IoT-Based Opioid Overdose Monitoring System
CPSC 503.08 Final Report

Jian Liao
Department of Computer Science
University of Calgary
Calgary AB, Canada
jian.liao1@ucalgary.ca

Advisor: Dr. Mea Wang
Department of Computer Science
University of Calgary
Calgary AB, Canada
meawang@ucalgary.ca

***Abstract*—The opioid epidemic in Canada has claimed the lives of over 8000 people in the last two years, and everyday 11 people die from an opioid overdose. This problem is clearly not improving, and it is urgently addressed that many individuals are trying to find a solution to opioid crisis. Fortunately, the Internet of Things (IoT) has provided a promising opportunity to tackle the opioid epidemic challenge. Therefore, with the use of IoT, this paper will try to explore a solution and implement an opioid overdose monitoring system with the goals of achieving high sclability and real-time services.**

***Index Terms*—Opioid overdose, health monitoring, Internet of Things (IoT), IoT database, cloud computing, data visualization**

I. Introduction

Over the past two decades, the United States population has experienced an extraordinary increase in the rates of death from opioid overdose. A steep rise in fatal overdoses caused by pharmaceutical opioids observed over the last 15 years [1] has been overtaken by rapidly increasing rates of heroin and illicit fentanyl overdose death as the supply of prescription opioids has been reduced [2]. In 2017, there were estimated to be 68,400 drug overdose deaths in the US [3], more than the annual number of deaths from AIDS at the height of that epidemic. These deaths have contributed to an overall decrease in life expectancy among middle-aged white people in the US [4]. Another report in 2018 shows that every day, 128 people in the United States die after overdosing on opioids [5]. The misuse of and addiction to opioids, including prescription pain relievers, heroin, and synthetic opioids such as fentanyl, which is a serious national crisis that affects public health as well as social and economic welfare. Meanwhile in Canada, the opioid epidemic has claimed the lives of over 8000 people in the past two years, and has resulted in 11 opioid overdose related deaths every day [6]. Of those deaths, almost 70% involved a prescription or illicit opioid.

The government of Canada is committed to fighting the opioid overdose epidemic and supporting states and communities as they continue work to identify outbreaks, collect data, respond to overdoses, provide care to those in their communities, and set up supervised consumption sites all over Canada for surveillance and prevention efforts [7]. These efforts include timelier tracking of nonfatal and fatal drug overdoses, improving toxicology to better track polysubstance-involved deaths, enhancing linkage to care for people with opioid use disorder and risk for opioid overdose, improving prescription drug monitoring programs, implementing health systems interventions, partnering with public safety, and implementing other innovative surveillance and prevention activities [8]. However, these federal actions responding to the opioid crisis still encounter a major deficiency which is the increasing requirement for continuous monitoring for the PWUD (People Who Use Drugs) and rapid medical emergency response system in case of opioid overdose occurs [9].

To assist the government of Canada in the fight against opioid crisis, it is necessary to propose a more efficient, and practical approach to tackling the problem. Originally, I teamed up in the Innovation4Health program and we proposed the idea of the opioid overdose monitoring system. And based on the previous work, this research project proposes a more satisfactory solution which is to develop an IoT-based opioid overdose monitoring system that is capable of automatically collecting and transmitting vital signs data (heart rate and respiration rate) using existing sensor solutions (ECG), storing and analyzing the data on the IoT cloud data platform to detect opioid overdose, and alerting a medical emergency so that the patients can get the rapid medical care they need. This proposed IoT-based opioid overdose monitoring system consists of 5 components in general.

First, the monitoring module (IoT device) will be composed of several physiological sensors and communication modules such as Electrocardiography sensor, accelerometer, and Bluetooth Low Energy 4.0 module. Second, a cross-platform technique (Dart & Flutter) will be used to develop the end-user mobile application layer. Third, IoT Middleware layer is implemented using MQTT protocol (Broker) as a gateway, which provides a medium for the communication between the edge device and the back-end server with reliable data transfer (QoS). Fourth, the Web-Tier application will be hosted on the server with different services implemented using Node.js [10]. These services include synchronizing and managing the gateway systems, transmitting data to and from the IoT data platform, handling all the requests and send it back to the requester. Fifth, the IoT data platform consists of two parts, cloud data engine (scientific computation module & cloud

database) and microservices. The cloud data engine contains a scientific computation module and cloud database which can store the sensor data and perform specific computation tasks on the sensor data.

The modules other than the IoT data platform in the system have been developed outside of this course and the goal of this research project is to develop the cloud-based IoT data platform. The overview of the IoT data platform in terms of technologies includes communication protocols design (MQTT & RESTful API), cloud database design, highly scalable broker server design (Load Balancer) and real-time data processing & visualization (Data Driven Document). And the expected outcome in terms of functionality and performance of this research project is to provide a full-featured ecosystem as well as the cloud IoT data platform, seamless sync from edge to cloud, guaranteed data availability, high scalability and real-time services. And for the rest of this paper, we will review related work in Sec. II, and propose the design of the IoT data platform in Sec. III. The evaluation of the platform is presented in Sec. IV, followed by conclusion in Sec. V.

