SmartAIR: Smart energy efficient framework for large network of air quality monitoring systems

Vikram Rao
University of California, Davis
CA 95616, USA
vikrao@ucdavis.edu

Munesh Singh
Indian Institute of Information Technology Design & Manufacturing, Chennai
munesh.singh@iiitdm.ac.in

Prasant Mohapatra
University of California, Davis
CA 95616, USA
pmohapatra@ucdavis.edu

***Abstract*—Live street-level air quality monitoring is important application of sensor networks. Such application reveals human exposure to hazardous air pollutants. It assists general public, army troops, environment agencies and the Government in decision-making every day. Live data visualization and data fusion plays crucial role in presenting pollution updates effectively for end-users. We propose efficient interactive, live data visualization in our application. Our application efficiently renders pollution data fast in under $10ms$. Users will be instantly aware of pollution levels in their desired location. Continuous data-logging at data centers from large-scale of sensor networks poses major challenges. We use policy based network management technique to reduce unwanted data-logging requests. We implement novel policies in identifying and rejecting numerous unwanted requests at data centers. Each data-logging involves computationally expensive database operations and with our policy specification we were able to cut down expensive operations significantly ($\geq 83\%$ reduction, especially in denser regions like traffic congested roads). Finally, we implement Lazy load scheme to make our application more energy efficient. With this scheme we save data and battery in end-users device over longer periods of time. We conducted several real-life trials and we observed negligible mobile data consumption ($\leq 1MB$ for $1-hour$). Similarly, we observed negligible power consumption ($\leq 4\%$ in $1-hour$ run) in end-users device. Our implementation of novel policies and schemes provide real-life benefits to data centers and end-users. Our end-users experience better, faster and lively pollution updates. Our data centers experience relatively lesser network load and less computation overheads on scaling up.**

***Keywords*—Sensor Networks, Large scale, Lazy load**

I. Introduction

Excessive pollutants in the air are harmful. WHO[1] identified air pollution as one of the most significant global health epidemics of our time. There is an increasing mortality, respiratory disease like asthma, heart disease, and even cancer [1]. As per WHO, NAAQ and U. S. APHC [1]–[3] recommendations we observe that there is an important need for air quality monitoring. Military Exposure Guidelines (MEG) gives critical and recommended levels of air pollution for army troops [4]. Hence air pollution monitoring is important not only for general public but also for army troop's health. Our model as shown in Fig. I provides live air pollution updates with energy efficient information processing at client-side as well as smart network management at server-side. This research is a small step towards smart energy efficient projects that delivers safety for the people living under the constant exposure of pollution.

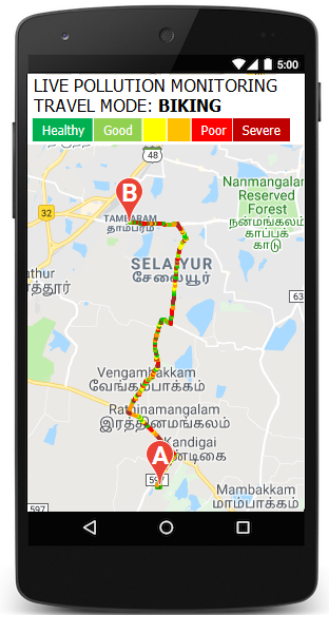


Fig. 1. Demonstrating live air pollution level in smart phone for a motorcyclist commuting between point A and B. Point A and B are starting point and destination of journey. Travel mode can be DRIVING, WALKING or BIKING. Pollution level can be Healthy(Green) to Severe(Maroon). In figure travel mode is BIKING and indicates overall live pollution level of trip up-to 5 meter accuracy.

