Research Article

Journal of Sensor Networks and Data Communications

Optimizing Telemedicine Framework Using Fog Computing for Smart Healthcare Systems

Michael Enbibel Kelkile*

International Research Scholar, Ethiopia

*Corresponding Author
Michael Enbibel Kelkile, International Research Scholar, Ethiopia.

Submitted: 2023, Apr 15; **Accepted**: 2023, May 01; **Published**: 2023, Jun 22

Citation: Kelkile, M. E. (2023). Optimizing Telemedicine Framework Using Fog Computing For Smart Healthcare Systems. *J Sen Net Data Comm, 3*(1), 01-24.

Abstract

The main aim of this research is to optimize the telemedicine framework using fog computing for smart healthcare systems in IoT ecosystem. In this research we have studied about the historical background, different architectural designs and approaches of the pre-existing telemedicine and smart healthcare systems. This research mainly focuses on the drawbacks and different issues that arise while developing and implementing a smart telemedicine system. We used Fog computing or fogging to solve the problems and also to optimize the telemedicine framework with 263 ms for smart healthcare systems. Generally this research work proposes, validates and evaluates telemedicine frameworks using Cloud Computing and Fog computing for smart healthcare systems. The research also shows and guides how to optimize and overcome the drawbacks of the issues that arises mainly the Network Latency issue. The finding in this project has a guide for the future in-order to come up with a sharp edged, most reliable and robust smart healthcare system in telemedicine framework for IoT ecosystem. It can also be used as a part of a smart healthcare system in the Smart cities in the future.

1. Introduction What is Telemedicine?

Telemedicine is a broad umbrella word that consists of different infrastructures like information communication technologies and healthcare institutions. The term was discovered in 1970 by the World Health Organization (WHO) for diagnosing, treating, and prevention of diseases [1]. As the term implies it is used to treat a patient from a distance [1]. In general, telemedicine uses information technologies to store, retrieve and execute the results that are uploaded by the medical personnel in a given cross-platform. The platforms can be a mobile application or a desktop application that uses different communicational tools that are available in information technology like cloud computing, big data analysis, and computer networking to maintain the system.

1.1. Typical Telemedicine Framework

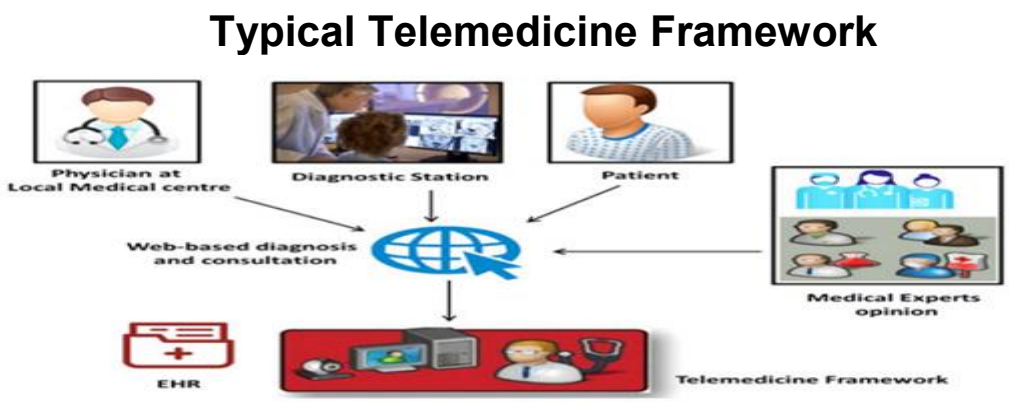

Figure 1: Typical Telemedicine Frameworks [10]

1.3. The Historical Background of Telemedicine

The term was originated in 1970 but the information was exchanged in Europe using heliographs or bonfires. During the civil war, a telegraph was used to order medical supplies. In 1930 radio was used to transfer medical information in rural areas. It was practically implemented by the National Aeronautics and Space Administration (NASA) when physicians on earth successfully monitored the health of astronauts in space [1]. They were able to monitor the blood pressure, heart rate, respiration rate, and temperature of the astronauts in space from earth.

In 1964 after the invention of television in 1950 there was successful video communication between two hospitals found in America. In 1967 there was a system deployed to link the United States Airport terminals to hospitals [1].

2. Types of Telemedicine

2.1. Asynchronous Telemedicine

This type of telemedicine mainly focuses on the store, retrieves and forward medical information when needed. This system doesn't need live communication or in other words, both parties don't need to communicate or connect to the system at the same time [1]. Datum can be collected, organized, and stored. Any party of the system can access or retrieve the datum for further diagnosis and analysis whenever it is feasible.

For example, an ECG image with the patient's medical history can be sent via email to a specialist that is located in a different location for further examination and diagnosis.

In this type of telemedicine smart healthcare system, we need to be careful on the Datum generated by both parties to store, retrieve and forward datum from the system and into the system. This is directly proportional to the number of users that use the system. In other words as the number of users that log in to the system increases the data that is going to be produced also increases so we need to design the database management system with the general architecture of the system. Big Data will be generated as the number of users increase. Hence we also have to use Big Data manipulation tools and techniques.

2.3. Synchronous Telemedicine

This type of telemedicine differs from the above because both parties should login simultaneously. In other words, it needs live communication between the health professional and the patient using wired or wireless devices for live communications such as Microphone, Speaker, and a video camera. In our context, Synchronous telemedicine can be implemented using LIVE communication tools like videoconference and Audios.

In this type of telemedicine, we need to focus more on the network architecture and network types that are useful for having live communication [1]. This includes the internet speed and the bandwidth has a significant impact in having a smooth LIVE communication. So this type of telemedicine application requires advanced wired and wireless communication network devices and network types are required with an optimal computer LIVE broadcasting network.

In this scenario, as both parties need to login and communicate at the same time simultaneously we need to have a LIVE communication so we need to think of the network capability and efficiency of the network so we need to think of different network types like cloud networks and hybrid approaches to maintain live communication. Better Delay than Loss for Live communications.

2.4. The Services of Telemedicine

Telemedicine services can be of two types that are **primary care service and specialist referral service** to achieve medical care [1]. The specialist and the patient can use video conference.

2.5. Primary Care Services

This type of telemedicine services uses **asynchronous telemedicine** where the health professionals and the patient uses the data centers for storing, retrieving and forwarding information from the patient to the health professional without LIVE communication just by mining data about the patient medical history for further diagnosis and treatment it can be combined with nurses' visits [20]. In this type of telemedicine service the main problem is the **database management** as the number of users increase so we need a systematic approach for managing and manipulating **Big Data.**

2.6. Specialist Referral Service

This type of telemedicine service uses **synchronous telemedicine** where both parties that are the patient and the Specialist should login to the system simultaneously at the same time. Hence it uses a LIVE communication between the two parties [20]. The main problem in this system is the type of network used meaning because we use video conferencing and online examination the network should be robust, reliable and accessible. So we need a wider bandwidth network connection and a fast internet access with a **cloud network** for cost minimization. Better delay than loss should be considered during communication. We can use a hybrid cloud network to minimize the cost and accessibility.

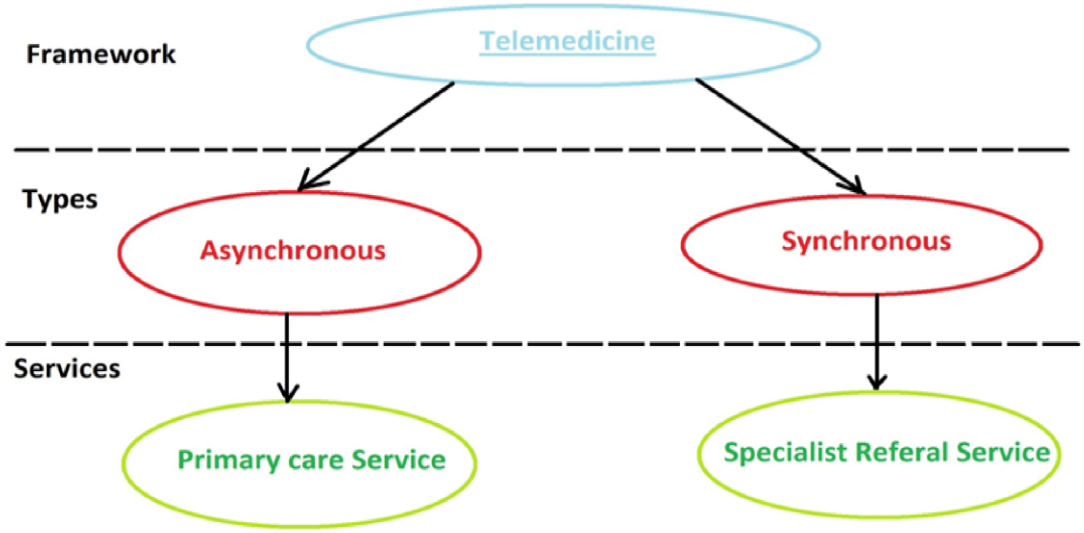

Figure 2: Block diagram of types and services of Telemedicine

3. Methodology Fog Computing

Fogging is a computer networking model that is found between the cloud service and the devices we use in the network .Fogging can be used for dumping processes and resources on the device before the cloud; mostly it is embedded on the network devices that are part of the system, while the data is going to be stored in the private or public cloud system provided by cloud service providers like Amazon or Google .Fog computing reduces the processing time with few resources consumed while manipulating a given system. In January 2014 it was first implemented by **Cisco Systems** [12].

The main advantage of using fog computing is for resource management and allocation through the network for minimizing the latency and increase the efficiency of the system by decentralizing the nodes while storing the information on the cloud [15].

3.1. Benefits of Fog Computing

Fogging has several benefits with reducing latency and increasing the speed of processing for many cross-platforms like Internet of Things (IoT), Big Data analysis, or Smart systems that are confined in Smart cities [6].

A. Reduced Latency

Fog computing reduce latency because it is much closer to the edges than a device to cloud approach, hence while we are using fog computing in between the device and the cloud the fog will act as an intermediary layer that can be used for facilitating the interaction of the devices with the cloud by communicating both ends [24]. This will result in reduction of latency during the data flow of the system.

B. Privacy

When we compare the direct cloud approach with fogging we can maintain a reduction of propagation of datum, the data center that can't be controlled by the system user is the one which makes the system less secured. Hence if we use fogging the system will become highly secured.

C. Energy Efficiency

Using fog computing we can easily increase the energy efficiency of the sensors on the system using sleep mode. We can also use gateways as proxies to extend the time of hibernation. While the sensors are in hibernate mode, the gateway will be responsible for any calls or system updates until the sensors wakes up.
On the other hand those who consume energy for computations on the system can be minimized by battery embedded nodes on the fog layer.

D. Bandwidth

With respect to a direct cloud approach, fogging reduces the datum which is going to be stored into the data centers. One of the ways in which we can reduce the datum that is going to be stored is by filtering, analyzing, pre-processing or compressing the datum that is going to be stored.

E. Scalability

Internal computation using fogging will increase the scalability of the telemedicine framework as well as the smart healthcare system by minimizing the centralized resource sharing that brought dead-locks to our system.

F. Dependability

By fogging we can also increase dependability issues of our smart healthcare system. We can also avoid redundancy by using nodes that can give same functionalities.

The nodes that are distributed closer to the sensor using distributed

data flow will result in independency of the system more than those who use centralized bottleneck architecture for resource allocation.

G. Context

As we are using fog computing the fog nodes are the first system components that can identifies about a given action held in the system [26].

4. Identified Issues

4.1. Infrastructural Issues: This issue arises during implementation phase while we try to implement our smart healthcare system. For example when we are using a cloud infrastructure we need to have a stable agreement with the cloud service vender as well as the internet service provider for sustainable service offer. In-Addition our smart healthcare system needs a higher bandwidth and in some areas while the cost might be expensive for both the system developer as well as the user.