II. Related Work

The Internet of Things is an emerging technology which is generally recognized as representing a revolution in Information and Communication Technology (ICT). It is expected to have a wide range of applications in various industrial sectors, including healthcare [11], [12] and energy industry [13]. According to research on IoT-based healthcare asset management system (IoT-HAMS), a general system architecture of IoT-HAMS is proposed [14]. The system components include RFID and Wireless Sensor Network (WSN) devices (IoT devices), IoT middle-ware, graphic user interface (GUI) for rules input and modification, cloud database, and core management engine. This state-of-the-art provides a very specific solution stack for IoT-HAMS which shares several similarities to the solution this paper is proposing such as IoT devices, IoT middle-ware layer, cloud database and GUI applications. However, what differs from IoT-HAMS is that our solution guarantees data reliability as well as instantaneity and focuses more on establishing a highly sclable system that is capable of performing specific data processing tasks on the real-time input data while IoT-HAMS focuses more on devices management.

Amazon (AWS), Microsoft (Azure) and Google also provide the state-of-the-art IoT platfrom solution for industrial, consumer and commercial use. The AWS IoT is the leader in IoT in terms of the number of services provided. AWS IoT solutions are available for both edge software and cloud services. Edge software allows developers to connect devices, collect data, and perform data analysis. Whereas, the AWS Cloud Services enable them to securely connect a group of devices, manage the devices, keep them secure, and detect, and respond to IoT events across IoT apps and sensors. The Azure IoT is a collection of cloud services that connect, monitor, and control billions of IoT assets which is managed by Microsoft. It is the second-largest IoT platform in terms of the market share. The IoT solution is made up of one or more IoT devices and one or more back-end services running in the cloud which communicate with each other. There are two paths for building the solution which are Platform as a Service (PaaS) and Software as a Service (SaaS). The Google IoT is one of the top IoT service providers around the world, making it easier for developers to build connected devices. It provides Cloud IoT Core as its flagship IoT solution for creating secure and innovative solutions. In details, the comparison of these solutions is as shown in Table I. All of these state-of-the-art IoT platform solutions share advanced and well-established techniques in terms of cloud services, data storage, cloud computation and key tools. What differs from our solution is that while these industrial solutions focus more on the breadth of provided IoT services for multiple scenarios, our solution explore the depth of the IoT health monitoring solution with high scalability and real-time services. However, these solutions have inspired and set the goal standard of the implementation and evaluation in this paper.

TABLE I
IoT Platform Solutions Comparison

IoT Solution	**IoT Platform Solution Provider**		
	AWS	**Azure**	**Google**
Services	AWS Web Services AWS Cloud Services	Azure PaaS Azure SaaS	Google Cloud (IoT Core)
Protocols	MQTT	HTTP MQTT AMQP	HTTP MQTT
Storage	Elastic Block Storage	Blob Storage	Cloud Storage
Database	Aurora ElastiCache	SQL CosmosDB	Cloud SQL Cloud Bigtable
Computation	AWS Beanstalk VMware Cloud Serverless Application	FaaS Service Fabric Azure Batch	App Engine Docker Compute Engine
Key Tools	IoT Core	IoT Hub	Cloud IoT Core

III. Design

This research project mainly focuses on the implementation of IoT Data Platform. The goal for the implementation is to develop a real-time and highly scalable system, to meet the requirements of continuous monitoring of opioid overdose. The proposed architecture of the IoT Data Platform is shown below, which is composed of two components, the Cloud Data Engine and the Microservices.

The Cloud Data Engine is an integrated server application that runs continuously ingesting as well as processing the incoming data flow. The Cloud Data Engine is connected with the Web-Tier synchronous gateway through MQTT protocol, which can subscribe and publish to the data flow. Inside the Cloud Data Engine, two components were provided.

- *Scientific Computation Module*
  The scientific computation module handles and processes the sensor data, which is the ECG data from the edge devices. The scientific computation module is designed as a Plug-in bundle that runs heterogeneous computation tasks on the incoming data flow, produces the expected data outcome, and pushes the results to the corresponding

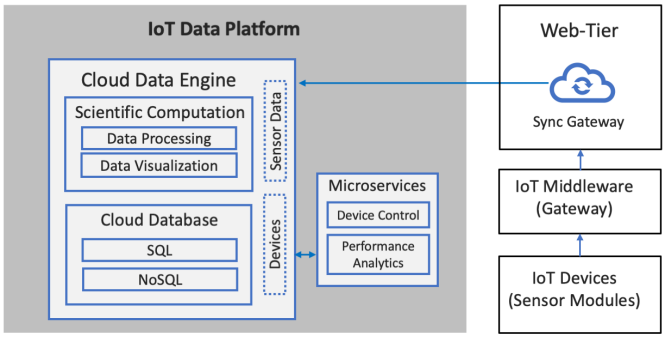

Fig. 1. Proposed Architecture of IoT Data Platform

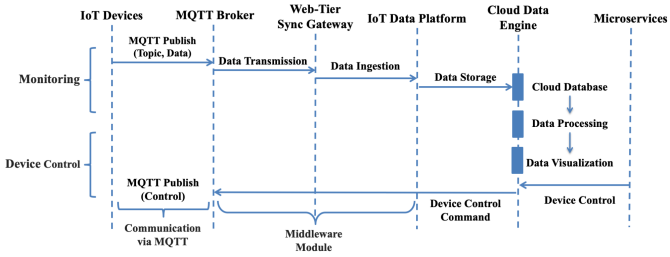

Fig. 2. Data Flow Sequence Diagram

destination i.e. feeding the data to the real-time data visualization.

