Scalable and Adaptive Context Delivery Mechanism for Context-aware Computing

Lenin Mehedy
Dept. of Computer Engineering, Kyung Hee University, 446-701, South Korea, Email: lenin@oslab.khu.ac.kr
Sungyoung Lee*
Dept. of Computer Engineering, Kyung Hee University, 446-701, South Korea, Email: sylee@oslab.khu.ac.kr
Salahuddin Muhammad Salim Zabir
Dept. of Computer Engineering, Kyung Hee University, 446-701, South Korea, Email: szabir@oslab.khu.ac.kr
Young-Koo Lee
Dept. of Computer Engineering, Kyung Hee University, 446-701, South Korea, Email: yklee@khu.ac.kr

Received: December XX 2006; revised: December XX 2006

Abstract— **Presence of innumerable sensors, complex deduction of contexts from sensor data, and reusability of contextual information impose the requirement of middleware for context-aware computing. Smart applications, hosted in myriad devices (e.g PDA, mobile, PCs), acquire different contexts from the middleware and act intelligently based on the available contexts in a context-aware computing environment. Therefore we believe, as the system grows larger in near future, scalable delivery of contexts from the middleware to numerous context-aware applications will be inevitable. But, pure unicast or pure broadcast based dissemination can not provide scalability as well as less average latency. Hence, in this paper we present a scalable context delivery mechanism for these middleware to facilitate the development of larger context-aware computing systems. Proposed scheme is based on hybrid data dissemination technique where the most requested data (e.g. HOT contexts) are multicast and the rest (e.g. COLD contexts) are delivered through unicast to reduce network traffic. We dynamically prioritize and classify the HOT and COLD context data depending on the number of requests and longest waiting time. Moreover, the division of bandwidth between the delivery of HOT and COLD contexts reduces average latency and we also decrease polling traffic by incorporating leasing mechanism. Extensive simulation proves the proposed scheme to perform better. We also present implementation detail of our prototype that is developed using the available tools such as *Jini* framework and *Java Reliable Multicast Service (JRMS)* library.**

Index Terms— **Context delivery, Context aware Middleware, Hybrid data dissemination, Bandwidth Division, Scalable Delivery**

I. INTRODUCTION

Context awareness is the key element to provide pervasive services (i.e. anywhere, any time) to users in context-aware computing era, where the system is supposed to have the ability to detect and sense, interpret and respond to the situation of an entity (e.g. user, applications etc.) [1]. Innumerable sensors will be deployed in a context-aware computing environment to collect various information and then deduce some higher level contexts such as user's contexts (e.g. location, speed, activity, preference etc.), environmental contexts (e.g. temperature, humidity etc.), systems contexts (e.g. network status, available resources etc.). However, collecting and processing huge sensor data imposes significant computational as well as design overhead for developing individual smart applications. Furthermore, some deduced contextual information may also be reusable for many other applications. Hence, context-aware computing environment, also termed as *ubiquitous computing* [2], is supposed to provide middleware support for context awareness [3]. Middleware solutions provide the system support, reusability and separation of concerns that are required for developing context aware systems (see [4] for a survey). For example, a middleware performs all the functions of context sensing and inferring and then the smart applications utilize these contextual information to provide intelligent support to the users. Hence, every context-aware middleware is supposed to have the following three main phases of execution (Fig. 1): a) Acquisition of raw sensor data, b) Context inference from the sensory data, c) Context delivery to applications. Therefore, the delivery of context is an indispensable part for any context aware middleware to facilitate the building up of "Context-aware Computing" environments. This paper focuses on the scalable context delivery mechanism for such middlewares.

While most of the middleware researches (e.g. Aura [5], ContextFabric [6], Context Toolkit[7], Gaia [8], iRos [9], mavHome [10], Solar [11] etc.) model small interactive environments such as home, class room, meeting room etc., we may envision a larger smart environment (such as a corporate office, academic building, shopping complex etc.) where numerous context-aware applications (we interchangeably use client or receivers), running on mobile devices like PDA or stationary devices (e.g. desktop PC), frequently request various contexts to the middleware. Size of context data may vary from few hundred bytes (e.g. XML representation of user's preference, location, activity, profile, and temperature

*Sungyoung Lee is the corresponding author

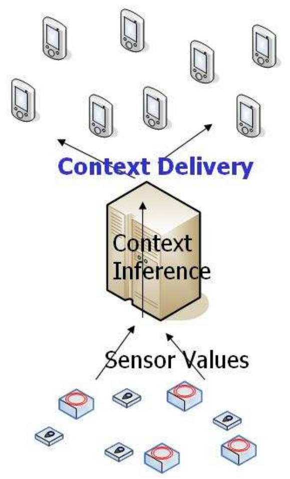

Fig. 1. Basic Functionalities of a Context-aware Middleware

of environment) to several kilobytes (e.g. image, video frame). Large number of users may request for the same context data and unicast of this context data will cause serious performance degradation in terms of access latency and bandwidth utilization. Moreover, a user may need particular context information for a long duration and polling in such case will also cause the misuse of valuable bandwidth due to large number of request messages. Hence, efficient and scalable dissemination of contextual information to large number of clients in such environment will be of utmost importance. However, our main focus of this scalable dissemination is to provide efficient use of bandwidth and provide less latency to clients while considering heterogeneous size of contexts and variable request rate.

