Brunel University London

EE5571 Workshop
Internet of Things Application Prototyping

Prof. Hongying Meng, Dionysios Satikidis

Julian Klissenbauer-Mathae (1841792)
Corvin Schapoehler (1841781)
Gianluca Panetta (1841790)
Oliver Johann Wasser (1841793)
Tino Stadelmaier (1841784)

Table of Contents

Abstract

Natural disasters like earthquakes cause thousands of deaths every year. A crucial step in the prevention is the early recognition of such events. This however, in the case of earthquakes, requires specialized hardware, that is expensive to obtain/run and therefore not available to all regions of the world. In order for people to quickly recognize such threads, we have created the “Distributed Disaster Detection System” or in short “D3S”.

D3S shall solve two main objectives:

1. Detection of earthquakes from large amounts of sensor data, that is being collected across multiple groups of smartphones located in close region to each other.
2. Provide a reporting platform for natural disasters that easily provides information to clients about events in their near proximity.

In the course of the workshop, which is part of the module “Embedded Systems Engineering”, we have focused ourselves on the first objective. As earthquakes cause resting objects, in this case smartphones, to move in a similar fashion, this enables us to recognize changes in their acceleration data in a set period of time. For this idea to work, a large group of clients is needed in order to eliminate false positives.

Additionally, D3S must be able to identify clients as “resting” in order to make correct assumptions on the basis of its provided acceleration data. When done correctly, this should enable us to recognize and even track the spread of automatically detected earthquakes.

The main infrastructure of the application is based on a Spring Boot backend in combination with a Python algorithm. The phones will continuously send sensor ping requests to the backend. If a client detects an abnormal movement, it will begin to send its sensor data to the backend, where this data is being further analysed and in the case of an incident, all affected clients are notified.

However, sensors of different devices work differently. This might be for example different sampling rates or sensitivities. Therefore, the solution must be able to adapt itself depending on the data source in order to correctly make assumptions about a potential earthquake.

As seen, this topic field offers a huge potential in added safety for a large group of people if D3S meets its expectations of a cheap and highly scalable earthquake detection system.

1 Problem Scope

Earthquakes and disasters in general cause a lot of harm when not recognized in time. Traditional earth quake detection systems mostly cover more developed areas. In order to maximize the area that is monitored for earthquakes and other various natural disasters we have created the “Distributed Disaster Detection System” or in short “D3S”. This system runs distributed across multiple smartphones, sending a variety of sensor data to a centralized server. This data then gets analysed and potential danger zones will be highlighted for all clients on a map and with a notification, which mainly serves the purpose of detecting earthquakes. Additionally, the client shall have the ability to manually report disasters from a pre-defined list including a description of the event. When the system detects enough similar reports, the disaster will also be shown/highlighted to other clients. This whole approach has the mayor advantage of not being dependent on multiple “centralized” earthquake detectors, but rather uses potentially thousands of smartphones that almost everybody nowadays uses and are evenly distributed across the land. Also, the cost is quite much less when compared to traditional systems, making D3S more attractive to country’s with lower incomes.

Figure 1: D3S Logo

2 Initial Concept

The main focus of this solution should be the recognition of certain patterns from large amounts of acceleration data, that is being collected across multiple groups of smartphones located in close region to each other. In future D3S should also serve as a sort of “reporting platform” for natural disasters of other kind.

As earthquakes cause resting objects, in this case smartphones, to move in a similar fashion, this enables us to recognize changes in their movement in a set period of time. For this idea to work, a large group of clients is needed in order to eliminate false positives. Additionally, D3S must be able to identify clients as “resting” in order to make correct assumptions on the basis of its provided acceleration data. When done correctly, this should enable us to recognize and even track the spread of automatically detected earthquakes.

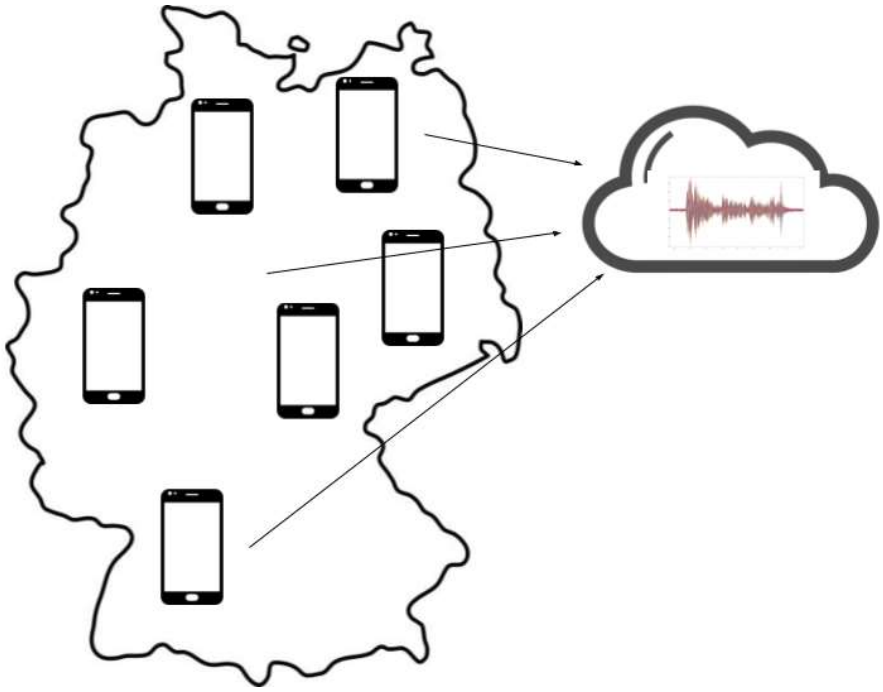

Figure 2: Concept Visualization

3 Accelerometer Experiments

To analyse the functionality of the sensors, the tests described in this chapter were made. For this, several phones were equipped with a sensor measurement app which recorded acceleration data to a csv file.