- *Cloud Database*
  The cloud databases are the database systems that run on a storage server that stores the incoming data flow simultaneously as the data being processed. The PostgreSQL (relational database) is used for storing the system data i.e. the system configuration data for the sensors. And the MongoDB (NoSQL) is used for storing the sensor data for the sake of improving the network performance.

The Microservices component is implemented on the back-end server between the cloud data engine and web-tier application (Sync Gateway), which includes two parts namely, the Device Control component and Performance Analytics component. The Microservices are called on-demand, which are provided with API endpoints and can be accessed using standard HTTP protocol. The Device Control component offers an approach to manage and control the connected devices, such as terminating the devices and fetching the device status [16]. And the Performance Analytics component can keep track of the network traffic, delay time and resource utilization of the system [17].

A. Data Flow

To fullfill the system requirement of real-time data transmission, effective transmission of data flow is proposed in Fig. 2. The data flow is implemented with two-way communication between the edge and the cloud. The IoT devices can publish the sensor data on a specific topic through MQTT protocol [18]. Then the MQTT broker will forward and transmit the data to the IoT Data Platform through the Sync Gateway. Then the IoT Platform will store the sensor data in the Cloud Database (NoSQL) and transmit the sensor data to the Scientific Computation Module simultaneously. Furthermore, the sensor data will be sent to the Data Processing component and after data process task is finished, the processed data will be fed into the Data Visualization component and visualize the data using Data Driven Document (D3). Similarly, the IoT devices can also subscribe to a specific topic which allows device control from the cloud (Microservices) to the edge. Each stage of the data flow is transmitted seamlessly from the edge to the cloud through MQTT protocol in real-time [19].

B. Communication Protocols

To achieve the goals of real-time data transmission as well as high concurrency, this paper is utilizing MQTT protocols that can meet the following expectations namely, efficient information distribution, high scalability, reliability, extremely lightweight overhead and reduced network bandwidth consumption [22].

First, MQTT is a publish/subscribe protocol that allows network edge devices to publish to a broker. Clients (IoT Devices) can connect to this broker, which then mediates communication from the edge to the cloud. Each device can subscribe, or register, to particular topics. When another client publishes a message on a subscribed topic, the broker forwards the message to any client that has subscribed. This provides an efficient information distribution for remote sensing and control. Second, the publish/subscribe model is ultra-scalable in an bandwidth-efficient way [20]. According to HiveMQ [21], a HiveMQ broker cluster can support more than 10 million concurrent connections as a distributed system, which scales in a linear fashion and can scale from two cluster nodes up to hundreds. Third, MQTT is bidirectional, and maintains stateful session awareness. If an edge device loses connectivity, all subscribed clients will be notified with the "Last Will and Testament" feature of the MQTT server so that any authorized client in the system can publish a new value back to the edge device, maintaining bidirectional connectivity. In addition, the feature Quality of Service (QoS) options can be used to guarantee data delivery and reliability. Fourth, the extremely lightweight overhead (2-byte header) of MQTT packet makes it possible to significantly increase the amount of data being monitored or controlled.

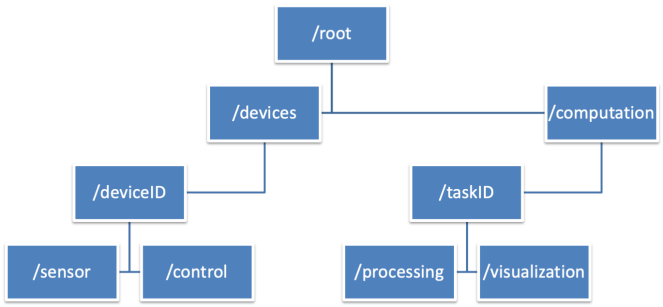

Fig. 3. MQTT Topic Hierarchy Design

1) MQTT Topic Design: This paper is proposing a MQTT topic design that is implemented in a hierarchy fasion as shown in Fig. 3. The advantage of using the hierarchy design is that it can reduce the system coupling and essentially enhance the system scalability [23]. The topic start with the root topic '/root' followed by two sub-topics namely, '/devices' and '/computation'. Under the '/devices' topic, the '/deviceID' topic is a universally unique identifier (UUID) which is used to distinguish the connected devices in the system. The '/deviceID' topic has two sub-topics specifically, '/sensor' and '/control', which allows the IoT devices publish the sensor data and subscribe to the device control command respectively. Comparably, under the '/computation' topic, the '/taskID' is also a UUID which is used to differentiate the computation tasks in the Scientific Computation module. The process of the data ingestion, processing and visulization is shown in Fig. 4. The Data Processing component subscribes to the '/root/devices/deviceID/sensor' topic and creates a computation task processing the incoming data in real-time. Then the Data Visualization component subscribes to the '/root/computation/taskID/processing' to obtain the processed data as well as forward (Publish) the data to an external front-end component (D3) for real-time data visualization.

