Prefetching and Caching in Internet of Things Environment using Publish Subscribe Model

Author: Poorn Pragya / Advisor: Dr. Mohan Kumar

ABSTRACT
With the rapid increase in wireless technologies and mobile devices in our daily lives, Internet of things has become very popular and is an emerging trend prevailing today. Since there are millions of devices sharing data in the Internet of Things and these devices can be stationary or mobile, therefore there are varying bandwidth connections. This varying bandwidth can have direct impact on performance of the applications running on these devices. In this project, we use publish subscribe model for prefetching and caching the real-time data that can be used in the future by the applications running on IoT devices. Prefetching relevant information before it is required by the applications can significantly improve the performance of applications and overcome delay caused by lower bandwidths situation or out of range devices. With publish subscribe model, the subscribers of the network have the advantage of getting the prefetched information which they can use and similarly the publishers get the benefits when they act as subscribers in future.

1. INTRODUCTION
With emergence of smart devices, Internet of Things (IoT) is one of the major technologies prevalent today [1]. IoT is networking between the smart devices which have inbuilt sensors and actuators along with software that can communicate with other devices to exchange information. IoT makes devices interconnected across existing network which can be controlled via networks and allows better integration of physical world with computer based system. This not only provides better integration of smart devices with our day to day activities but also helps in improving efficiency, accuracy and economic benefits.

Publish Subscribe Model [3] is an already existing messaging pattern where senders of the messages can send different class of messages. The receivers can express interest in receiving a certain type of messages by senders. In a pub-sub model, publishers are nodes that generate documents and submit it to the implementing system while subscribers are nodes that subscribe to certain types of documents to receive. The pub-sub models also consists a third actor, called brokers which are tasked with the duty of determining the subscribers along their subscription list. They also have the responsibility of forwarding the relevant documents to subscribers in case a certain relevant documents arrive. The main motive to use brokers is to enhance scalability.

Prefetching [4] [2] is a technique to fetch relevant data from other devices in anticipation prior to demand. Prefetching and caching [5] real-time data in IoT environment is essential because it not only helps in improving the efficiency of the applications running on IoT devices but also saves bandwidth thus helping other applications to run much faster. Prefetching data in wireless devices faces time and memory challenges meaning when to run the prefetch and what to prefetch. Since IoT devices are memory constrained, there is a need for prefetching just sufficient amount of information that is valid and relevant, and IoT devices interact based on real-time data and information accuracy ages and expires with respect to time.

This project utilizes the concept of prefetching and caching
real-time data to develop an application using publish subscribe model. The application developed is based on mobile devices and called Pub Sub marketplace. In this application, the publishers represent people who have visited marketplace to buy certain items and want to help the community. These items have certain properties that vary with time, for example, fruits’ prices vary with time. In return, when they act as subscribers at a later point of time in future, they can view the documents published by other publishers. The application will be able to keep track about the name of the item to be published, price, image and location. The idea is to keep the marketplace information at user’s fingertips in the community to help a certain person subscribe to the list of items he purchases frequently. Most of the android phones have sensors such as cameras, GPS etc to collect marketplace information.

2. REVIEW OF LITERATURE ON TOPIC

We started our literature search by reviewing the article by J.Gubb et al. [1]. The article describes about how IoT can be achieved by ubiquitous sensing in wireless sensor networks and describes the architecture and applications of IoT in various domains. The domains can be classified into personal and home, enterprise, utilities, mobile. The personal and home domains describe the applications of IoT in home network like home monitoring systems. The enterprise domain of IoT provides insights about how IoT can be used at enterprise level by creating environmental monitoring systems to smart cities. The utilities domain targets the use of IoT for service optimization rather than consumer consumption. Smart grid and smart meters can be potential application in this domain. Mobile domain of IoT focuses using smart phones as IoT devices that can be used in smart transportation using inbuilt mobile sensors like GPS, accelerometers, camera, Wi-Fi and blue-tooth technologies. The paper also describes about the cloud based architecture that can be used by future IoT devices to work seamlessly. The “Things” can be used to sense data that can be sent to the cloud infrastructure to perform analytics on it and provides users with useful information.

