Technische Universität Berlin
Fakultät IV
Institut für Wirtschaftsinformatik und Quantitative Methoden
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Prof. Dr.-Ing. habil. Dr. h.c. Sahin Albayrak

Proposal for bachelor thesis

Enabling QoS-Aware Task Execution on Distributed Node-RED Cluster for Fog Computing Environments

Daniel Sebastian Lienau
lienau@campus.tu-berlin.de
Course of studies: B. Sc. Wirtschaftsinformatik
Matriculation number: 378170
Supervisor: Cem Akpolat

Table of Content

1. Introduction **3**

2. Literature Review **4**

2.1 Fog infrastructure 4

2.2 Flow-based Programming 5

2.3 QoS-aware resource allocation & scheduling 5

2.3.1 FogTorch 6

2.3.2 Modified constrained optimization particle swarm optimization 6

2.4 Monitor-Analyze-Plan-Execute over a Knowledge base 7

3. Use Cases **7**

3.1 Notify a device using temperature sensor data 7

3.2 Object Detection 8

4. Problem Definition & Challenges **9**

4.1 Load Balancing among Fog Nodes 9

4.2 Dynamically Varying Network Conditions 10

4.3 Fog Node Failures 10

5. Solution Approach **10**

5.1 Containerisation and Deployment of Distributed Node-REDs and functions 11

5.2 Service Composition Panel 11

5.3 QoS-Aware Fog Orchestrator 12

5.4 Implementation of a resource allocation & scheduling algorithm 13

6. Schedule **14**

7. References **14**

1. Introduction

The number of interconnected devices on the internet is constantly increasing, more and more data is exchanged between different participants. While the Internet initially experienced a broad distribution during the 80s thanks to the world wide web service, where mainly personal computers and enterprise servers were involved in the communication, nowadays the number of offered services and online devices is countless. Over the last few years, *Internet of Things (IoT)* has become very popular and an end to this trend is not predictable. All kinds of different devices are connected to the internet (e.g. smartphones, smart watches, smart tvs, smart home devices, cars) and exchanging data with each others. Thus, the amount of exchanged data over the internet and between network nodes is constantly increasing as well, leading to higher requirements on the overall network infrastructure to ensure a high connection quality with little delays as well as to avoid network congestion.

A common scenario is that devices are communicating with each others via a cloud server in a big data center, which physical distance might be relatively long from the devices location. Often though, and this is especially the case for IoT scenarios, a lot of data is produced and consumed locally, on the networks edge. While this is not a big deal for services which don't have a high real-time requirement, a service that has a high real-time requirement relies on the network connection quality and low delay times between participating devices. If this can't be ensured, the service is unusable. This becomes even more important in applications where personal safety plays a role. For example, an autonomous driving car is told to break by a device at the side of the road. If these devices would be interconnected via a server in the cloud, the delay time would depend on the current network congestion level, thus the delay time is varying and the *quality of service (QoS)* could not be ensured. This issue is addressed by Fog computing.

Fog computing provides a distributed infrastructure at the edge of the network, resulting in low-latency access and faster response to application requests [1]. In a fog infrastructure, there is a variable number of so called fog nodes, which can provide resources like computation or storage. Fog nodes are usually small devices with limited resources compared to servers in the cloud. Furthermore, fog nodes in a fog infrastructure are not necessarily of the same kind, thus they have a different amount of resources available. For example, one fog node can have a relatively low computation power compared to another fog node. The challenge here is to distribute the services among different fog nodes while ensuring the QoS requirements are met. While some services can't be executed at all on certain nodes due to the lack of resources, other services are competing for resources on other fog nodes. One solution here is to *decompose* services into smaller *tasks*, so that these tasks could run on different fog nodes. However, these tasks need to be recomposed thereafter to provide the service as a whole. This part is called *service composition*.

