other plots are useless from our perspective, but it is still interesting to see the closely linear relation between the sent and received (Out and In Pkts) BC messages. In this case we know that in our network there is no other protocol which is using BC for communication. Routing protocols use Multicast and Unicast, the end users generate only Unicast type of traffic except the DHCP messages. So it can be seen on the plots that the linear relationship comes from the client-server type of communication. Discovery-Offer, Request-Acknowledgment, every question has its answer in the communication.

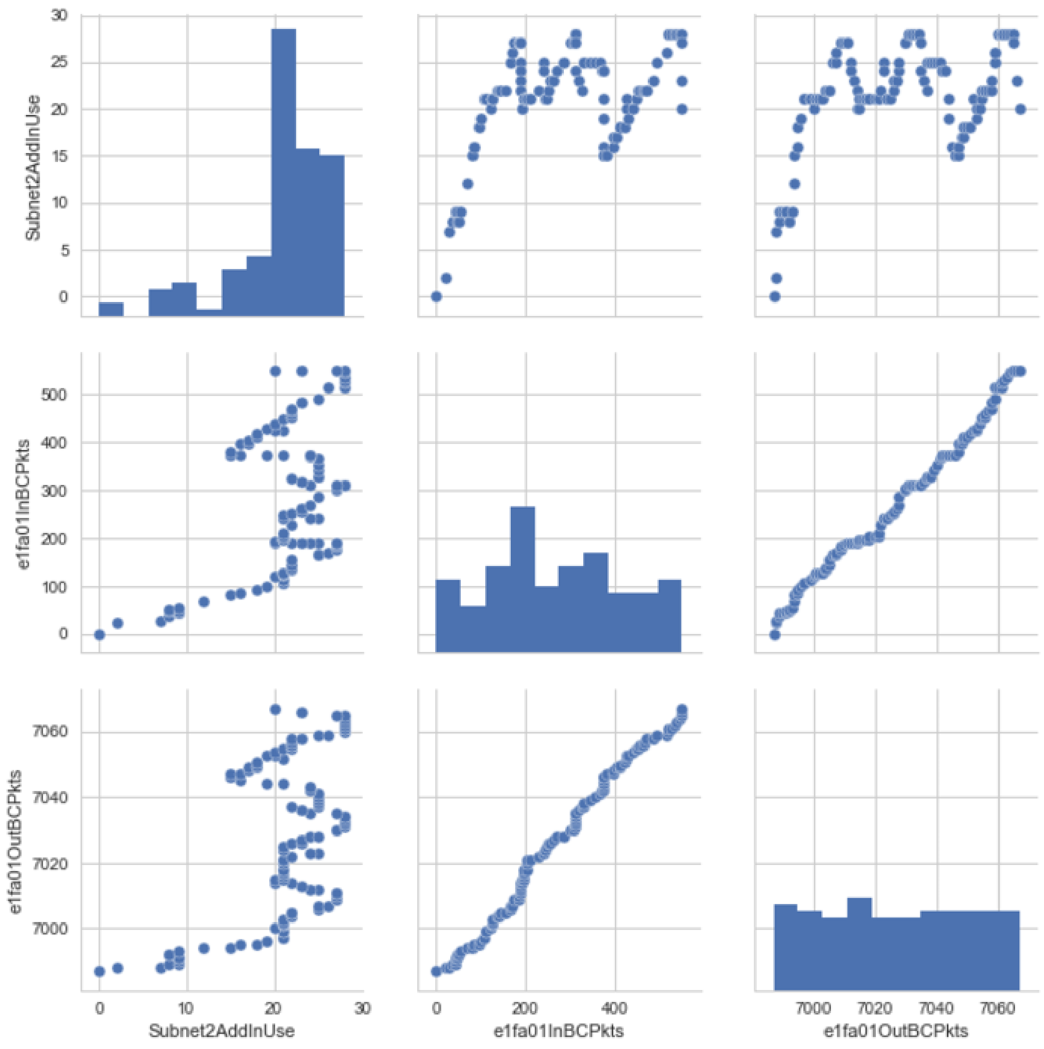


Figure 5.4.: Relations between the second DHCP pool usage and the BC packets

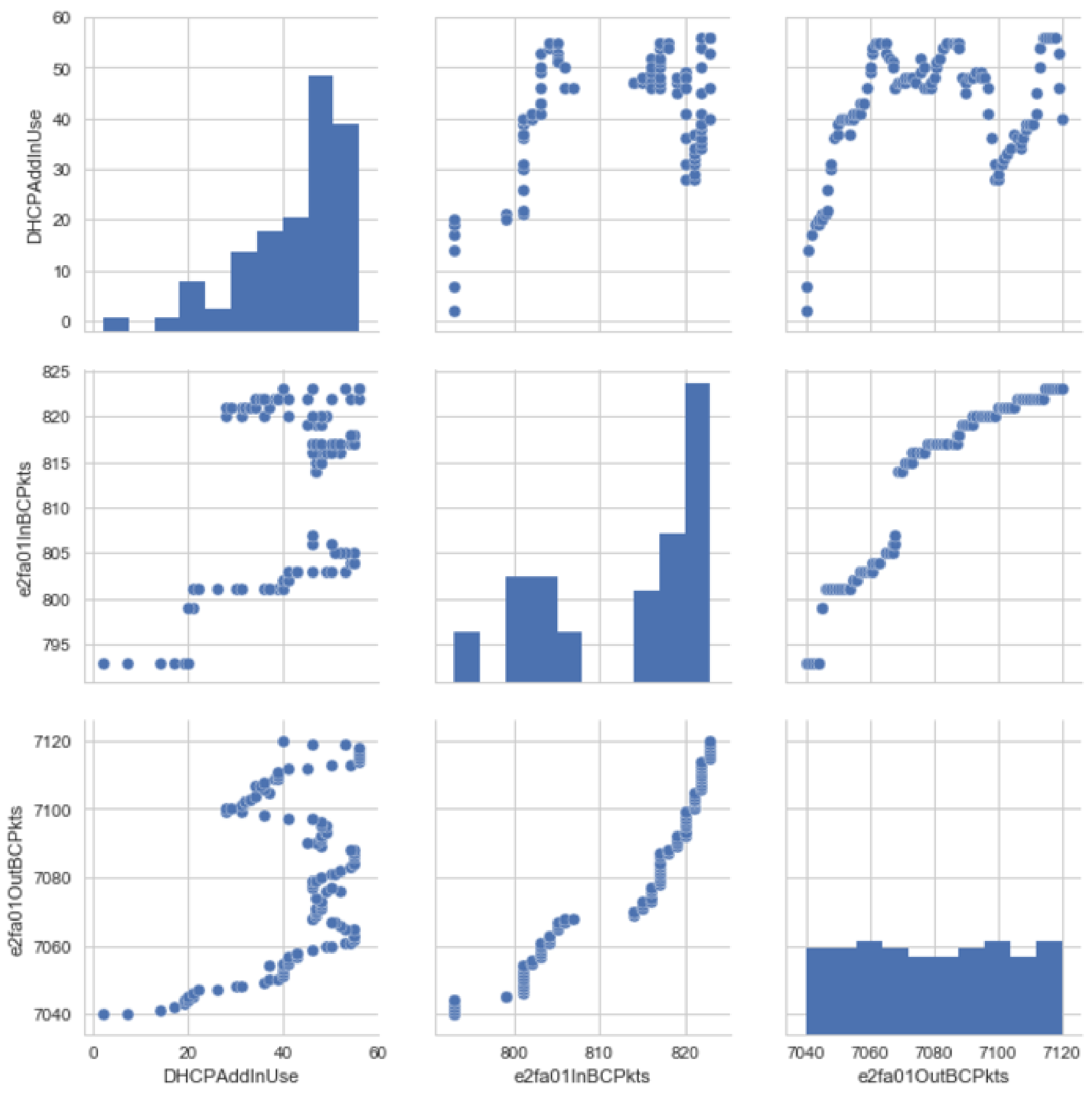
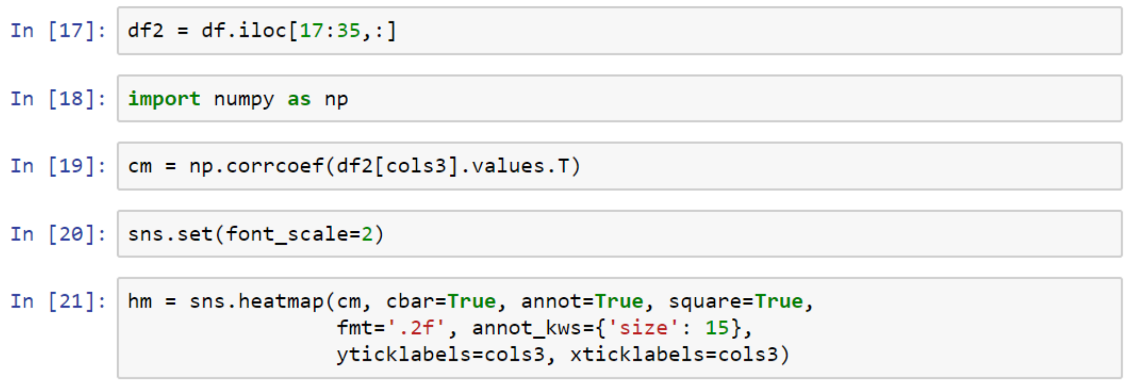