Now, the first challenge is to provide quick response time for clients by reducing network traffic. If the context information of interest is the same among different clients, traditional unicast (point to point or pull based) connection-oriented data services are uneconomical because it incurs a lot of unnecessary traffic from clients to server as well as on the reverse direction. Even if the current technology allows us to have high network bandwidth and server capacity, most of it would be under utilized and wasted during non-peak periods. Broadcast (push based) is an efficient and scalable dissemination method in a connectionless mode to any number of clients with no significant performance degradation in terms of access latency [12]; but a major concern for the success of such system is broadcasting the right set of data. Because, broadcasting less important data may cause network overload. On the other hand, on-demand broadcast (pull-push) method the server aggregates the requests of clients and broadcast the data. But broadcasting the context with lowest request rate (cold item) may also increase network traffic. So hybrid approach combines the benefit of broadcasting hot context data (having higher request rate) and that of unicasting the cold context data (having lower request rate) [13]. Even with this suitable and scalable approach, we have the problem of differentiating hot and cold context data and formulate a suitable broadcast scheduling algorithm for quick response time. These challenges are also considered for web databases and mobile computing [12], [13], [14], [15], [16], [17].

Besides, a smart environment is truly dynamic in nature where the context receivers (or, clients) may appear and disappear unpredictably. So the periodical delivery requires incorporating a leasing mechanism [18] or sending of periodical beacon from the clients [19] so that the contexts are not delivered indefinitely if the clients disappear without prior notice.

Furthermore, all the clients and middleware should share the same concepts of domain and context groups (e.g context ontology) for semantic inter-operability [20], [3]. This sharing of context ontology helps in reducing ambiguity and provides better matching between requests and associated contextual information.

Unavailability of a single comprehensive solution to these problems motivates us to devise a novel and scalable context delivery mechanism that resolves all these problems for context aware middleware in ubiquitous computing domain. In this paper we present our solution which is an effort towards developing a robust and comprehensive context delivery mechanism for the middleware CAMUS (Context Aware Middleware for Ubiquitous System) [21], [4]. Earlier version of this work appears in [22].

In brief, our delivery mechanism has the following properties:

1) It uses context ontology for semantic inter-operability.
2) We dynamically differentiate hot and cold context items to disseminate through multicast and unicast respectively. So, this adaptive delivery makes it more suitable for ubicomp application.
3) Lease mechanism is used instead of periodic context update request (polling) and copes up with dynamic environment.
4) We use *request rate* of an item and *longest waiting time* of any outstanding request for an item to prioritize as hot or cold items. Hence, we prioritize a data either because it is very popular or because it has at least one long-outstanding request
5) We further perform bandwidth division between hot and cold items for better performance in terms of average latency.
6) Our delivery technique also gives solutions to *push popularity problem.*

Some terms that we use in this paper are:

- *Average Waiting Time:* The amount of time on average from the instant that a client request arrives at the middleware, to the time that the data is delivered.
- *Longest Waiting Time:* The maximum amount of time that a request remains in the service queue before it is satisfied.
- *Average Latency:* The average latency for a data item on the push channel is half of the period of the multicast cycle if we assume that the items are multicast sequentially. However, the latency for pulled items are totally different because if an item of size is queued at the server for transmission, the corresponding queuing delay is either $O(Si)$ or unbounded [14].

The rest of this paper is organized as follows: Section II describes related works. Section III explains the proposed method of context delivery. Section IV presents the performance evaluation and Section V describes implementation detail of the prototype using some available tools. Then section VI concludes with some future work.

II. Related Work

Several middleware have been designed for ubiquitous (or, pervasive) computing [5], [6], [7], [8], [9], [10], [11]. Most of them focus on small interactive environments with small number of users such as class room [5], [8], meeting room [8], [9], home [10] etc. So their main concerns are about seamless communication among middleware components [5], [8], [9], abstraction to sensors [7], [6], [11] etc. However, large scale deployment of ubiquitous systems (e.g. corporate office, academic building, shopping complex, airports, subway etc.) with myriad context-aware clients (e.g. mobile or static) will be inevitable in near future. Hence in such large environments, scalable dissemination of contextual information based on available bandwidth, request rate etc. will be of great importance. A few works can be found regarding context query and aggregation [23], [24], [25], [19]. But none of them consider the important performance parameters such as request rate, available bandwidth, size of context items etc. during delivery of contextual information.

The most related research to ours is the Context Discovery Protocol (R-CDP) [19] that has been implemented and evaluated in "Reconfigurable Context Sensitive Middleware" (RCSM) [26]. The fundamental difference between R-CDP and our work is that R-CDP uses broadcast to request for a context and the middleware unicast the data to the requester, which is completely opposite to our mechanism as we use unicast for request and combination of multicast and unicast for delivery. We use the technique of RxW algorithm [17] to prioritize the delivery where they use *Refresh Priority*, which is based on the divergence of contexts and energy consumption of *Provider*. We also perform bandwidth division for hot and cold items for optimal average latency. The similarity with their work is that the motivation of their *Neighbor Validation Beacons (NVB)* is same as that of our *Lease Renewal* and we also have the way of specifying the update threshold for context update notification. However, they do not use context ontology for semantic interpretation. Moreover, R-CDP has not been tested for scalability [19], which we believe an important performance issue for large scale deployment of smart applications.