The paper by Vural et al. [5] gives insights about the in-network caching of IoT data at content routers. The paper describes the IoT data being identified not only by time and location tags but also by time range values set by applications. This paper describes one of the key property of IoT: transiency. Transiency means the IoT data becomes less fresh when data is not cached near to source generating the data, hence in-network caching of data should be done on content routers that are as close as possible to the data source. But using this approach causes increase multi-hop traffic load. Hence, there needs to be a trade off between transiency and multi-hop traffic load and paper goes on to derive the cost function based on these properties to store a particular IoT data at a particular content router.

The papers by Master et al. [2] and Tuah et al. [4] talks about speculative prefetching of data in wireless networks. Paper by Master et al. [2] talks about fetching multimedia data on fluctuating wireless networks on mobile devices which are memory constraint. When wireless networks are fluctuating, the bandwidth needed to download the multimedia data can be severely impacted and applications running on mobile devices can face some downtime in these scenarios. The paper goes on to discuss a dynamic programming solution to prefetch data during good bandwidth conditions based on importance of data and memory required to store that data. This solution uses probability distribution to provide randomness to fetch or not fetch policy. The dynamic programming solution uses four parameters to dynamically decide the fetch or not policy, namely, number of tasks on mobile devices waiting to be fetched, number of tasks waiting to be processed, state of wireless links and state of wireless link from central server to the mobile device. The paper by Tuah et al. [4] describes the above problem from prefetcher’s perspective. The paper develops a prefetch policy based on cache as one of the parameters and integrates prefetching and caching of data into a single model. Cache hit ratio is an important factor which needs to be taken into consideration while prefetching and using the performance evaluation of prefetcher which was not used in paper by Master et al. [2]. Paper by Tuah et al. [4] makes the model much more accurate as seen in the performance results.

The paper by Rezende et al. [3] proposes a Publish Subscribe design which is highly efficient in terms of transmission of messages from publisher to subscriber and keeps the nodes updated about published document and subscribed messages. The main motivation of this design is "mobility increases connectivity” [3]. As the number of nodes increases in the mobile environment, they can travel and increase the connectivity of mobile ad-hoc network. Each node acts as a publisher as well as subscriber. All the published messages are transmitted to other node through multicast messages within the radio range of particular node. Duplicate messages are not considered. When the node acts as publisher, it maintains a publisher table. This buffer maintains list of all publications. The node multicasts its publications in four phases as described in paper by Rezende et al. [3]; when a publication is created, when a node stops, when a node has publication and matches new incoming subscription, when a node receives a publication which matches any stored still valid subscription [3]. When a node acts as subscriber, it maintains a subscription table which stores all the subscriptions the node has received. Each foreign subscription in the list is associated with an expiry time stamp, and the subscription is removed from subscription table after this time has elapsed.

These papers provide a deep insight of prefetching, caching and publish subscribe model in wireless environments and we intend to use the basic designs as described while working in localized IoT environment.

3. DESIGN OF THE APPLICATION

The application developed demonstrates the prefetching and caching features in Publish Subscribe domain and is called Pub-Sub Market Place. It is a peer to peer application based on Publish Subscribe Model. It relies on the fact that not all nodes are within the each others’ range all the time with good bandwidth conditions. It is necessary to prefetch and cache published items before the publisher goes out of network. It is also necessary to prefetch before subscribers starts viewing the items in order to improve efficiency of the application. Users who use the app and act as subscribers can subscribe to the fruit items that they intend to buy. Users who act as publishers can publish can fruit items: bananas, mango, apple, strawberry, blueberry items. Each mobile node in the system will act as publisher as well as the subscriber. As soon as the application is opened,
the application would be able to join a multi-cast group using a class D IP address. Reliable multi-cast will be used instead of broadcast since we need to connect to only those nodes which have this application running on it. The communication between multiple nodes will be possible using an access point (AP). If a node lies in the range of more than 1 access point then, multi-cast will be done to the access point which has more number of active nodes. Each node is associated with a cache which would be quite small. The cache will use Least Recently Used (LRU) scheme.

The multi-cast algorithm runs periodically for publishers and subscribers. This mechanism would come in handy in the scenario when a publisher/subscriber is out of the network and just comes within the reach of access point and need to publish or subscribe data published by other nodes.

Since, the application is designed to name of the item, price, time of day, day of week, location, and images of fruit items, therefore the size of the packet might not conform the upper limit specified by the UDP multicast packet of 65,000 bytes. Moreover, since the MTU limit of IP layer is 1499 bytes, the IP layer further fragments each packet at network layer. Further, UDP doesn’t provide reliability and flow control, hence extra mechanism is required to add the flow control mechanism. This is done by chunking the original packet to multiples of 1499 bytes and sending it over the framework and buffering all the chunks on the subscriber side until all chunks are received.

