1. Einleitung (inkl. Motivation und Beschreibung der Aufgabenstellung)  
     
   2. Grundlagen (Dicovery Protocols und alles was sonst noch wichtig ist aber nicht von Ihnen entwickelt wurde)  
     
   3. Entwicklung der Software  
     
   4. Ergebnisse aus der Verwendung der Software  
     
   5. Zusammenfassung und Ausblick

**Motivation**:

Today, companies use ever-expanding networks and infrastructures, which implies constant monitoring of the network and services. A process called Network Discovery is used in order to have an overview of all those devices connected to a specific network, to know which of them has access to others and how they interact. More specifically we call Topology Discovery the process capable to discovery and map network devices and links.

To make a Network visible represents one of the most important task in order to understand it. The graphic visualization of this allows administrators to understand a lot of information about the network and solve network-related problems such as network congestion, otherwise the solving process would cost a lot of time. The process of monitoring the components of a Network, in order to optimize its performance, covers a very important role: knowing if a component is active, reachable and how it reacts gives the administrator an overview of the state of this component defining it healthy or not.

To facilitate the administration and configuration of a network in order to improve its performance and monitoring, a new approach, Software Defined Networking (SDN) technology, is chosen. The SDN separates the forwarding functions from the network control functions, with the aim of creating a centrally manageable and programmable network increasing the network efficiency.

Softwares/tools use specific search protocols such as Simple Network Management Protocol (SNMP), Link Layer Discovery Protocol (LLDP), and ping, to capture all those devices connected to a network. In the field of Software Defined Networking, Openflow Discovery Protocol (OFDP) is used in order to discovery the network.

The work includes the discovery of peers in a LAN, using the mDNS (Multicast Domain Name System) protocol, which allows all peers to find each other and connect with each other, sending useful information for analyzing and monitoring them.

1. **Network Discovery (Introduction)**

Network discovery refers to the process that allows computers or devices as well as routers, Switches, Access Points etc. to find each other’s under the same network. The discovery of the network and its components allows administrators to get a picture of the network, knowing which devices communicate with each other, which relationship they have and in what way they communicate, regardless of the size of the network. Detailed knowledge of the network and consequently its visibility is a fundamental task in solving any network-related problem. Therefore, knowing the devices connected to it and knowing how they relate allows the resolution of problem such as network congestion, downtime or interruption in a shorter time, otherwise this would cost a definitely longer time.

Another important aspect related to Network discovery definitely concerns network security. By scanning network devices and therefore IP addresses, we can figure out whether they are valid or not, thus preventing data breaches and therefore the presence of malicious attackers in the network. [What is network discovery? | N-able](https://www.n-able.com/what-is-network-discovery)

Starting with some basics

Later in this chapter the different discovery protocols are explained as well as some notions about Software defined networking and in particular the relationship of this to topology discovery and why this is important. Following a description of the implemented software.

1. **Basics**

This section provides useful information to understand how Network Discovery works, starting with the concept of Software Defined Networking (SDN), its relationship to topology Discovery, as well as the protocols used for Discovery and at least the two networks architecture: Client-Server and the Peer to Peer (P2P) architecture, used in the implemented software. These represent fundamental understandings of the technologies used in the development of the software described in Chapter 3.

**2.1 Software Defined Networking (SDN)**

“Software-defined networking (SDN) technology is an approach to [network management](https://en.wikipedia.org/wiki/Network_management) that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring, making it more like [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) than traditional network management” Wikipedia

Unlike classic traditional networking, which uses hardware such as routers or switches to control network traffic, the approach of software defined networking is purely software-based. In fact, the latter has the ability to control and monitor the network or its components, whether virtual or not, solely through software. One more point is also the flexibility of configuration unlike classical networking, thus allowing administrators to have control over the network, controlling its resources, as well as its configuration and capacity increase all through a centralized interface, without the need for external hardware. Therefore, in order to have a separation between the infrastructure and its configuration, (basic concept of SDN) it is necessary to understand the concepts of Control Plan and Data Plan.