Figure 5.5.: Relations between the sum of IP addresses' usage and the BC packets

We can see that in total that the distribution has changed between the DHCP server answers which are an allocated or released IP address and a sent BC message shown on the second plot of the first row. But we still can see a clear relationship between the BC messages sent out by the router and the allocated IP addresses performed by the DHCP server.

So let me continue the analysis with the upper right plot. Since we want to use for the first time linear- and nonlinear-regression to predict the values, we need to focus on the slopes of the plot. In the following examples I will shrink the dataset further, and I reduce the samples concentrating for only one slope.

To quantify the linear relationship between the features we will create a *correlation matrix*. The *correlation coefficients* are bounded to the range -1 and 1. Two features have perfect positive correlation if r = 1, no correlation if r = 0, and perfect negative correlation if r = -1. Now we can use NumPy's *corrcoef* function and visualize a so-called *heatmap* for the reduced dataset.



```
In [17]: df2 = df.iloc[17:35,:]

In [18]: import numpy as np

In [19]: cm = np.corrcoef(df2[cols3].values.T)

In [20]: sns.set(font_scale=2)

In [21]: hm = sns.heatmap(cm, cbar=True, annot=True, square=True,
                          fmt='.2f', annot_kws={'size': 15},
                          yticklabels=cols3, xticklabels=cols3)
```