To address the above design challenge, following multicast packet structure is used. Here Request id is unique id of the published/subscribed item. chunk id represents chunk number of the particular request. This is necessary in case the packets arrive out of order. The uniqueness of <Request ID, Chunk ID> will allow the application to identify the position of the packet in the stream while recreating it on the subscriber side. Remaining chunks field is used to tell the subscriber about how many chunks are still pending until it can start recreating the original data object. Destination IP is the IP address of the destination, Destination Port is port where the subscriber is listening, Source IP is the IP address of source publisher, Source Port are port of the publisher, data is the chunk of data being sent.

### 3.1 Publish Subscribe Design

As described earlier, in a publish subscribe model each node can act as a publisher, a subscriber or a broker. Publisher’s task is to publish different topics items to the framework. The subscribers task is to subscribe to a particular topic. The broker are all nodes that not a publisher or a subscriber and their responsibility is to make sure the published topics reach the subscriber that have subscribed to a particular topic.

Each node in the topology will act as a publisher, a subscriber and a broker. Therefore to conform to this design, each node will maintain a publisher table, a subscriber table and a broker table. The publisher table will be used to store all the publishing related information that the application needs to multi-cast to provide other nodes access to its data which other nodes would have requested. The subscriber table will store all the information that will required while prefetching. The node would use this table to send multi-cast request to other nodes about the information it needs to prefetch data. The broker table will be required to store the requests of publishers and subscribers and this table table would be used to route the information from publishers to subscribers.

The multi-cast algorithm runs periodically for publishers and subscribers. This mechanism would come in handy in the scenario when a publisher/subscriber is out of the network and just comes within the reach of access point and need to publish or subscribe data published by other nodes.

Since, the application is designed to name of the item, price, time of day, day of week, location, and images of fruit items, therefore the size of the packet might not conform the upper limit specified by the UDP multicast packet of 65,000 bytes. Moreover, since the MTU limit of IP layer is 1499 bytes, the IP layer further fragments each packet at network layer. Further, UDP doesn’t provide reliability and flow control, hence extra mechanism is required to add the flow control mechanism. This is done by chunking the original packet to multiples of 1499 bytes and sending it over the framework and buffering all the chunks on the subscriber side until all chunks are received.

### 3.2 Routing

While routing, a request there can be two cases possible
1. When all the nodes are reachable from access point.
2. When nodes share multiple access point.

In case 1, the publisher can simply publish its result of its publisher’s table using reliable multi-cast and all subscribers can get the desired information which requested for it. In case of subscriber, the subscriber can subscribe to the class of information that prefetching algorithm requests. All the information can be directly routed using the single access point which connects all the nodes.

In case 2, the need of the broker table as described in design section is evident. The mobile node 2 is one such node that lies in the range of access point 1 and access point 2.
The only way for mobile node 1 to communicate to mobile node 3 is through mobile node 2. When mobile node 3 subscribes to the information being published by mobile node 1, the mobile node 2 can save the subscribing request of mobile node 3 and the access point information from where the request came. Mobile node will multicast all contents of the broker table to all the access points within its reach. Once mobile node 1, receives the request, it can multicast the desired information, to mobile 2 using access point 1 and mobile node 2 can forward the received information to mobile node 3 using access point 2.

### 3.3 Prefetching

As described earlier, prefetching is technique to fetch relevant data in anticipation prior to demand. The relevant information that a node would require from other users would be other users marketplace information such as name of the fruit item, price, time of the day, day of week, location and image. This information is will be prefetched in anticipation that user might try buy the item which he has subscribed for. Furthermore users buying history, time, location and price is another important factor for prefetching since most of the users usually have a schedule and fixed location to visit market place and buy products that lie within their buying budget. Prefetching other user’s data would be at the maximum when the user is near a market place. This is because users are much more motivated to compare with other peers while purchasing items near their local marketplace. Hence, I propose to use the user’s spacial position to prefetch information since it would result in maximizing the cache hit.