A. Motivation

With the recent advancements of air pollution monitoring systems [5] and Government's need [1]–[4] to implement it, we are confident that government agencies will shortly deploy such devices at an large-scale to monitor pollution. So we can expect an sudden surge in number of such devices in every city that will continuously upload pollutant data into server every second. Hence server-side we will have high incoming traffic of data, which we have to control efficiently. Parallelly we can expect increasing number of users who want to view live pollution updates. Hence we will have very high data communication between millions of users and server, which we have to handle efficiently as well. Users can neither feel nor see harmful gases so effective data visualization plays a key role which motivates us to develop interactive, user friendly, faster application for users to visualize live pollution updates.

978-1-7281-0419-5/19/$31.00

B. Background

Notable amount of work is done by Google [6], [7] in mapping neighbourhood-level air quality. Since 2014, Messier *et al.*[6] used Google street car [7] for mapping pollutants emitted from cars, trucks and other sources in the city of Oakland, California. They have captured 3 million measurements spanning 15,000 miles every 31 days in the course of a year. Apart from pollutants, they have extended project to methane mapping to detect potential methane gas leakages.

Most of Messier *et al.*[6] research is focused in city of Oakland with the help of Google street car. Messier *et al.*[6] discuss briefly about scalability here we address scalability challenges in depth when air pollution monitoring systems are deployed in hundreds of thousands of cities across globe. Messier *et al.*[6] captured million readings over a span of year but in large-scale deployment we will receive several million readings every minute. With large scale deployment comes new set of challenges which we discuss further.

Google [6], [7] implementation gives us a glimpse of pollution data visualization using two-color gradient. Live data visualization plays critical role in presenting pollution updates effectively to end-users. Pollution data is dynamic with possibility to change every minute. To visualize effectively such innumerable and dynamic pollution data is a challenging task. In our model we aim to implement standard six-color indicator as recommended by National Ambient Air Quality Standard (NAAQS) [7]. However it evokes color-coding conflicts in Google maps *i.e.*some color-code overlap with other data layer(s) such as traffic data layer. To solve this, we look for simple and efficient data fusion. Data fusion makes end-users interpret data from multiple sources or sensors much better and easier without causing ambiguity [8]. We look in how effectively we can implement such method to combine pollution data-set with static or dynamic heterogeneous data-sets.

Survey by Wei *et al.*[5] on air pollution monitoring systems suggests that we have various stable implementations of wireless sensor network such as Static Sensor Network (SSN), Community Sensor Network (CSN) and Vehicle Sensor Network (VSN) for air pollution monitoring. For *e.g.*Google street car is a type of VSN. We observed greater advancements in networks with respect to mobility, data quality and maintenance. With these advancements and government's interest to deploy [1]–[4], we can expect high traffic of data-logging requests to server from variety of networks deployed across globe in near future.

Huge traffic of data-logging requests at data centers, if mishandled can directly affects server performance. ProgME [9] describe in their research that much of network traffic is mishandled due to insufficient network in-built intelligence. Networks inability to cope up with application-specific requirements or inability to adapt to network traffic can result inflexibility and limited scalability in server [9]. By efficiently handling network traffic we can avoid unnecessary server overloads.

With policy-based network management techniques we can control server's incoming traffic flexibly and in a simplified manner by defining new set of policies to govern the network traffic behavior [10]. Hence we attempt to devise new policies to manage incoming traffic. This policy based network provides us one solution to tackle network congestion problem, which frequently occurs in large scale of sensor networks.

Our end-user, who want to view pollution status continuously receives best experience when both data and battery is saved especially during longer journey. Hence we look for efficient data download schemes which avoids unnecessary data-downloads. Lazy load is an important design pattern commonly used in enterprise application architecture to defer object loading until it is really needed [11]. We see performance of this technique in operating systems to be significantly better in terms of memory and time [12]. Lazy load can provide us a solution by deferring data-download from server. Hence we check if this technique can significantly reduces client-server data-downloads. We implement and analyze the network performance of this technique in our model.