4.2. Implementation Issues: In this research Implementation is also one of the issues that arise due to vast need of System Requirements and financial budgets to develop and test the smart healthcare system. In-Addition we need to give training for village end technicians, IT staff and local doctors using a user guide manual of how to use the system.

4.3. Acceptance Issues: For village doctor and villagers, using high end technology may be too obstructing. However, once the benefits are seen, the acceptance rate will likely be high such as has been seen with mobile telephony and rural internet services.

4.4. Data Management Issues: - Big Datum will be produced while the health professionals as well as patients start signing up and using the system. In this scenario the patient history and the patient record we will be stored on the cloud and hence as the number of patients that use the system increases also the Datum will also increase because they are directly proportional.

4.5. Security Issues: - All the information about the patient should be kept safe and secured while storing all data in Data warehouse and Cloud. So we need to have a highly secured and authenticated login platform for both end users.

Bottleneck system approach issues: - This issue arises while our smart healthcare system uses directly the cloud all the users will login into same cloud service provided by the system and in this case while they are sharing a common resource at the same time **will result in deadlock [10]. Network latency issues: -** This issue arises while we are using and accessing Telemedicine framework for smart healthcare from the cloud. Latency is the time delay that happens when we are trying to access resources from the cloud and internet service provider [26]. In this research we have used this issue mainly in-order to optimize telemedicine framework for smart healthcare systems. Generally the latency is the time it takes to access a given resource from source to destination as we used Round Time Trip (RTT) to calculate the time delay using Time stamps.

5. Solutions

Optimizing the telemedicine framework by using the latest network technology like fog computing or fogging.

Using **Fog Computing** we can maintain agility across the network resulting in which it is going to reduce congestion and Latency [12].

Using **Fog Distributed Data Flow** will result in the elimination of Bottlenecks that come from centralized Computing systems [14].

6. Fog Computing Main Methodologies

6.1. Distributed Data Flow (DDF) Topology

This type of topology is mainly used to reduce the latency of the system. This type of network topology uses online servers and services that are provided by the cloud [10]. It also uses fog computing to distribute the system with various locations integrated with the edge layer that has sensors and different devices that makes up the smart healthcare system in telemedicine framework.

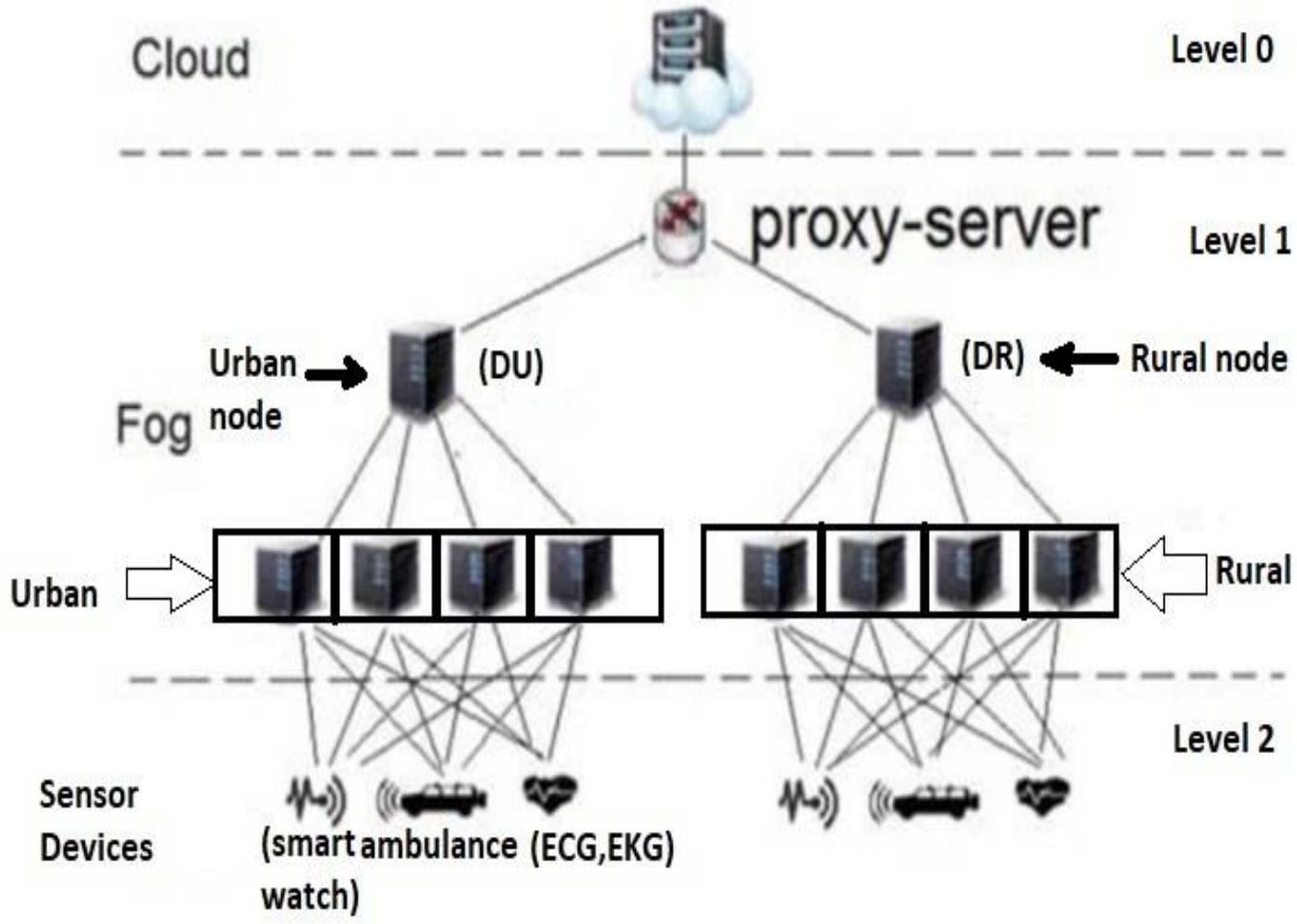

Figure 3: Topology of Distributed Data Flow (DDF) using Fog computing

6.1. System and Software requirements and steps for simulating fog computing in a smart healthcare environmen

Step 1:- . First we need to download the ifogsim simulator from the cloudsim lab that is provided by Melbourne university for simulating and testing fog computing with cloud computing so we need to visit The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne using http://www.cloudbus.org/cloudsim web link in- order to directly access resources that are provided by the Melbourne university CLOUDS lab [22].

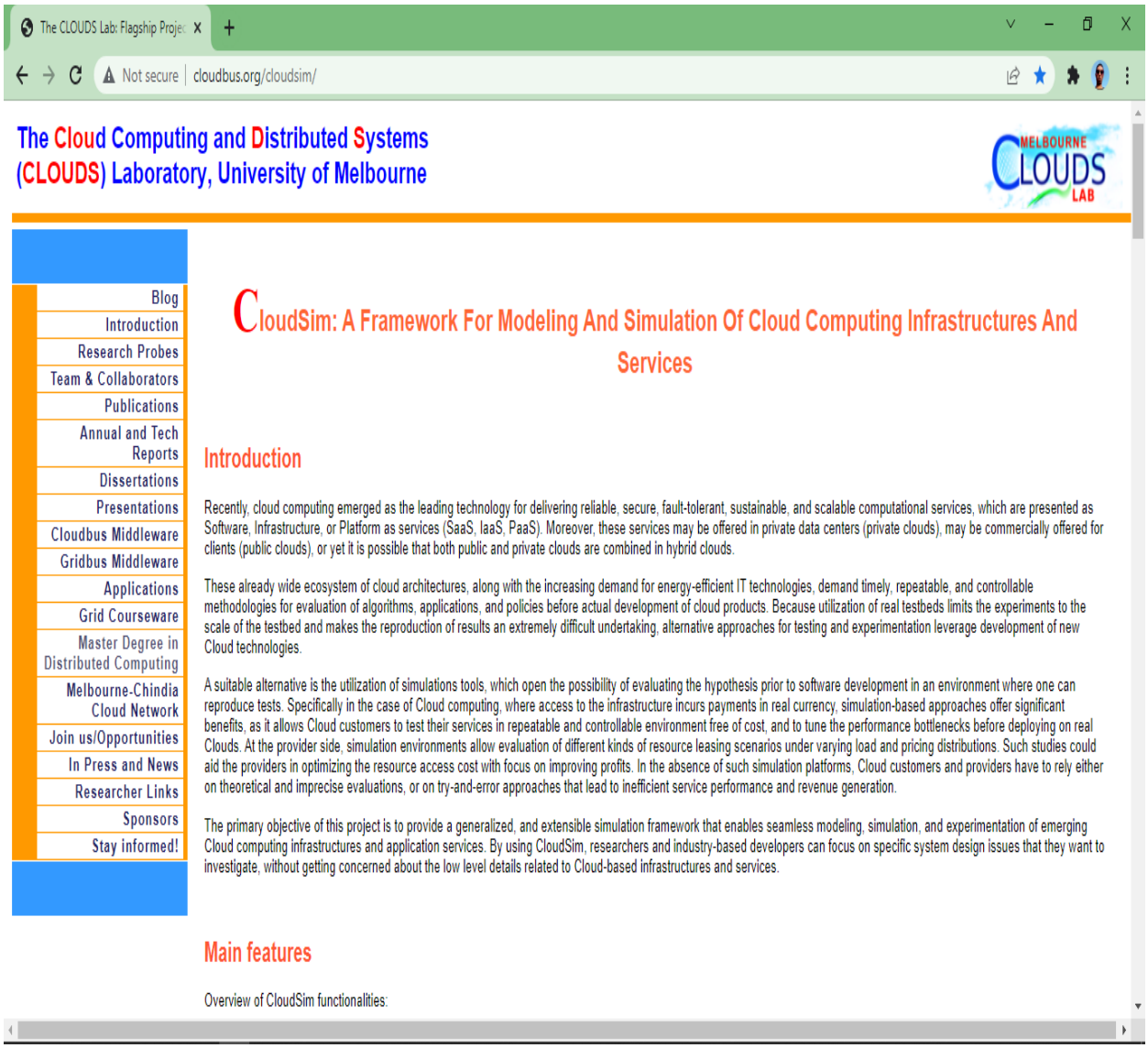

Figure 4: opening the CLOUDS Lab of Melbourne university

Step 2:- After we access the CLOUDS lab from Melbourne University in our case we are going to use fog computing for optimizing our telemedicine framework for smart healthcare system so we need to go for the fog simulator provided by the university. So we will scroll down until we found the ifogSim tool kit for fogging.