The idea of Hybrid data dissemination technique was first used in the Boston Community Information System [15] by combining broadcast and interactive communication to provide most updated information to an entire metropolitan area. This scheme is also adopted in [12] [13], [14], [15], [16] where the issue of mixing push and pull web documents together on a single broadcast channel was examined. But the document classification problem was introduced in [13] and later document classification along with bandwidth division was resolved in [14]. We employ these ideas of hybrid dissemination, scheduling, classification of data, bandwidth division etc. for scalable and efficient delivery of context for ubiquitous computing environment. Though our approach is very similar to [14], we differ in calculating the popularity of an item not only on the total number of requests but also on the longest waiting time of any outstanding request to avoid starvation of request for cold item . Moreover we also employ lease mechanism to reduce the periodic request (polling).

III. Proposed Scheme

Before detailed description of our context delivery scheme, we present our assumptions here:

- We assume that the clients can receive data though unicast as well as multicast.
- Items on the multicast channel are assumed to be HOT as well as useful for future. Hence, clients cache the items on the multicast channel that are delivered by the middleware
- Before sending request, a client at first checks its cache for that item. If it is found, client assumes it to be a fresh copy and uses that. Currently we do not consider client's cache management issues [27], [28].
- Clients discover the context delivery service using some discovery techniques (e.g. Jini Lookup [18]) and then submit requests. We also assume that the clients authenticate and exchange encryption keys to ensure secure delivery of data.

A. Context Delivery Scheme

Pure unicast (pull) or pure broadcast (push) based dissemination can not provide scalability as well as less average latency. Contemporary schemes [23], [24], [25], [19] for delivery of context may be viewed as pure pull based solutions. However, we use hybrid dissemination (push-pull)[13], [14] for scalable context delivery. In this data dissemination technique , the most popular data (e.g. HOT items) are multicast and the rest (e.g. COLD items) are delivered through unicast to reduce network traffic.

But this scheme introduces three inter-related data management problems at the middleware: *First:* The middleware must dynamically classify the requests between hot and cold context data and schedule the delivery according to priority or popularity (Prioritization). *Second:* The middleware should divide dynamically its bandwidth between unicast pull and multicast push for optimal use of bandwidth and ensure low latency (Bandwidth Division). *Third:* As the hot context data are multicast, some clients may receive the information passively in the sense that they do not send any request for that data. Therefore, the middleware lacks a lot of invaluable information about the data requirement that is used to decide the hot and cold data item dynamically (Push Popularity Problem [14]).

In the following sections we briefly describe our context model *Contel* [20] and solutions to the challenges as stated above (i.e. associated with hybrid dissemination).