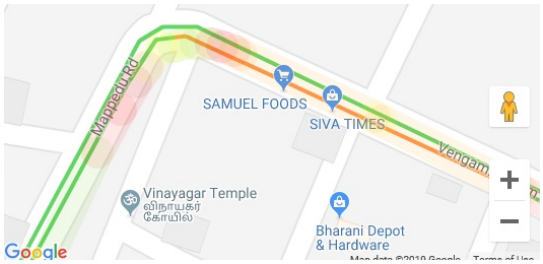
The rest of the paper is organized as follows: Section I present the motivation and background of the work. Section II presents the proposed model. Section III presents the results and analysis followed by conclusion section.

II. The Proposed Model

In this section, we discuss the development phases of our model:

A. Data Visualization and Fusion
B. Policy Based Network
C. Lazy load scheme

A. Data Visualization and Fusion



(a)

(b)

Fig. 2. Information fusion of several data layers: dynamic pollution data layer overlay (a) With dynamic traffic data layer (b) With static Google satellite images Map

Data visualization plays important role in presenting pollution updates to users. Data visualization model by Google [6], [7] and research done by Messier *et al.*[6] provides us summary of pollution in Oakland. We add novel features to make it practically usable in day-day lives. In our model we put in efforts to make pollution monitoring more interactive, lively and user friendly at the very best.

We demonstrate live air pollution mobile interface in Fig. I for a motorcyclist commuting from point A to B. NAAQS [7] standard six-color pollution level indicates pollution levels from Healthy(Green) to Severe(Maroon) as shown in Fig. I. User can choose commute mode (Either DRIVING, WALKING or BIKING) at the tap of a button. WALKING is useful for users exercising in outdoors. User can see the status and plan accordingly. In few taps user can change route or change mode and view live updates instantly.

Information fusion from multiple data sources gives us new perspectives. It is possible in our model to fuse both static and dynamic data-sets. We fuse dynamic pollution data-set with static Google Maps data as shown in Fig. 2(b). We overlay successfully by modifying line segment properties such as opacity and stroke-width. This minor modification brings amazing new information as shown in Fig. 2(b). Similarly we do fusion of our live data-set with Google's live traffic data-set as shown in Fig. 2(a). This information fusion provides new insights and draws interesting inferences such as possibility in Fig 2(a) shows that lesser-traffic on (Mappedu Rd.) is prone to heavy pollution at that moment.

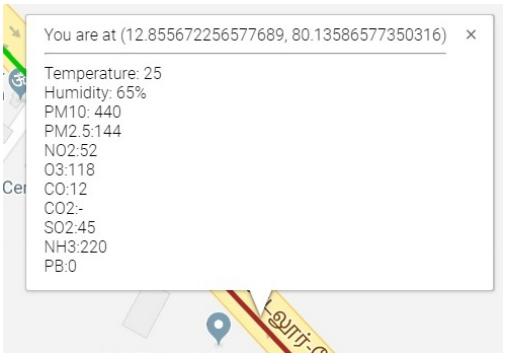


Fig. 3. Multi-modal air quality monitoring capability in our model. The information box shows reading captured by multiple sensorsat a particular Latitude and Longitude. Depending on country model can be reconfigured to accept different sensors, thresholds or even different logic

In our model we captured readings from Industry-standard battery-powered Particulate Matter (PM) sensor [13] but our model is futuristic with multi modality feature. Provision for data logging from multiple sensor is possible. As different countries have different pollutant monitoring standards, so this feature plays a vital role. We demonstrate the same in Fig. 3, we observe information box showing reading captured from many sensors PM, CO2, CO, SO2, NOx, etc.,

In our model, one novel feature is history management. We demonstrate it in Fig. 4, where user can replay/view entire history of pollution in the route. Fig. 4a shows our intuitive