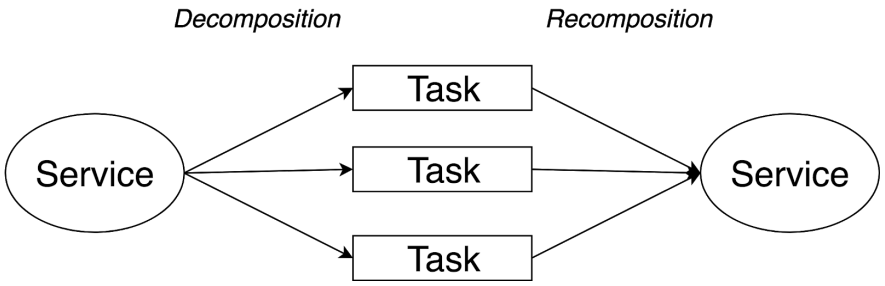

Figure 1: *Service Composition*

The challenge here is though, that the conditions inside of the fog infrastructure can change at any time. First of all, the demand for the offered services can increase or decrease at any point of time. Furthermore, fog nodes can join or leave the network at any point of time. Simply put, tasks have to be distributed *dynamically*. To ensure each services QoS requirements, every node has to be constantly monitored. If the demand for services increases, the average load on the fog nodes will be high. To still provide an acceptable service quality for each service, tasks of services which have lower QoS requirements have to be moved to another resource. A task could always be executed in the cloud, but this would lead to longer delay times.

2. Literature Review

In this section relevant topics for this work are discussed based on related work.

2.1 Fog infrastructure

Fog is an architecture that distributes computation, communication, control and storage closer to the end users along the cloud-to-things continuum [2]. Although cloud computing has established itself over the last few years, it cannot be used for certain use-cases, especially time-critical and bandwidth-intensive ones. Because of the closer physical distance to a fog node, fog computing can reach a much lower end-to-end latency than cloud computing [1][3]. This plays a vital role in applications such as vehicle-to-vehicle communication [3].

A network of fog nodes is a distributed heterogeneous network. Unlike in cloud computing, where hardware capabilities are virtually unlimited [4], resources on a fog device are usually constrained. They can’t execute every task because their computation power is limited. Furthermore, the available resources vary from node to node, making the whole network *heterogeneous*. Instead of executing a task on a central computer, it is *distributed* between several nodes in the network. It first has to be determined which fog node can execute which part of the task in terms of resource-constraints. At the end, all partial results are combined into the final result.