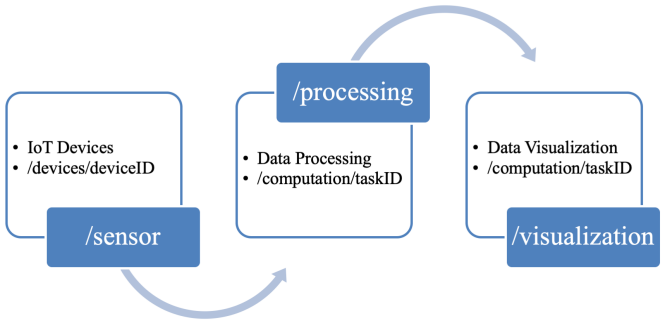

Fig. 4. Scientific Computation Process

2) RESTful API Design: The RESTful API is the endpoints for stateless operations on the Microservices, such as queuing the MQTT server information, controlling the devices and retrieving the performance analytics through standard HTTP protocols.

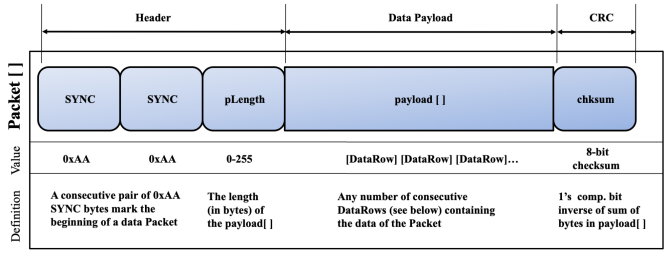

Fig. 5. Communication Packet Structure

3) Packet Structure: The packet structure of the sensor data being sent from the edge to the cloud is shown in Fig. 5. The packets are sent as an asynchronous serial stream of bytes and each packet begins with its Header, followed by its Data Payload, and ends with the Payload's Checksum Byte. The [PAYLOAD...] section is allowed to be up to 169 bytes long, while each of [SYNC], [PLENGTH], and [CHKSUM] are a single byte each. This means that a complete, valid packet is a minimum of 4 bytes long (possible if the Data Payload is zero bytes long, i.e. empty) and a maximum of 173 bytes long (possible if the Data Payload is the maximum 169 bytes long).

- *Packet Header*
  The Header of a packet consists of 3 bytes: two synchronization [SYNC] bytes (0xAA 0xAA), followed by a [PLENGTH] (Payload length) byte. The two [SYNC] bytes are used to signal the beginning of a new arriving packet and are bytes with the value 0xAA (decimal 170). And the [PLENGTH] byte indicates the length, in bytes, of the packet's Data Payload [PAYLOAD...] section, and may be any value from 0 up to 169.
- *Data Payload*
  The Data Payload of a packet is simply a series of bytes. The number of Data Payload bytes in the packet is given by the [PLENGTH] byte from the Packet Header. A DataRow consists of bytes in the following format as shown in Fig. 6:
  - *EXCODE*: The DataRow may begin with zero or more [EXCODE] (Extended Code) bytes, which are bytes with the value 0x55. The number of [EXCODE] bytes indicates the Extended Code Level. The Extended Code Level, in turn, is used in conjuction with the [CODE] byte to determine what type of Data Value this DataRow contains.
  - *CODE*: The [CODE] byte, in conjunction with the Extended Code Level, indicates the type of Data Value encoded in the DataRow.
  - *vLength*: A [VLENGTH] ("Value Length") byte immediately follows the [CODE] byte, and this is the number of bytes in [VALUE...].
  - *value[ ]*: The arrays of values, i.e. the sensor data.
- *Payload Checksum* The [CHKSUM] Byte must be used to verify the integrity of the packet's Data Payload. The Payload's Checksum is defined as:
  1. summing all the bytes of the Packet's Data Payload
  2. taking the lowest 8 bits of the sum
  3. performing the bit inverse (one's compliment inverse) on those lowest 8 bits

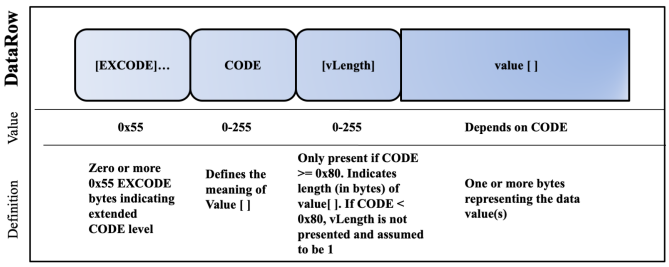

Fig. 6. Communication Packet Payload

C. Scalable System Design

To accomplish the system requirement of high concurrency, this paper chooses to use MQTT protocol to enhance the scalability. However, when using an MQTT protocol, the workload of an MQTT-based system can grow exponentially. The Fig. 7 below explains this concept graphically. For each device (n devices in total), it can choose to talk to the other n devices, which results in the maximum message workload of $n * n = n^2$.