Figure 5.6.: Creating a heatmap to show the correlation between the features

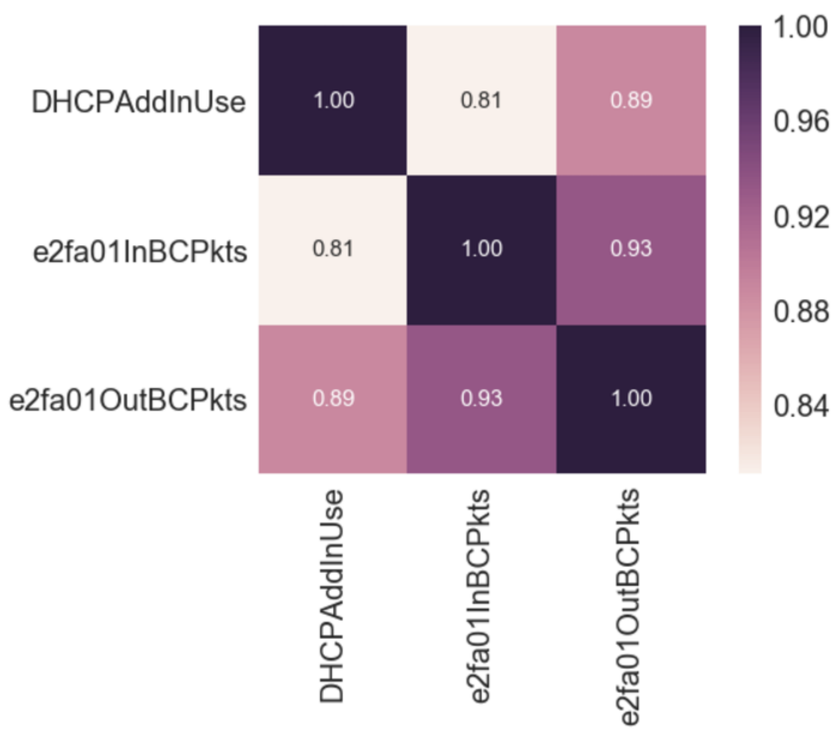


Figure 5.7.: A heatmap in case of increasing IP allocation

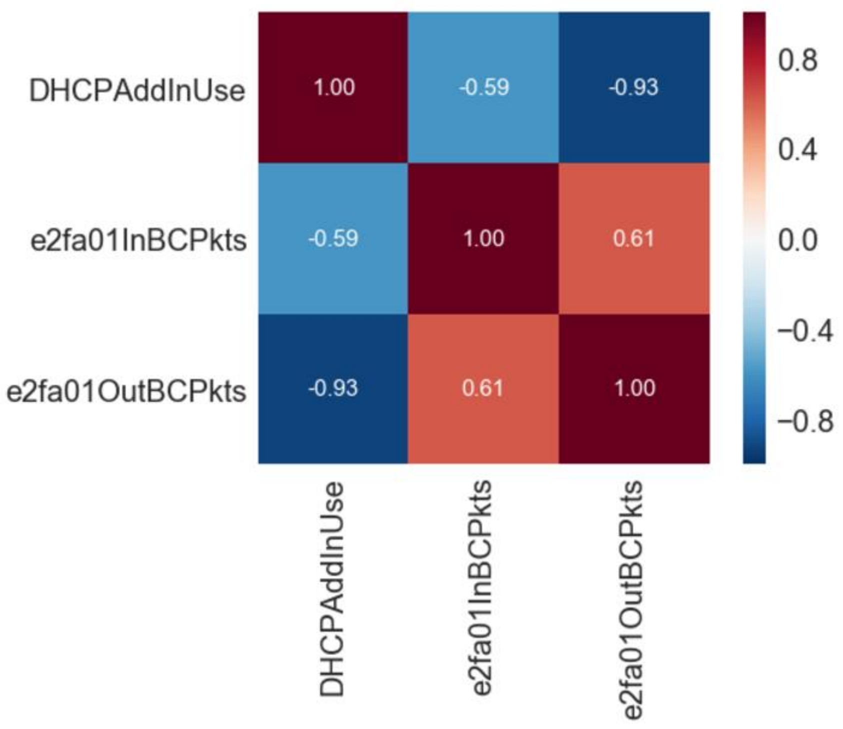


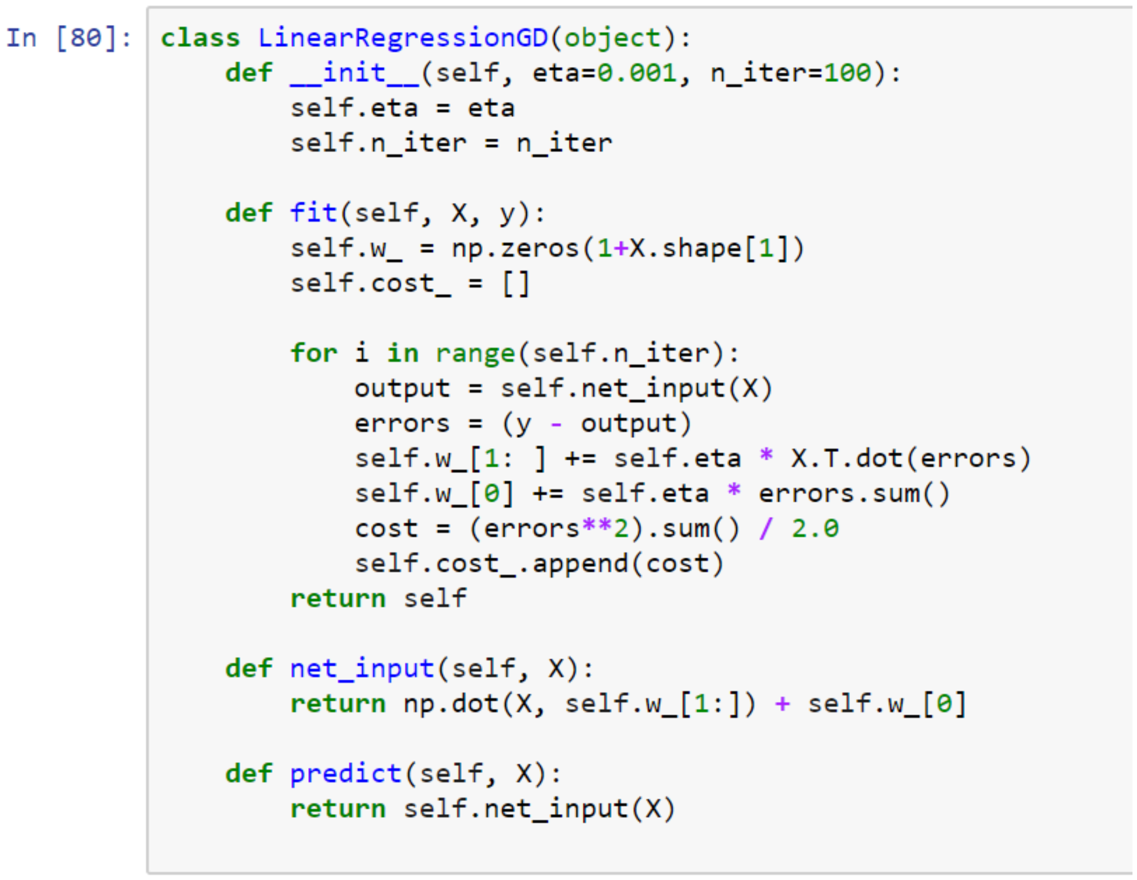
Figure 5.8.: A heatmap in case of decreasing IP allocation

We can see the high correlation between the allocated or released IP addresses and the sent BC messages towards the DHCP server. So we can see now, it makes sense to continue with these test cases and use the regression models for these to predict the next value of the allocated IP Addresses.

5.2 Applying ML algorithms on the measured networking data

5.2.1 Implementing an Ordinary Least Squares linear regression model

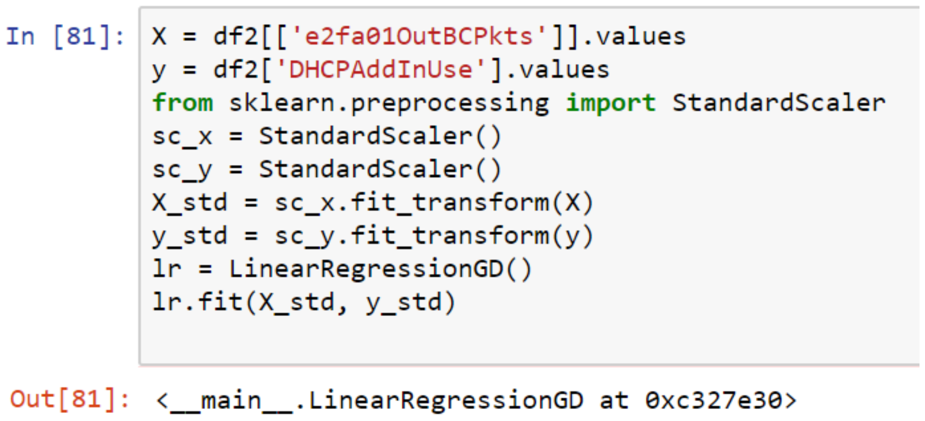
In the chapter 3.2. I described how the Adaline and the OLS model works. We understood that if we remove the unit step function from the Adaline, we can get an OLS model. Let us see, how a basic implementation looks alike of the OLS with the Gradient Descent weight optimization algorithm.



```
In [80]: class LinearRegressionGD(object):
             def __init__(self, eta=0.001, n_iter=100):
                 self.eta = eta
                 self.n_iter = n_iter

             def fit(self, X, y):
                 self.w_ = np.zeros(1+X.shape[1])
                 self.cost_ = []

                 for i in range(self.n_iter):
                     output = self.net_input(X)
                     errors = (y - output)
                     self.w_[1: ] += self.eta * X.T.dot(errors)
                     self.w_[0] += self.eta * errors.sum()
                     cost = (errors**2).sum() / 2.0
                     self.cost_.append(cost)
                 return self

             def net_input(self, X):
                 return np.dot(X, self.w_[1:]) + self.w_[0]

             def predict(self, X):
                 return self.net_input(X)
```

Figure 5.9.: Implementation of a Linear Regression model

To see how this model works in action, we use it for our simplified dataset. We need to standardize our variables for better convergence of the GD algorithm, so here we use the *StandardScaler* function of scikit-learn.



```
In [81]: X = df2[['e2fa01OutBCPkts']].values
         y = df2['DHCPAddInUse'].values
         from sklearn.preprocessing import StandardScaler
         sc_x = StandardScaler()
         sc_y = StandardScaler()
         X_std = sc_x.fit_transform(X)
         y_std = sc_y.fit_transform(y)
         lr = LinearRegressionGD()
         lr.fit(X_std, y_std)

Out[81]: <__main__.LinearRegressionGD at 0xc327e30>
```

Figure 5.10.: Using the model for the networking dataset

It looks a good idea to plot the cost as a function of the number of the epochs, when we are using optimization algorithms, such as gradient descent to check the convergence. The following plot shows us, that how many cycles are needed for the model to converge, in other words how many cycles needs the model to pass over and over the training dataset until the convergence happens.

```
In [82]: plt.plot(range(1, lr.n_iter+1), lr.cost_)
         plt.ylabel('SSE')
         plt.xlabel('Epoch')
         plt.show()
```

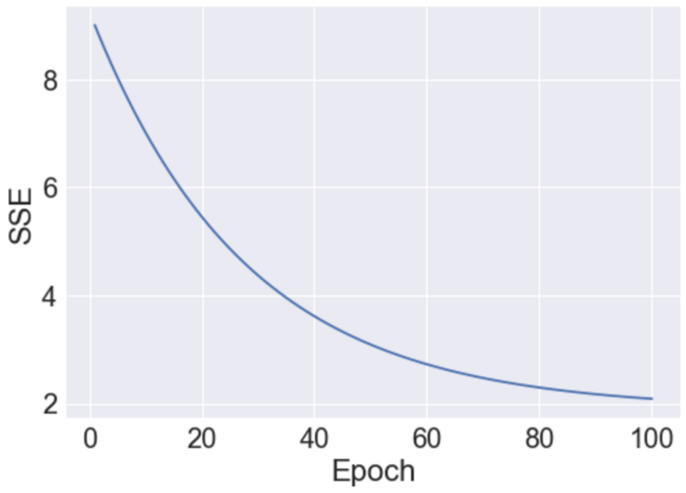


Figure 5.11.: Convergence of the OLS model

Now we can use the *lin_regplot* function to plot the number of sent BC messages against the IP address allocations:

```
In [83]: def lin_regplot(X, y, model):
             plt.scatter(X, y, c='blue')
             plt.plot(X, model.predict(X), color='red')
             return None
```

```
In [84]: lin_regplot(X_std, y_std, lr)
         plt.xlabel('e2fa01OutBCPkts (std)')
         plt.ylabel('DHCPAddInUse (std)')
         plt.show()
```