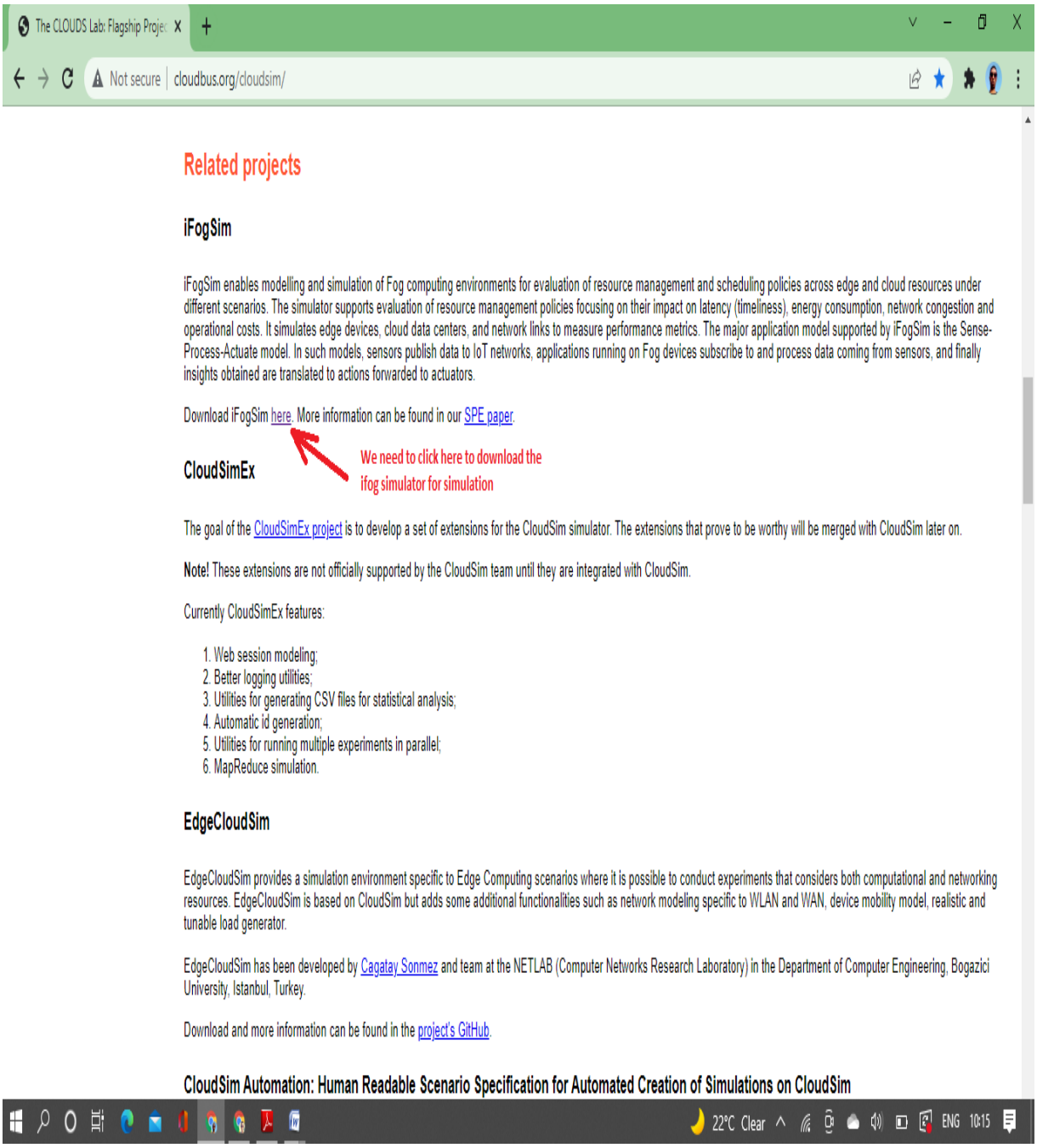

Figure 5: downloading iFogSim files

Step 3:- When we click the link that is provided by the university for IfogSim it will automatically redirect us to the GitHub that is linked with the zip file to be downloaded. [22]

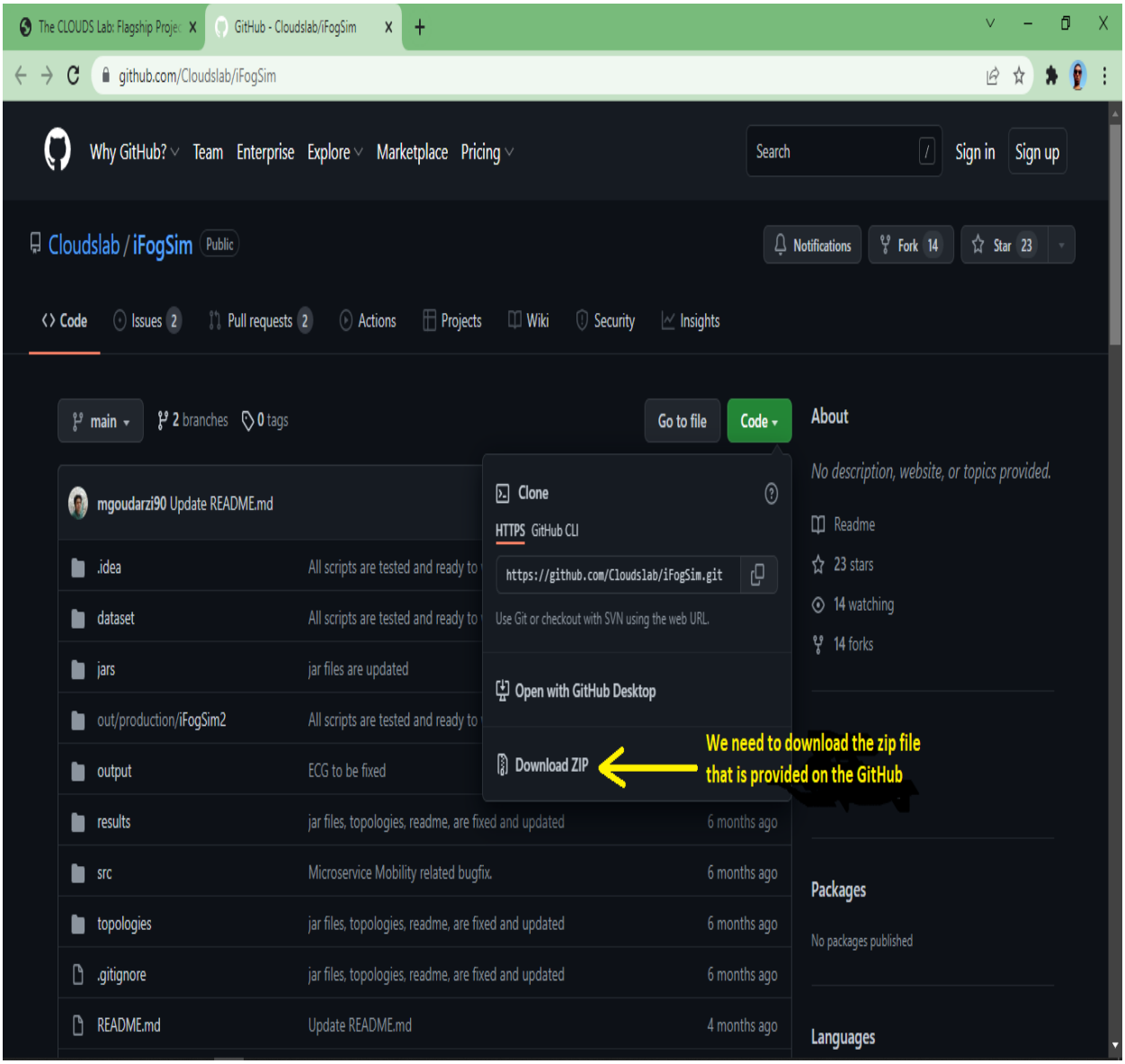

Figure 6: Redirecting to the GitHub repository to get the zip file

Step 4:- Then when we click download zip, it will automatically download the ifogSim simulator zip files into our computer and we will have a zip file which says IfogSim Main.

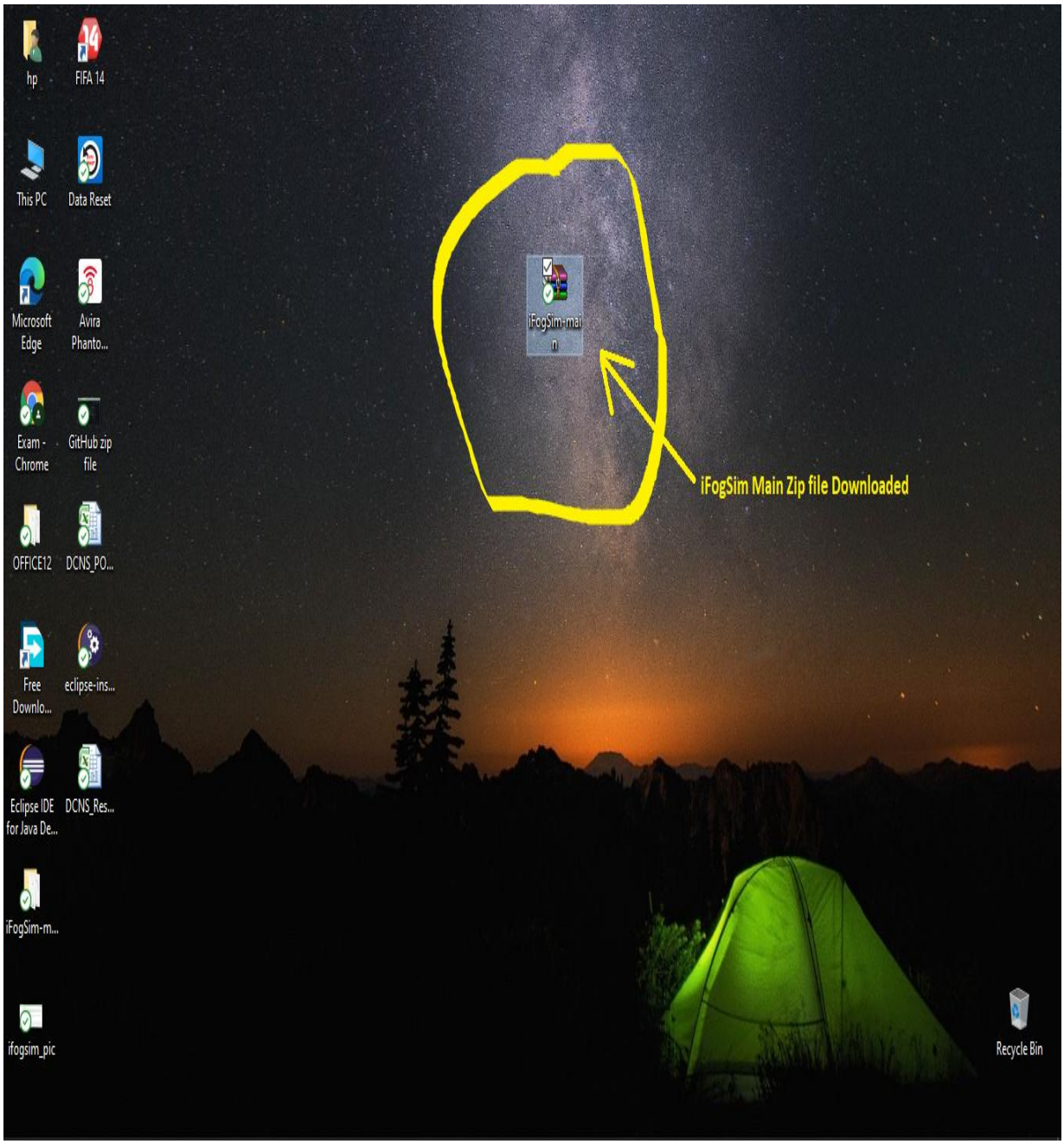

Figure 7: Fog zip file downloaded from the GitHub that was provided by Melbourne University CLOUDS Lab

Step 5:- After we download the zip file from the GitHub we need to extract the file for further use in our Java project.