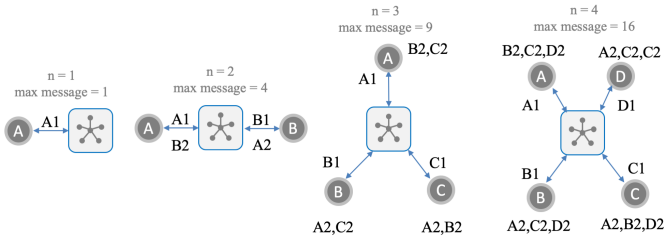

Fig. 7. Message Workload Grows Exponentially as MQTT Clients Increases

1) MQTT Broker Cluster: A scalable MQTT broker cluster is introduced to reduce the workload of the MQTT-based system as well as enhance the scalability of the system. The structure of a scalable MQTT broker cluster is shown in Fig. 8. A MQTT broker cluster is a distributed system that represents one logical MQTT broker. It consists of many different MQTT broker nodes that are typically installed on different physical machines and are connected over a network. From a MQTT client's perspective, a cluster of brokers behaves like a single MQTT broker. The MQTT Broker Cluster is used in this opioid monitoring system to guarantee the high scalability. Specifically, the IoT devices connect with a broker cluster which is subscribed by the IoT Data Platform. Inside the IoT Data Platform, the Cloud Database module and Scientific Computation module are subscribed by another broker cluster respectively, which enormously enhance the scalability of the proposedb system.

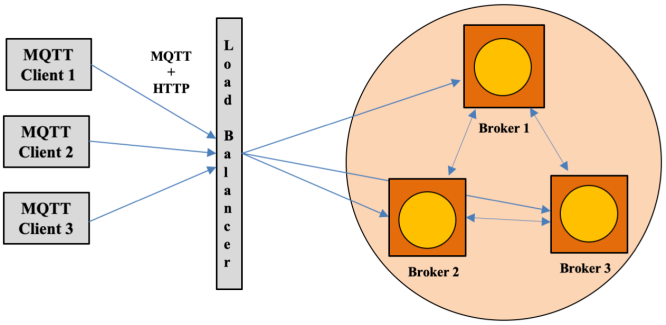

Fig. 8. The Structure of MQTT Broker Cluster

2) Load Balancer: Load balancers are often used together with MQTT broker clusters to have a single point of entry for all MQTT communication [24]. In the system implementation, load balancers are used in the entry point of each broker cluster. Any HTTP/TCP load balancer can be used together with any load balancing algorithm below.

- *Round Robin (RR)*
  A batch of servers are programmed to handle load in a rotating sequential manner. The algorithm assumes that each device is able to process the same number of requests and isn't able to account for active connections.
- *Weighted Round Robin (WRR)*
  Servers are rated based on the relative amount of requests each is able to process. Those having higher capacities are sent more requests.
- *Least Connections (LC)*
  Requests are sent to the server having the fewest number of active connections, assuming all connections generate an equal amount of server load.
- *Weighted Least Connections (WLC)*
  Servers are rated based on their processing capabilities. Load is distributed according to both the relative capacity of the servers and the number of active connections on each one.

This paper proposes a dynamic load balancing algorithm which can switch to different algorithms according to different scenarios. For the initialization stage, the load balancing algorithm is simply Round Robin due to low number of connections. Then if the number of the current connections surpasses the threshold T, then it will switch to Weighted Round Robin (Least Connections) algorithm.

Algorithm 1: Dynamic Load Balancing Algorithm

Input: A threshold T of the number of connections
Output: The broker entry point

```
PriorityQueue ← Q;
Bool ← INIT;
Array ← Brokers;
while New connections do
    if INIT == TRUE then
        Run Round Robin algorithm;
        if Number of connections > T then
            INIT ← FALSE;
        end
    end
    else
        if Q.Empty == TRUE then
            Brokers ← HTTPGet();
        end
        foreach broker ∈ Brokers and broker ∉ Q
         do
            Q.push(broker);
        end
        return Q.pop();
    end
end
```

The Algorithm. 1 will do HTTP Request (Get) through the RESTful API endpoints to obtain the information of the

brokers and uses Priority Queue to keep track of the number of the connections of each broker. If the Priority Queue is using Self-Balancing Binary Search Tree, the time complexity to construct a binary tree is $O(nlogn)$, and the time complexity of adding and deleting is $O(logn)$. Therefore, the average time complexity if $O(logn)$. The advantage of using such dynamic load balancing algorithm is that it can increase the resource utilization while reducing the time complexity, and it can prevent the broker from overflowing by an overwhelming amount of incoming connections spike.

D. EDR Algorithm

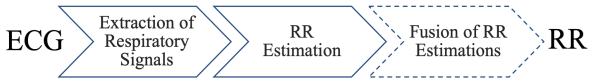

Fig. 9. The Three Stages of a Estimated RR Algorithm

Estimiated Respiratory Rate (RR) Algorithm can be divided into three stages, as illustrated in Fig. 9. Two of the stages namely, 'Extraction of Respiratory Signals' and 'RR Estimation', are compulsory whereas 'Fusion of RR Estimations' is optional [28]. The Data Processing component in the Scientific Computation module is performing the compuation tasks on the input ECG sensor data. The first stage is to extract the QRS wave from the ECG waveform by applying M-order Low-Pass filter (1) as well as M-order High-Pass filter (2) to remove the noise, where a_n, b_n are the coefficient of the transfer function. And then it will execute the Pan-Thomkins Algorithm [25] to calculate the heart rate (HR) and estimated respiratory rate (RR) from the input ECG waveform.

$$H_{LP}(z) = \Sigma_{n=0}^{M} a_n z^{-n} \quad (1)$$

$$H_{HP}(z) = \Sigma_{n=0}^{M} b_n z^{-n} \quad (2)$$

IV. Evaluation

A. Accuracy of the EDR Algorithm

The result of running the EDR Algorithm on a given data set (PhysioNet Fantasia [26]) is shown in Table II.