The Control Plan represents all those functions and processes that determine which routes to use to send a packet or frame, drawing then the network topology and using so-called routing tables in order to follow a route. The Control Plan then, implemented by default in the hardware components of the network, is extracted from them, with the purpose of separating the infrastructure and its configuration. It is also important to say that it performs its tasks independently through a Network Controller. For this very reason, being detached from the hardware increases programmability and consequently flexibility in Software defined Networking compared to other architectures. Routing is for example performed in the Control Plan.

The Data Plan, on the other hand, is based on the logic implemented in the Control Plan and refers to all those functions and processes that route a packet or frame from one interface to another. In essence it is responsible for moving packets from a source to a destination, performing its tasks dependently on the Control Plan. In SDN the Data Plan remains then part of the individual network devices. Switching unlike routing is performed in the Data Plan.

* + 1. **Topology Discovery in SDN**

To get a clear picture of the network, network discovery and management is a key feature of the SDN. The discovery picture is completed through the discovery of hosts, switches, and links between them. Knowledge of these elements represents a key factor that enables the controller (....) for example to direct network traffic to its destination. Another key aspect is to avoid possible attacks on the topology. The purpose of these attacks is to falsify the network topology, interfering in the management of network control.

**…**

* 1. **Discovery Protocols**

*“A****protocol****defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message”* [2]

So that a device in the network can be discovered and tracked, discovery protocols are used. Some of the most important and widely used discovery protocols are SNMP (Simple Network Management Protocol), LLDP (Link Layer Discovery Protocol), Ping (ICMP) [N-Able] and Open Flow protocol in the field of Software defined Network. Later in the section (...) another protocol used in the development of the Software, the so-called mDNS (multicast DNS), will be described in detail, understanding how it works, the fields in which it is applied, and why it was chosen as the discovery protocol. Some notions regarding the different protocols mentioned above will follow in order to get a clear view of how they work.

* + 1. **SNMP**

Simple Network Management Protocol (SNMP) is an application layer protocol belonging to the TCP/IP family of protocols. It is used to collect and organize the information of a device in a network, in essence management and monitoring of a device connected to the network such as a router, switches, servers, including anything accessible through its IP address. [[SNMP Monitoring: What It Is & How It Works | Datadog (datadoghq.com)](https://www.datadoghq.com/knowledge-center/network-monitoring/snmp-monitoring/#:~:text=SNMP%20relies%20on%20a%20client,the%20manager%20without%20being%20queried.), Wikipedia, [What is Simple Network Management Protocol (SNMP)? Definition from SearchNetworking (techtarget.com)](https://www.techtarget.com/searchnetworking/definition/SNMP)]

The architecture on which this protocol is based is the Client-Server architecture, explained in section (..). In order to provide this type of service, SNMP consists of several components: first of all appears the SNMP Manager, usually represented by a server that through requests to the devices, collects information about them. The client software installed on the SNMP-enabled-elements or devices is called SNMP Agent. The agent monitors device information, such as system status, configuration changes, CPU, Temperature, memory, etc., and transmits it to the SNMP Manager when requested. The devices on which the agent is installed are called managed devices. Lastly, the Management Information Base (MIB), a database that describes all the objects in a particular device that can be monitored and requested by the SNMP Manager. These objects are characterized by an Object Identifier (OID), an address in the device that represents a particular piece of information (Uptime, CPU...).

The agent software then constantly collects device information that will then be pushed when requested by the Manager through a series of commands (request-response), typically using UDP (User Datagram Protocol) as the transport protocol.

* + 1. **LLDP**

The Link Layer Discovery Protocol is a discovery protocol used in the 2 Layer with the function of discovering neighboring devices connected to the same network, allowing the enabled-device to advertise its identity and characteristics to the other devices. Unlike other protocols in the Link Layer it is vendor-neutral standard. LLDP messages are sent in a Layer 2 frame (OSI) to the multicast address "01:80:C2:00:00:0E" with the Ethertype "88-CC". [Wikipedia] Device information is sent through a software called LLDP agent installed in the device that supports the Link Layer Discovery Protocol. The agent´s task is then to periodically send the device information through each of their LLDP enabled interfaces, in form of an Ethernet frame, containing LLDP-DUs (Data Units) with a defined length and structure. [Geeks for Geeks] Data Units consist of several TLV (Type Length Value) blocks that describe a certain part of the information. The segments that must always be included in the Frame, so that topology discovery with LLDP is possible, are: Chassis ID TLV (Type = 1), Port ID TLV (Type = 2), Time to Live TLV (Type = 3), End of LLDPDU TLV (Type = 0). The information received via the LLDP-DUs (Data Units) may also contain the name and description of the system, the sending port, the name of the VLANs, the management IP address as well as the supported functions of the device like routing or switching. [Ip-Insider] Typically stored locally on each device in a data structure, the Management Information Base (MIB), represents a key component of this protocol as well as in the SNMT. Typical applications of the Link Layer Discovery Protocol are network management, troubleshooting or the automatic detection of devices in a network such as VoIP telephones. [[Was ist LLDP (Link-Layer Discovery Protocol / 802.1AB)? (ip-insider.de)](https://www.ip-insider.de/was-ist-lldp-link-layer-discovery-protocol-8021ab-a-928511/)]