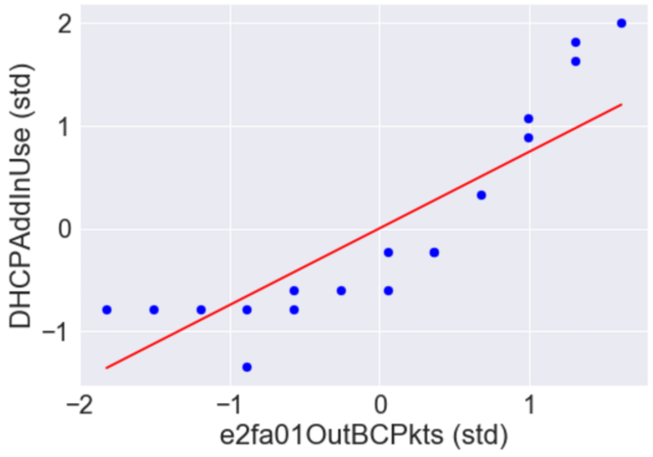
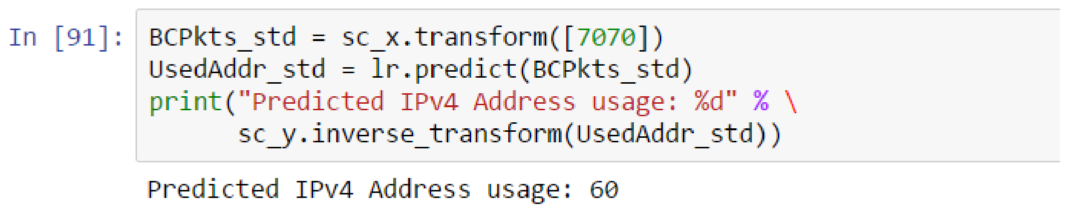


Figure 5.11.: The Linear Regression model fits the training data

As we can see this model does not fit well on the training data or explain the relationship between the features. But it clearly shows the increasing nature of the IP address allocation. Let us see, how the prediction works, after transforming back the data:

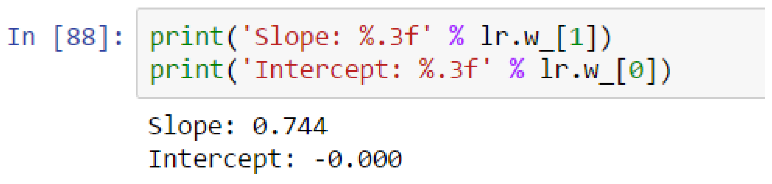


```
In [91]: BCPkts_std = sc_x.transform([7070])
         UsedAddr_std = lr.predict(BCPkts_std)
         print("Predicted IPv4 Address usage: %d" % \
               sc_y.inverse_transform(UsedAddr_std))

         Predicted IPv4 Address usage: 60
```

Figure 5.12.: Predicting output value with the implemented Linear Regression model

In this example we tried to predict the number of allocated addresses if the BC message counter of the Fa0/1 interface on the edge2-isp router reaches the 7070 value. From the original database we can see, that this value is only 47, so we need further refinements of the model.

On a side note, it is worth mentioning that we do not have to update the weights of the intercept if we work with standardized variables. The y axis intercept is always 0 in these cases. Let me prove it by printing the weights:



```
In [88]: print('Slope: %.3f' % lr.w_[1])
         print('Intercept: %.3f' % lr.w_[0])

         Slope: 0.744
         Intercept: -0.000
```

Figure 5.13.: the weights of the Linear Regression model (in case of standardization)

5.2.2 Using the Linear Regression model of Scikit-Learn

As we can see above, we need more efficient implementations of the Linear Regression model, so let us try scikit-lear's *LinearRegression* object, that uses the LIBLINEAR library and advanced optimization algorithms that work better with unstandardized variables.

```
In [89]: from sklearn.linear_model import LinearRegression
         slr = LinearRegression()
         slr.fit(X, y)
         print('Slope: %.3f' % slr.coef_[0])
         print('Intercept: %.3f' % slr.intercept_)

         Slope: 1.500
         Intercept: -10537.840
```

Figure 5.14.: Fitting the network dataset to the sklearn Linear Regression model and displaying the weights

If we plot how the model fits the training dataset, we can see that the result looks identical, to our own implementation, but fits better a bit.

```
In [90]: lin_regplot(X, y, slr)
         plt.xlabel('e2fa01OutBCPkts')
         plt.ylabel('DHCPAddInUse')
         plt.show()
```

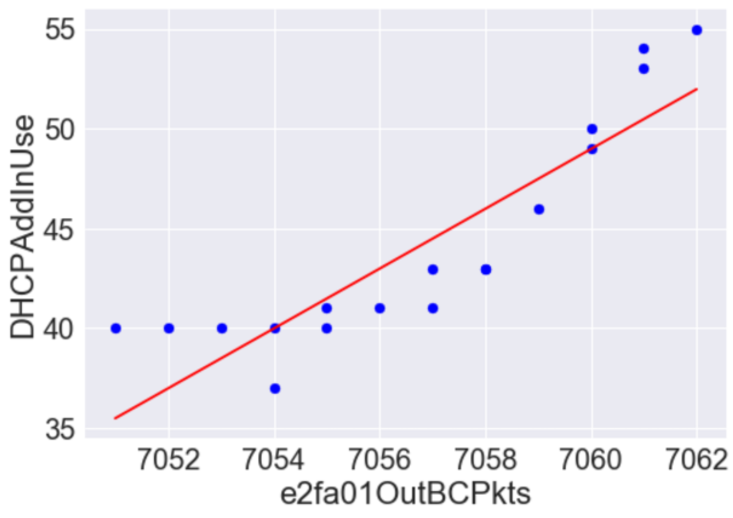
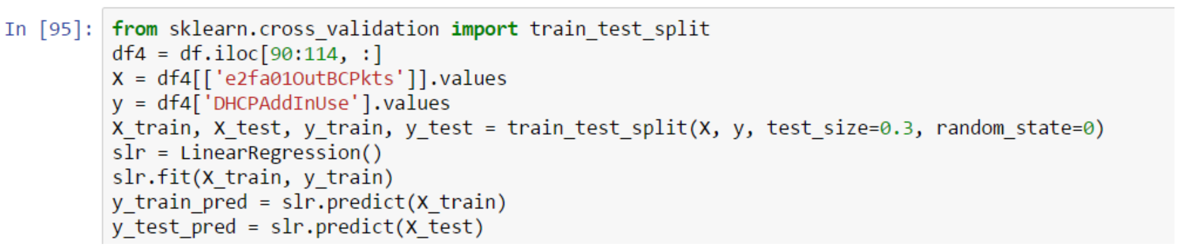


Figure 5.15.: sklearn LinearRegression model on the networking dataset

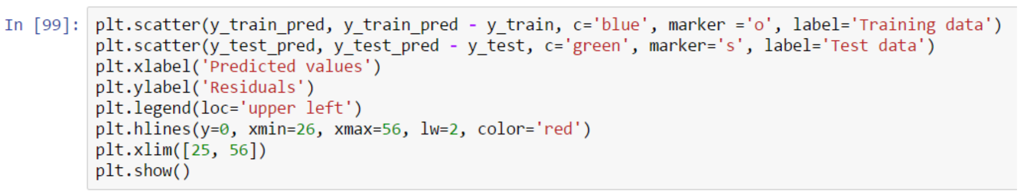
It is instructive to show, how this LinearRegression model works with a training and test set of data. In the following example I imported more samples of the whole dataset, where the IP address usage shows an increasing trend. I splitted the data, 70% of the samples is used for training and the other 30% is used for test. Then I collected the predicted data in the *y_train_pred* and *y_test_pred* vectors.



```
In [95]: from sklearn.cross_validation import train_test_split
         df4 = df.iloc[90:114, :]
         X = df4[['e2fa01OutBCPkts']].values
         y = df4['DHCPAddInUse'].values
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
         slr = LinearRegression()
         slr.fit(X_train, y_train)
         y_train_pred = slr.predict(X_train)
         y_test_pred = slr.predict(X_test)
```