The application would need to record a window of information about what time of the day, day of week, location the user goes to the market place along with price and image of the item. Another important factor would be how close (distance) is the user to his routine marketplace. Based on these statistics, we can create a probabilistic model to classify whether a the person is heading to marketplace for buy certain products or not. The probabilistic model utilizes logistic regression model based on Naive Bayes algorithm to find the probability of user going to the marketplace. The same is shown in the diagram below:

![Figure 7: Probabilistic Model for prefetching](image)

The model to compute the prefetch probability is calculated by model described below:

\[
P(\text{Prefetch} = \text{Yes} | \text{Data Item}) = \frac{P(\text{Data Item} | \text{Yes}) \times P(\text{Yes})}{P(\text{Data Item})}
\]

\[
P(\text{Prefetch} = \text{No} | \text{Data Item}) = \frac{P(\text{Data Item} | \text{No}) \times P(\text{No})}{P(\text{Data Item})}
\]

Here, \(P(\text{Data Item} | \text{Yes}) = P(\text{Data Item category (intersection) no. of Yes}) / P(\text{Data Item category})\)

Finally, the Posterior Probability(\text{Prefetch} = \text{Yes}) = P\text{i} (P(\text{Prefetch} = \text{Yes Data Item Category = Yes})).

Here \(\text{P}i\) represents multiplication.

Prefetcher would be activated if Posterior Probability(Prefetch
A sample run of the algorithm to create a Prefetcher model is shown below:

![Sample Model](image)

**Figure 8:** A sample model created using Naive Bayes to compute prefetch

Input variables are a list of attributes which provide some value in making the decision whether to prefetch a certain item or not. As shown, for each category of input variables, the posterior probability is calculated from the historical data that is saved when the user buys a certain item. Suppose, the user buys mango on Monday at 7:00 pm at a price of 3.99 at a certain market which lies within 5 mile radius of the user’s home location. Then this information along with other similar history of the user’s activity will be used to compute the posterior probability about the likelihood of the user buying a certain item (Prefetch = YES) or likelihood of the user not buying the item. This is done by counting all the different categories of input variables from the history and dividing it by the total. Then all positive posterior probabilities are multiplied and similarly negative posterior probabilities are multiplied to get the negative posterior probability. If the positive probability is greater than the negative probability for a certain item, then it is put in the queue to prefetch else it is rejected. The prefetch becomes active when it is within a certain threshold miles from the market place with prefetch value “yes”.

### 3.4 Caching

![LRU Cache](image)

**Figure 9:** LRU Cache

The application uses least recently used (LRU) scheme with a variation to use data size along with data oldness as one of the parameters for cache replacement. As shown in the figure, the data structure used to implement the cache is Hash Table backed by a doubly linked list. The Hash Table stores the data item ID as key along with pointer to node block where actual data is stored in the linked list. The linked list is organized in such a way that the front of the list always points to the most recently used item while the end of the list points to the least recently used item. When a new item is inserted, the least recently used item is removed and a new item is placed at the front. In case when the data already exists in the cache, then that particular item is moved in the front.

### 4. IMPLEMENTATION

The application will be used for to run on android devices. The implementation can be grouped into the following components:

#### 4.1 Application Architecture

![Application Architecture](image)

**Figure 10:** Application Architecture

#### 4.2 Publisher

This class provides the functionality for a node to behave like a publisher and publish requests when the user needs to publish any item. This class uses Datagram Packet class of Java to send the packets to a predefined multi-cast address that all applications agree on joining. Since multi-cast uses UDP protocol to transfer data, extra care has to be taken to improve reliability. As soon as the application is started, this class is initialized in the main activity and a singleton object reference is passed across all the threads of the application to ensure reliable sending of packets from each thread in a thread-safe way.

The maximum amount of data that can be sent in one data packet is 65,535 bytes. But the maximum Transmission Unit (MTU) size that is enforced on the IP layer is 1500 bytes. If we try to send a packet greater than 1500 bytes, the IP layer fragments it and sends it in chunks. Due to lack of flow control in UDP, if any one packet is even lost, the entire data packet is dropped. So to overcome this issue, in the publisher client serialized data packet greater than 1490 bytes is being chunked and sent serially so as to ensure proper flow control.