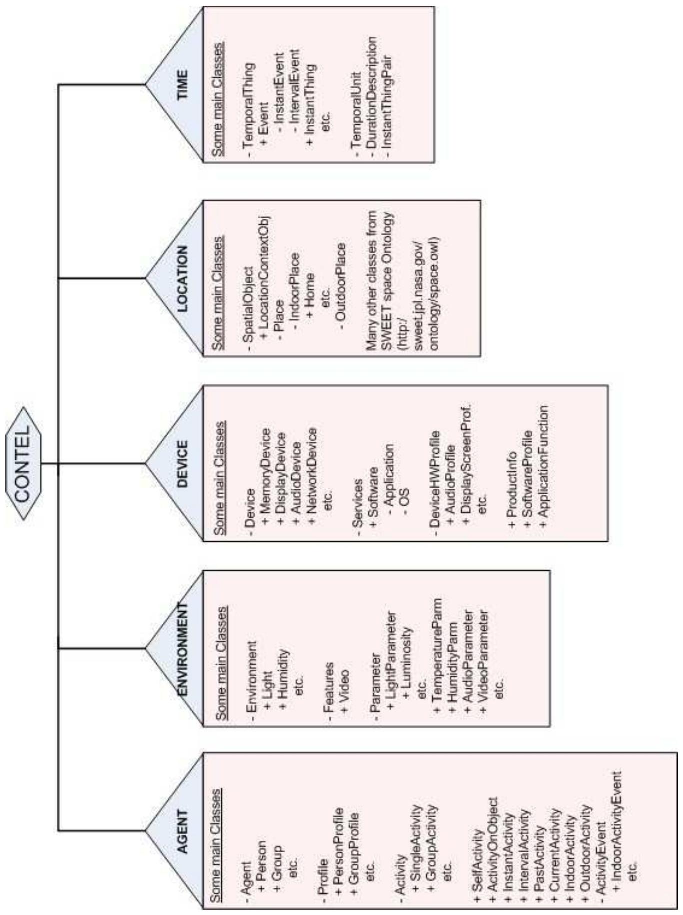

Fig. 2. Basic Categorizations and Domain Concepts in *Contel*

B. Formal Context Model and Semantics of Context

The middleware as well as the application should share the same context ontology for interchanging the information. In this regard, we use *Contel* [20] as the context ontology. *Contel* is extensible and as well as reusable for any context-aware system. Though more detail about *Contel* can be found in [20], we briefly present it here for completeness. *Contel* has categorized formal modeling of context into five top level concepts such as agents, environment, device, location and time. Fig. 2 describes a partial diagram of *Contel*. Here, the Agent class has been further classified into SoftwareAgent, Person, Organization, and Group. Each Agent has the property hasProfile associated with it whose range is AgentProfile. Also, an Agent isActorOf some Activity. Activity class, representing any Activity, can be classified based on the Actor of it e.g. SingleActivity (which has only one actor), groupActivity (which has Group as its actor and can have many SinlgeActivity instances). An Activity having some object of action on which it is done called ActivityOnObject like CookingDinner, TurnOnLight, or WatchingTV etc., while SelftActivity has no object of action e.g. Sleeping, or Bathing. The Device ontology in Contel is based on FIPA device ontology specification. The environmental context is provided by the various classes in the Environment ontology. Humidity, Sound, Light and Temperature are different environmental information we are utilizing in our framework. This sensed information is available though different sensors deployed in the smart environment, and used by the applications to adapt their behavior. Location ontology is extended from NASA Jet Propulsion Lab space ontology. We are also using the concepts from DAML-Time ontology for temporal context.

TABLE I
REQUEST/REPLY MESSAGE FORMAT

Type	Content
Request	CT, ET, EID, LD, UT
Reply	CT, ET, EID, V, MLD, MUT, RP

```
<contextType> temperature </contextType>
<Entity>
    <EntityType> room </EntityType>
    <EntityId> B340 </EntityId>
</Entity>
<leaseDuration> 3600 </leaseDuration>
<updateThreshold>0.5</updateThreshold>
```

Fig. 3. Request format in XML

C. Message Format

All the clients request for context information according to the context ontology. Clients specify (Table I) the type of context (e.g. Temperature) and as well as the entity of which this context is related to (e.g. Room). ”Lease Duration” and ”Update Threshold” are also to be specified if a client needs a context for a certain amount of time to avoid polling. Thus the Request message contains the following basic information: Context Type (CT), Entity Type (ET), Entity Id (EID), Lease Duration(LD) and Update Threshold (UT) (see Table I). If any client does not need periodical update notification, it should specify the ”Lease Duration” field and ”Update Threshold” field as zero. It should be noted that the middleware (context server) does not give lease to a client for more than a predefined maximum period (e.g. 5 hours) to block the delivery of context for an indefinite duration.

Now let’s consider an example where a smart assistant running on student’s PDA (client) may specify that it is interested to 0.5 degree Celsius change in the temperature of a room (e.g. B340) during the class period of one hour (Fig. 3 shows the XML format of this request). Here we can see that the client takes lease for the the temperature context for a long duration (e.g. one hour) to avoid polling. Consequently, it will certainly improve efficiency by saving valuable bandwidth when number of clients (e.g. students) is large.

To the contrary, the Reply message (see Fig. 4) contains CT, ET, EID, value (V), Maximum Lease Duration (MLD), Minimum Update Threshold (MUT) and Report Probability (RP). In our current prototype implementation (see Section V) the data type of the Value (V) field is a Java Object to accommodate any data type (e.g. Integer, Float, String etc.) and Java Serialization is used for transmitting the whole reply packet. When the size of the reply packet is large (e.g. compared to a smaller packet with just numerical value), bandwidth saving is significant. An example of large reply

```
<contextType> temperature </contextType>
<Entity>
    <EntityType> room </EntityType>
    <EntityId> B340 </EntityId>
</Entity>
<Value>25</Value>
<MaxLease>3600<MaxLease>
<MinUThreshold>0.5</MinUThreshold>
<ReportProbability>0.2</ReportProbability>
```

Fig. 4. Reply format in XML

packet may be a video frame of a particular location (e.g. class room) that is collected using a video camera mounted at the requested location.

It is notable that these formats of the request/reply messages are extendable for comprehensive representation. For example, we may include measurement unit information (e.g. second, Celsius etc.) for Value, LD, UT etc. fields in the messages. Investigation of such comprehensive message formats may be another direction of extending this work. However, in this work we only consider some basic information in the messages as described above.

D. Prioritization

To overcome the item prioritization problem as stated in section III-A, we at first categorize the requests into groups, in the form of a tree, depending on the context type (e.g. CT='temperature') and entity information (e.g. ET='Room', EID='B07'). These groups form the leaf nodes of our context request hierarchy (e.g. request tree). Motivation of using tree structure in storing requests is to make the searching faster. However, if multiple requests for the same context are received from the same client, the system keeps only one entry (e.g. most recent one) for that request in the request lists (see below for different request lists: TLR, TPR). This is because, the delivery of this data will satisfy the duplicate requests from the same client at the same time. Thus it also helps to prevent the *false popularity problem* that may happen if a client sends many duplicate requests to increase the popularity of a item of its interest. Polling may also cause the generation of these duplicate requests; but however, no duplicate request is stored in the request tree.