Figure 5.16.: Creating training and test splits from the dataset

In the following figure I plotted the residuals against the predicted values, to see how big mistakes have been done by the model:



```
In [99]: plt.scatter(y_train_pred, y_train_pred - y_train, c='blue', marker ='o', label='Training data')
         plt.scatter(y_test_pred, y_test_pred - y_test, c='green', marker='s', label='Test data')
         plt.xlabel('Predicted values')
         plt.ylabel('Residuals')
         plt.legend(loc='upper left')
         plt.hlines(y=0, xmin=26, xmax=56, lw=2, color='red')
         plt.xlim([25, 56])
         plt.show()
```

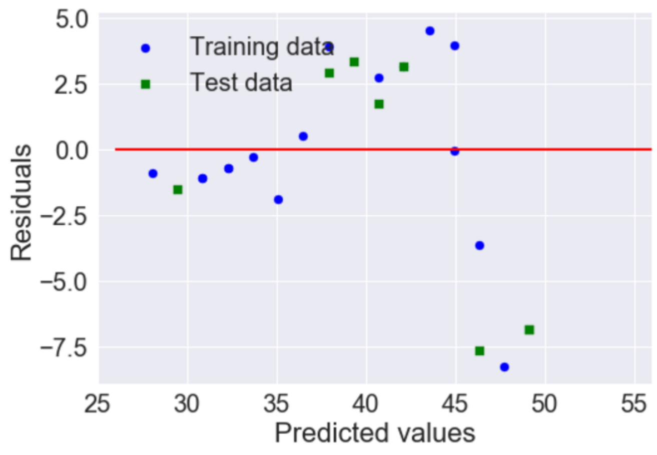
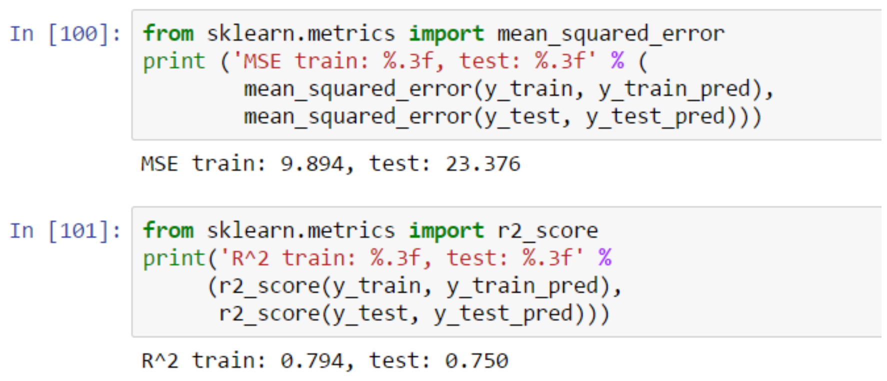


Figure 5.17.: Evaluating the performance of the sklearn LinearRegression model

In the case of perfect prediction the residuals would be exactly zero. For a good regression model we expect, that the errors are randomly distributed as I mentioned earlier. From this viewpoint, this model seems to work fine.

We also mentioned earlier the R^2 and *MSE* values. Our model is better when *MSE* is closer to 0 and R^2 is closer to 1. Let me calculate these values for the current case:

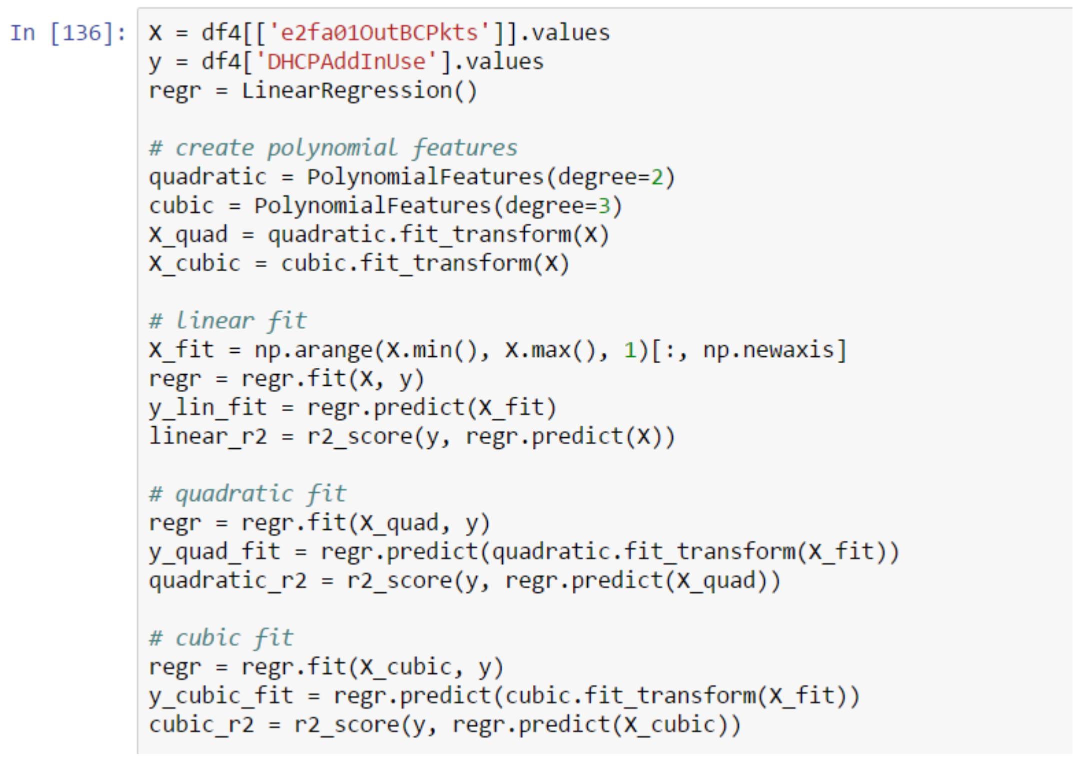


```
In [100]: from sklearn.metrics import mean_squared_error
          print ('MSE train: %.3f, test: %.3f' % (
                  mean_squared_error(y_train, y_train_pred),
                  mean_squared_error(y_test, y_test_pred)))

          MSE train: 9.894, test: 23.376

In [101]: from sklearn.metrics import r2_score
          print('R^2 train: %.3f, test: %.3f' %
               (r2_score(y_train, y_train_pred),
                r2_score(y_test, y_test_pred)))

          R^2 train: 0.794, test: 0.750
```

Figure 5.18.: Calculating the mean squared errors and the coefficients of determination for the training and test dataset

5.2.3 Modelling nonlinear relationships

In this section I use the *PolynomialFeatures* class from sklearn to show quadratic and cubic fit on the same dataset. I will compare the results in one plot.



```
In [136]: X = df4[['e2fa01OutBCPkts']].values
          y = df4['DHCPAddInUse'].values
          regr = LinearRegression()

          # create polynomial features
          quadratic = PolynomialFeatures(degree=2)
          cubic = PolynomialFeatures(degree=3)
          X_quad = quadratic.fit_transform(X)
          X_cubic = cubic.fit_transform(X)

          # linear fit
          X_fit = np.arange(X.min(), X.max(), 1)[:, np.newaxis]
          regr = regr.fit(X, y)
          y_lin_fit = regr.predict(X_fit)
          linear_r2 = r2_score(y, regr.predict(X))

          # quadratic fit
          regr = regr.fit(X_quad, y)
          y_quad_fit = regr.predict(quadratic.fit_transform(X_fit))
          quadratic_r2 = r2_score(y, regr.predict(X_quad))

          # cubic fit
          regr = regr.fit(X_cubic, y)
          y_cubic_fit = regr.predict(cubic.fit_transform(X_fit))
          cubic_r2 = r2_score(y, regr.predict(X_cubic))
```

Figure 5.19.: Creating nonlinear regression models

On the figure below we can see the linear, quadratic and cubic models. The calculated R^2 values are also shown.

