International Journal of Control and Automation
Vol. 11, No. 1 (2018), pp.123-130
http//dx.doi.org/10.14257/ijca.2018.11.1.11

Event Detection System for Mountain Safety based on Continuous Query Processing of Moving Objects

Jiwon Wee[1], Daesik Ko[2] and Seokil Song[1]

[1]*National University of Transportation, Republic of Korea*
upsidejiwon@gmail.com, seokil.song@gmail.com
[2]*Mokwon University, Republic of Korea*
kds@mokwon.ac.kr

Abstract

In this paper, we propose a location-based real-time safety system to contribute to the safety of hikers in mountainous areas. Based on an internet-of-things (IoT), which considers the signal shadowing in the mountainous regions, we developed a method to detect safety events by analyzing the hikers' location stream in real time. For this purpose, we define event types that can be detected using the hikers' location stream. In addition, we propose a method to continuously process queries by indexing the hikers' locations on the grid at the mountainous region to achieve an effective event detection. It is also designed to detect events based on a big data stream processing platform that can handle scalable high-speed large-capacity location streams. Using a structured query language (SQL) database management system, we verify the proposed event detection algorithm.

Keywords*: Mountain Safety, Continuous Query, Moving Object, Event Detection*

1. Introduction

A total of 28,287 mountain hiking accidents occurred from 2010 to 2014, killing 568 people, as announced by the Ministry of Public Safety and Security in 2016. Compared with 3,088 accidents in 2010, 7,442 accidents occurred in 2014, which is an increase of ~140%. According to the type of hiking accidents, 33% of the total number of accidents were slip-and-fall accidents, 13% were caused by personal diseases such as hypothermia, and 8% occurred owing to safety regulations.

According to the Hiking Safety Manual, in the event of an accident, the first step is to recognize the emergency and decide how to act, the second step is to ask for help, and the third step is to take appropriate actions until the injured person is handed over to the emergency medical institution. When a person asks for an emergency rescue during a mountain hike, he/she should notify the emergency institution about the exact location of the accident, the type of the accident, the condition of the injured person, his/her name and contact information, how many people were injured, and how first aid was provided.

In this study, we analyze the location stream of a mountain hiker in real time, and design and verify a system that can recognize hikers' emergency situations and the location of their occurrence to inform a person concerned about the accident. The proposed system is aimed to automatically recognize the first step of the Hiking Safety Manual, and accordingly generate the aid request in the second step. Then, the system, according to the Manual, approximates the location and details of the accident, and informs the person concerned about the accident.

Received (October 15, 2017), Review Result (December 10, 2017), Accepted (December 18, 2017)

ISSN: 2005-4297 IJCA

Shadowing communication and GPS areas often exist in mountainous regions; hence, it is not always possible to identify hikers' positions in mountainous regions [1, 2]. This study proposes a method to detect safety events by assuming that the positions of hikers can be tracked in no shadowing areas, using a mountainous area internet-of-things (IoT) network composed of long range (LoRa) devices and gateways, and beacons, as proposed in Ref. [1]. LoRa is a network protocol for large-scale low-power long-distance wireless communications [3].

The proposed location-based detection method for mountain safety events stores and manages a user's location data collected in the target mountain region using the grid index [4, 5]. Furthermore, a continuous query [6] is performed on the user's location data collected in real time to detect the mountain safety events defined in this study. Three mountain safety events are considered. First, the system detects hikers located in a particular area of the mountain longer than a certain period of time. Second, the system detects a situation where a mountain hiker stays in an area for an unusually long time. Third, the system detects a situation where the hiker's position is no longer observed. The proposed location-based detection method for mountain safety events is designed for real-time detection of the abovementioned types of events using a continuous query processing method to continuously receive the location data stream of hikers.

This article is organized as follows. After this introductory section, Chapter 2 presents the structure of the proposed location tracking system for mountain hikers. Chapter 3 presents the method for the detection of mountain safety events. Chapter 4 describes the performed experiments, which can be employed to verify the proposed event detection method. The conclusions of this study are provided in Chapter 5.