To facilitate the delivery mechanism described in this paper, the middleware maintains the following information for each of the groups in the request tree:

- Context Id (CID): A unique identifier is assigned to each group.
- Total Leased Request (TLR): Total number of leased requests for this specific context. This value is used in determining whether this data should be scheduled for multicast (hot item) or unicast (cold item).
- Leased Request List (LRL): This list contains the re-questers' ids (IP address) that have been leased along with their lease duration.
- Max Lease Duration (MLD): Maximum lease duration among the leased duration. This information is also sent along with the data to let the clients know how long this data will be delivered. If any client is receiving the data passively and wants to use longer than this time, it will renew the lease with longer period.
- Total Pending Request (TPR): This field denotes the total number of requests that have been received but no delivery of the context has been done yet. This field is reset to zero after each delivery of the context and incremented after receiving of each new request for this context. The larger the value of this field, the higher the priority of delivery of this context should be.
- Pending Request List (PRL): This list is similar to LRL but contains the re-questers' ids (IP address) and requested lease duration of the pending re-quests. After the delivery, the requests with lease duration greater than zero will be added to the LRL and TLR will be updated accordingly.
- First Arrival Time (FAT): This is the arrival time of the first request which is still pending.
- Longest waiting time (LWT) of any pending request for this context is the difference of current time and FAT. LWT is used to determine the priority of delivering this context together with TPR. FAT is reset to zero after each delivery and set to the arrival time of the first request as it is queued. The larger the value of this field, the higher the priority of delivery of this context should be.
- Last Delivery Time (LDT): The most recent time when the context was delivered. This is used to calculate the longest waiting time of the leased re-quests in LRL.
- Min Update Threshold (MUT): The minimum of update threshold values among the requests. If the context is changed by this amount, it is then scheduled for delivery.
- Candidate for Scheduling (CS): This is a binary value. If the context change exceeds the threshold MUT, the value of this field becomes one (true) and implies this data to be delivered. This field becomes zero (false) with the next delivery of the context data.
- Last Update Time (LUT): This field denotes the time of last update of this context data.
- Frequency of Update (FUT): The frequency of update of this context data.
- Value (V): The current updated value of this context. This field may contain any kind of data (i.e. character, string, double, integer, boolean etc).
- Size (S): Size of this data item.

To set the priorities of the requested context data, we use the total number of requests (R) and longest waiting time of the outstanding request (W) for that item and it is motivated by algorithm [17]. In RxW algorithm, the item with higher R*W value has higher priority. Thus we prioritize a data either because it is very popular or because it has at least one long-outstanding request. We consider both Total Pending Request (TPR) and Total Leased Request (TLR) when CS is one (true) to be the total number of request (R), otherwise we only consider TPR to be the R value for the context item (see Equation 1). This is because TLR comes into account as soon as the amount of change exceeds Min Update threshold (MUT) and CS becomes one (true). Similarly, as long as CS is zero, difference of current time (CT) and FAT (First Arrival Time) is the value of longest waiting time (W); but as soon as CS becomes one, the longest wait time (W) is the difference of CT and LDT (Last Delivery Time) as all the leased requests have been waiting since LDT. Hence we define with the following equation:

$$\begin{aligned} RxW = {} & (TPR + CS * TLR)* \\ & ((CT - FAT)(1 - CS) + CS(CT - LDT)) \end{aligned} \tag{1}$$

We calculate value of each group (data item) and sort them in descending order. We update the list each time a request comes and use the same data structure proposed in [17] for efficient maintenance of such list. The RxW value of ith group is denoted as popularity (or priority) p_i in the following sections.

E. Bandwidth Division

The motivation of bandwidth division comes from the fact that the average latency (L) of a data item is less when HOT items are assigned to push, COLD items to unicast pull and the bandwidth is divided appropriately between the two channels. We use the bandwidth division algorithm based on the prioritization described in section III-D rather than using the prioritization based on request rate only as it is used in [14]. The bandwidth division algorithm uses the sorted list of items with decreasing order of popularity, i.e. $p_i \geq p_{i+1}, (1 \leq i \leq n)$, where n is the current size of the list. It is intuitive that if item i is pushed, then $j \leq i$ should also be pushed. So, the algorithm tries to partition the list at index k such that the push set $1, 2, \ldots, k$ minimizes the latency L given a certain bandwidth B and *pull over-provisioning factor* $\alpha > 1$. The *pull over-provisioning factor* denotes the actual bandwidth we reserve for pull is α times what an idealized estimate predicts and queuing theory asserts that $\alpha > 1$ guarantees bounded queuing delays [14]. The optimal value k^* is found by trying all possible values of L and finally the algorithm determines the pull bandwidth $\alpha \sum_{i=k+1}^{n} \lambda p_i S_i$, which leaves bandwidth $pushBW = B - \alpha \sum_{i=k+1}^{n} \lambda p_i S_i$ for the push channel and average latency for the pushed documents is then $\sum_{i=1}^{k} \frac{S_i}{2pushBW}$. Here λ and S_i denote request rate and size of the item respectively. The algorithm runs in $O(n)$ as it performs binary search over all possible values of L and maintains an internal array that stores the total size of each possible partition using binary tree techniques [14].

F. Push Popularity Problem

As the HOT items are delivered through multicast, some clients will use this data without sending explicit request. Thus middleware may not know the actual number of clients that are using an item. Hence, middleware will be misguided to lower the priority of an item even though that is being used by a large number of clients. This problem is known as *push popularity problem* [14].