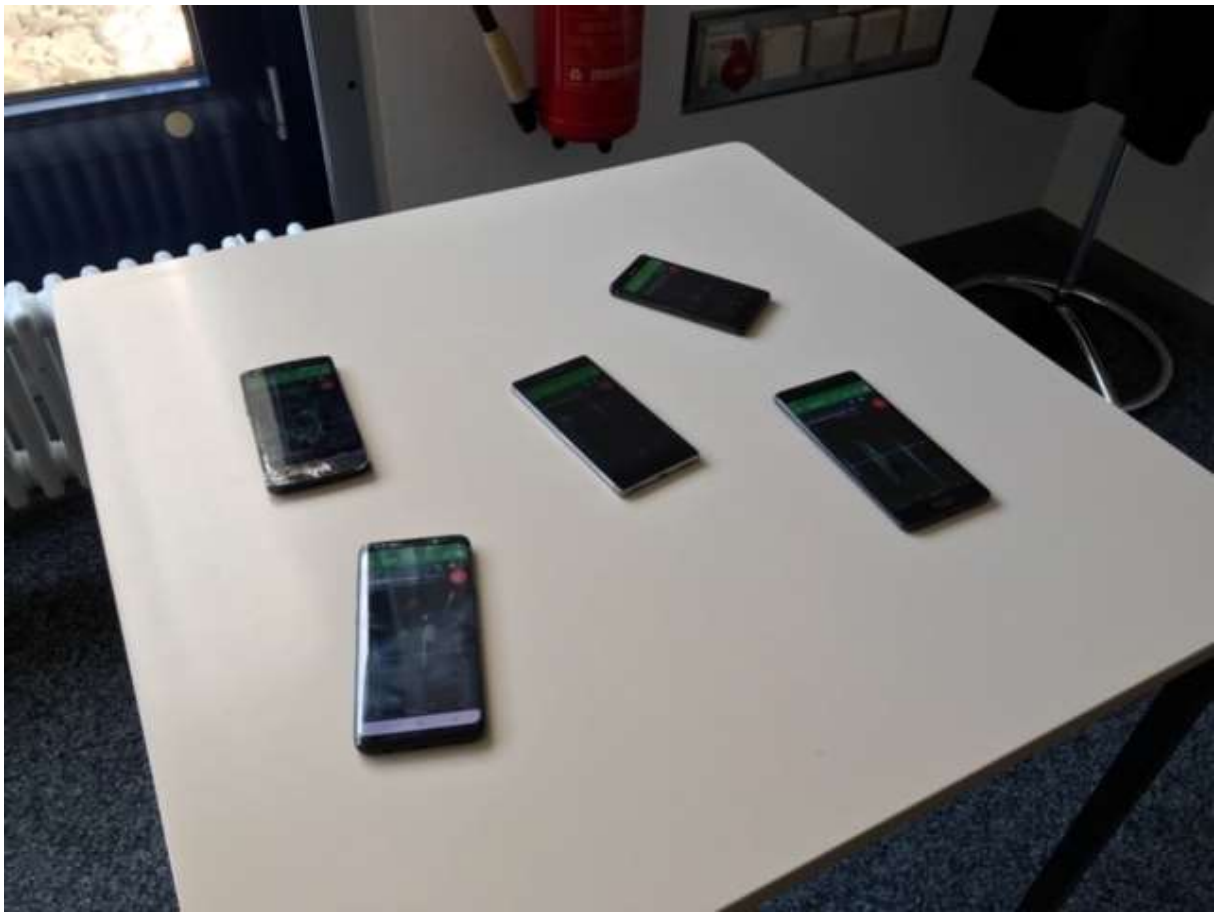

Figure 3: First evaluation of the concept

3.1 Phone on a table, trampling on the ground

With this experiment, we wanted to analyse if shockwaves can be measured if the phone is resting flat on a table and one tramples on the ground nearby the table. The experiment showed that such waves are measured, but the strength depends on the weight and stiffness of the table as well as the ground. This makes sense, because these parameters change the way the table reacts to immediate shockwaves. Mainly the z-axis was affected by the trampling. This makes sense, because trampling is usually done by moving the foot towards the ground.

3.2 Phone on a table, tapping on the table

The experiment was supposed to show the effects of very faint tapping on the same table the phone rests on. The results shows that the sensors indeed measure the "taps". The taps affected mainly the z-axis.

3.3 Phone on a table, shaking the table

This experiment showed us how the sensors react if the table is heavily shaken. Here, the x- and y-axes show clear signs of movement, while the z-axis stays relatively calm.

3.4 Result Analysis

After the measurements were made and saved into one csv file per phone, they were transferred to a PC and imported into MATLAB. With the MATLAB plot() function, the measurements from each device were plotted overlapping in different colors, so that a qualitative comparison could be made.

In Figure 4, three measurements with different intensities are shown. It is clearly visible, that the data of each sensor differs extremely regarding absolute acceleration values. But the overall plot of each sensor data looks similar regarding the ratio between maxima and minima.

The experiments showed that it is possible to even detect faint movement with the sensors in a smartphone. Also, heavier movement can clearly be separated from the background noise. However, it is necessary to somehow look at all axes, because motion can also be detected in the z-direction.