* + 1. **Ping and ICMP**

Ping is a utility tool that is used to verify that a host is reachable in an IP network. The key component of this tool is the use of the Internet Control Message Protocol (ICMP). ICMP is a support protocol belonging to the TCP/IP family [Wikipedia], used to exchange information about the status and problems of a network. It is connectionless, that means one device does not need to open a connection with another host, as it happens by TCP with the handshaking before a packet is sent. Technically, Ping sends one or more ICMP echo request packets to the target host, waiting for an echo reply (usually programmed automatically on receipt of an echo request).

C:\Users\s.osman>ping www.google.com

Pinging www.google.com [172.217.23.100] with 32 bytes of data:

Reply from 172.217.23.100: bytes=32 time=25ms TTL=115

Reply from 172.217.23.100: bytes=32 time=18ms TTL=115

Reply from 172.217.23.100: bytes=32 time=16ms TTL=115

Reply from 172.217.23.100: bytes=32 time=19ms TTL=115

Ping statistics for 172.217.23.100:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 16ms, Maximum = 25ms, Average = 19ms

As shown in the output above, the program typically reports, based on the ping response, the number of packets sent and received, the number of packets lost, as well as the minimum, maximum and average Round Trip Time (RTT). RTT is the time taken from a data packet to reach the target IP and return to the source, expressed in milliseconds. If the recipient cannot be reached, an ICMP error is sent to the source.

**2.2.4 Multicast DNS Protocol (mDNS)**

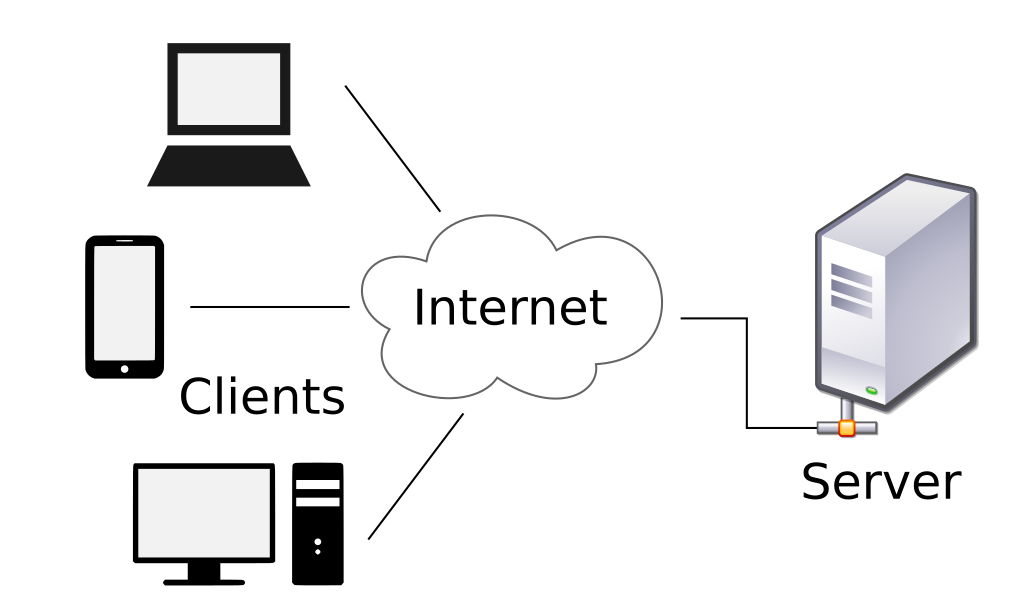
Multicast Domain Name System (mDNS) is a service/protocol part of the TCP/IP Suite, which deals with name resolution in small networks. Unlike classic DNS, in mDSN a request is not sent to a name server but the participants in a network are contacted directly. When a peer connects to a network or detects a network change, it sends a 'multicast' to it, requesting which networks node the respective hostname corresponds to. Multicast refers to a special form of communication in which a single message is addressed to a group of receivers. This group may include, for instance, a network or a sub-network. In this way, the request reaches the desired node characterized by the respective hostname. The latter responds to the entire network again via a multicast, so that all peers belonging to the network receive the IP and the respective hostname, adding an entry to their mDNS cache. No other node will then send a request to that host, as long as its entry remains valid. In the context of libp2p and consequently in the implemented software the mDNS is used to automatically discover other peers (that support libp2p) in the local network and to add them to the topology. This type of discovery was chosen in the software because mDNS belongs to the Zeroconf (Zero Configuration Networking) context, allowing computers to communicate with each other without prior configuration, resulting then very simple to use.