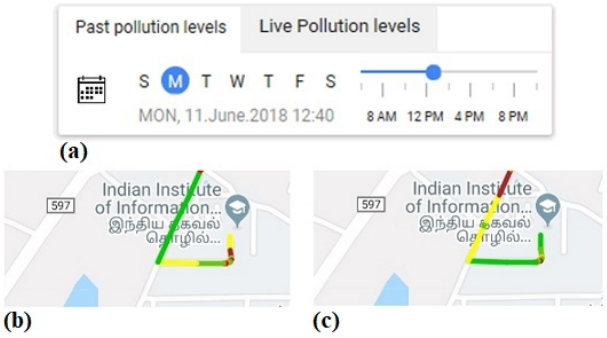


Fig. 4. Unique history management of pollution data. (a) Intuitive interface to quickly navigate to current/past pollution level (b) Showing past pollution levels dated 11.June.2018 12:40PM IST. In single tap, user can switch from past to live mode (c) Live pollution level.

interface where user can pick any day in calender then tap on any day of week and then slide to any time to view history. Data queried is retrieved Fig. 4b shows the result. User can switch back to Live mode just by tapping 'LIVE' Tab as shown in Fig. 4(a,c) shows the results of this action. This kind of visualization narrates entire pollution history of selected route in a unique way.

Our Data visualization implementation is very efficient, it takes only 10 millisecond average to plot pollution data points. Performance with various implementations are discussed further in the results section.

B. Policy Based Network

In next few years smart air pollution monitoring devices will be everywhere. There will be surge in network traffic because these devices will continuously post recorded data from sensor to server's shared databases. This shared database will be leveraged by other applications such as the read-intensive application shown in Fig. I which shows pollution levels to end-user.

To manage rising network traffic efficiently we need policy based network management [10]. We propose four novel policies in four-step sequence as shown in Fig. 5 that filters out all unnecessary data-logging requests step by step. Policy structure as shown in Fig. 5a is simple query to check whether set of conditions are met. If met then request is processed else request is rejected. All the incoming data-log requests first hits Policy I. Policy I to III are read-intensive whereas Policy IV is write-intensive. Policy I to III is least expensive database (Fig. 5b) operations whereas Policy IV is most expensive and time consuming database write (Fig. 5c) operation. Policy I to III uses memcached high speed read-only database but Policy IV does computationally expensive tasks like creation of indexes, checking if record exists accordingly insert/update record. Hence we carefully define policy in such a way that all unnecessary requests filter out within Policy I to III. In this manner, we execute Policy IV only for qualified requests and not every data-logging request gets logged thus saving lots of valuable network resources.

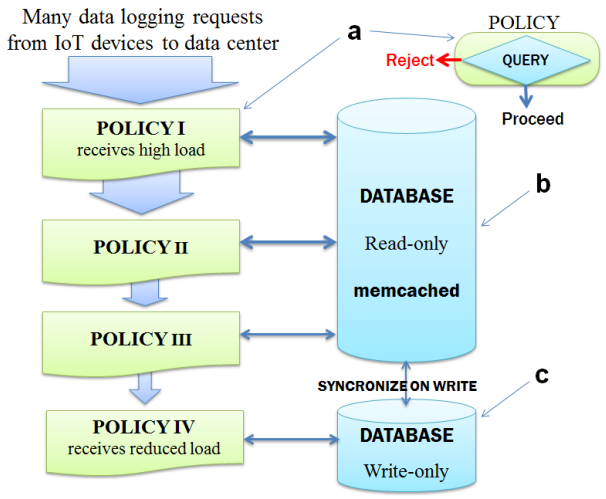


Fig. 5. Policy based network to streamline network load. (a) Represent policy structure (b) Database for read intensive operations (c) Database for write intensive operation. These novel policies I - IV schemes reduces network overloads

Algorithm 1 Policy Collection

Input: R : HTTP Request Object
Entry Point: SECURE-ACCESS(R)