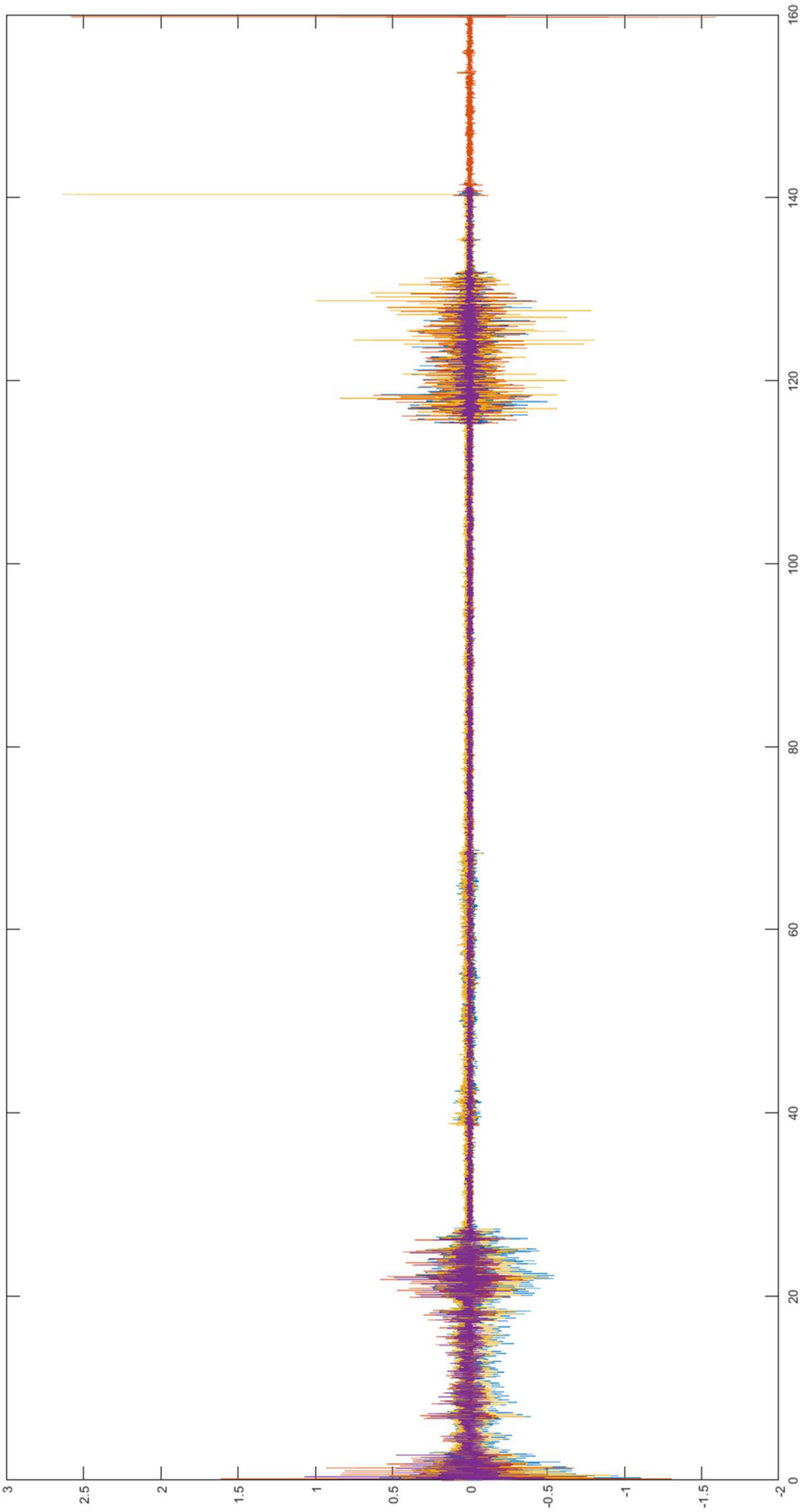

Figure 4: Three measurements with different intensities

4 System Overview

The Distributed Disaster Detection System (D3S) is a centralized, distributed application. It requires multiple phones to be connected to a central server. Each device will evaluate the current sensor data and send abnormal acceleration data to the backend. The backend will then decide whether or not a disaster is likely and send according instructions to the phones being affected. The phones will react by displaying a warning message.