* 1. **Network-Architecture**

In this section some notions regarding the two paradigms of Network Architecture are explained, focusing on differences between them and the fields in which they are applied. Building a network application means to choose basically between two different approaches or styles of interaction, which describe the pieces of the network application that are going to interact with each other. The first approach is the client-server model and the second is the Peer-to-Peer (P2P). The latter was chosen as the network-architecture in the developed software, which will be specified in chapter three.

**2.3.1** **Client-Server**

The Client-Server model is a type of network architecture in which a client connects to a server to take advantage of a given service. Thus, the term Client is used to refer to any component (Hosts) that takes advantage of certain services, made available by another component that is called a server.



Wikipedia Client-Server model

In this architecture the server can be a physical device or a software program. Typical servers are: File-Servers, Webservers, Mail-Servers and Database-Servers. Important is that it is always on host, meaning it is continuously available [1, 2]. Providing a certain service e.g. one of the above mentioned, puts the server in the central role in the Client-Server architecture, which is also called Server-Based Network. Generally it has a permanent IP-Address so that clients will know where to reach it. If the server has a public address it can be reached by accessing from the web, in the case of a private IP address it can only be accessed from inside the LAN of the clients.

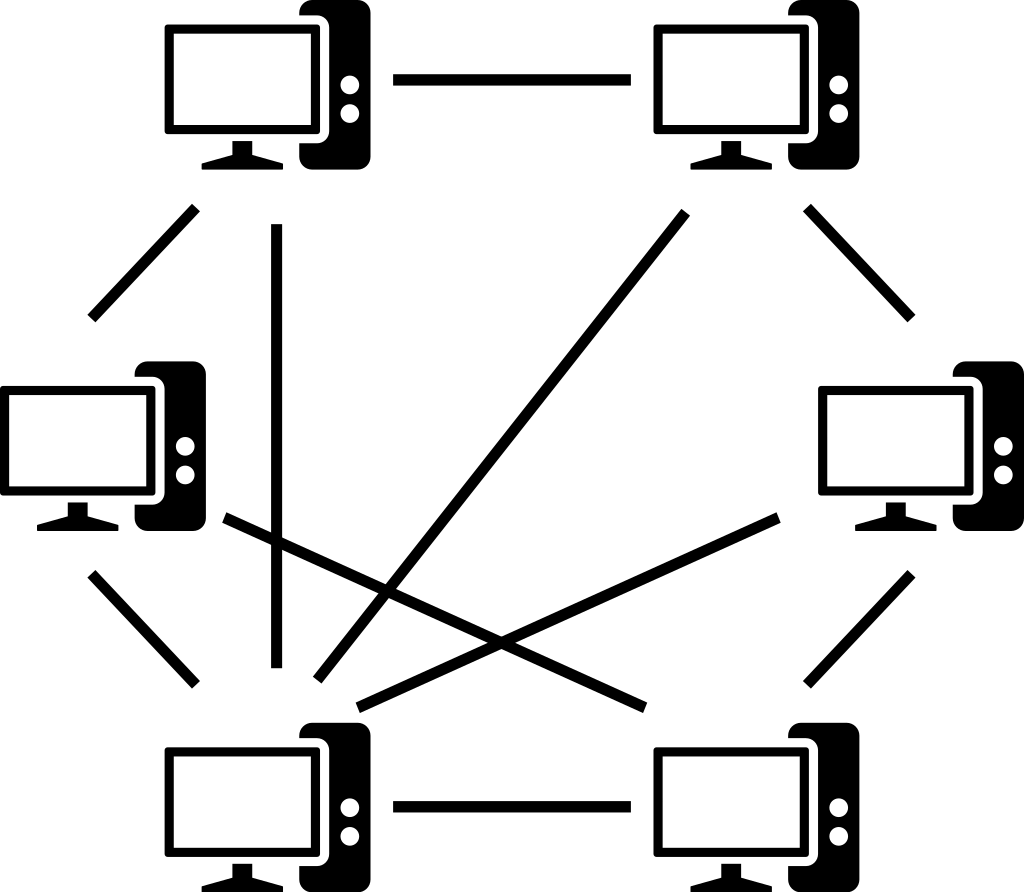
Clients are going to operate by contacting and communicating with a server by sending requests, the server on the other side, will elaborate the requests and send responses back. In contrast to servers the clients have no permanent IP-Address because they are typically intermittently connected to the internet. Most importantly, in the Client-Server architecture, clients do not communicate with each other, instead they are going to interact with servers. A classical example of a Client-Server protocol is HTTP (Hyper Transfer Text Protocol), where the client is the web browser and the server is the webserver [2].