procedure SECURE-ACCESS(R) ▷ Policy-I
  $id \leftarrow R.UniqueDeviceIdentifier$
  $key \leftarrow R.SecretAuthenticationToken$
  $status \leftarrow$ DATABASE-LOOKUP$(id, status)$
  if $status.registered$ **and** $status.active$ **then**
    if ISAUTHENTIC(key) **then**
      return LOG-FOR-LIMITEDTIME(R, id)
  return $null$
procedure LOG-FOR-LIMITEDTIME(R, id) ▷ Policy-II
  $K_1 \leftarrow 60$ ▷ Log device every 60 seconds only
  $record \leftarrow$ DATABASE-LOOKUP$(id, lastRecord)$
  if $K_1 \geq ServerTime - record.lastLogTime$ **then**
    return LOG-LIMITEDPOINTS(R)
  return $null$
procedure LOG-LIMITEDPOINTS(R) ▷ Policy-III
  $K_2 \leftarrow 60$ ▷ Log region every 60 seconds only
  $L \leftarrow 5$ ▷ Threshold limit of 5 points
  $M \leftarrow 110$ ▷ Rounding radius of 110 meter
  $latLng \leftarrow R.LatitudeLongitude$
  $rLatLng \leftarrow$ ROUNDOFF$(latLng, M)$
  $records \leftarrow$ DATABASE-LOOKUP$(rLatLng, lastRecords)$
  if $K_2 \geq ServerTime - records.lastLogTime$ **then**
    if $L \leq records.count$ **then**
      return DATA-LOGGING$(R, rLatLng)$
  return $null$
procedure DATA-LOGGING$(R, rLatLng)$ ▷ Policy-IV
  $id \leftarrow R.UniqueDeviceIdentifier$
  $data \leftarrow R.SensorValues$
  DATABASE-WRITE$(id, rLatLng, ServerTime, data)$
  return 1

Policies Collection defined in Algorithm 1 has four simplified policies procedures which we have implemented and tested successfully. Policy-I SECURE-ACCESS(R) being the entry point does the preliminary security checks. By cross-checking in database we check device status and we verify authenticity of the device with the help of secret key/password. If request is authentic we proceed further else request is rejected.

Policy-II LOG-FOR-LIMITEDTIME checks in database if device has already logged past K_1 time (Here 60 seconds). If true, the request is forwarded to next policy else rejected.

Policy-III LOG-LIMITEDPOINTS is the most important policy. It checks if requested region has already logged L points in a radius of M meters within time-span of K_2 seconds. We query mentioned condition in database and if satisfied we proceed to Policy-IV else reject request. Policy-IV DATA-LOGGING is the final policy where requested sensor data along with coordinates and time-stamp is written into database. This step involves most expensive database operation so by filtering out Policy I-III we data-log only genuine and qualified requests.

With our novel Policy schemes I-IV we improve quality of service reducing unwanted use of network resources ultimately lowering power consumption at data centers. We provide detailed results and analysis in the results section.

C. Lazy load scheme

Pollution data is dynamic with possibility to change every minute. In a map between point A and B as shown in Fig. 6 (inset) there is possibility of hundreds to thousands of dynamic pollution data points. We do not want to download all data at once, hence we apply Lazy load [11] to download data fragment by fragment when needed.

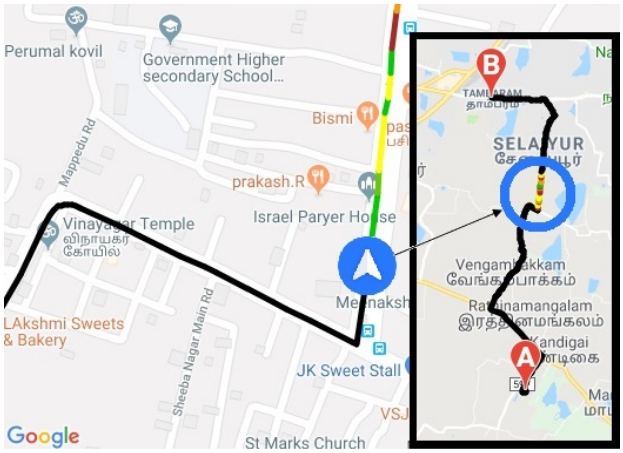


Fig. 6. Figure shows Lazy loaded map where only necessary data points are downloaded, updated and displayed. Image inset shows complete route map from point A to point B where only small fragment of data is downloaded when necessary.