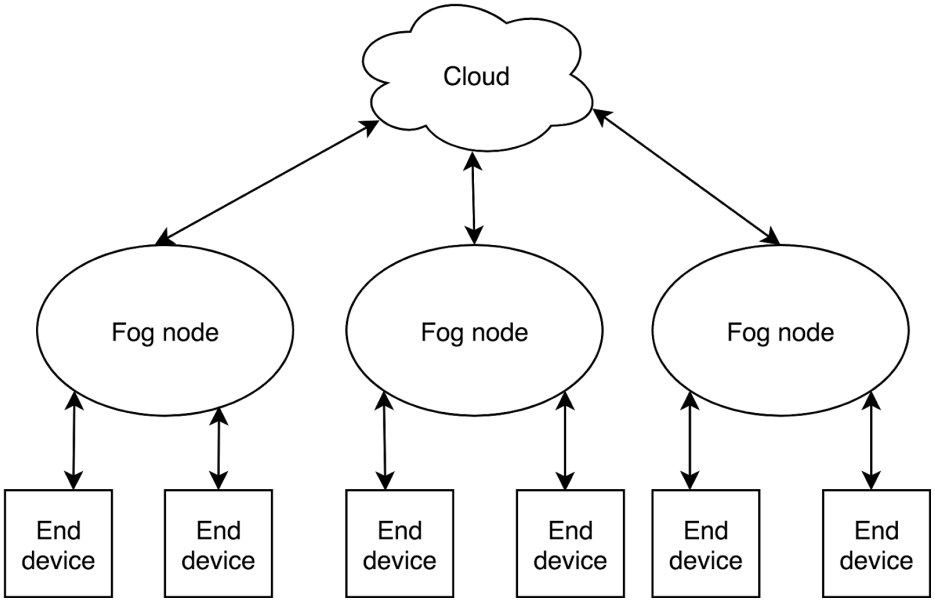

Figure 2: *Fog infrastructure*

2.2 Flow-based Programming

Flow-based programming (FBP) allows the developer to structurally define the informational flow within a system. While it is not important *where* a specific task is physically executed, FBP focuses on the *path* the data takes from one process to another. Any number of further processes can be involved in between, while every process manipulates or just reads the data to trigger further actions. Each process has a specific in- and output. For the developer, all execution units are “black box” processes, which can consume and create data. The processes can be executed and recomposed in any order, as long as output and input formats match each others. It is also possible to split the data flow at one point and combine it back together at a later point, making parallel computation possible.

Data between processes is transferred in *Information Packets (IPs)*. IPs belong to a single process or are in a transfer state, where they are owned by no process. As soon as a process receives an IP, it can start processing it without the need to communicate with other processes, because the IP contains all necessary information the process needs to fulfil the task. This makes FBP highly attractive for Fog computing environments, where each fog node can be used to execute one or more process types, depending on the process requirements like computation power, bandwidth or latency.

2.3 QoS-aware resource allocation & scheduling

In this section, related work on QoS, load-balancing, and resource allocation is discussed. Although fog is considered as a cloud close to the ground, load balancing strategies of cloud computing can not be directly adopted in the fog network since the heterogeneity of fog [3].

2.3.1 FogTorch

In [4] a model to support QoS-aware deployment of multicomponent IoT applications to a Fog infrastructures is proposed. Furthermore, a Java tool called *FogTorch*[1] has been prototyped which implements that model. The model allows to define QoS profiles, Fog infrastructures and IoT applications, which are used to determine eligible deployments.

A *QoS profile* defines bandwidth and average latency required for an application, or offered by a communication link. A *Fog infrastructure* includes IoT devices, Fog nodes, Cloud data centers, and communication links, while each link is associated to its QoS profile. Cloud computing is modeled according to the hypothesis that it can offer a virtually unlimited amount of hardware capabilities. An *IoT application* is a set of independently deployable components that are working together and must meet some QoS constraints. For this, software components as well as required interactions among components, including the desired QoS profile, are defined. At the end, an *eligible deployment* for the IoT application is calculated by an algorithm.

An algorithm selects where a component is to be deployed within the Cloud to Things continuum. For this, a *preprocessing* procedure which reduces the search space for eligible deployments runs before a *backtracking* procedure and *heuristics* are used to determine a single eligible deployment. Because the proposed backtracking algorithm follows a heuristic approach to get a solution faster, it shows greedy behavior.

[5] presents *FogTorchΠ*[2], an open source prototype based on [4]. FogTorchΠ, compared to FogTorch, additionally allows to express processing capabilities and average QoS attributes of a Fog infrastructure, along with processing and QoS requirements of an application, and it determines deployments of the application over the Fog infrastructure that meet all such requirements [5]. It models the QoS of communication links by using probability distributions (based on historical data) repeatedly to simulate different runtime behaviours. At the end, it aggregates the results for deployments generated over a large number runs. The output of FogTorchΠ contains eligible deployments (like FogTorch), but additionally outputs QoS assurance and resource consumption over Fog nodes which allows to compare possible deployments and evaluate the impact of possible changes.