However, we can not expect to solve this push popularity problem completely as it will require all the clients to send requests explicitly and hinder the benefit of multicast. So, a portion of the passive clients should send requests even though the data is ensured to be delivered. The middleware sends a report probability (RP) with the data and a passive client submits an explicit request for this data with probability RP. It is proved in [14] that RP should be set inversely proportional to the predicted access probability for that data and the equation to calculate RP is :

$$RP_i = \begin{cases} \frac{\beta}{\lambda p_i k} & \text{if } \lambda p_i k > \beta \\ 0.2 & \text{otherwise} \end{cases} \tag{2}$$

Where β is the difference of Maximum acceptable TCP connection and request arrival rate, λ denotes aggregate request rate and p_i denotes the priority (or popularity) of group i based on total request and k denotes the current number of multicast items. Here we notice that if $\lambda p_i k > \beta$, the probability will exceed one and hence we specified RP to be 0.2 as a default. It should also be noted that whenever the client sends a request, it sends a complete request with its desired update threshold (UT) and desired lease extension (LE). LE denotes the desired extension after the expiry of current MLD.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Total Client	3000
Total Item, N	50
Size of Each Item	200 bytes
Zipf Parameter θ	15
System Bandwidth	512,000 bits/sec
Exponential mean M	12
α	2
ϵ	0.005
Lease Duration of a Client	10 $\sim$ 100 ms (uniform)
Lease Renewal Probability	0.7
Update Threshold	0.5 $\sim$ 2 unit (uniform)

IV. EVALUATION

In order to establish the potential of our proposed context delivery mechanism, we have built a simulation model of the proposed system and evaluated using the simulation tool OMNET++ [29]. All the graphs presented here are generated using the *Plov* tool of OMNET++.

A. Simulation Model

In our client-server model the server (our middleware) acts as a data server and delivers self identifying context data items of equal size either by multicast or unicast upon explicit request. The clients request an item according to Zipf distribution [30] and the time of requests is exponentially distributed with mean, where is the average request rate of each client. We present the analysis of average latency and network traffic of the proposed system in the following subsections. Table II presents all the simulation parameters. Here the *pull over-provisioning factor* α and the *tolerance factor* ϵ are used by the bandwidth division algorithm described in [14].

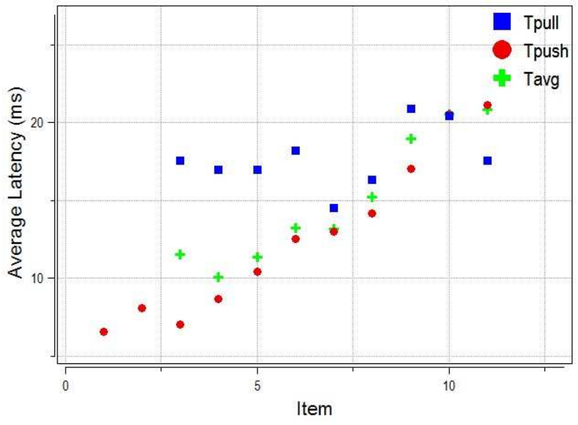

Fig. 5. Relation of average latency of Push and Pull as the number of multicast items changes according to our experiments. Here the intersection of and occurs at and before k=3, grows arbitrarily large.

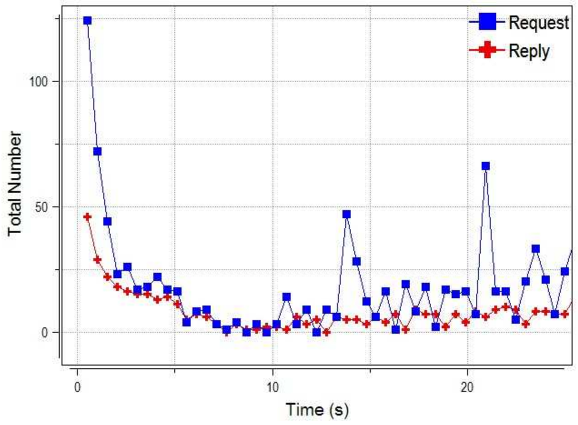

Fig. 6. Number of request and reply with change of time. The number of reply denotes total of multicast and unicast items.

B. Average Latency

Let $G(k)$ be the average latency (T_{avg}) if the k most popular items are multicast. The function $G(k)$ is a weighted average of the average latency of pushed items $T_{push} = \sum_{i=1}^{k} \frac{S_i}{2pushBW}$ and the average latency for the pulled items $T_{pull} = \frac{1}{\mu - \left(\sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{k} \lambda_i\right)}$, where λ_i is the Poisson request rate for each item i [13], [14]. Our result is shown in Fig. 5. Notice that the minimum of $G(k)$ is to the left of the intersection (at k=10) of the push and pull curves though theoretically it should be on the right side of the intersection [13]. The minimum of $G(k)$ occurs at a relatively small value of k and precedes the intersection due to two complementary reasons. First, the most popular items are chosen for push and are also those to which a Zipf distribution gives substantially more weight. So, if an item is delivered using multicast, it will also have the largest impact on the globally average delays. As the numbers of the most popular items are small and are multicast first, the overall minimum delay occurs for small values of k . Second, pull delays are actually minimized at the points $k^{'}$ where the pull-curve flattens out. However $k^{'}$ precedes the intersection in our graph, and so the overall minimum occurs before that intersection.