TABLE II
Estimated Respiratory Rate (RR) of EDR Algorithm

Age	Sex	EST. RR/min	ACT. RR/min	Error/ min
23	F	23	20	3
28	F	16	15	1
30	M	19	17	2
32	F	24	18	6
68	M	17	17	0

B. Delay of the MQTT Broker Server

To test the delay between the MQTT Client and the MQTT Broker Server, we utilized Wireshark [27] to capture the packets between the device and the broker. The filter used in Wireshark, was the port number of the broker server. From Wireshark we determined that the delay was 73 milliseconds. This can vary as we were using the school wifi network which can cause longer processing delays. The data transmission path is shown in Fig. 10 and since the broker server is hosted in Ashburn, Virginia which is 2,293 miles away, this can cause higher transmission delay.

$$d_{prop} = (2000\ miles/3.0 * 10^8 m/s) \approx 10\ ms * 2 = 20\ ms.$$

Since the propagation delay is around 20 ms, but the full delay is 73 ms so the rest of the delay is from d_{queue}, d_{trans} and d_{proc}. There are 30 hops between the MQTT Client and the MQTT Broker server so the queuing delay will be higher. Since the delay of sending and receiving a packet is 73 ms, so just sending would be approximately 36.5 ms. Therefore the sending rate is 1000 ms / 36.5 ms = 27 packets per second.

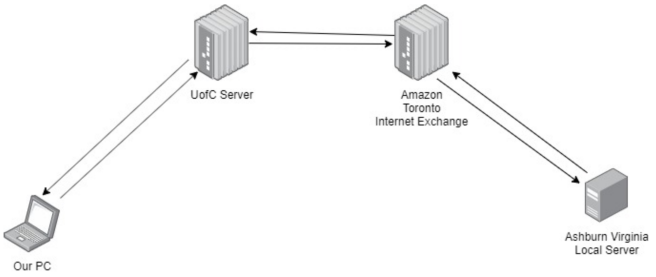

Fig. 10. The Data Transmission Path of Sending and Receiving

C. MQTT Broker Server Scalability

A network analysis tool was used to measure the following metric namely, Throughput, Latency, and Packets Dropped [27]. For an experimental comparison, a logarithmic scale of clients was chosen to send 1, 10, 100, and 1000 consecutive messages with a reliability of Quality of Service: QoS-0 and QoS-1 in each case. QoS-2 was excluded in test cases due to its large overhead. Table III represents different test case scenarios indicating that the system was able to accept 1000 concurrent MQTT clients publishing messages to a subscribed topic of a single MQTT broker server. From the results as shown in Table III, it is evident that the overall latency for QoS-0 is always lower than that for QoS-1 irrespective of the number of concurrent client requests.

TABLE III
MQTT Test Cases and Results

Client Number	Throughput (kbps)		Latency (ms)		Packets Dropped (%)	
	QoS-0	QoS-1	QoS-0	QoS-1	QoS-0	QoS-1
1	2	2.6	37	38	0	0
10	3.6	3.9	66	78	0	0
100	3.5	4.9	70	74	0	0
1000	105.5	57.9	92	112	2.9	2.6
10000	124.3	101.5	126	135	3.1	3.8

Similarly, another network analysis with the same metric has been measured with a MQTT Broker Cluster of 10 brokers using Round Robin load balancing algorithm. Table IV, Fig. 11 and Fig. 12 present a result that the MQTT Broker Cluster

of 10 brokers approximately increase the capacity by a log_{10}^{10} scale. That is because the Round Robin load balancing algorithm spread the work load evenly into 10 brokers compared to a single broker.

TABLE IV
MQTT BROKER CLUSTER WITH LOAD BALANCING

Client Number	Throughput (kbps)		Latency (ms)		Packets Dropped (%)	
	QoS-0	QoS-1	QoS-0	QoS-1	QoS-0	QoS-1
1	1.9	2.7	35	36	0	0
10	2.0	2.9	41	48	0	0
100	2.1	3.3	45	52	0	0
1000	5.0	5.7	71	73	0	0
10000	108.9	61.2	88	107	2.9	2.8