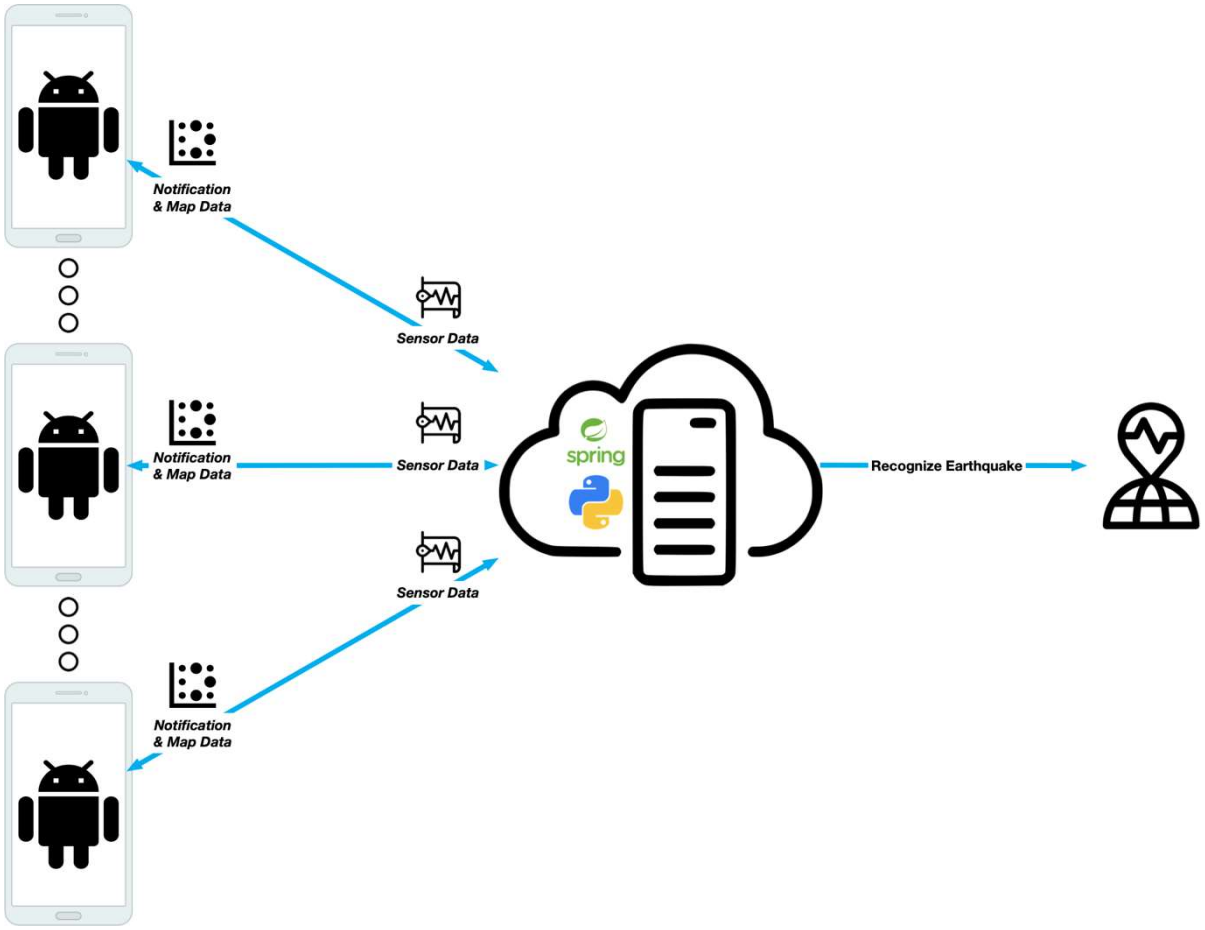

Figure 5: System Overview

4.1 Infrastructure and Data Exchange

The main infrastructure of the application is based on a Spring Boot backend in combination with a Python algorithm. The phones (referred to as devices) will continuously send sensor ping requests to the backend. If a client detects an abnormal movement, it will begin to send

its sensor data to the backend. Every second one bulk of data (referred to as a measurement entry) is sent.

The interface between device and backend is a simple REST-API that stores devices and measurement entries as entities. Additionally, there are two endpoints that are used by devices to register or unregister.

The Python algorithm periodically checks the backend and fetches the current measurement entries. Based on the measurement data, warnings are generated. If a warning is generated after analysing the data, an entity is stored in the backend. Clients will then be informed of the incident after issuing the next ping request.

If a device did not call the ping endpoint for more than ten seconds, it will be removed from the database. This is necessary, because the algorithm needs to know how many devices are present in a region to calculate the probability an incident currently happens in a region.

4.2 Data Objects

This section contains information about the data objects the backend and algorithm uses. The exchange format for all data objects is JSON.

4.2.1 Device

A device is the representation of a phone in the backend. It contains the GPS-coordinates and the last time the device issued a ping. Additionally, measurement entries and a sampling rate can be linked to a device.

4.2.2 Measurement Entry

A single measurement entry contains three arrays. Each containing the recorded acceleration data in one dimension. Such an entry is roughly generated once per second.

4.2.3 Warning

If the algorithm detects an abnormal behaviour, it will add a warning to the database. This warning contains a time and a description.

5 Movement Detection

To detect movements with a smartphone, the accelerometer and gyrometer of a device are used. On most modern smartphones, those sensors deliver precise values for the acceleration and the rotation. The state machine behind the whole D3S movement detection, which is taking place on each mobile device, is visualized in figure 6. The following chapter 5.1 provides a closer look into each state and in its entry/exit methods.

The basic idea of the motion detection is to detect movement of the device and send the mentioned acceleration/rotation data to a Spring Boot server. This is only done, when the device is not currently in use by a user, meaning it is resting on a table. The android application detects whether or not a device is actively used by analysing the gyrometer of the respective device.

A small evaluation showed, that it is indeed possible to distinguish between a resting phone or one that is actively used. This is necessary to get a reliable data source for the later earthquake detection process, as a moving phone obviously emits a large amount of acceleration data.

The basic flow of data between each state can be seen in figure 7.

5.1 State Machine

As described, the Android App has three states:

1. **Measure**
2. **Send**
3. **Idle**

Following are two figures with the first showing an overview of the mentioned states along with all state transitions present. The second figure gives an insight into the data flow present between each state as well as the order in which state transitions will likely be executed during normal system operation.