At the same time we want to clear the pollution data of explored tracks. In Fig. 6 you can observe black strokes in explored route indicating data is cleared off map. In

comparison, Fig. I is map without Lazy load containing 500 data points and Fig. 6 is map with Lazy load containing only 20 data points at the moment. Now instead of updating 500 points we update only 20 points periodically. In this way we significantly reduce download of unnecessary data points therefore we save data, bandwidth, battery at client side and stop unnecessary download requests at server. By this scheme we bring higher efficiency in a large scale scenario where we drastically cut down millions of unnecessary communication between client-server thus saving millions of dollars. Detailed one hour experimental run is presented in next section.

III. Results and Analysis

A. Data Visualization Performance

There are many implementation to plot data points onto static maps like Google Maps. We can either use either buffer array or special files in GeoJSON format for loading Geo Spatial data (Latitude, Longitude, Pollutant readings). We can plot data either using Data Layer or without data layer [7]. We see that these implementations perform significantly different for larger data set as shown in Fig. 7. We have implemented these methods in various combinations to plot 500 points and conducted performance tests as shown in Fig. 7. Incrementally numerous iterations has been done as shown in Fig. 7 and We observe that Data Layer with GeoJSON file loading is efficient and takes under $10ms$ on average to plot and display data. Basically at this speed our pollution data gets displayed much before Google maps static data. Hence we use this particular implementation extensively in our model to improve efficiency

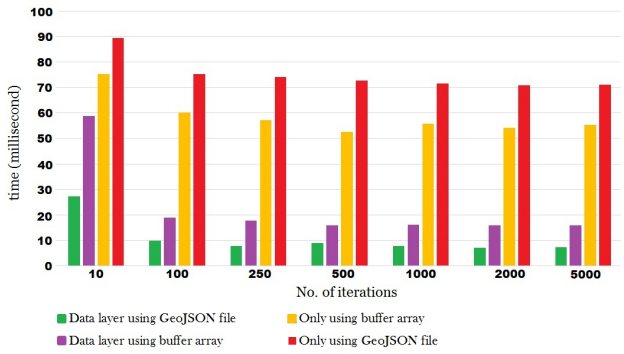


Fig. 7. Performance of various implementations for Mapping data onto Data layer (a) using GeoJSON file (b) using buffer array. Mapping data without Data layer (c) using buffer array (d) using GeoJSON file. The number of iterations (X) verses average time (Y) to plot 500 Geo-spatial(Latitude, Longitude, Data) points.

B. Policy Based Network Performance

As sensor networks across globe rise, there will be increasing number of data-logging into server. During data-logging data flows from sensor network to server. Before server logs such request, four policy checks are performed as defined in Algorithm 1. To simulate, we define density: sensors per $M * M\ meter^2$) (Here $M = 110\ meter$). Fig. 8 shows plot

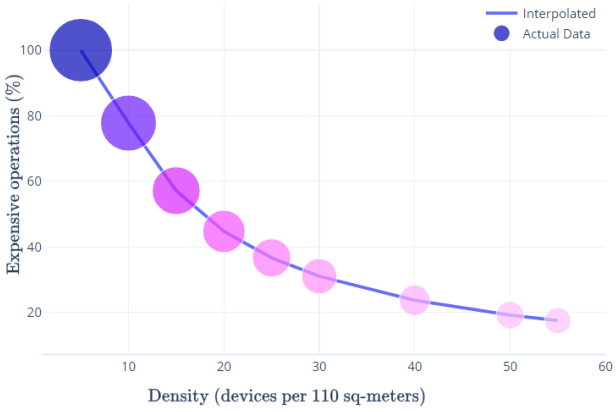


Fig. 8. Plot showing total expensive operations(%) at server vs. density of network (Number of sensors per sq. meters)

of expensive operations vs. density. For *e.g.*, density of 20 means there are 20 sensor network in a radius of $110 * 110$ m^2). We observe in Fig. 8 that as networks gets denser, server cuts down up-to 83% operations. The small bubble in Fig. 8 indicates only 17% expensive operation happens due to Policy-III condition where we already receive required samples from sensors within radius of $110{*}110\ m^2$. Less dense sensor network areas (big bubble in Fig. 8) have relatively lesser populations so we can conveniently execute all database operations in such regions.