2.3.2 Modified constrained optimization particle swarm optimization

[3] proposes the *modified constrained optimization particle swarm optimization* (MPSO-CO) load balancing algorithm, which is *software defined networking* (SDN)-based. Compared to the *constrained optimization particle swarm optimization* (PSO-CO) algorithm, which it is based on, it is able to effectively decrease latency and improve the QoS in a *software defined cloud/fog networking* (SDCFN) architecture. It was developed for applications in the Internet of Vehicles (IoV) which still stuffers from high processing latency. SDN is used for centralized control and to get the required information before load balancing. The key

[1] https://github.com/di-unipi-socc/FogTorch
[2] https://github.com/di-unipi-socc/FogTorchPI

technology of SDN is decoupling data and control plane. The controller collects real-time information of the network including load, processing speed, and communication latency. Based on that, it can formulate optimal load balancing strategies for the network. It is shown that the MPSO-CO algorithm obtains lower latency compared to PSO-CO, Max-Min load balancing algorithm (LBMM) and greedy load balancing algorithm (Greedy-LB), since LBMM and Greedy-LB don't take the transmission latency into account, and PSO-CO may fall into the local optimum. However, on a low task load (lower than 0.05GB), where transmission latency is not a relevant factor, there is no big difference regarding the latencies of the different algorithms and they all perform pretty much the same.

2.4 Monitor-Analyze-Plan-Execute over a Knowledge base

A self-adaptive system consists of the two layers *managed subsystem* and *managing subsystem*, whereas the managing subsystem resides on top of the managed subsystem and monitors the managed subsystem as well as the environment. It realizes a feedback loop which adapts to changes, e.g. environmental changes like congestion or failures, or to goal changes [6].

The *Monitor-Analyze-Plan-Execute over a Knowledge base* (MAPE-K) [7] reference control model is the most influential reference control model for autonomic and self-adaptive systems. It is commonly used to realize feedback loops [6].

The component *Knowledge* (K) is responsible for storing and providing data from/to other components. The component *Monitor* (M) collects data through probes or sensors from the environment as well as data from the managed subsystem. The collected data is saved in K. The component *Analyze* (A) analyzes the collected data to check if the system needs an adaption. If this is the case, the component *Plan* (P) will determine which actions are required to put the system in the desired state. The component *Execution* (E) then carries out these actions. [6]

3. Use Cases

This section describes two different use cases. Because this work focuses on QoS, it is necessary to have use cases of different importance, e.g. real-time and non-realtime services.

3.1 Notify a device using temperature sensor data

Sensor networks are used to monitor an (industrial) environment for various measurable parameters. In a wireless sensor network (WSN) different types of sensors (e.g. microphones, CO2, pressure, humidity, thermometers) can be used. The measured values are used to decide (calculate) whether an action should take place or not. Since the sensors themselves usually have few resources for the calculation, this calculation takes place externally. This can be done either centrally or decentrally by distributing the calculation between different nodes.

A task execution on a local node rather than on a cloud server makes sense especially for time-critical or data-intensive applications. Data-intensive because the datastream remains in the local network and does not require an internet connection and therefore does not occupy any bandwidth of the internet connection. Time critical because the round trip time to a cloud server is usually higher than to a local server. This work focuses on the time-critical aspect of task distribution and execution.

To distribute services between different nodes, a service must first be split into different functions or tasks, which can then be executed on different fog nodes. At the end the partial results must be merged to an overall result. The fog nodes must therefore be able to communicate with each other and forward the final result to another unit, which then takes further action or not based on the result. The possible actions can be of different priority. For example, a temperature adjustment is less important than an emergency stop or emergency braking of a machine.

It is very important to use a distributed deployment here because the computational resources of the measurement taking node are limited. However, the result of a calculation must be available within a specified time window if time-critical actions have to be executed. The node itself can not guarantee to calculate the result in time, so it has to offload the task to another node.

Task: Notify a device using x sensor data

- Collect raw data from devices. Devices can be simulated. If we find a dataset that can be used through these services, they can only forward the data to a specified cloud server.
- A cloud service should operate some processes. Based on the data structure provided by those sensors, the cloud service should evaluate different parameters and end up with a result that will be used by another local service/device in the smart factory.