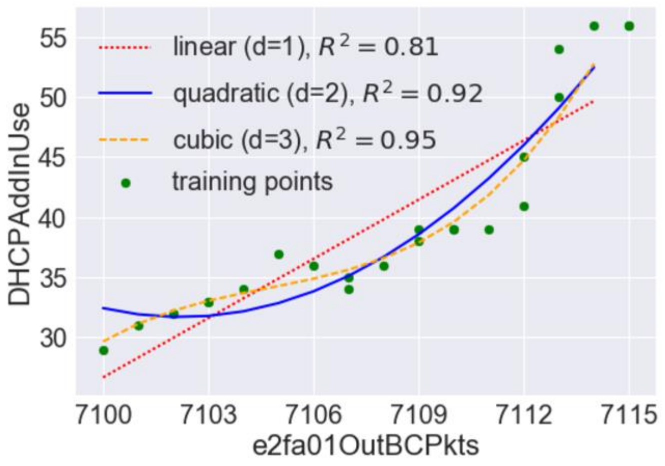


Figure 5.20.: Linear and nonlinear regression comparison

The nonlinear models are working really well with the networking dataset. We can see that the R^2 values are close to 1. All in all we can declare that for optimal prediction we can use the quadratic and cubic regression models. Of course, there is always a chance to increase the degree of the polynomial, but we have to take care of our hardware resources and calculation time as well, when we work with big databases.

5.2.4 Using decision tree regression

Decision tree algorithm does not require any transformation of the features. We can easily use it for the whole dataset, so it can be a better model instead of manually shrinking the dataset based on the steepness of the slopes:

```
In [143]: from sklearn.tree import DecisionTreeRegressor
          X = df[['e2fa01OutBCPkts']].values
          y = df['DHCPAddInUse'].values
          tree = DecisionTreeRegressor(max_depth=5)
          tree.fit(X, y)
          sort_idx = X.flatten().argsort()
          lin_regplot(X[sort_idx], y[sort_idx], tree)
          plt.xlabel('e2fa01OutBCPkts')
          plt.ylabel('DHCPAddInUse')
          plt.show()
```

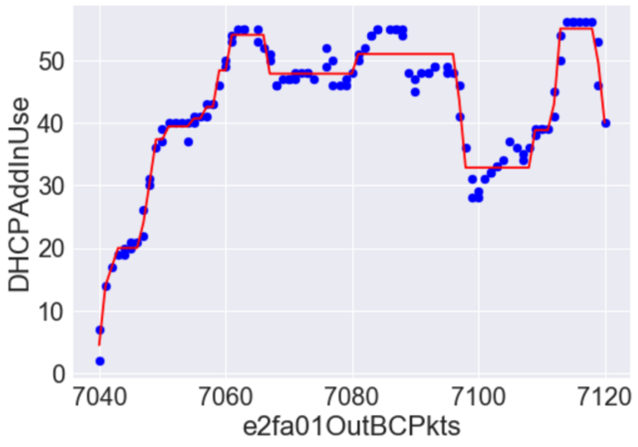


Figure 5.21.: Using decision tree regression on the networking dataset

With this depth the fitted line is really rough especially in the [7080:7100] range. But as we can see the fitted line tracks well the whole dataset, so just go deeper with the random forest regression.

5.2.5 Random forest regression as the final solution

So let us see, how the random forest algorithm, which we talked about in Chapter 3.2. performs with the whole dataset.

```
In [144]: X = df[['e2fa01OutBCPkts']].values
          y = df['DHCPAddInUse'].values
          X_train, X_test, y_train, y_test =\
              train_test_split(X, y, test_size=0.4,
                               random_state=1)
          from sklearn.ensemble import RandomForestRegressor
          forest = RandomForestRegressor(n_estimators=1000,
                                         criterion='mse',
                                         random_state=1,
                                         n_jobs=-1)
          forest.fit(X_train, y_train)
          y_train_pred = forest.predict(X_train)
          y_test_pred = forest.predict(X_test)
          print('MSE train: %.3f, test: %.3f' % (
                  mean_squared_error(y_train, y_train_pred),
                  mean_squared_error(y_test, y_test_pred)))
          print('R^2 train: %.3f, test: %.3f' % (
                  r2_score(y_train, y_train_pred),
                  r2_score(y_test, y_test_pred)))

          MSE train: 1.552, test: 7.508
          R^2 train: 0.989, test: 0.917
```

Figure 5.22.: MSE and R^2 calculation of the random rorest regression model

We can see that the model still overfits the training data, but we got so far the best scores compared to the others. In the following figure I show the errors of the prediction and we will see the improvement.

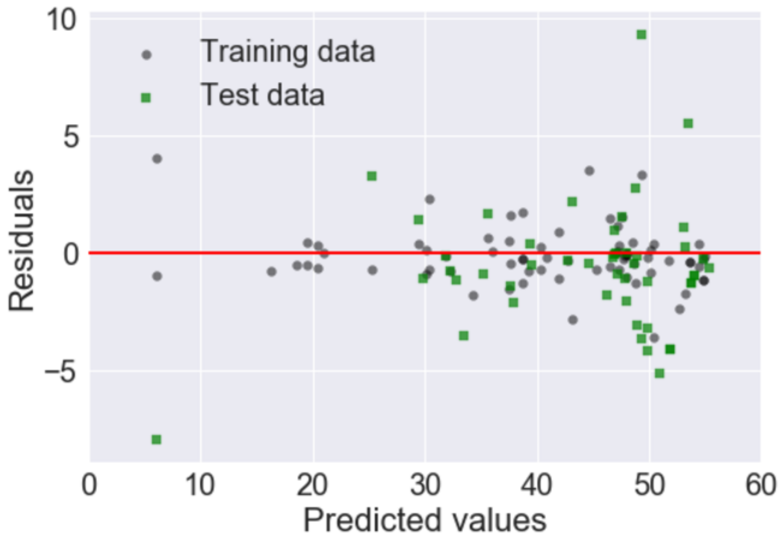


Figure 5.23.: The errors of prediction for training and test data using random forest regression

Finally, we have to notice, that my dataset was quite small compared to other datasets that are usually analyzed. Knowing this we can have the presumption, that these algorithms can perform well in a real network environment and can learn the behaviour of the network giving good enough predictions about the problematic events. At least this kind of approach to the network operation can enlighten the hope of the proactive management and may maximalize the customer experience.

Conclusions

The sections above showed clearly how powerful ML can be processing data produced by IP networks. All the python libraries were easy to use without any modification of the functions and to be honest, implementing the measurement system and collecting the necessary data were the most complicated tasks.

I have to admit, that the platforms (SW images in my case) and networking technologies/solutions I used are more than 10 years old, and since these were not prepared for feeding big databases I had to spend more time on data preprocessing than I expected. But dealing with all the problems were not a waste of time, because in most of the cases this is the technology which is still used by network operators - of course they have newer equipments and management softwares, but the basics and logics are more or less the same.