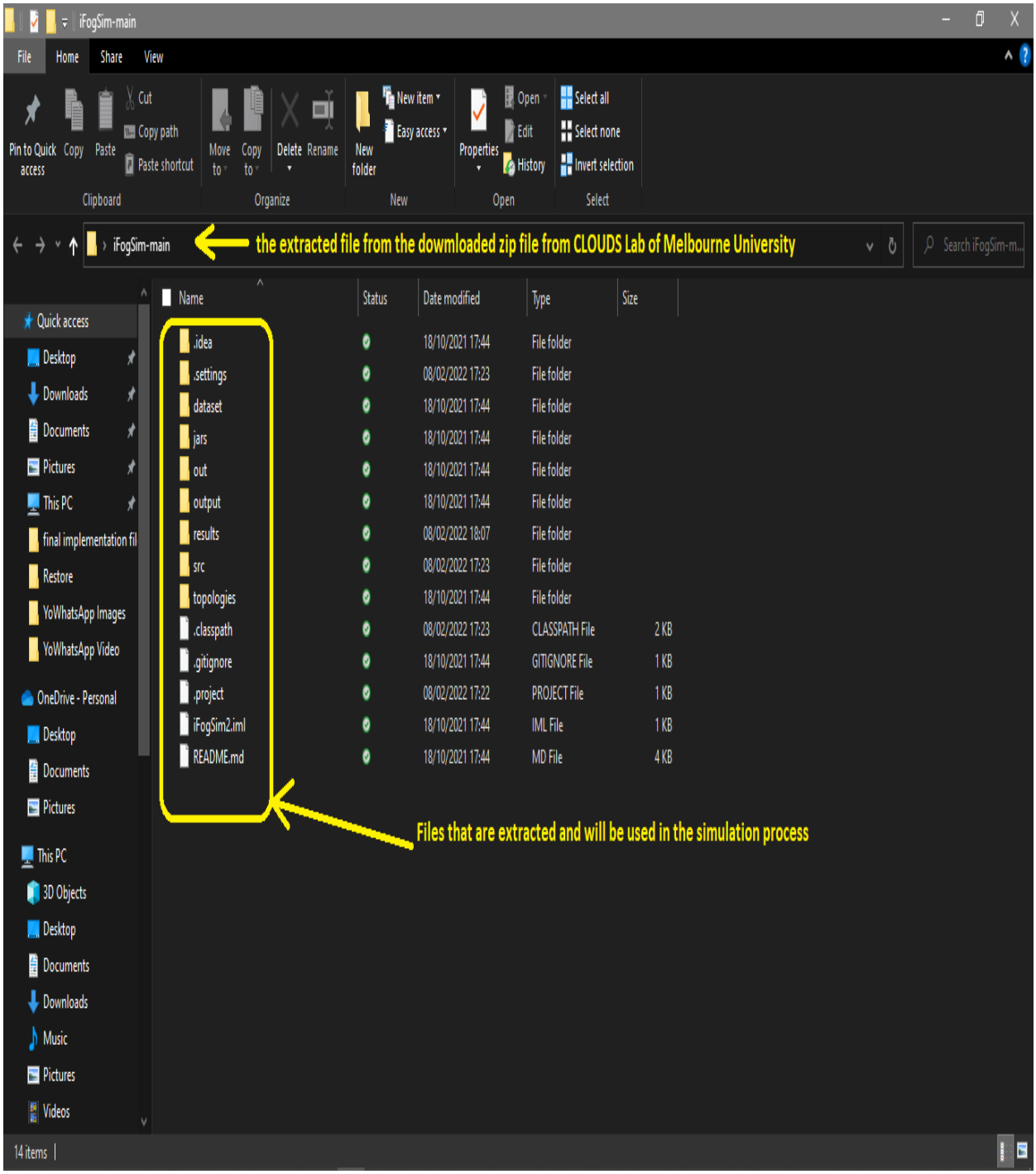

Figure 8: extracting the downloaded iFogSim

Step 6:- After downloading and extracting iFogSim Zip files from Melbourne University CLOUDS lab the next step is to install Eclipse software for java developers in order to create and manage our project using the iFogSim files that are already downloaded. So we need to download and install Eclipse for Java developers' software.

Step 7:- After downloading and installing the Eclipse java developers' software we need to start the Eclipse software to create a project for simulation.

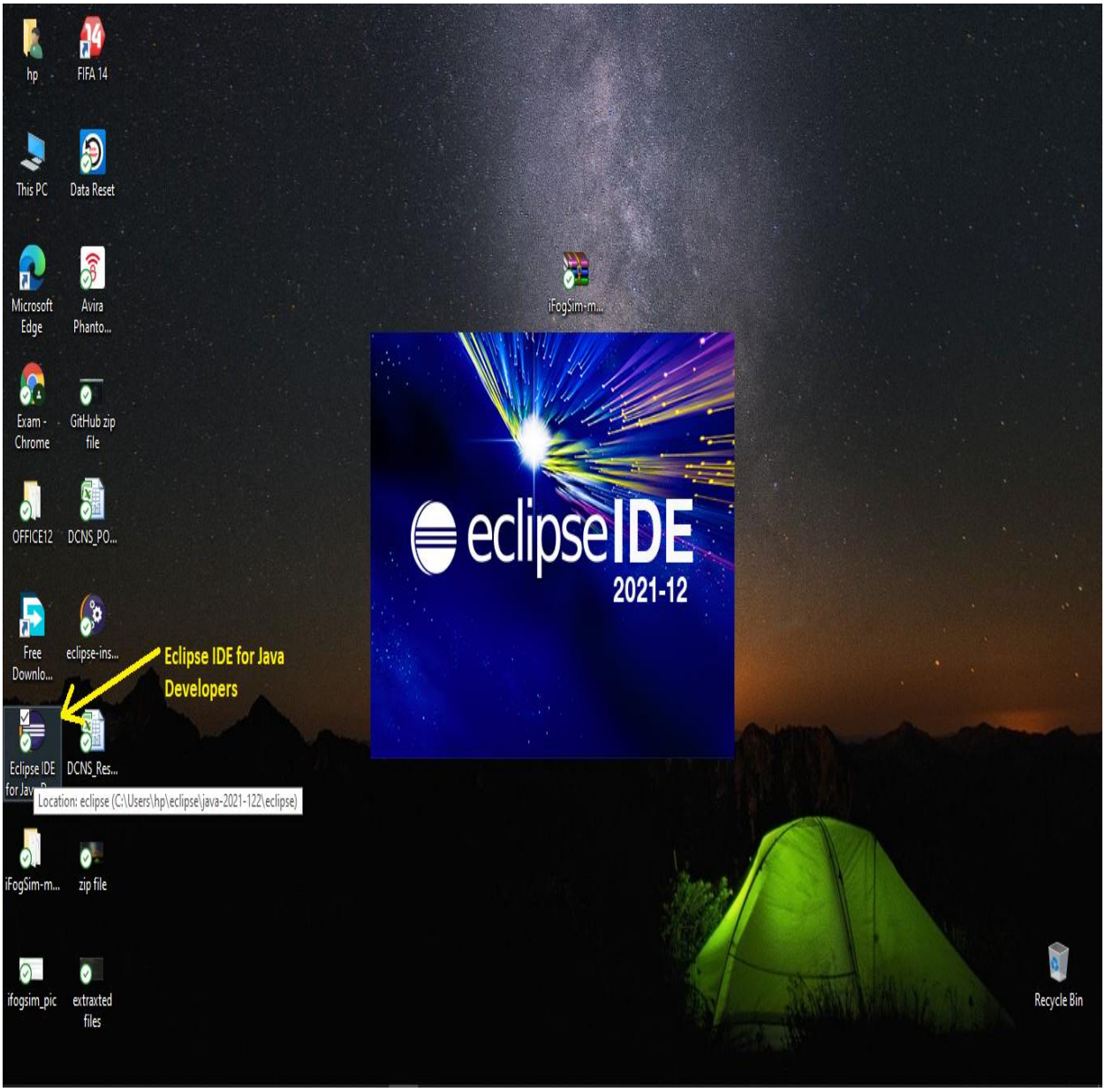

Figure 9: initiating our Eclipse IDE java developers software

Step 8:- After opening the Eclipse IDE for Java Developers we will have a wizard that asks for our confirmation in launching or executing the Eclipse IDE workspace.

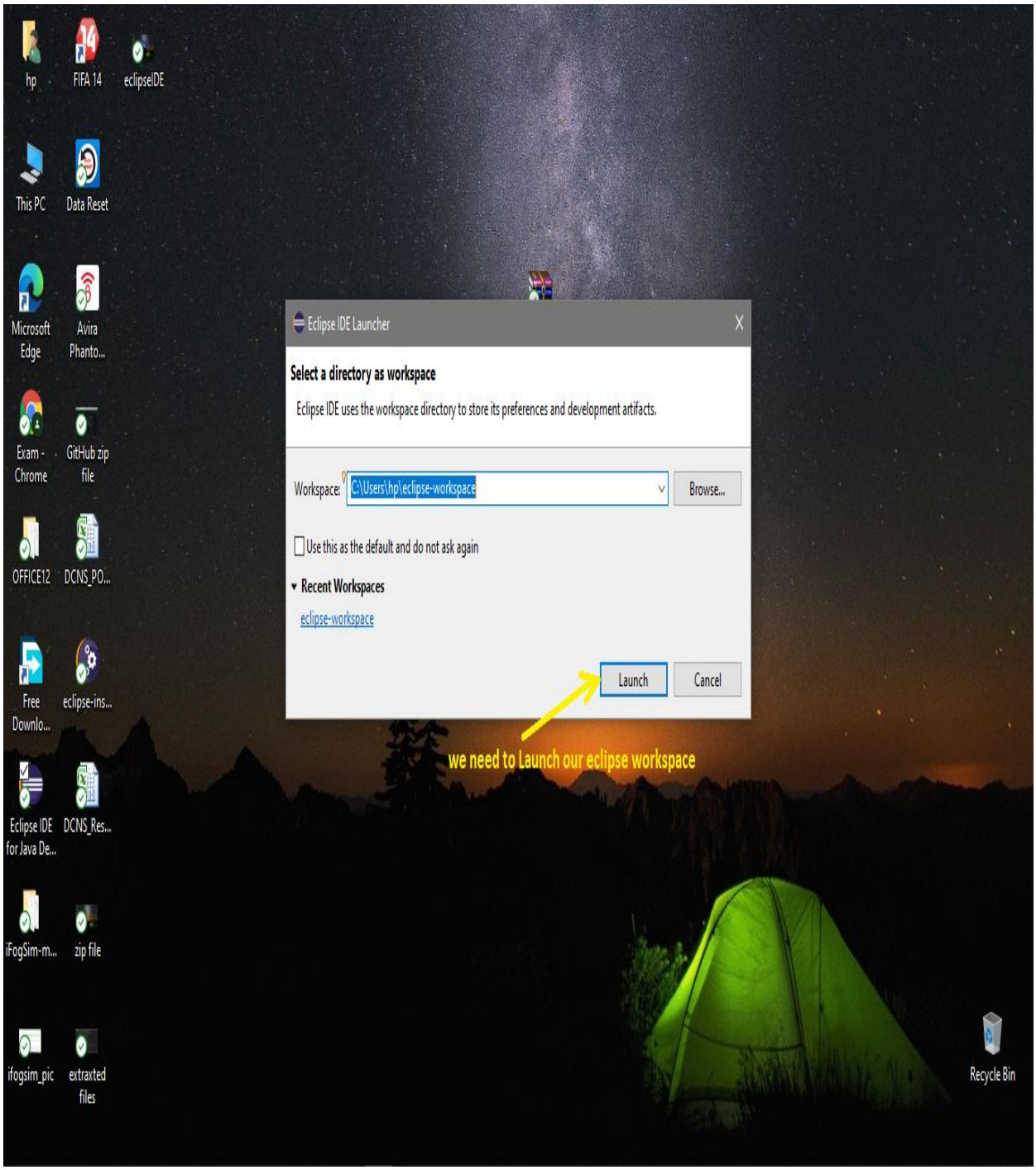

Figure 10: Launching our Eclipse Java Workspace

Step 9:- After launching our Eclipse IDE for java developers we need to create a project which is going to be connected with the iFogSim Simulator on the Melbourne University. So first we need to create a new java project as shown below.