Challenge: We will increase the number of the devices that send the data to the cloud service, at the same time the network will be dynamically manipulated through the DITG/X tool in order to enforce the fog orchestrator to take a decision. Possible decisions are: moving service from cloud to fog or fog to another fog.

3.2 Object Detection

The object detection is used nearly in most of the fields in our daily life. It is also quite relevant for a smart factory and factory devices where the information has to be extracted from the recorded picture. This use case aims at reflecting a simple object detection process that will be operated using *Object Detection Service* running in Cloud and Fog Network. The essential goal is to compare the total delay for the object recognition process in different locations of the network.

Use-case: Factory prints the products, worker takes the product from the conveyor belt and stores it in one of the container locating in front of the worker. A IoT-enabled camera tracks the worker object placement to figure out whether the products are correctly placed. Object Detection Service should analyse the object and informs the user/robot if there is a mistake while placing the objects.

Alternative Use-Case: Factory worker doesn't know where to place the object, since the objects differ from each other. Therefore, the printed object should be recognized by the Object Detection Service beforehand and then inform the worker to which container the product should be placed.

Task: Video processing

- Video recording: a local computer sends a video stream to the cloud or to a fog node running an object detection service
- Object detection: the service returns the name of the object in text format as well as the correct container for the object

Challenge: The task should first be executed in the cloud. Afterwards, the system should decide if the service requirements are satisfied. Due to varying network conditions which are created by the same tools mentioned in 3.1, the system should decide to move the service from the cloud or a fog node to another fog node which can satisfy the service requirements.

4. Problem Definition & Challenges

This part structures the problem. Our problem here is mainly resource allocation and its usage. What are the technical the theoretical challenges?

4.1 Load Balancing among Fog Nodes

A service can be seen as a flow and consists of several tasks (functions in a flow), whereas each function could be executed on a different node. The challenge here is to decide which node should execute which function. The goal is to meet the services QoS requirements. For the MWSN use cases it has to be ensured that a result is available within the defined maximum latency timespan, e.g. within 10ms for condition monitoring for safety.

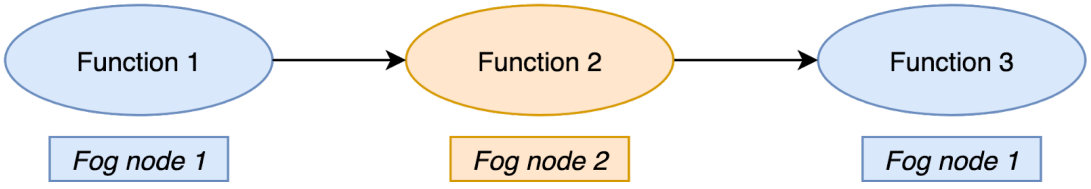

***Figure 3**: Distributed flow*

Since we are dealing with a Fog infrastructure, it is by nature that the different nodes offer different hardware and software resources. In order to assign tasks to nodes, it must first be checked whether a node is able to execute the task with given QoS requirements.

The challenge here is to find and use a suitable resource allocation algorithm for the given use cases.

4.2 Dynamically Varying Network Conditions

The network conditions might change at any point of time. The increased demand for services plays a major role here, but there may also be increased network traffic. For instance, in the MWSN scenario, the amount of sensors (end devices) requesting services might increase, which leads to a higher load on the fog nodes hardware until the point of congestion. To avoid congestion and ensure QoS, tasks of services with a lower priority have to be moved elsewhere to free resources for high priority services. Apart from that, the network traffic might increase due to other network services which are not part of our Fog infrastructure but are using the same link.

The challenge here is to adapt to the mentioned changes, for which they must be identified first before the load balancing algorithm is executed to calculate an eventually different, optimized deployment.