#### 4.3 Subscriber

As soon as the application is initialized, a subscriber client is initialized in a separate thread and listens to a predefined...
All the transactions will be stored in a list and naive bayes will be based on the historical window of users history of subscriber, subscriber activity, publisher activity and thread purpose of this activity is to initialize the handler, publisher, subscriber receives the number of chunks that publisher wants to multicast first and then receives each chunk of size with a maximum of 1490 bytes to ensure reliability. To ensure flow control, the subscriber has the option to send request the publisher to decrease their sending rate. Once all the chunks are successfully received in the buffer, a serialized object is created based on the received chunks bytes. This process is especially necessary to ensure images are published since the size of the image will most likely exceed 1490 bytes.

4.4 Publisher Activity

The publisher activity marks start of the publisher documents. This activity provides features to collect name of the item, price of the item, location of the item, the image of the item and time of the item when it was published. Using this activity, the publishers of the item can use to multicast the request of their and item and the corresponding subscribers who have subscribed to this item will receive an update.

As soon as the end user, provides the specifics of the item he wants to publish, and clicks on the publish button, all the information is collected about the data item into a realizable object. This object is then serialized by the publisher client as described above in form chunk of bytes to be multi-casted among the peers within the reach of the publisher. The user can’t publish another item until the current item is completely published.

4.5 Subscriber Activity

The subscriber activity marks the start of subscribing the documents. The application users would require be required to subscribe to the list of documents at the start of the application. Once each user has marked which items they are interested in, this activity would allow only those class of item to be received on their application. This list of items will be also be modified based on their history of item purchase i.e. the application would look at the past history of items which user has been interested in the past and adjust the list of items to subscribe based on this list. This activity contains a list view to display the subscribed items i.e. the name of the item, price of the item, location where the item can be found, image of the item and freshness based on time when the item was last published.

4.6 Main Activity

This activity marks the start of application itself. The purpose of this activity is to initialize the handler, publisher, subscriber, subscriber activity, publisher activity and thread policy.

4.7 Prefetcher

As discussed in the design section, the prefetching of items will be based on the historical window of users history of buying items to identify the items which user buys the most. All the transactions will be stored in a list and naive bayes algorithm as discussed in the design section will be used to calculate the probability of each item whether to prefetch or not. The prefetching will based on location, time and name of the item that user has used earlier.

Once the probabilities of each item is identified, it will be compared with certain threshold to identify whether to prefetch the item. The prefetch would run in background periodic thread will calculate the probability on a periodic basis in order to save the battery life.

4.8 LRU Cache

The prefetcher communicates with the LRU cache module to once it prefetches data and replaces the old cache entry as described in design section.

5. RESULTS

The application was tested on 4 mobile nodes with 2 nodes acting as publishers and 2 nodes acting as subscribers. The testing framework was set up in such a way that allow all subscribers to subscribe all the items and publishers publish all the items. The amount of item published in each run by each publisher was close to 50 percent. The cache hit ratio was observed with prefetching enabled and prefetching disabled and total time required to fetch the item was calculated. The below table shows the reading recorded.

![Image](image.png)

**Figure 11: Results**

As shown in the results, the cache hits with prefetching is greater than cache hits without prefetching in most of the cases for both the subscribers. This made the fetch time to fetch remaining item lesser as can be observed in total time taken which 213.98 ms with prefetching compared to 331.16 ms without prefetching for subscriber 1. Similarly, for second subscriber, the total time taken from fetch all 75 items is reduced to 292.05 ms with prefetching compared to 430.06 ms without prefetching because of increased number of cache hits.

6. CONCLUSION AND FUTURE WORK

As seen in the above results, prefetching and caching definitely improves the cache hits ratio of the application which in turn helps in increasing the run time of the application.
This is especially important in IoT environment since most of the sensors are mobile these days and data lacks freshness. It is highly possible that sensors keep on moving in and out of networks or the bandwidth to communicate with peers might drop significantly. In those situations, it is necessary to predict what data the application / sensor might need in the future and cache it when the other nodes are within the radio range.

Currently, in this application we have used naive bayes algorithm to create the prefetch model and predict data for future use. Naive Bayes algorithm is based on the assumption that all the attributes on the posterior probability are independent. This is not the case in all the scenarios. There are a lot of applications like coordination among the UAV drones where the attribute selection is quite complex and most of the attributes are dependent of each other. In those cases, Naive Bayes model might not provide a better efficiency in predict correct data. Hence, for complex problems different regression models like SVM, Decision Trees etc. might be tested to check which model performs best.

7. REFERENCES