2. Location Tracking System for Mountain Safety System

Figure 1 shows the structure of the proposed system employed for the tracking of the location of a mountain hiker, and detecting safety events in mountainous areas. In the mountainous region, there is an IoT network consisting of LoRa devices, LoRa gateways, and beacon devices. A LoRa device is a device that enables a low-power long-distance communication. It can transmit the beacon signal received by the beacon device to the infrastructure network through the LoRa gateway. The identifier (ID) of a beacon device, installed in the mountainous region, is determined by its location. The user moves along the trails with a terminal (*e.g.*, smartphone) capable of transmitting/receiving beacons. The beacon devices receive the beacon signals transmitted from the user's smartphone, and transmit the beacon device's ID and received ID of the user's beacon through the LoRa gateway. The server that receives the user beacon's ID and beacon device's ID can recognize the current position of the specific user using the location mapped with the beacon device's ID.

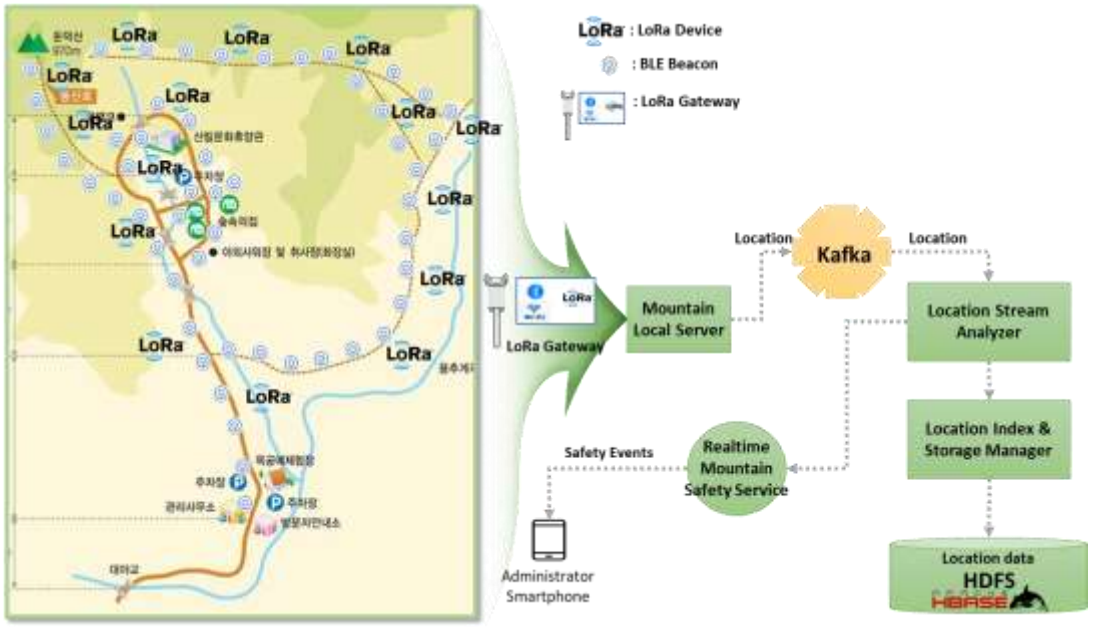


Figure 1. Location Tracking and Event Extraction System Architecture in LoRa- and Beacon-based Mountain IoT Networks

The data transmitted through the LoRa gateway in the mountainous region is further transmitted to the local server installed in the mountainous region. The local server sends the received data to the central processing server. The server uses an open-source stream processing platform, Kafka [7], to process the real-time location streams that are transmitted from multiple mountain regions during the data collection. The data collected through the Kafka platform are pre-processed through a spatial streaming analysis manager based on the Spark Streaming [8]. The preprocessed data is then used for the event detection, and indexed and stored with the position data index and storage manager, respectively. Spark Streaming is a stream processing tool based on Spark [8], an in-memory distributed parallel-processing framework. Location data streams are stored and managed through the Hadoop distributed file system (HDFS) [9] and HBase [10], which is a NoSQL database.