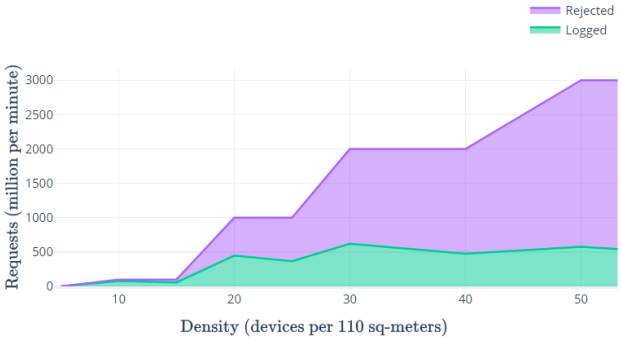


Fig. 9. Plot showing total number of requests (in million) at server vs. density of network

Similarly we do analysis on amount of rejected requests, accepted requests vs. density as shown in Fig. 9. We observe in Fig. 9 that in denser areas we allow around 525 million requests and reject 2.4 billion requests. In contrast in least denser areas we allow 1 million requests and reject none. However in reality and in both cases we will reject much more requests due to Policy-I and II because some devices are not authentic and some devices have already logged previously within stipulated time.

C. Lazy load Performance

We tested on Android 8.0 (Orea) operating system having a regular mobile data pack. We navigated across various routes

and captured $1^{st} - hour$ application-specific usage snapshot during the journey. This snapshot includes log of application data usage and application power usage. With mobile data toggled on during the entire trip, Fig. 10 shows $1^{st} - hour$ application data usage statistics.

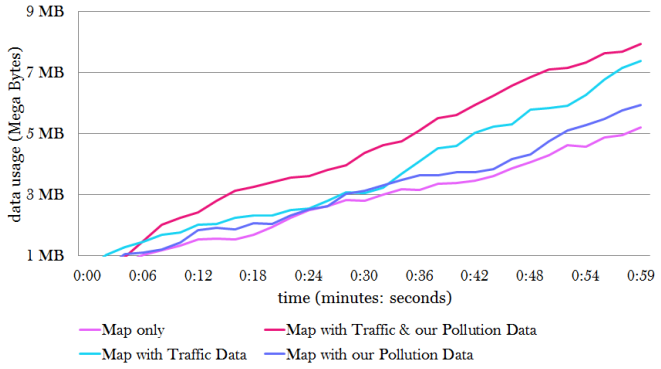


Fig. 10. Application data usage plot in its 1^{st}-hour in several Map modes

Four cases presented in Fig. 10 Trip with Map only mode, Map with traffic data, Map with our pollution data and Map with traffic as well as pollution data. We observe in Fig. 10 that in its $1^{st} - hour$ we observed only ($0.7MB$ in $1^{st} - hour$) data was consumed which is very less unlike other dynamic data such as traffic data which consumed triple more ($2.2MB$ in $1^{st} - hour$). This is because traffic data other than its main route is also loaded during the trip.