Client-Server communication in a Client-Server protocol implies that a request is followed by a response. This type of communication can take place in different ways. In synchronous communication, the client makes no further requests without first receiving a response from the server. In asynchronous communication on the other hand, the client sends several requests to the server, but receives unordered responses at different times. Furthermore there is the pipeline type where the client sends several requests in succession but waits for ordered responses of the requests to arrive from the server. [1]

[1][Client-Server-Architektur (elektronik-kompendium.de)](https://www.elektronik-kompendium.de/sites/net/2101151.htm)

## [2] Computer Networking: a Top-Down Approach (8th ed.) J.F. Kurose, K.W. Ross, Pearson, 2020, Principle of Network Applications

**2.3.2 Peer-to-Peer (P2P)**

In a Peer-to-Peer architecture, unlike the Client-Server one, there is no server. Instead Peers are interacting in systems that are going to directly communicate with each other. In essence Peers are intermittently connected Hosts, also known as participants of a Network that communicate with each other without passing through a dedicated server. 

Wikipedia peer-to-peer (P2P) Network

In fact, each Peer acts as client and server: (as shown in Figure …) requesting service from other peers and providing service in return. File sharing like BitTorrent is a typical example of a P2P architecture, where one can request files from other participants but also provides files to those. These Participants, as already mentioned, are going to be intermittently connected to the internet and they are going to change their IP-Address. That means that the management of these Peers going online and offline, represent a more complex scenario than in a Client-Server architecture. Since it does not require a major server structure and significant bandwidth, the P2P model is significantly more convenient than the Client-Server one which is characterized by datacenters. Furthermore, a key feature of P2P is self-scaling: returning to the File Sharing example, each peer generates requests but at the same time adds capacity to the services on the system by providing Files to the other participants. Another feature is that an eventual failure of a peer in a P2P architecture does not usually have a major impact on the application, unlike the Client-Server where a failure of the Server is equivalent to no longer having a given service. [2]

1. **Development of the software**

This chapter deals with information on software implementation. Starting with the approach to Software Defined Networking and including configurations, design choices as well as functionalities. A subsection also deals with the key role of the mDNS protocol, which is used to discover other nodes in the network. LÖSCHEN?????

* 1. **Specifics**

What is libp2p, Network Architecture, Programming Language,

The software was implemented based on libp2p: “a modular system of protocols, specifications and libraries that enable the development of peer-to-peer network applications. [libp2p-documentation]

The purpose of the software is to monitor the participants or nodes in the network, in this case LAN in various aspects.

* 1. **Components of the software**

The software is basically divided into two components called agents: the Sender-Agent and the DB-Agent, remembering that each participant in a P2P architecture is connected to other nodes and provides as well as receives certain services.