Figure 2 shows how a LoRa device collects the beacon ID of the user, received from the beacon device, and how it transmits the data to the server. The LoRa device can receive a list of user beacons received from multiple beacon devices. As shown in Figure 2, the LoRa device periodically calls the procedure Send() to transmit the numerous userBeaconList {recvBeaconID, userBeaconCnt, (userBeaconID0, userBeaconID1, … userBeaconIDN)} (a list of beacon devices' IDs and user beacon's IDs received by the beacon devices) to the server. Before the transmission, the LoRa device checks the value of PrevUserBeaconCnt, which shows whether there was a list of user beacons collected from the corresponding beacon devices in the previous cycle. It transmits the list if PrevUserBeaconCnt is different than 0; it does not transmit the list if PrevUserBeaconCnt is 0.

```
Send()
Input : recvBeaconList {recvBeaconCnt, userBeaconList0, userBeaconList1, …
userBeaconListM}
{
For (recvBeaconCnt)
SendBeaconList(userBeaconList)
}

SendBeaconList()
Input : userBeaconList {recvBeaconID, userBeaconCnt, (userBeaconID0, userBeaconID1,
… userBeaconIDN)}
{
Static prevUserBeaconCnt;

If (prevUserBeaconCnt != 0)
{
Send beaconList;
prevUserBeaconCnt = beaconList.userBeaconCnt;
}
Else // prevUserBeaconCnt()
{
Return;
}
}
```

Figure 2. Transmission Procedure of the User Beacon ID List by the LoRa Device

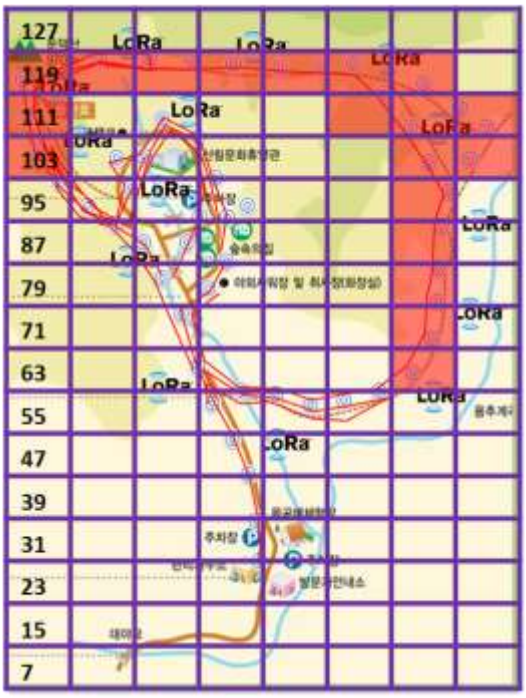
As shown in Figure 1, the user location data stream analysis manager is employed to detect, in real time, the abovementioned three event types. For the detection of the events, the latest positions of each user, times at which each user enters the corresponding location, and time periods of stay of the users at each location are recorded. Specific details are described in Chapter 3.

3. Proposed Mountain Safety Event Detection Method

As discussed in Chapter 1, in this study, three event types are detected using the proposed mountain safety event detection method. In the first event (Event 1), hikers located in a particular area of the mountain longer than a certain period of time are detected. Therefore, Event 1 detects hikers who stay within the dangerous area of the mountain for a duration longer than the safe time limit. In the second event (Event 2), situations where a mountain hiker stays in an area for an unusually long time are detected. If a mountain hiker stays in a dangerous area longer than necessary, there is a possibility of a safety accident, which should be detected and monitored. In the third event (Event 3), situations where a hiker's position is no longer observed are detected. In these situations, the position of a hiker in the dangerous area of the mountain suddenly becomes non-detectable.

For event detection, we construct a grid-based index and data structure, as shown in Figure 3. The monitored mountainous region is divided into grid cells, as shown in Figure 3. Each cell is assigned with a unique cell ID. The manager specifies dangerous areas in the mountainous region that must be always monitored, and stores them in the cells of interest (COI) table. The COI table contains the time moment (start_time) when the hiker enters the corresponding cell for the first time, and the time (last_time) when the hiker leaves the

considered cell. Furthermore, the COI table contains the values of the numbers of hikers currently staying in each of the cells (user_cnt) and hikers' IDs (user_list). The COI table further contains the time limit (time_limit) for each cell. The COI is a hash table with cell IDs as the keys. The UserINCOI records the ID of the current cell (cellID) of the hiker, first entry time (start_time) into the cell, and time moment when the location was identified in the cell for the last time (last_time). Further, the Location Table records the user's location in all areas, regardless of the COI. It is worth noting that UserINCOI is a hash table with userIDs as the keys.