However my dataset was a bit small and I focused only on IP address pool utilization, I have good impressions about how useful these predictions can be. All kind of regression algorithms could showed good results and as I improved my model with different regression algorithms, I was always able to reduce the error of predictions.

In the future I plan to extend my model to more networking use cases and to involve datasets produced by real networks. I hope that NETCONF/Yang models (today it seems to be the new standard for describing networking devices and their behavior) will be unified and supported by most of the vendors soon, so it will be much easier to collect standardized information and step in the world of data driven network management.

Bibliography

[1] J. Postel: *Internet Protocol,* RFC791, Internet Engineering Task Force, September 1981

[2] S. Thomson, T. Narten, T. Jinmei: *IPv6 Stateless Address Autoconfiguration*, RFC4862, Internet Engineering Task Force, September 2007

[3] R.Droms: *Dynamic Host Configuration Protocol,* RF2131, Internet Engineering Task Force, March 1997

[4] R. Rdoms, Ed., J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney: *Dynamic Host Configuration Protocol for IPv6 (DHCPv6),* RFC3315, Internet Engineering Task Force, July 2003

[5] J. Moy: *OSPF Version 2,* RFC2328, Internet Engineering Task Force, April 1998

[6] A. Lindem, Ed., S. Mirtorabi, A. Roy, M. Barnes, R. Aggarwal: *Support of Address families in OSPFv3,* RFC5838, Internet Engineering Task Force, April 2010

[7] Y. Rekhter, T. Li: *A Border Gateway Protocol 4 (BGP-4),* RFC1771, Internet Engineering Task Force, March 1995

[8] E. Rosen, A. Viswanathan, R. Callon: *Multiprotocol Label Switching Architecture,* RFC3031, Internet Engineering Task Force, January 2001

[9] M. Bjorklund, Ed..: *YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF),* RFC6020, Internet Engineering Task Force, October 2010

[10] J.D. Case, M. Fedor, M. L. Schoffstall, J. Davin: *Simple Network Management Protocol (SNMP),* RFC1157, Internet Engineering Task Force, May 1990

[11] Ansible: http://www.ansible.com (revision 21:00, 20 May 2017)

[12] K.McCloghrie, M. Rose: *Management Information Base for Network Management of TCP/IP-based internets: MIB-II,* RFC1213, Internet Engineering Task Force, March 1991

[13] B. Claise, Ed..: *Cisco Systems NetFlow Services Export Version 9.,* RFC3954, Internet Engineering Task Force, October 2004

[14] C. Londvick: *The BSD Syslog Protocol,* RFC3164, Internet Engineering Task Force, August 2001

[15] S. Raschka: *Python Machine Learning,* 1st edition, ISBN 978-1-78355-513-0, Birmingham, 2015

[16] W. S. McCulloch and W. Pitts: *A Logical Calculus of the Ideas Immanent in Nervous Activity*, The bulletin of mathematical biophysics, 5(4):115–133, 1943

[17] F. Rosenblatt: *The Perceptron, a Perceiving and Recognizing Automaton*, Cornell Aeronautical Laboratory, 1957

[18] Cacti: http://www.cacti.net (revision 21:02, 20 May 2017)

[19] RRDTool: http://oss.oetiker.ch/rrdtool/ (revision 21:03, 20 May 2017)

[20] VirtualBox: https://www.virtualbox.org/ (revision 21:04, 20 May 2017)

[21] GNS3: https://www.gns3.com/ (revision 21:05, 20 May 2017)

[22] R. B. Hibbs, G. Waters: *Dynamic Host Configuration Protocol (DHCP) Server MIB*, Internet-draft, Internet Engineering Task Force, November 2000

[23] E. Rosen, Y. Rekhter: *BGP/MPLS IP Virtual Private Networks (VPNs)*, RFC4364, Internet Engineering Task Force, February 2006

[24] VRF-lite: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/12-2/54sg/configuration/guide/config/vrf.pdf (revision 21:13, 20 May 2017)

[25] The Institute of Electrical and Electronics Engineers: *Virtual Bridged Local Area Networks*, IEEE Std 802.1Q, ISBN 0-7381-3663-8, May 2003

[26] J.Arkko, M. Cotton, L. Vegoda: *IPv4 Address Blocks Reserved for Documenation*, RFC5737, Internet Engineering Task Force, January 2010

[27] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear: *Address Allocation for Private Internets*, RFC1918, Internet Engineering Task Force, February 1996

[28] D. Eastlake 3rd, A. Panitz: *Reserved Top Level DNS Names*, RFC2606, Internet Engineering Task Force, June 1999

[29] G. Huston, A. Lord, P. Smith: *IPv6 Address Prefix Reserved for Documentation*, RFC3849, Internet Engineering Task Force, July 2004

[30] G. Huston: *Autonomous System (AS) Number Reservation for Documentation Use*, Internet Engineering Task Force, December 2008

[31] J. Mitchell: *Autonomous System (AS) Number Reservation for Private Use*, RFC6996, Internet Engineering Task Force, July 2013

[32] A. Bruno, S. Jordan: *CCDA 640-864 Official Cert Guide*, 1st edition, ISBN 978-1-58714-257-4, Indianapolis, May 2011

Appendix

Configurations of the networking devices:

```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption      !!! the configured passwords are not !!!
hashed
!
hostname edge2-isp
!
boot-start-marker
boot-end-marker
!
no aaa new-model
memory-size iomem 5
ip cef              !!! enabling CEF to support MPLS and dynamic routing
!
ip vrf mgmt         !!! VRF-lite configuration for mgmt IP-VPN
 rd 64496:10                  !!! route-distinguisher
 route-target export 64496:10
 route-target import 64496:10
!
ip domain name net.isp.example      !!! domain name configuration
!
mpls label protocol ldp             !!! MPLS ldp configuration
multilink bundle-name authenticated
!
archive
 log config
  hidekeys
!
!!! Interface configurations !!!
!
interface Loopback0
 ip address 203.0.113.3 255.255.255.255
!
interface FastEthernet0/0
 description Management
 ip vrf forwarding mgmt
 ip address 192.168.1.73 255.255.255.0
 speed 100
 full-duplex
 no cdp enable
!
interface FastEthernet0/1
 description DHCP_server
 ip address 198.51.100.65 255.255.255.224
 speed 100
 full-duplex
!
interface FastEthernet1/0
 description MPLS core0-isp fa0_0
 ip address 192.0.2.1 255.255.255.254
 ip ospf network point-to-point
 speed 100
 full-duplex
 mpls ip                 !!! enabling mpls on the interface
!
interface FastEthernet2/0
 ip address 203.0.113.8 255.255.255.254
 speed 100
 full-duplex
!
!!! Dynamic Routing Configuration !!!
!
router ospf 64496    !!! unique process ID
 mpls ldp sync       !!! synchronizing the IGP with MPLS
 router-id 203.0.113.3    !!! router-id based on the public IP addressed !!!
Loopback
 log-adjacency-changes
 passive-interface default!!! by default we disable OSPF Hello packets on
!!! all interfaces
 no passive-interface FastEthernet1/0
 no passive-interface Loopback0
 network 192.0.2.0 0.0.0.1 area 0   !!! networks should be advertised
 network 192.168.1.0 0.0.0.255 area 40
 network 198.51.100.64 0.0.0.3 area 30
 network 203.0.113.3 0.0.0.0 area 0
!
router bgp 64496     !!! BGP configuration, identifying the ASN
 bgp router-id 203.0.113.3     !!! unique router-id as above
 bgp log-neighbor-changes      !!! logging adjacency changes in the syslog
 neighbor 203.0.113.1 remote-as 64496    !!! iBGP neighbor - note that the
!!! ASN is the same
 neighbor 203.0.113.1 description edge0-isp
 neighbor 203.0.113.1 update-source Loopback0
 neighbor 203.0.113.2 remote-as 64496
 neighbor 203.0.113.2 description edge1-isp
 neighbor 203.0.113.2 update-source Loopback0
 neighbor 203.0.113.9 remote-as 64497 !!! eBGP configuration -  note ASN
 neighbor 203.0.113.9 description Internet
 neighbor 203.0.113.9 update-source Loopback0
 !
 address-family ipv4           !!! ipv4 AF configuration
  redistribute connected route-map redist-conn !!! redistributing only the
!!! allowed connected routes
  neighbor 203.0.113.1 activate          !!! activating BGP neighborship
  neighbor 203.0.113.1 default-originate !!! propagating default route to
!!! the neighbor
  neighbor 203.0.113.2 activate
  neighbor 203.0.113.2 default-originate
  neighbor 203.0.113.9 activate
  no auto-summary
  no synchronization
 exit-address-family
!
ip forward-protocol nd
!
no ip http server    !!! disabling the TCP port 80 from security reasons
no ip http secure-server  !!! disabling TCP port 8080
!
!!! prefix-list that describes the allowed prefixes for redistribution
!
```

```
ip prefix-list red-conn seq 10 permit 192.0.2.0/31
ip prefix-list red-conn seq 20 permit 198.51.100.64/30
ip prefix-list red-conn seq 30 permit 203.0.113.3/32
ip prefix-list red-conn seq 40 permit 192.168.1.0/24
ip prefix-list red-conn seq 50 deny 0.0.0.0/0
!
!!! configuring SNMP Agent for Manager community with read-only privileges
!
snmp-server community Manager RO
!
!!! route-map for describing what should we do with the matching prefixes
!!! specified in the prefix-set above
!
route-map redist-conn permit 10
 match ip address prefix-list red-conn
!
control-plane
!
!!! configuring console, aux and terminal access
!!! mandatory in real networks, but in this simple model we didn't secure
!!! anything
line con 0
line aux 0
line vty 0 4
!
end
```