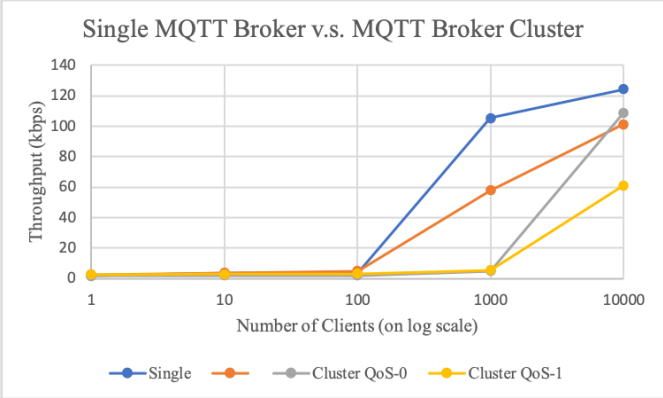

Fig. 11. Throughput vs. Number of Clients

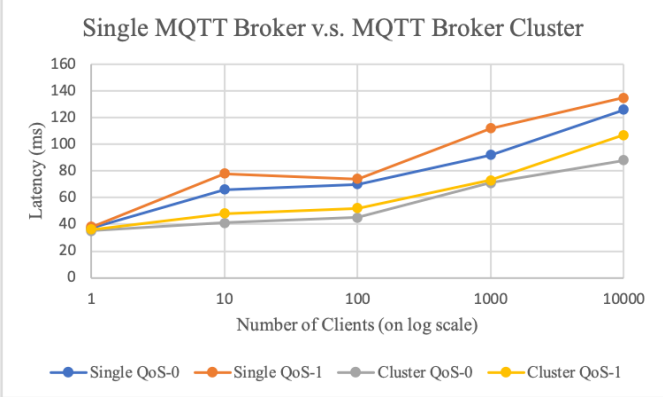

Fig. 12. Latency vs. Number of Clients

D. Resource Utilization

Many users were simulated concurrently using a Webserver Stress Tool on the back-end server which is hosted on Heroku. Each user in this test makes a HTTP GET Request involving retrieving the information from the relational database. CPU and Network utilization were high when all requests started coming as shown in Fig. 13. Almost all requests were answered immediately up to 4000 users. Beyond 4000 users the communication was terminated due to data traffic congestion and testing tool limitation.

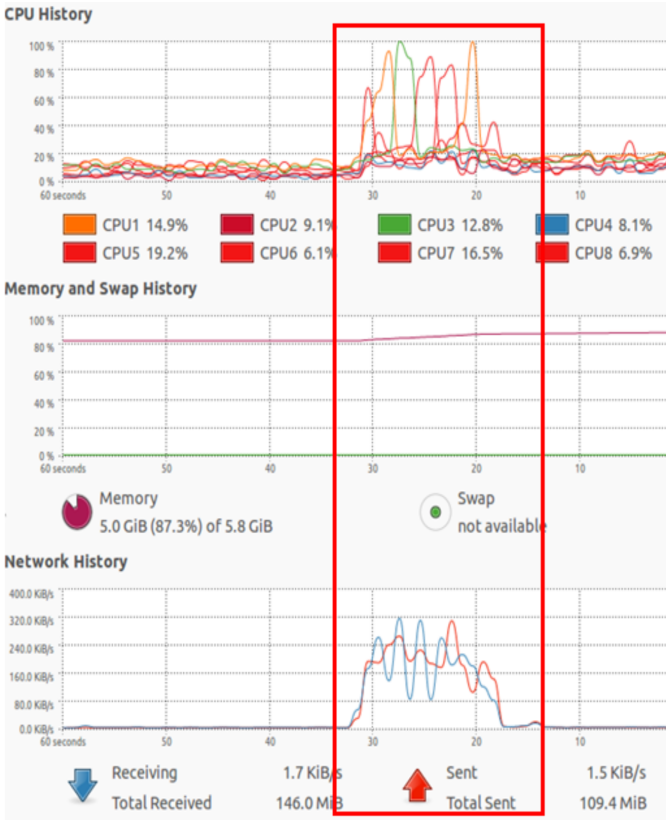

Fig. 13. Server Performance During Request Phase

E. Response Time

TABLE V
RESPONSE TIME FOR 4000 CONCURRENT REQUESTS

Maximum response time	322.382
Average response time	165.925
Minimum response time	080.329

The tested results of HTTP GET Request response time of simulating 4000 cuncurrent users are shown in Table V.

V. CONCLUSION

Internet of Things (IoT) has been rapidly developed in the last few years and is increasingly impacting various sectors especially in health monitoring. The proposed work in this paper is to develop an IoT-Based Opioid Overdose Monitoring System as there has been a spike in opioid overdose related deaths, triggering a Public Health Emergency in some provinces, including Alberta. The high scalability and the real-time health monitoring of the proposed system in this paper can help the PWUD reduce the risk of opioid-related deaths, which will not only ease the burden of the opioid epidemic on our healthcare system but also on the lives of those affected.

REFERENCES

[1] Paulozzi LJ, Weisler RH, Patkar AA. A national epidemic of unintentional prescription opioid overdose deaths: how physicians can help control it. J Clin Psychiatry. 2011; 72(5): 589-592. https://doi.org/10.4088/JCP.10com06560 PMID: 21536000

[2] Rudd RA, Aleshire N, Zibbell JE, Gladden RM. Increases in drug and opioid overdose deaths in the United States, 2000±2014.MMWRMorb Mortal Wkly Rep. 2016; 64(50±51): 1378±1382. https://doi.org/10.15585/mmwr.mm6450a3 PMID: 26720857

[3] Ahmad F, Rossen L, Spencer M, Warner M, Sutton P. Provisional drug overdose death counts. Atlanta, Georgia: National Center for Health Statistics, Centres for Disease Control and Prevention, 2018. Available from: https://www.cdc.gov/nchs/nvss/vsrr/drug-overdose-data.htm.

[4] Case A, Deaton A. Rising morbidity and mortality in midlife among white non-Hispanic Americans in the 21st century. Proc Natl Acad Sci USA. 2015; 112(49): 15078±15083. https://doi.org/10.1073/pnas.1518393112 PMID: 26575631

[5] Hall, Wayne D., and Michael Farrell. "Reducing the Opioid Overdose Death Toll in North America." PLoS Medicine 15.7 (2018): E1002626. Web.