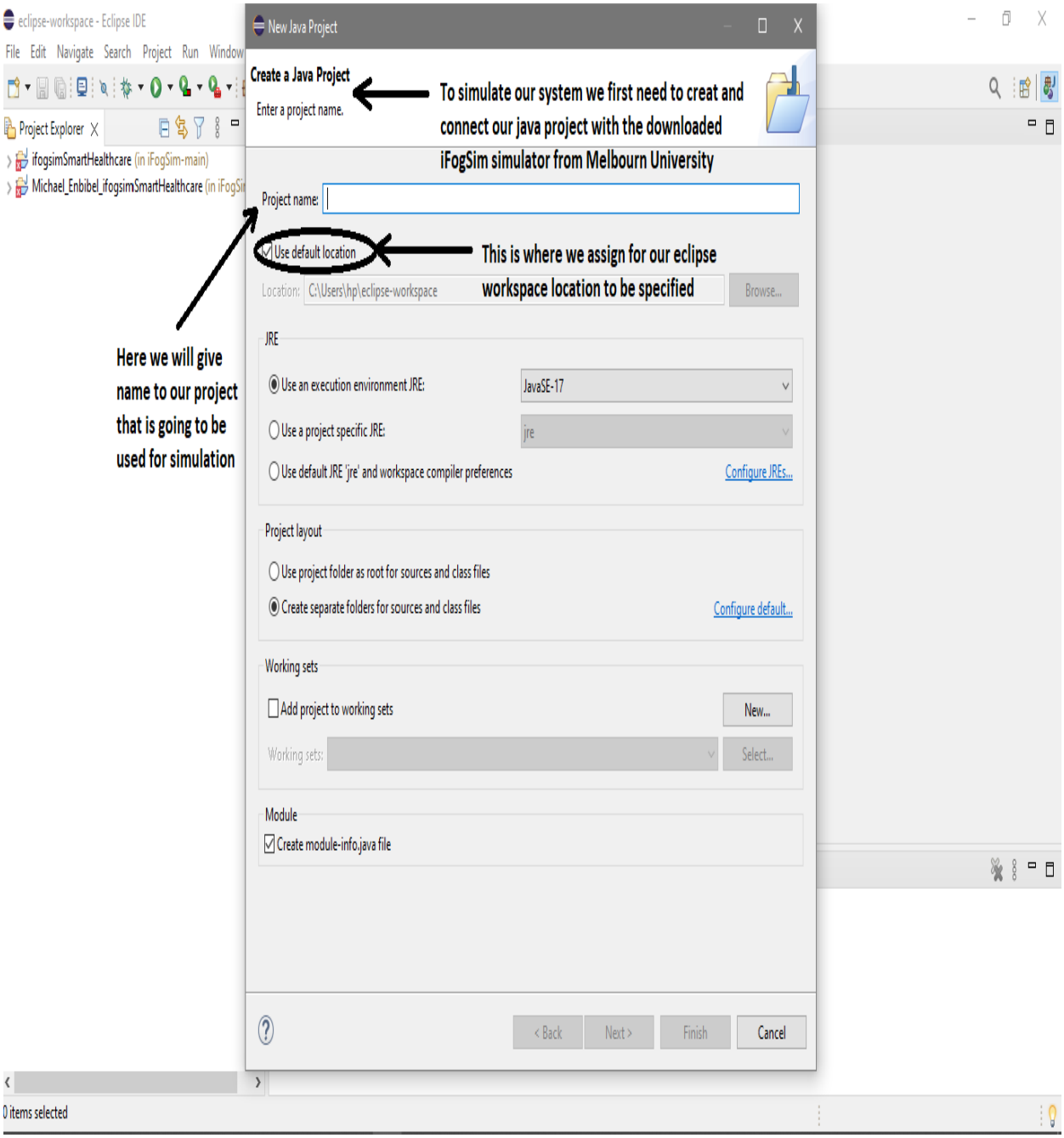

Figure 11: creating a Java project by connecting to the iFogSim files

Step 10:- At this moment while we are giving the location information we need to use and link our project with theiFogSim extracted file location.

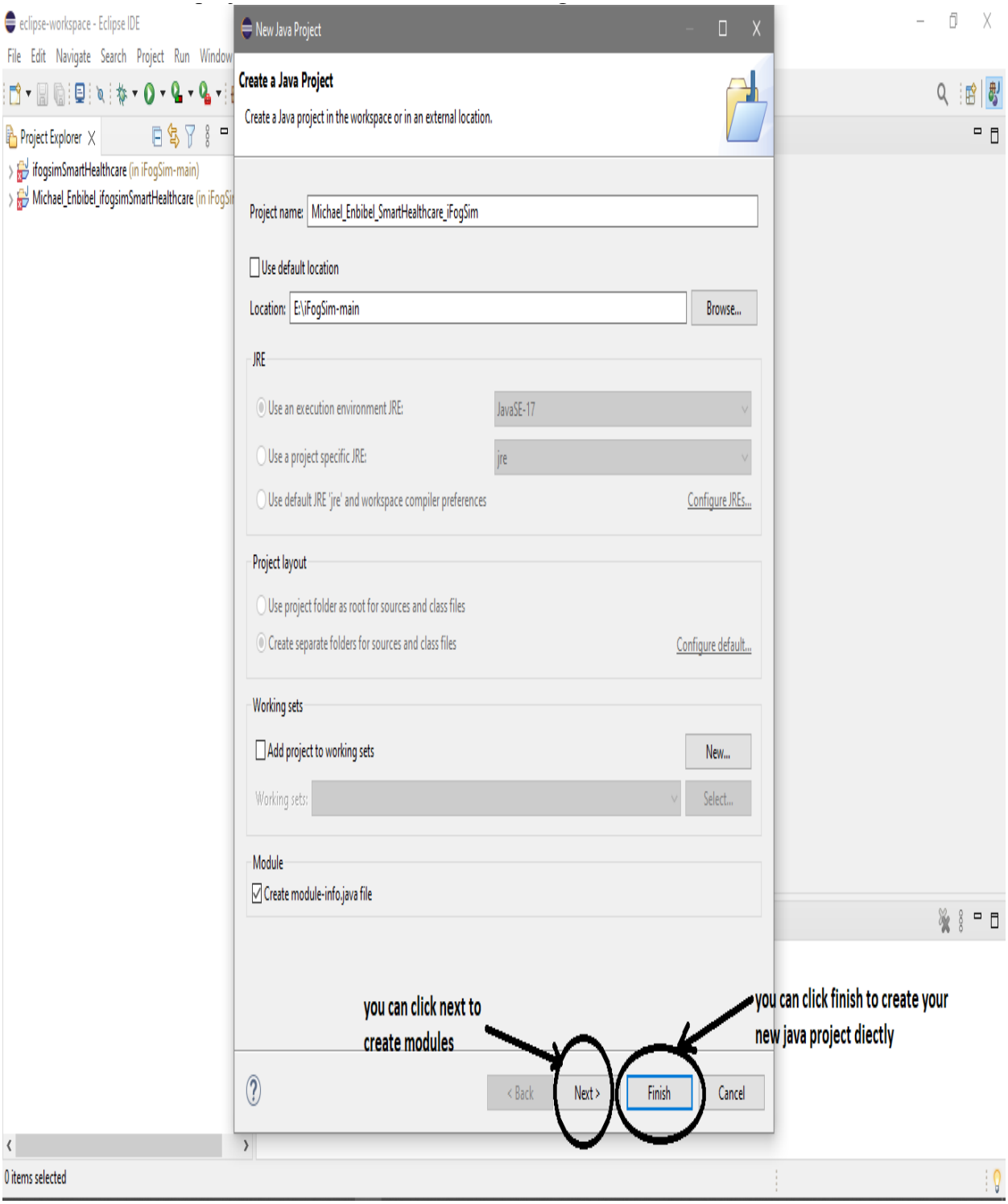

Figure 12: creating modules our java project

Step 11:- If we click the next command button it will redirect us to the java build settings where we can define our modules and our output folder for the source folders.

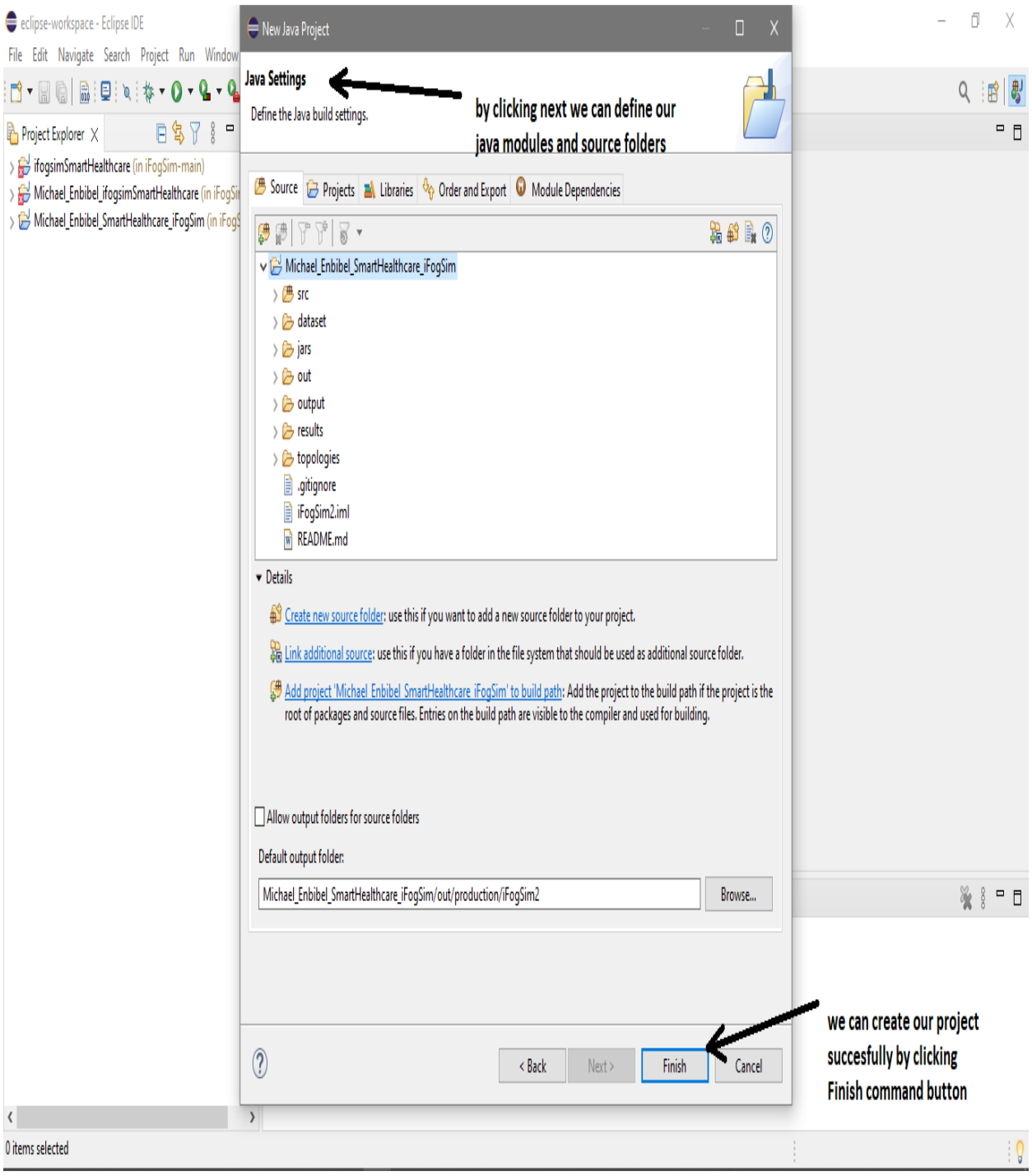

Figure 13: finishing and creating a simulation project with the iFogSim files

Step 12:- After we created our project on our java developer environment now we can manipulate and assign different values while we want to simulate depending on the environment we are testing our smart healthcare scenarios. For example in the following screen shoot I have tried to show how we can monitor a patient heartbeat by assigning heartbeat level for the sensor so that it will automatically alert the health professional when the patient has a lower or higher heartbeat on the EEG.