Configuration 4.1.: edge2-isp running-config

```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname edge1-isp
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
memory-size iomem 5
ip cef
!
ip vrf mgmt
 rd 64496:10
 route-target export 64496:10
 route-target import 64496:10
!
ip domain name net.isp.example
ip auth-proxy max-nodata-conns 3
ip admission max-nodata-conns 3
!
mpls label protocol ldp
!
interface Loopback0
 ip address 203.0.113.2 255.255.255.255
!
```

```
interface FastEthernet0/0
 description MPLS core0-isp fa1_1
 ip address 192.0.2.5 255.255.255.254
 ip ospf network point-to-point
 speed 100
 full-duplex
 mpls ip
!
interface FastEthernet0/1
 description Customers
 ip address 198.51.100.62 255.255.255.224      !!! GW for subscribers
 ip helper-address 198.51.100.66               !!! DHCP relay configuration
 duplex auto
 speed auto
!
interface FastEthernet1/0
 description Management
 ip vrf forwarding mgmt
 ip address 192.168.1.72 255.255.255.0
 duplex auto
 speed auto
!
router ospf 64496
 mpls ldp sync
 router-id 203.0.113.2
 log-adjacency-changes
 passive-interface default
 no passive-interface FastEthernet0/0
 no passive-interface Loopback0
 network 192.0.2.4 0.0.0.1 area 0
 network 198.51.100.32 0.0.0.31 area 20
 network 203.0.113.2 0.0.0.0 area 0
!
router bgp 64496
 bgp router-id 203.0.113.2
 bgp log-neighbor-changes
 neighbor 203.0.113.1 remote-as 64496
 neighbor 203.0.113.1 description edge0-isp
 neighbor 203.0.113.1 update-source Loopback0
 neighbor 203.0.113.3 remote-as 64496
 neighbor 203.0.113.3 description edge2-isp
 neighbor 203.0.113.3 update-source Loopback0
 !
 address-family ipv4
  redistribute connected route-map redist-conn
  neighbor 203.0.113.1 activate
  neighbor 203.0.113.3 activate
  no auto-summary
  no synchronization
 exit-address-family
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
ip prefix-list red-conn seq 10 permit 192.0.2.4/31
ip prefix-list red-conn seq 20 permit 198.51.100.32/27
ip prefix-list red-conn seq 30 permit 203.0.113.2/32
ip prefix-list red-conn seq 40 deny 0.0.0.0/0
snmp-server community Manager RO
!
route-map redist-conn permit 10
 match ip address prefix-list red-conn
!
control-plane
!
line con 0
line aux 0
line vty 0 4
 login
!
!
end
```

Configuration 4.2.: edge1-isp running-config

```
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname core0-isp
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
!
ip vrf mgmt
 rd 64496:10
 route-target export 64496:10
 route-target import 64496:10
!
no ip domain lookup
ip domain name isp.net.example
ip cef
no ipv6 cef
!
mpls label protocol ldp
multilink bundle-name authenticated
!
interface Loopback0
 ip address 203.0.113.0 255.255.255.255
!
interface FastEthernet0/0
 description MPLS edge2-isp fa0_1
 ip address 192.0.2.0 255.255.255.254
 ip ospf network point-to-point
 speed auto
 duplex auto
 mpls ip
!
interface FastEthernet0/1
 description Management
 ip vrf forwarding mgmt
 ip address 192.168.1.70 255.255.255.0
 speed auto
 duplex auto
 no cdp enable
!
interface FastEthernet1/0
 description MPLS edge0-isp fa0_0
 ip address 192.0.2.2 255.255.255.254
 ip ospf network point-to-point
 speed auto
 duplex auto
 mpls ip
!
interface FastEthernet1/1
 description MPLS edge1-isp fa1_1
 ip address 192.0.2.4 255.255.255.254
 ip ospf network point-to-point
 speed auto
 duplex auto
 mpls ip
!
router ospf 64496
 router-id 203.0.113.0
 passive-interface default
 no passive-interface FastEthernet0/0
 no passive-interface FastEthernet1/0
 no passive-interface FastEthernet1/1
 no passive-interface Loopback0
!
!!! because of the proper IP addressing, we can use aggregation here
!
 network 192.0.2.0 0.0.0.7 area 0
 network 203.0.113.0 0.0.0.0 area 0
 mpls ldp sync
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
snmp-server community Manager RO
!
control-plane
!
line con 0
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 login
!
end
```

Configuration 4.3.: core0-isp running-config

```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
```

```
no service password-encryption
!
hostname Internet
!
boot-start-marker
boot-end-marker
!
no aaa new-model
memory-size iomem 5
ip cef
!
ip auth-proxy max-nodata-conns 3
ip admission max-nodata-conns 3
!
interface Loopback1
 ip address 1.1.1.1 255.255.255.255
!
interface Loopback2
 ip address 2.2.2.2 255.255.255.255
!
interface Loopback3
 ip address 3.3.3.3 255.255.255.255
!
interface Loopback4
 ip address 4.4.4.4 255.255.255.255
!
interface Loopback5
 ip address 5.5.5.5 255.255.255.255
!
interface Loopback6
 ip address 6.6.6.6 255.255.255.255
!
interface Loopback7
 ip address 7.7.7.7 255.255.255.255
!
interface Loopback8
 ip address 8.8.8.8 255.255.255.255
!
interface FastEthernet0/0
 ip address 203.0.113.9 255.255.255.254
 speed 100
 full-duplex
!
interface FastEthernet0/1
 no ip address
 shutdown
 duplex auto
 speed auto
!
router bgp 64497
 bgp log-neighbor-changes
 neighbor 203.0.113.8 remote-as 64496
 neighbor 203.0.113.8 description ISP-network
 !
 address-family ipv4
  redistribute connected
  neighbor 203.0.113.8 activate
  no auto-summary
  no synchronization
 exit-address-family
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
control-plane
!
line con 0
line aux 0
line vty 0 4
 login
!
end
```

Configuration 4.4.: Internet router running-config