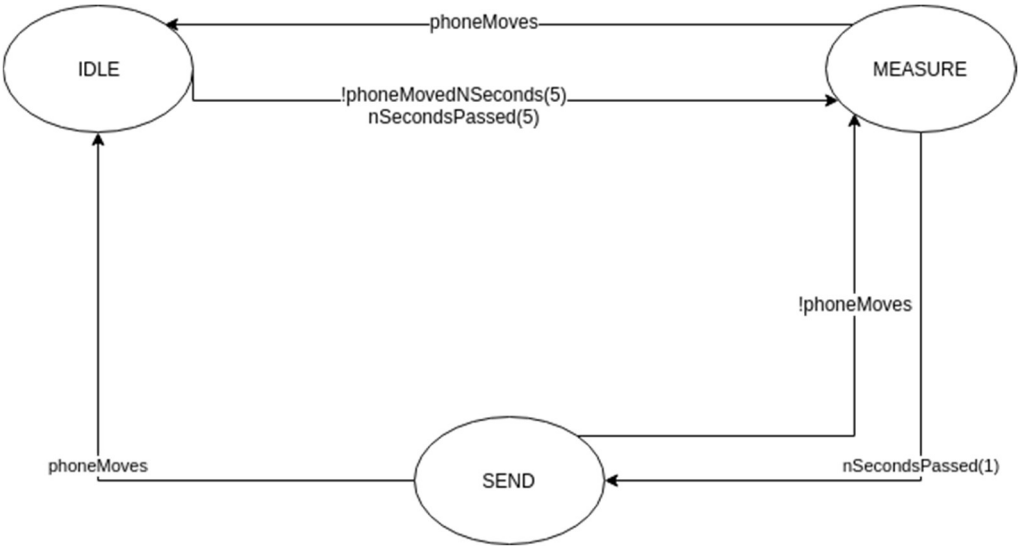

Figure 6: D3S State Machine

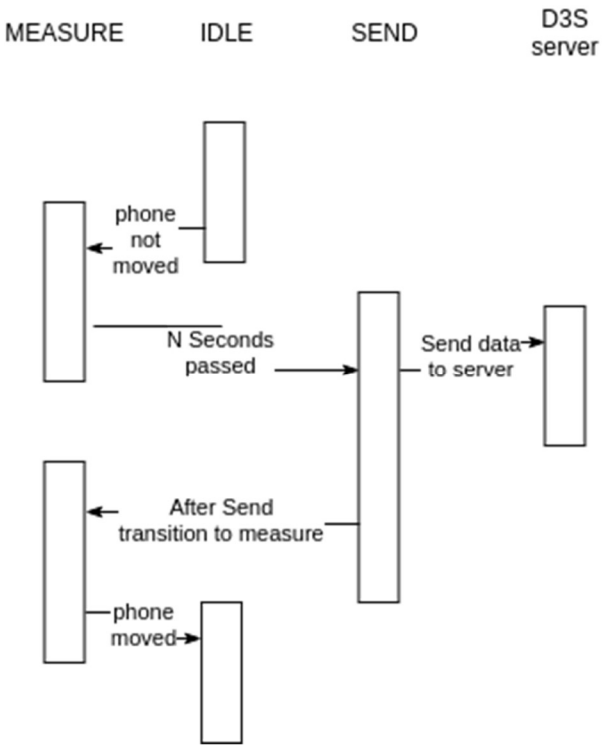

Figure 7: D3S Data Flow

5.1.1 State: Measure

The state “Measure” represents the most important part of the state machine. In this state, the application collects data of the accelerometers and bundles them together into one package. In addition, if the app is not registered when entering this state, which is checked by validating

the UUID assigned from the server, the app registers itself against the server. This is necessary to connect the measured data to a device representation in the backend.

The collected data is normalized to reduce false-positives within the earthquake detection part of D3S. This is done by storing the accelerometer values of X, Y and Z on entry of this state. The collected data is later biased by this amount, which makes detection of patterns easier.

5.1.2 State: Idle

This state is used to represent the phone moving or not collecting data. As a moving phone would break the algorithm and deliver a lot of false-positives, this is essential to keep D3S trustworthy for the users.

Whenever the device is moved, meaning the rotation / orientation is changed, the application transitions into this state. Once in this state, the D3S application unregisters the device from the Spring Boot server and checks periodically if the device stops moving. This would lead into a transition to the “Measure” state.

5.1.3 State: Send

This state represents the device sending the collected data package to the server. For sending this bulk of data, a REST client implementation for Android is used. The app generates a Measurement Entry as described in chapter 4.2.2 and send this entry, together with the current timestamp and the collected data, as an JSON object to the Spring Boot server.

5.1.4 Potential Problems and Mitigation

The major problem with this implementation of the state machine is the sample time. Currently the application is locked to a sample rate of around 60Hz (every 20ms). This, as described in chapter 8, makes it impossible to detect a direction within close range.

In addition, the sample rate as well as all parameters are hard coded within the app. This can lead to problems, where the Spring Boot server cannot be reached and the device cannot transmit its data.

The last problem with the current implementation is the amount of data. Bandwidth is not unlimited, especially in underdeveloped countries. Currently the data is not cleaned and, with the exception of it being normalized, is sent as is from the device. In the future, each device could calculate its own vector of motion and by that reduce the data sent through the network.

6 User Interface

The user interface of the application is kept quite simple. It shows the current backend warning in the middle of the screen. The most recent state of the state machine is shown on top of the screen, while the sensor data is displayed at the bottom. The following figure shows a screenshot of the application.

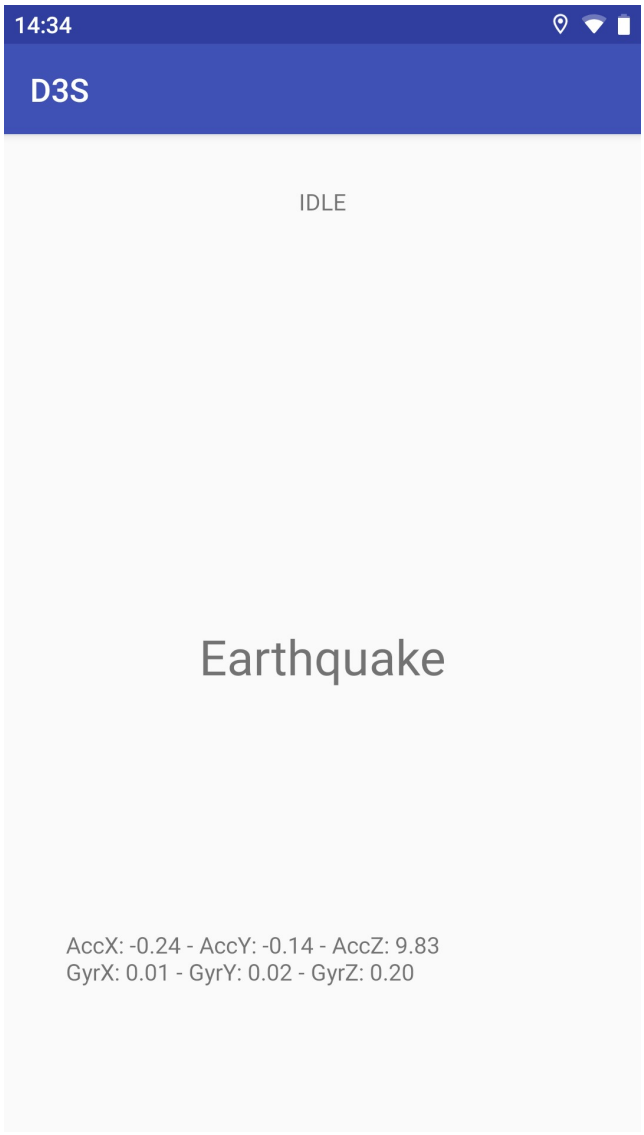

Figure 8: D3S Android Application

7 Earthquake Detection

The earthquake detection, also referred as the “D3S Core Component”, is a Python application split into several threads.

One thread called “Poll” polls the Spring Boot server every second for new devices. When a new device is registered, Poll creates a new thread of type “Slave”, which represents the newly registered device. When a device is deregistered, the according Slave thread will be shut down.

The relationship between those threads is shown in Figure 9.