4.3 Fog Node Failures

A Fog infrastructure is not as reliable as a Cloud infrastructure, for instance. Fog nodes can join or leave the network at any point of time, which leads to a dynamic change in the available resources. Since Fog nodes can be any type of hardware, it is not unlikely that small and relatively cheap devices (e.g. Raspberry Pi) are used. Thus, it is more likely that they might fail since the hardware components are not of high quality compared to enterprise server hardware. Furthermore, the environment is neither monitored nor controlled like a Cloud server data center, which could lead to node failures (e.g. network connection errors or power supply failures). For instance, a fuse could fail due to some physical environment circumstances that have nothing to do with the nodes hard- or software which causes a node failure.

The challenge here is to adapt to those dynamic changes. If a node leaves the network, fewer resources are available and therefore another deployment might be the optimum. The same when a node joins the network, more resources are available and again, another deployment might be ideal.

5. Solution Approach

In this work a solution for distributing different tasks onto different fog nodes while meeting the QoS requirements should be found. The challenges described in the previous chapter are to be solved.

This section describes the system architecture and the functionalities of the different components.

5.1 Containerisation and Deployment of Distributed Node-REDs and functions

Since the tasks are to be executed on different fog nodes, therefore the tasks must be known and dynamically distributed to the different fog nodes. Each node will run an instance of node-RED, which is able to execute JavaScript functions and connect these functions' in- and outputs in a flow-based editor. To be able to distribute these functions dynamically, the functions have to be stored in a database first. To execute these functions, a component which will read the functions from the database and distribute them has to be implemented. The fog nodes hardware will be a couple of Raspberry PIs and Virtual Machines, which all have different resource capabilities. Since the underlying hardware architecture may vary, a containerisation tool and a base image containing node-RED will be needed. For this, Node-RED-Docker[3] will be used. Because the different tasks will be executed on different fog nodes, but have to offer a service at the end, the distributed functions have to communicate with each others. This will be done by using *distributed node-RED*[4] (DNR).

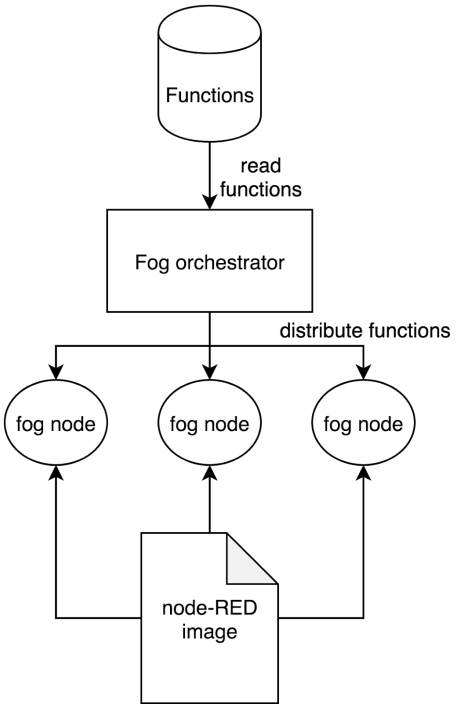

Figure 4: *Function/Task distribution*

5.2 Service Composition Panel

A service composition panel is a requirement for a service developer. This panel should offer a flow editor, where different functions could be composed into a flow (service). Furthermore, it must be possible so set the required QoS parameters like maximum latency or bandwidth.

[3] https://hub.docker.com/r/nodered/node-red-docker/
[4] https://github.com/namgk/dnr-editor

The service developer does not care about where a function is actually executed. He only cares that the service is working and that it fulfils the QoS requirements.

The distributed Node-RED editor offers such a panel which will be used for this purpose. Since DNR offers a REST API, it is possible to hook into a flow and (dynamically) change the placement of functions from one node to another.

5.3 QoS-Aware Fog Orchestrator

The heart of the architecture is the QoS aware `Fog orchestrator`. Its main task is to distribute flows between different nodes. Determining which node should execute which function is the job of the `QoS Orchestrator` module, which runs inside of the `Fog orchestrator`. Figure 5 shows the system architecture with focus on the `Fog orchestrator`, which is shown in detail, whereas the nodes are displayed as black boxes.