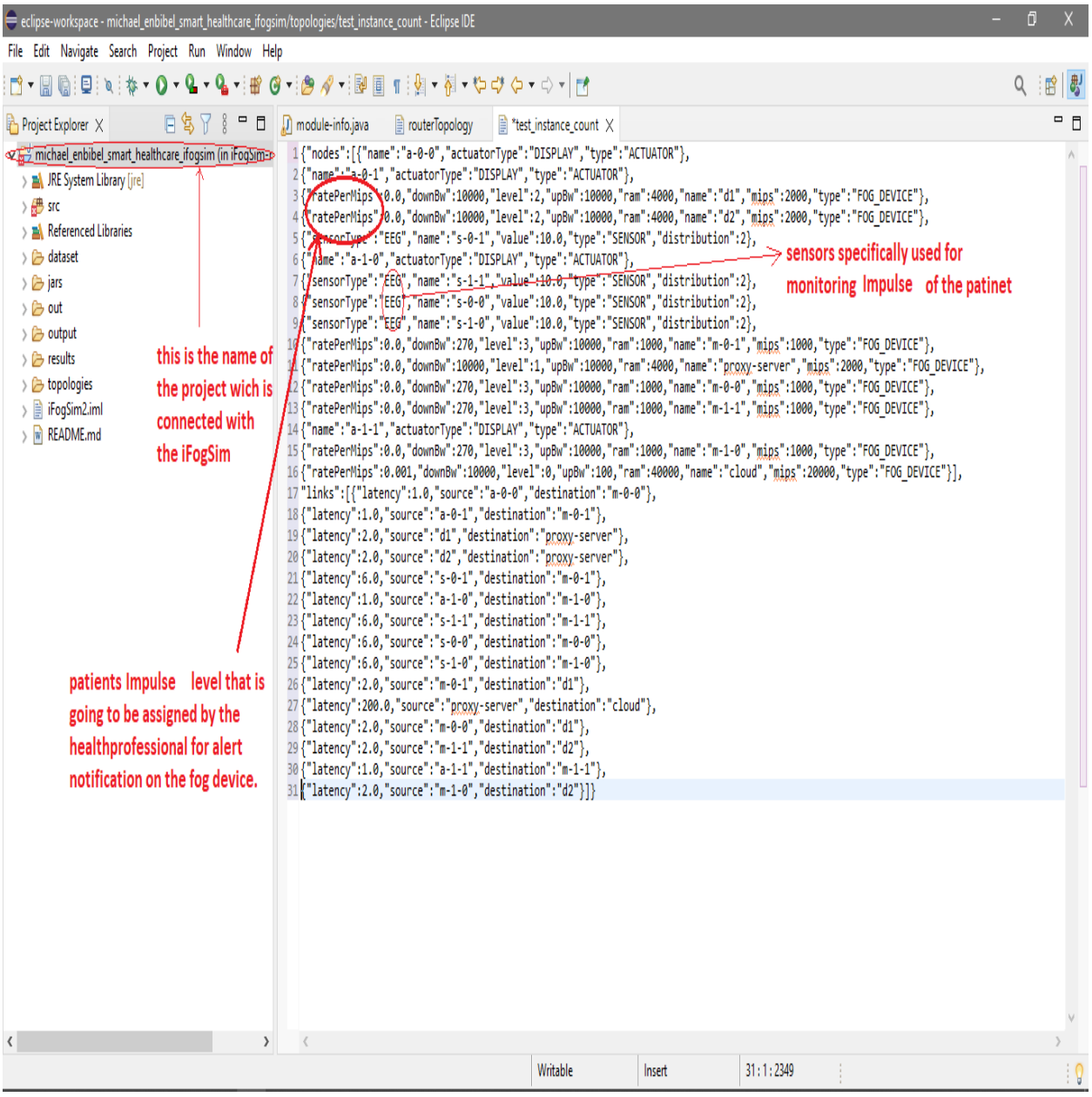

Figure 14: how we can assign the heartbeat level for monitoring the patient

Step 13:- Creating topology for both cloud and fog

Creating a cloud topology for comparison as and for knowing the system topological design while using a resource from the cloud and monitoring the pain of the patient.

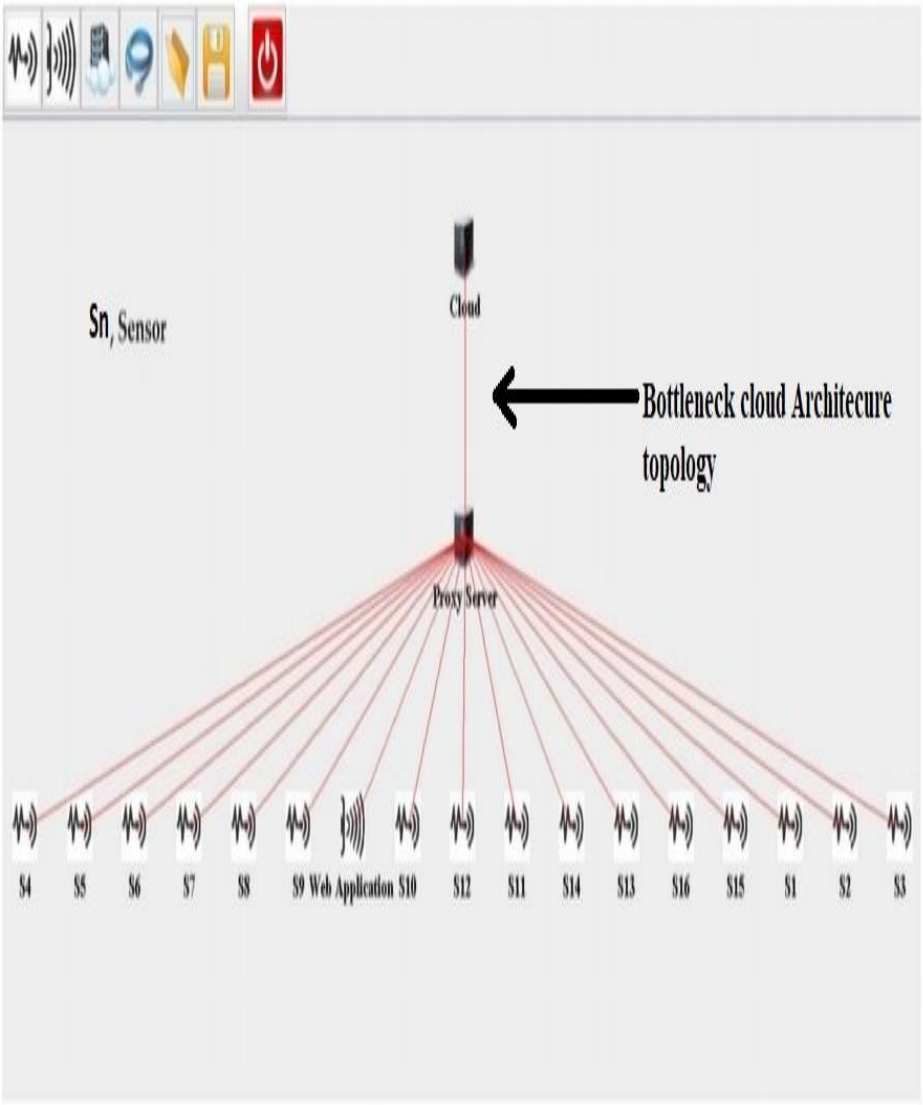

Figure 15: CloudSim topology for cloud based pain monitoring

Step 14:- iFogSim topology design for comparison and showing the distributed data flow, hence we used fog Devices for distribution for the proposed patient health pain monitoring.

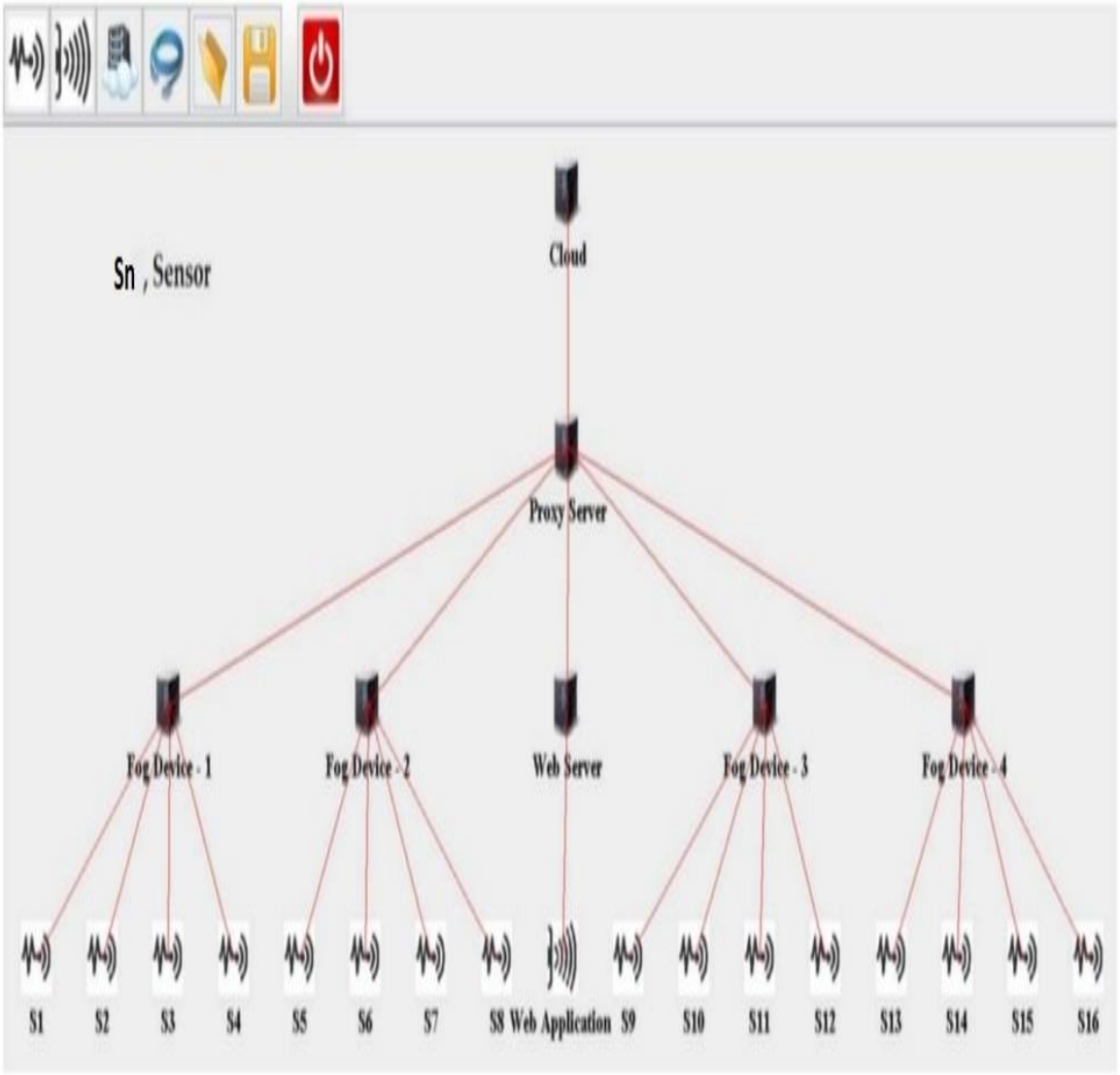

Figure 16: Fog topology for comparison with cloud topology with a distributed data flow approach

From the above topological design there is a big difference on the way the **edges** are connected to **proxy gateway** in order to share resource that has been provided by the **cloud service vendor**. For example as the number of patients and health professional increases the data that is going to be stored on the cloud also increase in creating Big Data. Typical a cloud system approach uses a **bottleneck** approach by just giving **one proxy gateway** for all sensors and devices that are integrated in the system [10]. So this show that whenever all the sensors and edges are trying to access a same resource there will be traffic as well conflictions in the system due to priority assignment. In this scenarios while retrieving and sharing same resources from the cloud will have a **dead lock.**

6.2. System Overview and Architectural Design in Real World Implementation

In this optimization process we used fogging in-order to solve different issues that were listed in the Article. We mainly focus on the latency issue in our research and we have designed and come up with new approach that can help the professionals who are interested in this particular area. We also recommend different ways and methodologies for the upcoming era as a future work.

6.3. Architectural Design

To build and test the optimized smart healthcare System we mainly need to have three **client modules.**

1. The patient client: an indoor positioning system, with Bluetooth beacons planted in the test site, will be used to localize a patient's Android phone.