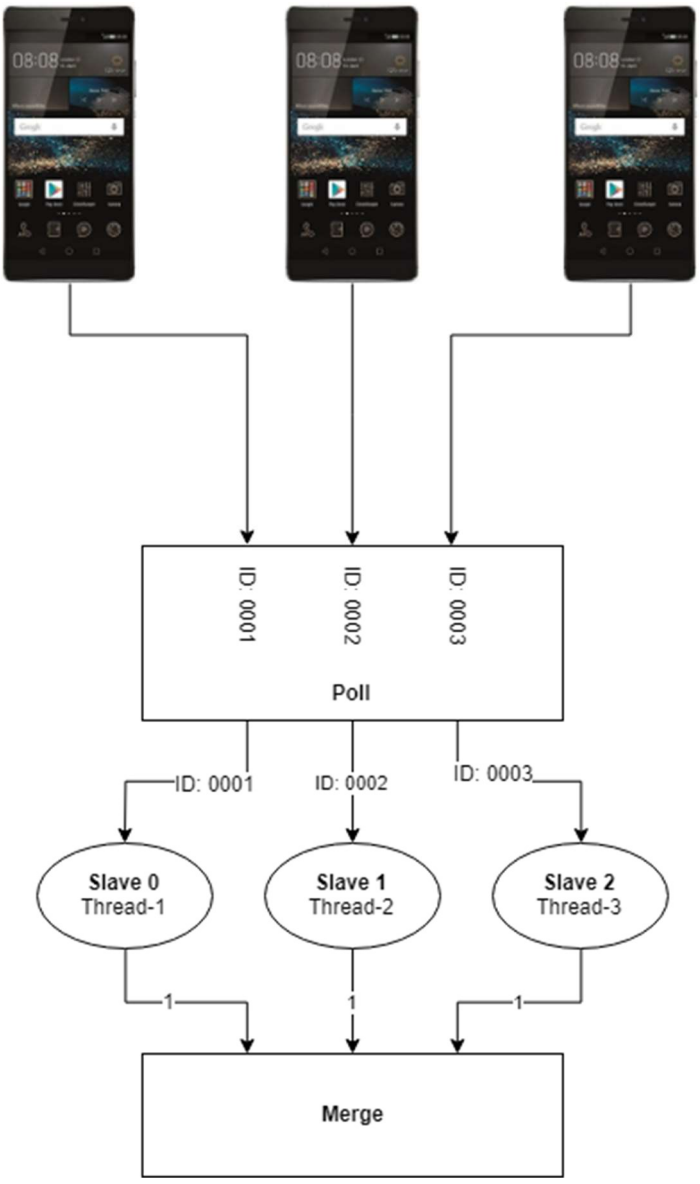

Figure 9: Relationship between threads

Each Slave thread is given the unique identifier (UID) with which the according device is registered at the server. It polls the server for new measurement data every second. As soon as the Slave thread receives new data, which will be 60 values for x, y and z (1 second of measurement), it calculates the mean value and standard deviation of it.

For merging the data, a single artificial neuron is used which is modelled after a “Spiking Neuron Model” based on the eponymic book from Wulfram Gerstner and Werner M. Kistler (Gerstner & Werner M., 2002). The basic idea behind this model is that a neuron only gets spikes from its inputs. The higher the firing rate of a neuron on the input of another neuron is, the higher the hypothetical weight between those two neurons would be. For this, a neuron does not only sum up all inputs. The weight of a neuron depends on the time since the last input spike occurred. This means the neuron weight decreases over time and it only increases, when input spikes occur in a rather high frequency (see Figure 11).

For this case, every Slave marks a postsynaptic neuron and with that an input for our neuron. A spike is modelled by an “1”. Every received “1” will be added to the weight of the neuron, which decreases by 0.1 every 100 milliseconds.

$$g(t,x) = \sum(x) - 0.1t$$

This means a single input spike is nullified after one second. This is important, so that no single device can produce an alarm by repeatedly (every 1 second) sending spikes to the neuron.

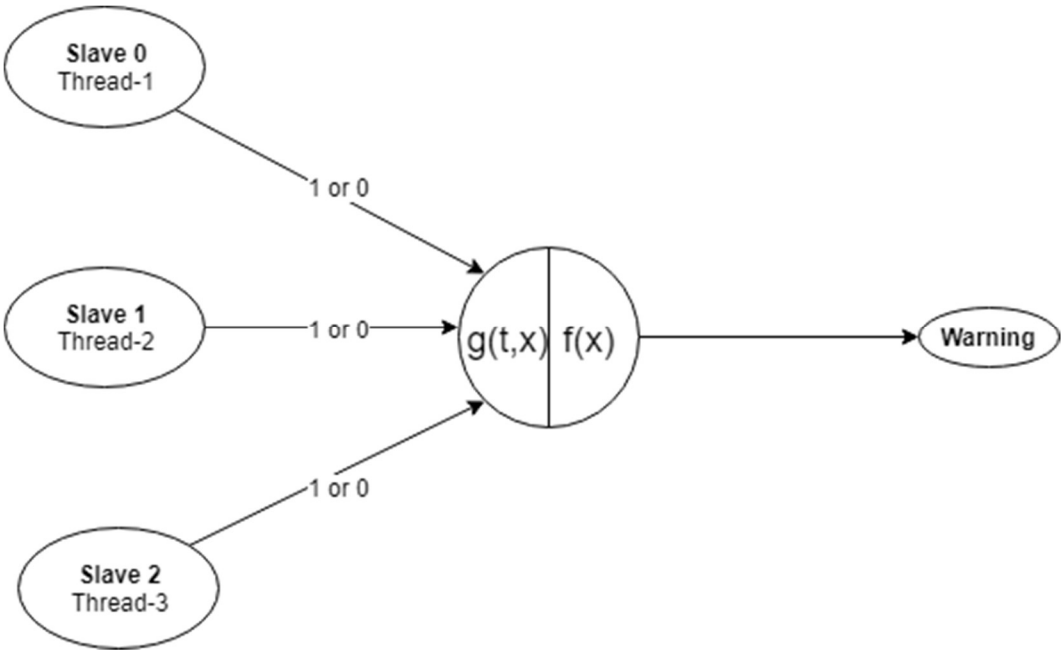

Figure 10: Neuron in D3S.