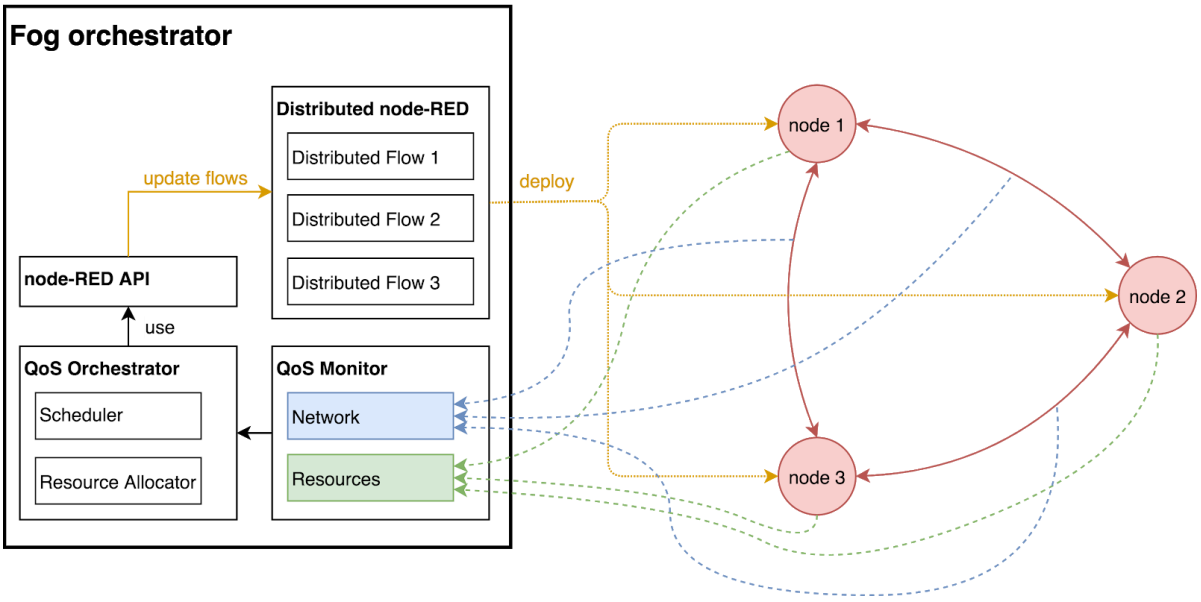

Figure 5: *QoS-Aware Distributed Node-RED Architecture*

The `Fog orchestrator` will be aware of the current resource load of each node as well as of the network connection quality between each node. Each node will continuously transmit its current measurement data via MQTT to the `QoS monitor` running on the `Fog orchestrator`. Based on this data the `QoS Orchestrator` will decide how flows should be distributed between the nodes to ensure best possible QoS values. For this decision the `QoS Orchestrator` uses a resource allocation and a scheduling algorithm. The result contains multiple distributed flows, which are represented as JSON and transmitted to the `Distributed node-RED Editor` via the node-RED API, if the most recent result differs from the previous one. The `Distributed node-RED Editor` also runs on the `Fog orchestrator` and deploys the updated flows to the fog nodes. This is a dynamic process since measurement data is sent at a given interval and might change. To see how the `QoS Orchestrator` reacts to changing network conditions described in Section 4, it is necessary to artificially influence the network quality as well as the request for services.

Figure 6 shows the system architecture with more detail on the modules running on the `Fog node`.