COI (Cells of Interest)

cellID	start_time	last_time	user_cnt	time_limit	user_list
57	20171112-10:55	20171112-11:15	3	19:20	1, 2
65	20171112-10:25	20171112-11:05	2	19:20	4, 10
73	20171112-10:45	20171112-11:15	1	19:20	3,
…	…	…	…		
119	20171112-19:05	20171112-19:10	2	19:20	5, 11

UserINCOI

userID	start_time	last_time	cellID
1	20171112-11:10	20171112-11:10	57->X
2	20171112-11:05	20171112-11:15	57
…			

Location

timestamp	beaconID	userID	cellID
20171112-10:05	3	1	1
20171112-10:15	3	4	4
20171112-10:25	2	2	1
20171112-10:35	3	5	103
20171112-10:45	54	2	111
20171112-10:55	54	7	4
20171112-11:05	3	8	2
20171112-11:15	4	9	119
…			

Figure 3. Detection Method of Mountain Safety Events

The COI and UserINCOI tables are data structures used for the continuous query processing to detect mountain safety events. When the server receives the userBeaconList, collected and transmitted by the LoRa device, it executes the procedure shown in Figure 4, employed for the events detection. As shown in Figure 4, the server first obtains the corresponding cellID through userBeaconList.recvBeaconID. Although they are not shown in Figure 4, each beacon device installed in the mountainous region has a separate table that maps between the beacon device's ID and geographic location. For the conversion into a cellID, the server first obtains the geographical location through this mapping table, and then converts it to cellID.

```
ProcessContinuousQuery()
Input : userBeaconList
callback function : called whenever userBeaconList is received
{
cellID = GetCellID (userBeaconList.recvBeaconID);
userIDList = GetUserList (userBeaconList)

if (userBeaconList.user_cnt == 0) {
Reset values of cellID in COI table;
Return;
}

diffUserIDList = diff (COI(cellID).user_list, userIDList);
for (userID in diffUserIDList)
update UserINCOI(userID).cellID = MOVED;

for (userID in userIDList) {
Add userID to COI(cellID).user_list;
Update COI(cellID).start_time & COI(cellID).last_time;
If (COI(cellID).last_time > COI(cellID).time_limit)
Add (Event1, cellID) to eventList;

if (userID does not exist in UserINCOI) {
Add userID to UserINCOI;
update UserINCOI(userID).start_time, UserINCOI(userID).end__time,
UserINCOI(userID).cellID;
        }
Else {
If (cellID != UseINCOI(userID).cellID)
update UserINCOI(userID).cellID;

update UserINCOI(userID).start_time, UserINCOI(userID).end__time;
}
If (threshold1 < delta(UserINCOI(userID).start_time,
UserINCOI(userID).end__time)
Add (Event2, cellID) to eventList;
}

Add userIDs in UserINCOI whose cellID is MOVED to disappearedUserList;
Add (Event3, disappearedUserList) to eventList;

Return eventList;
}
```

Figure 4. Algorithm to Detect Events

After the conversion into cellID, the server calculates the difference (for the records existing only in the user_list, not in the userIDList) between the userIDList of the userBeaconList and user_list of the retrieved record, by searching for the records that correspond to the cellID of the COI table. Then, it adds the difference into the diffUserIDList. Subsequently, the server marks the cellID as MOVED for the userID of the diffUserIDList in the UserINCOI table. If user_cnt in the userBeaconList is 0, all values of the retrieved records are initialized to 0. Otherwise, the server deletes the user_list of the record, replaces it with userIDList, and changes the last_time of the record to the current time. If the last_time of the

record exceeds the time_limit, it is considered that an Event 1 has occurred, the event is created together with the corresponding cell ID, and then returned.

Moreover, for each userID of the userIDList, the server changes the cellID, last_time, and end_time using the records retrieved from the UserINCOI. After the change, if the difference between the start_time and end_time of the corresponding record is larger than the threshold1, it is considered that an Event 2 has occurred; the event is created together with the corresponding cell ID and then returned. After all changes are completed, if any cellID of each userID of userINCOI is marked as MOVED, it is considered that the Event 3 has occurred; the event is created together with the corresponding cellID, and then returned.

4. Experiments

A simple experiment was performed to verify the proposed mountain safety event detection method. For the experiment, first, we developed a path generator that can generate a path by simulating the locations of the hikers using the proposed detection system with LoRa- and beacon-based IoT network. The path generator involves 50 beacon devices installed in the mountain region, which are employed to detect the location of the user. Each beacon device is characterized with a relative position within the virtual mountainous region. Figure 5 shows the user interface of the path generator, where the red square indicates the hiker's current location.