The output function is a single comparison with a bias based on the number of registered devices. If more than 2 devices are registered and the weight of the neuron exceeds the number of devices times 0.7, the neuron fires.

$$f(x) = \begin{cases} 1, & x > bias \\ 0, & x \leq bias \end{cases}$$

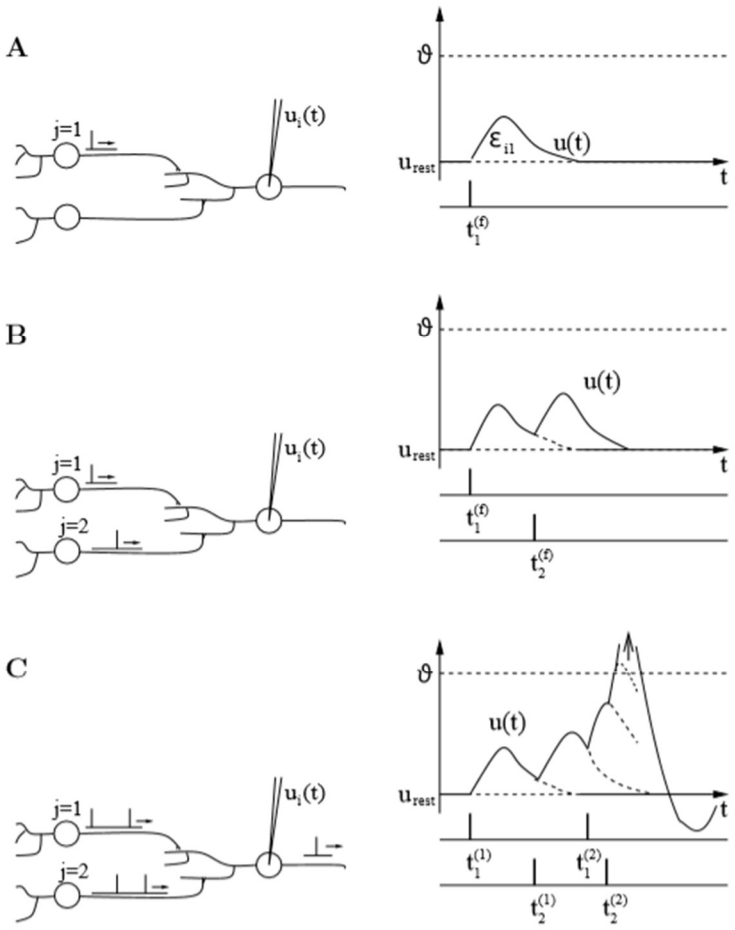

Fig. 1.3: A postsynaptic neuron i receives input from two presynaptic neurons $j = 1, 2$. **A**. Each presynaptic spike evokes an excitatory postsynaptic potential (EPSP) that can be measured with an electrode as a potential difference $u_i(t) - u_{\text{rest}}$. The time course of the EPSP caused by the spike of neuron $j = 1$ is $\epsilon_{i1}(t - t_1^{(f)})$. **B**. An input spike from a second presynaptic neuron $j = 2$ that arrives shortly after the spike from neuron $j = 1$, causes a second postsynaptic potential that adds to the first one. **C**. If $u_i(t)$ reaches the threshold ϑ, an action potential is triggered. As a consequence, the membrane potential starts a large positive pulse-like excursion (arrow). On the voltage scale of the graph, the peak of the pulse is out of bounds. After the pulse the voltage returns to a value below the resting potential.

Figure 11: Draft from Spiking Neuron Models by Gerstner and Kistler (Gerstner & Werner M., 2002)

The current implementation of the prototype is only based on this single neuron, which receives an input spike for every measurement with a standard deviation over 0.05. Although planned, it does not evaluate the GPS coordinates. This could be solved by implementing a layer of several neurons, where each neuron represents an area and the height of one spike could be proportional to the distance between the geographic centre of the area covered by it and the coordinates returned by the corresponding measurement.

Experiments were made with sending the mean value or standard deviation instead of a “1” to the neuron, but the results were not as satisfying as using the Spiking Neuron Model. One main problem by letting the input weights correspond to the measured data is, that a small number of devices can trigger an earthquake alarm, when they were stimulated too hard. For the idea behind D3S, a vague tremor should be not counted lesser than a strong one.

As soon as the neuron fired at least 10 times consecutively (due to a low pass filter), the D3S Core Algorithm sends a warning to the Spring Boot server, which forwards it to every connected phone (see Figure 12).