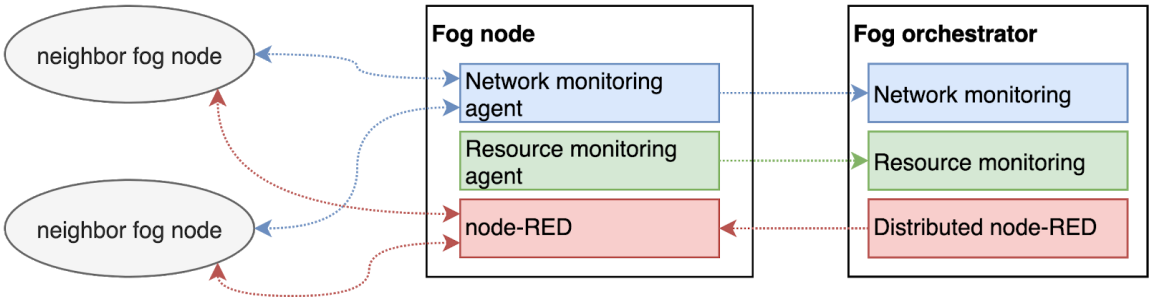

Figure 6: *QoS-Aware Distributed Node-RED Architecture*

There are two agents running on each node, the `Network monitoring agent` and the `Resource monitoring agent`. The first one monitors the network connection quality to its neighbor fog nodes, while the second one monitors local resources like CPU and RAM. The measurement values are constantly updated and transmitted to the `Fog orchestrator` via MQTT.
Furthermore, a `node-RED` instance containing the `Distributed Node-RED` package runs on each fog node. This package connects a fog node to one master `Distributed node-RED Editor` running on the `Fog orchestrator`. This makes it possible to provide a fog node as a possible task execution unit. All node-RED communication between the fog orchestrator and the nodes, as well as between the nodes themselves is managed by `Distributed node-RED` and is not part of the thesis. Only the connection from the fog node to the `Distributed node-RED Editor` must be set up once.

The described architecture applies the MAPE-K feedback loop reference control model, while the Fog orchestrator is the managing layer, and the managed layer consists of the Fog nodes. The network and resource monitoring components build up the *Monitor* component, while the resource allocation algorithm is part of the *Analyze* and *Plan* component. Finally, the *Execution* component is realized by distributed Node-RED, which deploys a new flow distribution if necessary. The *Knowledge* component holds the actual fog network state which is collected by the monitor. The recorded data is stored in a database and provided to the other components.

5.4 Implementation of a resource allocation & scheduling algorithm

The task distribution will be based on the implemented resource allocation algorithm. Since the MWSN use cases described in section 3 do not require much bandwidth, but low latency, the model proposed in [5] seems to be suitable, even though it does not consider transmission latency. The idea is to build on the open source Java prototype FogTorchΠ, which is only used to determine an eligible deployment for a theoretical Fog infrastructure so far. However, it still needs to be validated through real world use cases in order to compare the automatedly computed deployments with those determined by IT experts [4].

6. Schedule

#	Effort	Completion
1	Literature research and the development of a concept	31.05.2019
2	Submission of the proposal	07.06.2019
3	First milestone: Beta version of the project	30.06.2019
4	Second milestone: Final version of the project	31.07.2019
5	Submission of the Bachelor thesis	15.08.2019
6	Defense of the Bachelor thesis	17.09.2019

7. References

[1] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-Aware Application Scheduling in Fog Computing," *IEEE Cloud Computing*, vol. 4, no. 2. pp. 26–35, 2017.

[2] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6. pp. 854–864, 2016.

[3] X. He, Z. Ren, C. Shi, and J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles," *China Communications*, vol. 13, no. 2. pp. 140–149, 2016.

[4] A. Brogi and S. Forti, "QoS-Aware Deployment of IoT Applications Through the Fog," *IEEE Internet of Things Journal*, vol. 4, no. 5. pp. 1185–1192, 2017.

[5] A. Brogi, S. Forti, and A. Ibrahim, "How to Best Deploy Your Fog Applications, Probably," *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. 2017.

[6] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation," *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2015.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1. pp. 41–50, 2003.

[8] [Online]. Available: http://www.3gpp.org/ftp//Specs/archive/22_series/22.804/22804-g10.zip. [Accessed: 17-Apr-2019].

[9] "3GPP." [Online]. Available: https://www.3gpp.org/. [Accessed: 17-Apr-2019].