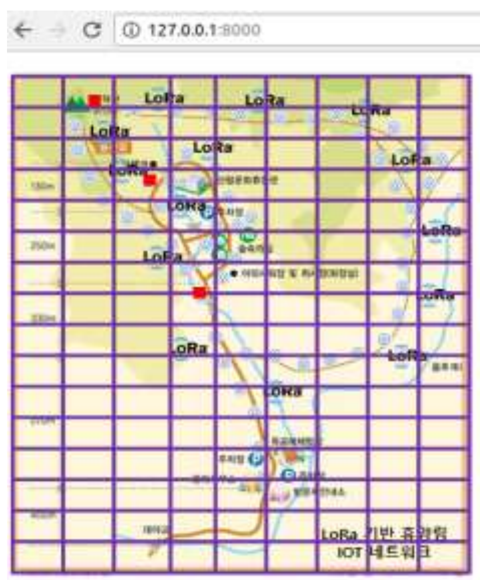


Figure 5. Path Generator Web Interface

The path is generated using a random number generator, based on the number of beacon devices along the path and time periods required for signals exchanges with each beacon device. The paths were generated assuming that the 100 hikers have 30–50 routes, the time to maintain communication with each beacon device is in the range of 5–10 s, and each collected location data is transmitted every 2 s. The total size of cells in the mountainous region was 8 × 16, while the number of COI cells was 19. Owing to its simplicity, we did not perform a detection experiment for Event 1. However, we performed detection tests for Event 2 and Event 3. The results of the experiments confirmed that the events are properly detected using the proposed method.

5. Conclusion

This study proposed and verified a location-based real-time method to contribute to the safety of hikers in mountainous regions. Based on the IoT network, considering the presence of shadowing in the mountainous region, the proposed system detects hikers' safety events through a real-time analysis of the hikers' location stream in an environment that provides the real-time hikers' locations. For this purpose, we defined three event types which can be detected using hikers' locations, and proposed a grid-based index and storage structure in order to achieve an effective detection. In order to verify the proposed event detection algorithm, we employed a database management system. In future studies, we aim to implement the proposed method using a big data platform with a significantly increased number of location data.

Acknowledgements

This research was supported by the Forestry Science and Technology Research and Development Project (2017063B10-1719-AB01) of the Korea Forest Service (Korea Forestry Promotion Institute), and, also, this was supported by Korea National University of Transportation in 2016.

References

[1] D. Ko, D. Lee and P. Park, "A study on the network design for IoT services in mountainous region", J. of Platform Technology, vol. 5, no. 3, **(2003)**, pp. 32–39.
[2] L. Soon, K. Min and H. Beom, "Mobile phone positioning technique trend for location-based service", Aerospace Industry Trends, vol. 12, no. 1, **(2014)**, pp. 220–231.
[3] L. Yi, G. Lee and H. Kim, "A study on the LoRa systems", Proceedings of Korea Institute of Communication and Information Conference, **(2013)**, pp. 217–218.
[4] P. Rigaux, M. Scholl and A. Voisard, "Spatial Databases - with application to GIS", Morgan Kaufmann, San Francisco, US.
[5] M. Esch, J. Botev, H. Schloss and I. Scholtes, "Gp3-a distributed grid-based spatial index infrastructure for massive multiuser virtual environments", Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS), **(2008)**, pp. 811–816.
[6] H. Hu, J. Xu and D. Lee, 'A generic framework for monitoring continuous spatial queries over moving objects', Proceedings of the 2005 ACM international conference on Management of data (SIGMOD), **(2005)**, pp. 479–490
[7] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system", International Journal of Scientific Engineering and Technology Research, vol. 3, no. 47, **(2014)**, pp. 9478–9483.
[8] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin and A. Ghodsi, "Apache Spark: A unified engine for big data processing", Communications of the ACM, **(2016)**, vol. 59, no. 11, pp. 56–65.
[9] T. Harter, D. Borthakur, S. Dong, A. S. Aiyer, L. Tang, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Analysis of HDFS under HBase: a facebook messages case study", Proceedings of FAST, vol. 14, **(2014)**, pp. 12.
[10] A. K. Karun and K. Chitharanjan, "A review on Hadoop—HDFS infrastructure extensions", Proceedingsof the IEEE Conference on Information & Communication Technologies (ICT), **(2013)**, pp. 132–137.