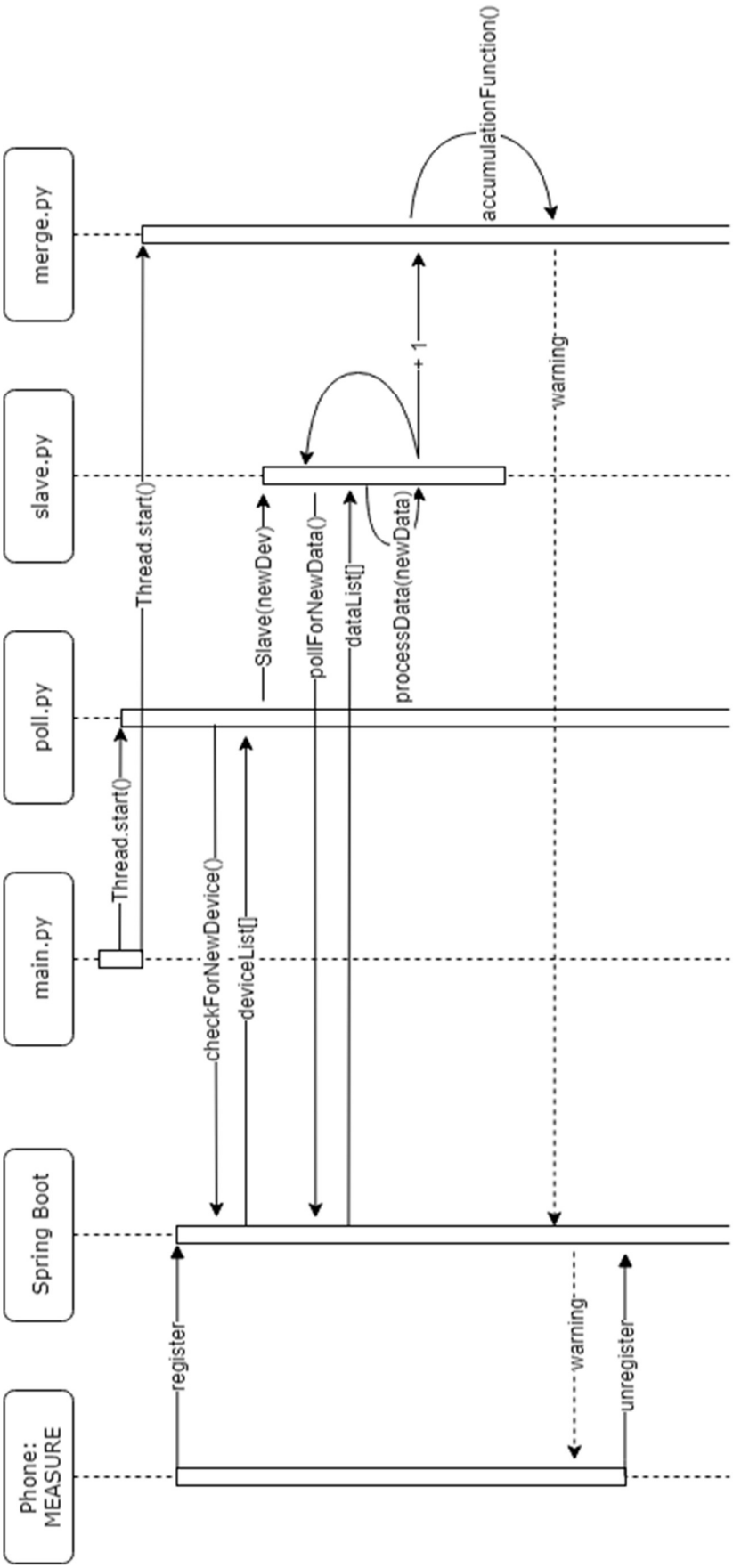

Figure 12: Sequence diagram for the D3S backend

8 Problems and Further Improvements

One large problem was the fact that the sensors of different devices work differently. Some devices have a larger sampling rate. Others have only a very limited sampling rate. Additionally, the detection range (maximum amplitude) varies from devices to device. For large movement, bad sensors start to clip. This essentially means they generate high frequency, high amplitude measurement data which is hard to compare to sensors that have a higher tolerance.

Also, depending on the resolution of the sensor, small movements might not be detected with the same precision. This can complicate the comparison of data of different devices.

The provided application is able to detect if many devices detect abnormal movement at the same time. To generalize this, the algorithm could group “affected” devices together and only send notifications to the devices that are nearby. This could potentially be done by forming the convex hull of the affected devices and expanding the radius by a certain percentage.

Additionally, a forecast could be made by capturing the “movement” of the zones over time. The application would then be able to warn users that an earthquake is likely happening at their location in a short amount of time.

Finding the direction the shockwaves are travelling, however, is not easy, because it requires a large scale environment. Simulating this on a table is impossible, because of the speed the waves travel. Shockwaves generally travel with the speed of sound. For wood this is around 4500m/s. This means the shockwaves travel with a speed of around 4.5 meters per millisecond. The sampling rate, however is at best 250Hz (usually 50Hz). This means every four milliseconds one sample is generated.

So the wave travelled already around 18 meters before the first sample is generated. The largest, non-segmented table we had at our disposal was around two meters in length, which was far too short to capture the wave travel movement.

9 Conclusion

The Distributed Disaster Detection System (D3S) is a working prototype of a smartphone-assisted system to aid the detection of earthquakes. The system presents a possible solution to enhance early-warning systems for earthquakes or other catastrophes using a network of connected smartphones.

With this work, we have shown that it is possible to use smartphones to detect earthquakes. Therefore, an application was developed that collects data from the sensors (accelerometer, gyroscope, GPS, etc.) of a smartphone and sends the data to a server which is responsible for the evaluation of the received data. When the server concludes from the cumulated data of a certain region that an earthquake is about to arise or already in progress an alert will be sent to all connected devices in that region informing the users of the app of an incoming or ongoing earthquake. To detect earthquakes an algorithm was developed evaluating data collected by the sensors of connected devices (insert ref to algorithm section).

During development, we noticed that the sensors in the devices differ greatly in regard to accuracy, sampling rate, frequency range, etc. One of the challenges was to find a way to extenuate the varying measurements without falsifying the collected data. To mitigate those differences and to ensure the correctness of the algorithm we ran a variety of tests using smartphones from different brands (LG, Xiaomi, Sony, Huawei, Samsung) to find reasonable limits and thresholds for the algorithm to produce reliable results.

The current version of the application is periodically sending data to the server, which is then processed by the earthquake detection algorithm on the server. Since the data being sent is rather small and does not require a lot of bandwidth the impact on the users data plan or smartphone performance is almost negligible. In the future, this behaviour could be improved to further limit the data exchanged between server and client by implementing the application to pre-process the data collected from the smartphones sensors to detect ongoing or upcoming earthquakes directly on the device. Furthermore, the current version of the application constantly polls the server for status updates resulting in a rather larger number of unnecessary requests. To improve this behaviour future versions could be implemented such that the server sends status updates to each connected device as soon as the server detects an event changing the state of the application (such as the detection of an earthquake) which will greatly reduce the load on the server and network. Additionally, the system could be extended to not be limited to a regional scale. Through the usage of widely available commodity hardware (smartphones) it is possible to extend D3S to work on a global scale at relatively cheap cost.

Bibliography

Gerstner, W., & Werner M., K. (2002). *Spiking Neuron Models.*