* + 1. **Sender-Agent**

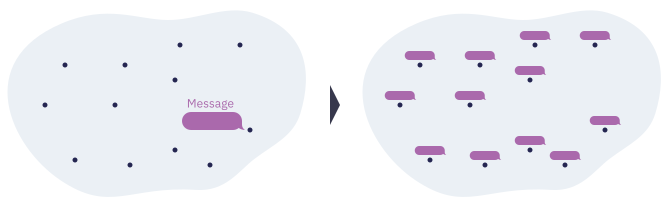
The purpose of the Sender-Agent is to transmit various information regularly about the system in which the program is installed in order to monitor it. To start the program and the monitoring a new host is configured, which listens on a given address parsed to the function.

node, err := libp2p.New(libp2p.ListenAddrStrings("/ip4/0.0.0.0/tcp/0"))

The second step is to configure the discovery for that particular node, creating a new mDNS Service based on the mDNS protocol to search for other participants in the network. The node will be listening until it is notified of the presence of a new peer registered to the same mDNS service name, in this case called "roomDiscovery" (in the section ..., a description is provided of how discovery via multicast DNS takes place VORHER ERKLÄREN). At this point it is possible to interact with the newly discovered node by deciding what action to take with the "HandlePeerFound" function. The created host will connect to the newly discovered peer thanks to the node's connect method.

**func** (d \*discoveryNotifee) HandlePeerFound(info peer.AddrInfo) {  
 fmt.Printf("discovered a new peer %s\n", info.ID.Pretty())  
  
 **if** d.node.ID().Pretty() != info.ID.Pretty() {  
 d.node.Connect(context.Background(), info)  
 PeerList = append(PeerList, info)  
   
 log.Printf("connected to Peer %s ", info.ID.Pretty())  
  
 service.SendPing(context.Background(), d.node, info, pingTopic)  
 }  
}

The participant to which the host has just connected will then be saved and he is accessible in the Peerstore, a hosts` repository of Peers Addresses and keys. In addition, for convenience, the peers discovered and connected by the node will be saved in the list called PeerList, which will be used later. Once connected to another participant, the communication between the nodes begins via a messaging system. The software uses the PubSub system (Publish/Subscribe) to communicate and exchange messages between connected peers.



Peers will therefore have to subscribe to a topic once they have registered a PubSubService. Once subscribed, they will be able to participate in sending and receiving messages throughout the network. PubSub is used e.g. in the field of chatrooms and filesharing. [PS documentation] In the developed software, each participant subscribes to a certain topic and uses the Publish function of PubSub to publish a message containing precise information about their system. Subsequently this information will be read and stored by the second component of the software, the DB-Agent, explained in the next subchapter. The PubSub system was chosen because of its fundamental properties such as: reliability, speed, efficiency, resilience, scale and simplicity.

As mentioned above, the function of the sender is to send or publish messages about particular information regarding his system. This information will be sent and published in different topics by calling Send-Functions that are running asynchronously in separate Go-Routines (a lightweight thread of execution). This procedure will usually be repeated every minute, in order to monitor the following features of the system over time:

* **System information** includes the hostname of the system in which the Sender-Agent is installed, its IP-Address, unique peer ID, platform and system version as well as the number of logged in users. This information is useful to be able to identify the host in the network and in particular to know what kind of machine is online. The identification and classification of the system would certainly help during a troubleshooting of compatibility, the installation of other software or the release of a patch. Furthermore, knowing how many users are currently logged into the system can help to identify an unverified and illegitimate user.
* **CPU** (Central Processing Unit) information which includes the model of the processor and in particular the CPU percentage. The latter indicates what percentage of the processors or a cores total working time is actually used for data processing. This value is significant for understanding how much stress the processor is currently under. A high CPU value also leads to rising CPU temperature. Very high temperatures could lead to computing performance problems, in worst case leading to a freeze of programs. To contain a high working performance of the company the monitoring of the CPU percentage seems necessary. [High CPU usage: What are the potential causes? - IONOS](https://www.ionos.com/digitalguide/server/know-how/cpu-usage/#:~:text=If%20CPU%20usage%20is%20too,by%20checking%20the%20CPU%20temperature.)
* **RAM** (Random Access Memory) percentage. The task of the RAM is to temporarily store all data of running programs and processes. \*\*\*to complete\*\*\*
* **Ping status**. When a peer connects to another peer, it starts pinging it regularly to see if it is active (See Figure mDNS Discovery), how this process works is already described in Chapter two Protocols/Ping. This makes it possible to actively check if the host is still connected to all other peers, but above all that they are reachable. After 10 consecutive 'negative' pings, the host will stop pinging the other peer, considering him as absent. The PingStatus[[1]](#footnote-1) structure in the software is characterized by the fields: source and target node of the ping, a Boolean representing whether the peer is alive or not and an RTT expressed in milliseconds. To avoid an overload of inactive peers and a host who is constantly busy with pinging them and uses resources for that, it is useful to mark them as absent. Monitoring active and absent peers helps to identify whether a device, host or connection is broken or not, maximizing the stability and functionality of them.
* **TCP Status** includes the number of TCP connection queues, the number of segments sent and received by the peer. Before the application data is sent to the network, the TCP/IP stack collects this data in order to limit the sent amount of data. For this purpose, each connection has a queue, where the data waits before being transmitted. [TCP/IP overview (usenix.org)](https://www.usenix.org/legacy/publications/library/proceedings/mobisys03/tech/full_papers/dunkels/dunkels_html/node3.html). The number of TCP connection queues with the number of segments sent and received makes it possible to know whether the system is working by sending and receiving TCP-packets and how many connections are waiting to send data to the network.This data is extracted from the system by running “netstat” commands on machines using Linux and Windows. In Linux “netstat –na | grep ESTAB” is used to see how many TCP established connections are in the queue and “netstat –st” in order to display TCP networking statistics. \*\*\*WOFÜR IST DAS GUT?\*\*\*
* **Bandwidth** includes ingoing and outgoing data transferred by the local peer. The values sent include the Total-In and Total-Out fields which record the cumulative bytes sent and received. The Rate-In and Rate-Out records on the other hand show the bytes sent and received per second by the Peer. This function is actually used by the DB-Agent, the second component of the software, which request the Peers for the above-mentioned parameters. \*\*\*WOFÜR IST DAS GUT?\*\*\*
  + 1. **DB-Agent**

The primary task of the DB-Agent is to listen on the various topics discussed in the previous section, waiting for incoming messages sent by other peers through the Sender-Agent and storing the received information in a database.

As already described in the previous section, a node (libp2p host) is also created in this case, which has not only the option to listen on a TCP port (example in the sender), but also the additional Bandwidth Reporter option provided by libp2p. The latter allows to call the function GetBandwidthForPeer(id peer.ID) which returns a Stats struct with bandwidth metrics associated with the given peer.ID, including all traffic sent and received for the peer, regardless of used protocol.

Through the use of the so-called go-routines the listening functions such as “go collector.ReadSystemInfo(timeSubscribe, context, node)” are executed in a separate thread and thus asynchronously. Once a Go-Routine has been called, the program will wait in an infinite loop for incoming messages from other peers in a certain topic.

Once a message has been received from a peer, it is 'Unmarshalled' i.e. parsing the JSON-encoded data in the form of []bytes and saved in the respective data structure, e.g. json.Unmarshal(message.Data, cpu). At this point the data structure attributes are stored in a Postgres database.

Whenever a message is published by a peer, a timestamp is always attached, which monitors the time frame of that particular property. The timestamp will be stored in the database and will be used by the Graphana tool to display graphs over time. \*\*\*Warum will ich das zeigen\*\*\*

The timestamp received from the sender-Agent will be used by the DB-Agent to calculate the latency between the two peers. Latency is the time difference expressed in milliseconds between when the message was sent by the sender agent and when it is received by the data collector. A high latency is a symptom of poor performance. Conversely, the lower the value, the better the network performance.

In order for all machines to be synchronized at the same time, the library https://github.com/beevik/ntp is used, which implements SNTP (Simple Time Network Protocol), making it possible to connect to a remote NTP server and request information on the current time.

* 1. **SDN approach**

Configuration, AWS, VPC, VMs, installation

1. Here written together because it refers to how it is used in code developed and not the general Ping status. [↑](#footnote-ref-1)