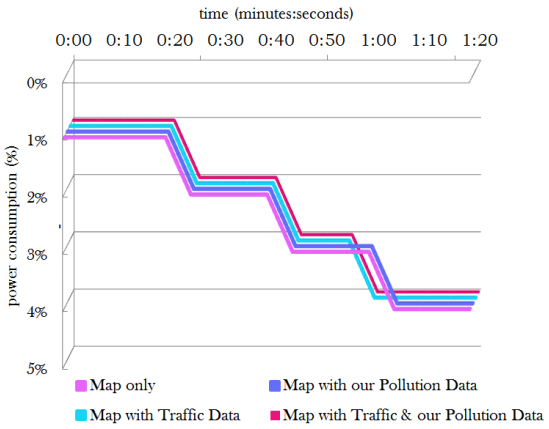


Fig. 11. Application power usage plot in its 1^{st}-hour in several Map modes

Similarly, Fig. 11 show plot of application power consumption (since last charged) in its $1^{st} - hour$. We test in all map modes at various time slots in a smart phone that solely runs on battery during the entire trip. We observe in Fig. 11 that there is low battery consumption in user's device especially in longer term. Power consumption is same as Map without any Data layers, it is almost negligible because we transmit/receive very less mobile data packets. Average of $85mAh$ was consumed in the entire 1-hour trip. Hence we observed greater efficiency in our Lazy load implementation.

Conclusion

We observed the consequences of air pollutants on our health and find the importance of monitoring air pollution. Live data visualization and fusion plays key role in presenting pollution updates effectively to end-users. So user will be more aware of the situation. Our model provides interactive, faster and live pollutant updates which means our pollution data appears to user much before Google maps data. Advancements in sensor networks and governments interest to deploy across globe suggests us that we can expect high traffic of data-logging at server. We devise efficient policies in identifying and rejecting millions of unwanted requests. Data-logging involves computationally expensive database operations. With our policies we are able to cut down such operations significantly especially in a dense sensor network. Thus saving valuable network resources, time and power in data centers. End-user's experience is faster and efficient with our Lazy load scheme. Using this user's both data and battery life is saved especially during longer journey. In 1-hour trial run we observed that not much data was consumed unlike other dynamic data such as traffic data. We observed almost negligible power consumption as well. Our framework provides great efficiency to both server and end-users. Thus saving millions of dollars in maintenance at data-centers as well as our end-users experience better, faster and live pollution updates.

References

[1] WHO. (2018, Aug.) 7 million premature deaths annually linked to air pollution. [Online]. Available: http://www.who.int/mediacentre/news/releases/2014/air-pollution/en/

[2] NAAQS. (2018, May.) Naaqs table. [Online]. Available: https://www.epa.gov/criteria-air-pollutants/naaqs-table

[3] USAPHC. (2018, May.) Air quality. [Online]. Available: https://phc.amedd.army.mil/topics/envirohealth/aq/Pages/default.aspx

[4] M. J. Morris, L. L. Zacher, and D. A. Jackson, "Investigating the respiratory health of deployed military personnel," *Military medicine*, vol. 176, no. 10, pp. 1157–1161, 2011.

[5] W. Yi, K. Lo, T. Mak, K. Leung, Y. Leung, and M. Meng, "A survey of wireless sensor network based air pollution monitoring systems," *Sensors*, vol. 15, no. 12, pp. 31 392–31 427, 2015.

[6] K. P. Messier, S. E. Chambliss, S. Gani, R. Alvarez, M. Brauer, J. J. Choi, S. P. Hamburg, J. Kerckhoffs, B. LaFranchi, M. M. Lunden *et al.*, "Mapping air pollution with google street view cars: Efficient approaches with mobile monitoring and land use regression," *Environmental science & technology*, vol. 52, no. 21, pp. 12 563–12 572, 2018.

[7] Google. (2018, Jun.) Air quality monitoring. [Online]. Available: https://www.google.com/earth/outreach/special-projects/air-quality/

[8] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, Jan 1997.

[9] L. Yuan, C. Chuah, and P. Mohapatra, "Progme: Towards programmable network measurement," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 115–128, Feb 2011.

[10] D. C. Verma, "Simplifying network administration using policy-based management," *IEEE network*, vol. 16, no. 2, pp. 20–26, 2002.

[11] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley, 2013.

[12] J. W. Arendt, P. P. Giangarra, R. K. Manikundalam, D. R. Padgett, and J. M. Phelan, "System and method for lazy loading of shared libraries," Jan. 13 1998, .Patent US 5,708,811A.

[13] Inovafitness. (2018, Apr.) Particulate matter sensor. [Online]. Available: http://www.inovafitness.com/en/a/chanpinzhongxin/95.html