2. The data Collector client: A raspberry Pi is going to be used

as a data collector client from different simulated sensors and send data to the server to be further processed.

3. The Smart healthcare client: An Android tablet is going to be used to receive notification of the patients alarm and collect position information of the patient for finding the patient.

Cloud services from Amazon Web Services (AWS)-EC2 instance, is going to be used as a cloud back-end server for the healthcare system. Further **three Raspberry Pi's** are going to be used as a fog infrastructure to be installed in the site.

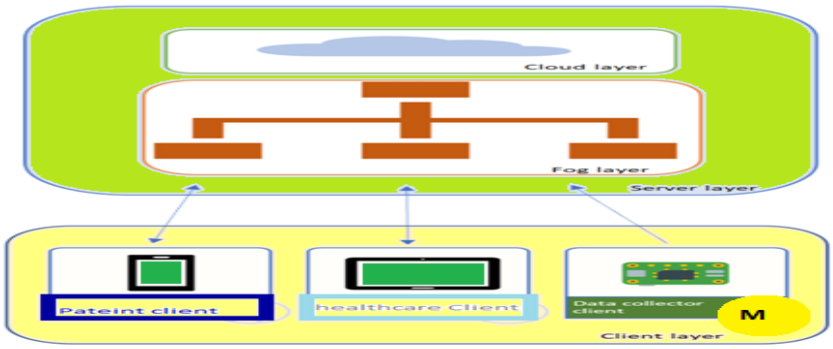

Figure 17: High level architectural design of the smart healthcare system

6.4. Delimitation

The Requirement elicitation process for the implementation of the smart healthcare system has produced a large number of requirements that **do not** fit in this article due to the size of its contents. Therefore, only the **special requirements** that are significant for the explanation of the design of the system are explained in this Article. This Article focuses only **one** of the issues that were listed previously that is **latency**. The Technical implementation needs an IT Expert and a network administrator while setting up and testing the real world implementation who is well experienced in cloud computing and fog computing while setting and configuring each clients on the system.

6.5. Latency

Latency in communication is **the time delay** that takes place during communication over a given network. There are mainly four types of delays that affect end-to-end delay in a given communication these are

- **Processing delay**
- **Queue delay**
- **Serialization delay**
- **Propagation delay**

Thus the overall end-to-end delay that is $\mathbf{D_{end\text{-}to\text{-}end}}$, will be the summation of the above four delays times the number of nodes the packet passes through.

Dend-to-end=N*(dproc+dqueue+dseri+dprop)

In the above equation $\mathbf{d_{proc}}$ is the nodal processing delay, $\mathbf{d_{queue}}$ is the queue delay, $\mathbf{d_{seri}}$ is the serialization delay, dprop is the propagation delay and N is the number of network segment a packet must go through along the **IoT ecosystem** [27].

In this Article I used **RTT (Round Trip Time)** is adopted from Network Coordinate System

(NCS) that establishes virtual positioning system for every node.

6.6. End-to-End Delay Estimation with Timestamps in RTT (Round Trip Time)

According to the Network Working Group (NWG) the RTT between a Source and a destination node is defined as follows

$\mathbf{\Delta T=T_1-T_2}$

Where T_1 is the time at which the first bit of the request packet sent by the source node approaches the physical layer and T_2 is the time at which the last bit of the response sequence, immediately sent back by the destination node, leaves the physical layer.

Simulation Parameters for the procedure in the RTT in the above given equation are:

1. The source should have a test packet with the given length containing the IP address of both the source and the destination.
2. The destination should be able to receive the test packets and answer as soon as possible. The source host should also be able to receive the response test packet.
3. The source host should be able to save the time reference value just before sending the test packet.
4. The final time stamp (T2) is considered upon the receipt of the packet, if the response packet arrives at source within a predetermined time interval.
5. RTT is calculated by subtracting T2 from T1

When considering the **RTT** measurement method that is sending test packets across the network and time stamping the test packets in each node the packet passes by, there is an important aspect to consider when it comes to time stamping: we have to make sure that the **CPU clock** runs at a **constant rate** across all socket in a node. We can do this by configuring it as the clock source for the Linux Kernel at boot time. In case of Android environment where the **patient client** and the **Smart healthcare client** run on, the library class timestamp can be used. This class adds the ability to hold the **SQL TIMESTAMP** fractional second value, by allowing the specification of fractional seconds to precision of **nanoseconds**.

6.7. Ping

A way to measure/approximate network latency is the ***ping*** command. A user can communicate with an underlying operating system with this standard command, ***Ping***, which is used to either detect whether the network is operational or/ and detect if the network has an actual connection to a far end destination across the network. This command uses the Internet Control Message Protocol (**ICMP**).

In summary, by using the Ping command one can identify that both ends of the network are connected and operational. Additionally by utilizing a ping command, one could find the total round trip time (**RTT**).

6.8. Unified Modeling Language (UML) for Sequence of operations

The sequence diagram **figure 18** below shows the normal sequence of events and the data flow between the three clients and the server which contains the smart healthcare system. Notice that the server layer runs the two different architectures that is the cloud and fog architectures. We mainly used an application container that is deployed-either directly to the cloud, or the fog cells. The sensor data collected by the data collector client is sent to the server continuously.

The data is then filtered through the Smart healthcare client module.

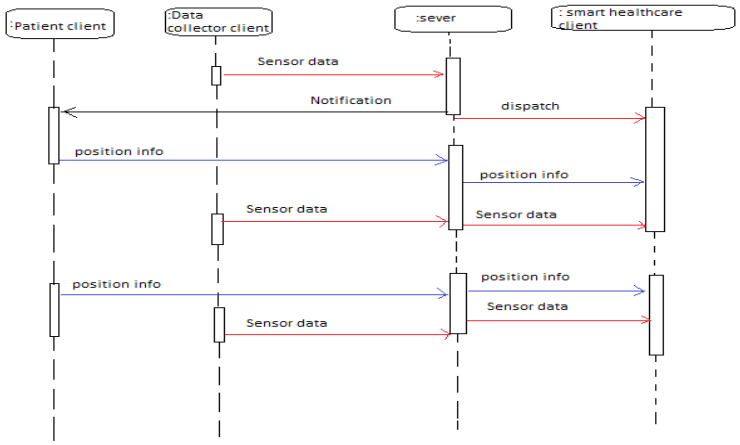

Figure 18: sequence diagram for emergency mode

6.9. Latency Measurement Method

As mentioned earlier we choose **RTT** for measuring latency in this Article. This was done by sending **JSON-file** with different packet sizes from the **patients' client** to the **smart healthcare client** through a **fog cell** and later on through **the cloud server** and having all the nodes print their timestamps as shown in figure 20. Then after collecting the timestamps data, Equation 4.1 was applied to send message, and the average of all the sent message was calculated by dividing the total sum with the number of message sent. We have done this for **100, 500** and **1000** bytes of packages.

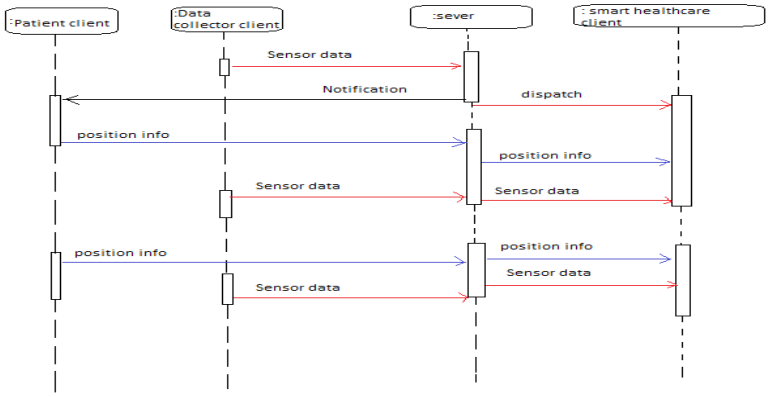

Figure 19: Sequence diagram for emergency mode

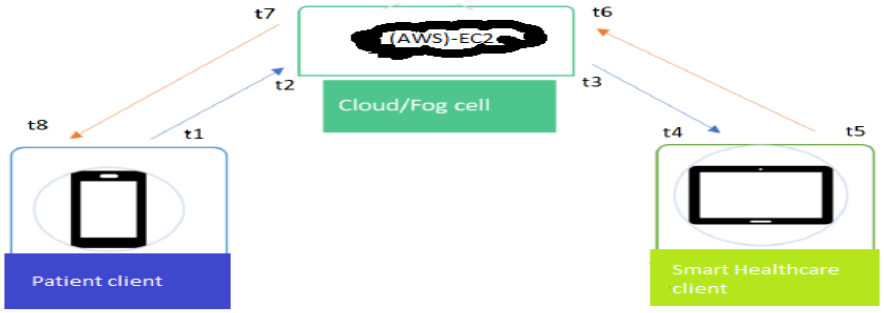

Figure 20: Latency estimation timestamp for RTT.

$$D_{t4\text{-}t1} = (t_2\text{-}t_1) + (t_3\text{-}t_2) + (t_4\text{-}t_3) \quad (1)$$

$$D_{t_4-t_1} = \frac{(t_2 - t_1) + (t_8 - t_7)}{2} + (t_3 - t_2) + \frac{(t_4 - t_3) + (t_6 - t_5)}{2} \quad (2)$$

7. Result

The test result for checking if the smart healthcare system works as intended, that the patient- client sends its location information to the smart healthcare client, was confirmed by the observation made on the test site. There are mainly two ways to check the correctness of the received information:

1. By connecting and live debugging the Android tablet and observing the coordinate message flow in as received message.
2. An observing temporal actor that received the message that the smart healthcare client received, by connecting a chrome-extension called smart Web Socket and connecting it to the cloud and fog cells as a smart healthcare client.

When it comes to latency test a significant difference with **263 ms** in latency was observed in test result shown in bar graph in figure 21. The timestamps were collected from log files, and a script that calculates the average time, according to the formula above, were executed. The average time it took for $D_{t4\text{-}t1}$ that is location information to arrive at the smart healthcare client was **277 ms** for the **cloud infrastructure**, and for **fog computing** the corresponding measurements was **14 ms**. This shows that using fog computing for smart healthcare system has reduced the latency with **263 ms** with respect to cloud computing. The average value obtained by those three packages with their standard devotion of **38.4 ms** for the **cloud** and **4.8 ms** for the **fog**. There was no significant difference in value observed between the different package sizes. Especially in the cloud architecture the 1000 bytes message arrived 5-20 ms faster than the 100 bytes. Generally this shows that the packet-size, between 100-1000 bytes, did not matter in latency.