[6] Davidson, Peter J, Eliza Wheeler, James Proudfoot, Ronghui Xu, and Karla D Wagner. "Naloxone Distribution to Drug Users in California and Opioid-related Overdose Death Rates." Drug and Alcohol Dependence 156 (2015): E54. Web.

[7] Bratberg, Jeffrey, Bill McLaughlin, and Scott Brewster. "Opioid Overdose Prevention." Journal of the American Pharmacists Association : JAPhA 55.5 (2015): 470-76. Web.

[8] Walley, Alexander Y, Maya Doe-Simkins, Emily Quinn, Courtney Pierce, Ziming Xuan, and Al Ozonoff. "Opioid Overdose Prevention with Intranasal Naloxone among People Who Take Methadone." Journal of Substance Abuse Treatment 44.2 (2013): 241-47. Web.

[9] Clark, Angela, Erin L Winstanley, Donna S Martsolf, and Michael Rosen. "Implementation of an Inpatient Opioid Overdose Prevention Program." Addictive Behaviors 53 (2016): 141-45. Web.

[10] N. Foundation, "Node.js," Nodejs.org, 2016. [Online]. Available: https://nodejs.org/en/.

[11] Xu, L. D., He, W. & Li, S. (2014), "Internet of Things in Industries: A Survey", Industrial Informatics, IEEE Transactions on 10(4), 2233-2243.

[12] Yang, G., Xie, L., Mantysalo, M., Zhou, X. L., Pang, Z. B., Xu, L. D., Kao-Walter, S., Chen, Q. & Zheng, L. R. (2014), "A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box", Industrial Informatics, IEEE Transactions on 10(4), 2180-2191.

[13] Cho, Wei-Ting, Lai, Ying-Xun, Lai, Chin-Feng, and Huang, Yueh-Min. "Appliance-Aware Activity Recognition Mechanism for IoT Energy Management System." The Computer Journal 56.8 (2013): 1020-1033. Web.

[14] Man, Lee Carman Ka, Cheng Mei Na, and Ng Chun Kit. "IoT-Based Asset Management System for Healthcare-Related Industries." International Journal of Engineering Business Management 7 (2015): International Journal of Engineering Business Management, 20 November 2015, Vol.7. Web.

[15] Al-Ali, A. R, Imran A Zualkernan, Mohammed Rashid, Ragini Gupta, and Mazin Alikarar. "A Smart Home Energy Management System Using IoT and Big Data Analytics Approach." IEEE Transactions on Consumer Electronics 63.4 (2017): 426-34. Web.

[16] Abdelgawad, Ahmed, and Kumar Yelamarthi. "Internet of Things (IoT) Platform for Structure Health Monitoring." Wireless Communications and Mobile Computing 2017 (2017): 10. Web.

[17] Meng, Xiao Jing J., Huan Qing Q. Cui, and Rong Hua. "An IoT-based Remote Health Monitoring and Management System." Applied Mechanics and Materials 571-572 (2014): 1176-179. Web.

[18] Thirunavukkarasu, Gokul, Benjamin Champion, Ben Horan, Mehdi Seyedmahmoudian, and Alex Stojcevski. "IoT-Based System Health Management Infrastructure as a Service." Proceedings of the 2018 International Conference on Cloud Computing and Internet of Things (2018): 55-61. Web.

[19] Yang, Yang, Xianghan Zheng, and Chunming Tang. "Lightweight Distributed Secure Data Management System for Health Internet of Things." Journal of Network and Computer Applications 89 (2017): 26-37. Web.

[20] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga and D. Kobayashi, "A scalable and low-cost MQTT broker clustering system," 2017 2nd International Conference on Information Technology (INCIT), Nakhonpathom, 2017, pp. 1-5.

[21] HiveMQ, "Creating highly available and ultra-scalable MQTT clusters". [Online]. Available: https://www.hivemq.com/blog/clustering-mqtt-introduction-benefits/.

[22] Rathore, M., Mazhar Ahmad, Awais Paul, Anand Wan, and Jiafu Zhang. "Real-time Medical Emergency Response System: Exploiting IoT and Big Data for Public Health." Journal of Medical Systems 40.12 (2016): 1-10. Web.

[23] Daeil Kwon, M., Hodkiewicz, Jiajie Fan, Shibutani, and Pecht. "IoT-Based Prognostics and Systems Health Management for Industrial Applications." IEEE Access 4 (2016): 3659-670. Web.

[24] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu and B. Xu, "An IoT-Oriented Data Storage Framework in Cloud Computing Platform," in IEEE Transactions on Industrial Informatics, vol. 10, no. 2, pp. 1443-1451, May 2014.

[25] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," in IEEE Transactions on Biomedical Engineering, vol. BME-32, no. 3, pp. 230-236, March 1985.

[26] GOLDBERGER A L, AMARAL L A, GLASS L, et al. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals [J]. Circulation, 2000, 101(23): 215-220.

[27] "Wireshark Go Deep." Wireshark.org, 2016. [Online]. Available:https://www.wireshark.org/.

[28] Charlton P.H., Villarroel M., Salguiero F. (2016) Waveform Analysis to Estimate Respiratory Rate. In: Secondary Analysis of Electronic Health Records. Springer, Cham.