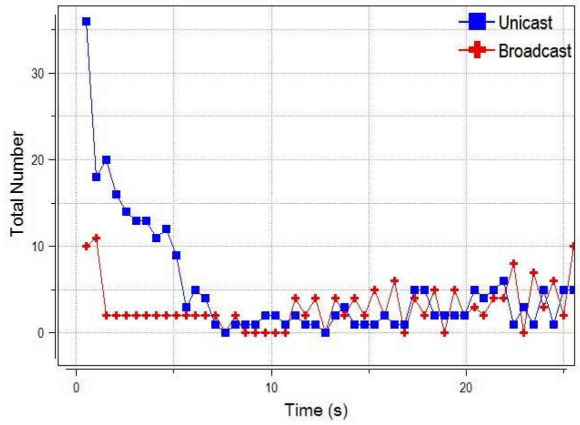

Fig. 7. Number of multicast and Unicast items with change of time. According to our approach, some of the requested items are multicast while the remaining items are delivered by unicast. Total numbers of multicast and unicast replies are shown in Fig. 6

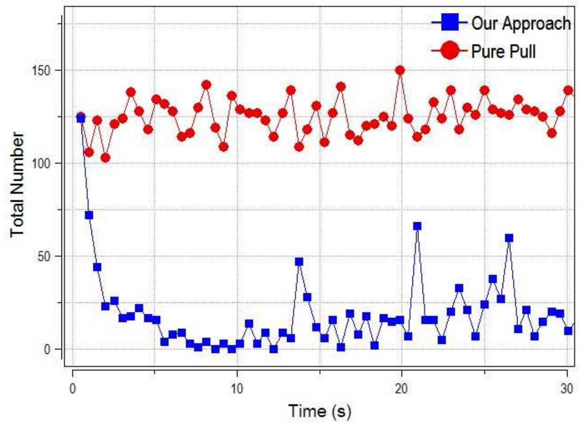

Fig. 8. Number of requests in our approach and in pure pull approach. Simulation results show that our approach causes fewer requests as it avoids polling and uses lease mechanism.

C. Network Traffic

Fig. 6 presents our simulation result regarding network traffic. Here we can see that in the beginning of time, the number of request is high. But as the server starts to deliver items, the number of request decreases due to two reasons. First, the replies contain the maximum lease period and minimum threshold value for the context items and the clients do not need to send explicit request again until the lease expires. Second, as the most popular items are multicast, the clients that also need the data do not send request but uses the data passively. But we can see some spikes in the request graph because of the lease renewal requests and the requests sent by the passive clients due to *Report Probability* as we have already discussed. Here we see that incorporation of lease mechanism and threshold reduces overall network traffic from client as well as from server. Besides, the lease renewal and Report Probability cause the generation of necessary traffic from clients to determine the hot and cold item at server. Fig. 7 shows the number of multicast and unicast items with the change of time.

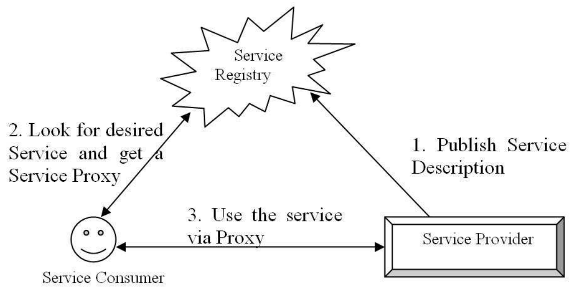
Service Registry
2. Look for desired Service and get a Service Proxy
1. Publish Service Description
3. Use the service via Proxy
Service Provider
Service Consumer

Fig. 9. Service Oriented Architecture

V. Implementation of Prototype

Now, as our proposed mechanism shows convincing performance in a simulated environment, we engage in developing the prototype of context delivery module for our middleware CAMUS [21], [4].

We design the context delivery module (CDM) as a middleware service based on Service Oriented Architecture (SOA) [31]. The SOA interaction comprise of a service provider, a service consumer (or, client) and a registry (Fig. 9). A service consumer look for a service provider using the registry. A *service lease* specifies the amount of time or contract for which the interaction with the service is valid. The service provider supplies a service proxy to the service consumer and the service consumer executes the request by calling an API function on the proxy. Then the proxy formats the request message and executes that on behalf of the consumer. Figure 9 shows a typical setup of a service oriented system.

Motivation of using SOA for our prototype is to enable dynamic discovery of the *Context Delivery Service* of the middleware. For example, if the contextual information is provided through a static address such as URL or IP address, all clients are to be pre-configured to operate using that address. Moreover, the users are also to be notified whenever the address is changed. But the problem of notifying all the users about the changed address is very challenging. However, SOA rescues us from this problem as the applications (or, clients) may themselves discover the context delivery service from the well known service registry such as Jini Lookup service [18]. In our implementation, we provide context delivery module as a Jini service [18] assuming that clients will be able to lookup for this context delivery service using the Jini Lookup.

A. Architecture and System Work Flow

The main entities of context delivery module are *Jini Service Interface*, *Context Delivery Manager*, *Request Queue*, *Schedule Manager*, *Priority Calculator*, *Bandwidth Allocator* and *Dissemination Manager* (Fig. 10). The data flow among these modules is described below.