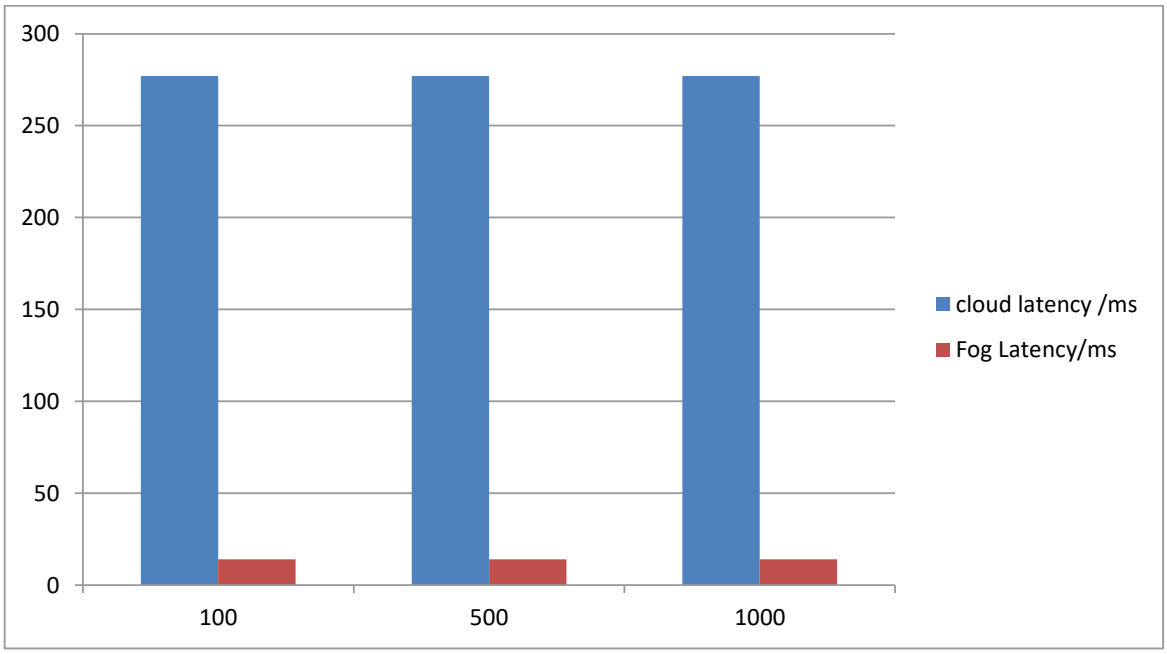

Figure 21: Cloud and fog latency measurement with timestamp

8. Conclusion

In this Article, we proposed Fog Computing to optimize Telemedicine framework for Smart healthcare Systems which assures reduced latency, high privacy, energy efficient, better bandwidth, high scalability, Less dependable, and precise context for sharp edged telemedicine applications with low cost. Our solution focuses on fogging with respect to cloud computing and we found a significant result in reducing latency with 263 ms and optimizing telemedicine framework for smart healthcare systems. We have clearly stated and showed that using a distributed data flow with fog computing can reduce the time delay from 277 ms to 14 ms by using fog computing. This solution can be further extended to be used for any smart healthcare services which may be used to govern other remote healthcare centers. In addition using fog computing for smart healthcare centers reduces the cost and time it takes during implementation and maintaining the system. Generally this research shows the different approaches like fogging to optimize the telemedicine framework for smart healthcare systems while designing and implementing. So this Smart healthcare system can be further upgraded and can be used in the future by the coming era in the IoT ecosystem when we built a Smart City.

References

1. Aziz, H. A., & Abochar, H. (2015). Telemedicine. Clinical Laboratory Science, 28(4), 256-259.
2. Harris, G. (2002). India: telemedicine's great new frontier. IEEE Spectrum, 39(4), 16-17.
3. Ahad, A., Tahir, M., & Yau, K. L. A. (2019). 5G-based smart healthcare network: architecture, taxonomy, challenges and future research directions. IEEE access, 7, 100747-100762.
4. Zhai, Y., Xu, X., Chen, B., Lu, H., Wang, Y., Li, S., ... & Zhao, J. (2021). 5G-network-enabled smart ambulance: architecture, application, and evaluation. IEEE Network, 35(1), 190-196.
5. Matlani, P., & Londhe, N. D. (2013, January). A cloud computing based telemedicine service. In 2013 IEEE Point-of-Care Healthcare Technologies (PHT) (pp. 326-330). IEEE.
6. Qiao, L., & Koutsakis, P. (2010). Adaptive bandwidth reservation and scheduling for efficient wireless telemedicine traffic transmission. IEEE Transactions on Vehicular Technology, 60(2), 632-643.
7. Galletta, A., Carnevale, L., Bramanti, A., & Fazio, M. (2018). An innovative methodology for big data visualization for telemedicine. IEEE Transactions on Industrial Informatics, 15(1), 490-497.
8. Hao, Z., Novak, E., Yi, S., & Li, Q. (2017). Challenges and software architecture for fog computing. IEEE Internet Computing, 21(2), 44-53.
9. Armfield, Nigel R, et al. "Studies in Health Technology and Informatics." Clinicians' perceptions of telemedicine for remote neonatal consultation, vol. 7 2019, p. 10, 10.3233/978-1- 60750-659-1-1. Accessed 6 June 2019.
10. Abugabah, A., Nizamuddin, N., & Alzubi, A. A. (2020). Decentralized telemedicine framework for a smart healthcare ecosystem. IEEE Access, 8, 166575-166588.
11. Qi, S., Lu, Y., Wei, W., & Chen, X. (2020). Efficient data access control with fine-grained data protection in cloud-assisted IIoT. IEEE Internet of Things Journal, 8(4), 2886-2899.
12. Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., & Ranjan, R. (2018). Fog computing:

Survey of trends, architectures, requirements, and research directions. IEEE access, 6, 47980-48009.
13. Sharma, N., & Bhatt, R. (2020, November). FoG Computing based IoT in Healthcare Application. In 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC) (pp. 442-446). IEEE.
14. Kumari, A., Tanwar, S., Tyagi, S., & Kumar, N. (2018). Fog computing for Healthcare 4.0 environment: Opportunities and challenges. Computers & Electrical Engineering, 72, 1-13.
15. Kraemer, F. A., Braten, A. E., Tamkittikhun, N., & Palma, D. (2017). Fog computing in healthcare–a review and discussion. IEEE Access, 5, 9206-9222.
16. Dubey, H., Monteiro, A., Constant, N., Abtahi, M., Borthakur, D., Mahler, L., ... & Mankodiya, K. (2017). Fog computing in medical internet-of-things: architecture, implementation, and applications. Handbook of Large-Scale Distributed Computing in Smart Healthcare, 281-321. Dubey, H., & Constant, N. P. (2016). Enhancing Telehealth Big Data Through Fog Computing. Fog Data, 2, 6.
17. " Verma, P., & Sood, S. K. (2018). Fog assisted-IoT enabled patient health monitoring in smart homes. IEEE Internet Things J 5 (3): 1789–1796.
18. Dastjerdi, A. V., & Buyya, R. (2016). Fog computing: Helping the Internet of Things realize its potential. Computer, 49(8), 112-116.
19. Craig, J., & Petterson, V. (2005). Introduction to the practice of telemedicine. Journal of telemedicine and telecare, 11(1), 3-9.
20. Xue, M., Yuan, C., Wu, H., Zhang, Y., & Liu, W. (2020). Machine learning security: Threats, countermeasures, and evaluations. IEEE Access, 8, 74720-74742.
21. Gupta, H., VahidDastjerdi, A., & Ghosh, S. K. (2017). RajkumarBuyya "iFogSim: A toolkit for modelling and simulation of resource management techniques in the Internet of Things. Edge and Fog computing environments.
22. Freed, J., Lowe, C., Flodgren, G. M., Binks, R., Doughty, K., & Kolsi, J. (2018). Telemedicine: is it really worth it? A perspective from evidence and experience.
23. George, A., Dhanasekaran, H., Chittiappa, J. P., Challagundla, L. A., Nikkam, S. S., & Abuzaghleh, O. (2018, May). Internet of Things in health care using fog computing. In 2018 IEEE Long Island Systems, Applications and Technology conference (LISAT) (pp. 1-6). IEEE.
24. Buldakova, T. I., & Sokolova, A. V. (2019, November). Network services for interaction of the telemedicine system users. In 2019 1st International Conference on Control Systems, Mathematical Modelling, Automation and Energy Efficiency (SUMMA) (pp. 387-391). IEEE.
25. Aazam, M., & Fernando, X. (2019, December). oHealth: opportunistic healthcare in public transit through fog and edge computing. In 2019 IEEE International Conference on Smart Cloud (SmartCloud) (pp. 59-64). IEEE.
26. Yu, H., & Zhou, Z. (2021). Optimization of IoT-based artificial intelligence assisted telemedicine health analysis system. IEEE access, 9, 85034-85048.
27. Schiza, E. C., Kyprianou, T. C., Petkov, N., & Schizas, C. N. (2018). Proposal for an ehealth based ecosystem serving national healthcare. IEEE journal of biomedical and health informatics, 23(3), 1346-1357.
28. Lian, W., Xue, T., Lu, Y., Wang, M., & Deng, W. (2019). Research on hierarchical data fusion of intelligent medical monitoring. IEEE Access, 8, 38355-38367.
29. Martinez, N. P., Martinez, M., Kuebler, S. M., Touma, J. E., Rumpf, R. C., & Lentz, J. K. (2018, August). Spatially-variant photonic crystals and possible applications. In 2018 IEEE Research and Applications of Photonics In Defense Conference (RAPID) (pp. 1-4). IEEE.
30. Touil, M., Bahatti, L., & El Magri, A. (2020, December). Telemedicine application to reduce the spread of Covid-19. In 2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS) (pp. 1-4). IEEE.
31. Jin, Z., & Chen, Y. (2015). Telemedicine in the cloud era: Prospects and challenges. IEEE Pervasive Computing, 14(1), 54-61.
32. World Health Organization. (2010). Telemedicine: opportunities and developments in member states. Report on the second global survey on eHealth. World Health Organization.
33. Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2008). The cost of a cloud: research problems in data center networks. ACM SIGCOMM computer communication review, 39(1), 68-73.

Scripts and commands

1. Dockers file for application container on cloud architecture

FROM ubuntu : 1 2 . 0 4 # Install dependencies RUN apt_get update _y
RUN apt_get i n s t a l l _y apache2 # Configure apache
RUN a2enmod r ewr i t e
RUN chown _R www_data :www_data /var/www ENV APACHE_ RUN_USER www_data
ENV APACHE_RUN_GROUP www_data ENV APACHE_ LOG_DIR /var/log/apache2 # Set the working directory to /app WORKDIR /app
Copy the current directory content s into the container at /app ADD . /app
I n s t a l l any needed packages specified in requirements. t x t RUN pip install trusted host pypi . python . org _r requirements . t x t # Make port 80 a v a i l a b l e to the world outside this container EXPOSE 80
Define environment variable ENV NAME World

2. Application container builds and run on cloud architecture.

\$docker build _t application container.
\$docker run _p 8000:80 application container

3. Dockers file for application container on fog architecture.

Use an official Python runtime as a parent image FROM python:2.7_slim
Set the working directory to /app WORKDIR /app
Copy the current directory content s into the container at /app ADD . /app
I n s t a l l any needed packages specified in requirements. t x t RUN pip install trusted host pypi . python . org _r requirements . t x t # Make por t 80 a v a i l a b l e to the world outside this container EXPOSE 80
Define environment variable ENV NAME World
Run server.py when the container launches CMD [" python " , " server.py "]

4. Dockerfile for cell connector container on fog architecture.

Use an official Python runtime as a parent image FROM python: 2.7 slim
Set the working directory to /app WORKDIR /app
Copy the current directory content s into the container at /app ADD./app
Install any needed packages specified in requirements. t x t RUN pip install trusted host pypi.python.org r requirements. txt # Make port 80 available to the world outside this container EXPOSE 80
Define environment variable ENV NAME World
Run connector.py when the container launches CMD [" python " , " connector . py "]

5. Dockerfile for controller container on fog architecure.

FROM ubuntu : 1 2 . 0 4 # Install dependencies RUN apt_get update _y
RUN apt_get install _y apache2 # Configure apache
RUN a2enmod rewrite
RUN chown _R www_data :www_data /var/www ENV APACHE_ RUN_USER www_data
ENV APACHE_RUN_GROUP www_data ENV APACHE_ LOG_DIR /var/log/apache2 # Set the working directory to /app WORKDIR /app
Copy the current directory content s into the container at /app ADD . /app
I n s t a l l any needed packages specified in requirements . t x t RUN pip i n s t a l l trusted host pypi.python.org _r requirements . t x t # Make port 80 available to the world outside this container EXPOSE 80
Define environment variable ENV NAME World
Run controller.py when the container launches CMD [" python " , " controller.py "]

Annexure

Abbreviations with their full format WHO World Health Organization

NASA	National Aeronautics and Space Administration
IoT	Internet of Things
FC	Fog Computing
RTT	Round Trip Time
UML	Unified Modelling Language
ICMP	Internet Control Message Protocol
AWS	Amazon Web Services
CPU	Central Processing Unit
SQL	Standard Query Language