Context Delivery Module publishes a *Jini Service Interface* through which clients submit their requests. Clients at first discover this interface using Jini Lookup service [18] and get a proxy of this interface. This proxy, on behalf of the clients, perform all the task of remote procedure call (RPC) to handover the requests to *Context Delivery Manager*. *Context Delivery Manager* controls the dataflow among various modules. As

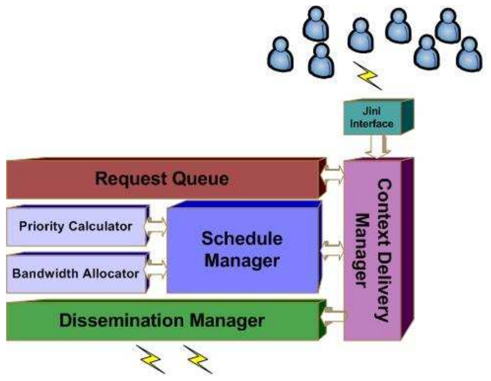

Fig. 10. Architecture of Context Delivery Module

soon as *Context Delivery Manager* gets a request from *Jini Interface*, it enqueues that request into the *Request Queue*. *Request Queue* maintains a tree like data structure to store the requests. Tree like structure helps to optimize the searching and grouping of requests. *Schedule Manager* prioritizes the requests using the *Priority Calculator module* and *Bandwidth Allocator* allocates the requests to be delivered by multicast or unicast based on available bandwidth. Then the actual delivery is performed by the *Dissemination Manager*. *Dissemination Manager* has the ability to deliver the context using multicast channel or using unicast. In our implementation, *Java Reliable Multicast Service (JRMS)* library [32] is used for reliable multicast delivery.

VI. Conclusion

In this paper we present a scalable context delivery mechanism for context-aware middlewares based on hybrid data dissemination technique where the most requested data are multicast and the rest are delivered through unicast to reduce network traffic. Our mechanism dynamically prioritizes and classifies the hot and cold context data depending on the request rate and longest waiting time. We further perform division of bandwidth depending on the hot and cold items to reduce average latency dynamically. Our solution also addresses the push popularity problem that occurs as the passive client access data without sending explicit requests. We incorporate the leasing mechanism to reduce the periodical requests (polling) for better performance. We further describe the implementation detail of the prototype using Jini framework [18] and *Java Reliable Multicast Service (JRMS)* library [32].

There is a lot of interesting works to be done in the near future for efficient context delivery. We plan to investigate indexing scheme, cache invalidation report (IR) scheme, real time delivery, predictive multicast of context and secure delivery of context in future.

Acknowledgment

Authors would like to thank Kamrul Hasan, Yllias Chali for their valuable suggestions. Moreover, authors are also grateful to the anonymous reviewers for their useful comments to improve the quality of this paper. This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement), IITA-2006-(C1090-0602-0002).

References

[1] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *Proc. of the 2000 Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, April 2000.

[2] M. Weiser, "The computer for the 21st century," *Scientific America*, pp. 94–104, Sept 1991.

[3] A. Ranganathan and R. H. Campbell, "A middleware for context-aware agents in ubiquitous computing environments," in *ACM/IFIP/USENIX International Middleware Conference*, Brazil, June 2003.

[4] A. Shehzad, N. Q. Hung, K. A. Pham, M. Riaz, S. L. Kiani, S. Y. Lee, and Y. K. Lee, "Middleware infrastructure for context-aware ubiquitous computing systems," Kyung Hee University, Seoul, Korea, CAMUS Technical Report TR-V3.2, February 2005. [Online]. Available: http://oslab.khu.ac.kr/camus

[5] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project aura: Toward distraction-free pervasive computing," *IEEE Pervasive Computing*, April-June 2002.

[6] J. I. Hong and J. A. Landay, "An infrastructure approach to context-aware computing," *HCI Journal*, vol. 16, 2001.

[7] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *Proc. of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, Pittsburgh, PA, May 15-20.

[8] A. Ranganathan and R. H. Campbell, "A middleware for context-aware agents in ubiquitous computing environments," in *Proc. of ACM/IFIP/USENIX International Middleware Conference*, Brazil, June 2003.

[9] B. Johanson, A. Fox, and T. Winograd, "The interactive workspaces project: Experiences with ubiquitous computing rooms," *IEEE Pervasive Computing*, 2002.

[10] S. K. Das, D. J. Cook, A. Bhattacharya, E. O. Heierman-III, and T. Y. Lin, "The role of prediction algorithms in the mavhome smart home architecture," *IEEE Wireless Communications Special Issue on Smart Homes*, vol. 9, pp. 77–84, 2002.

[11] G. Chen and D. Kotz, "Solar: An open platform for context-aware mobile applications," in *Proc. of the First International Conference on Pervasive Computing (Pervasive 2002)*, Switzerland, June 2002.

[12] S. Acharya, M. Franklin, and S. Zdonik, "Balancing push and pull data broadcast," *ACM SIGMOD*, May 1997.

[13] K. Stanthatos, N. Roussopoulos, and J. S. Baras, "Adaptive data broadcast in hybrid networks," in *Proc. of the 23rd International Conf. on VLDB*, September 1997, pp. 326–335.

[14] J. Beaver, N. Morsillo, K. Pruhs, and P. K. Chrysanthis, "Scalable dissemination: What's hot and what's not," in *Proc. of the Seventh International Workshop on the Web and Databases (WebDB 2004)*, Paris, France, June 17-18 2004.

[15] D. Gifford, "Polychannel systems for mass digital communications," *Communications of ACM*, vol. 37, October 1994.

[16] P. Triantafillou, R. Harpantidou, and M. Paterakis, "High performance data broadcasting systems," *Mobile Networks and Applications*, pp. 279–290, July 2002.

[17] D. Aksoy and M. Franklin, "RxW: A scheduling approach for large-scale on-demand data broadcast," *IEEE/ACM Transactions On Networking*, vol. 7, pp. 846–860, December 1999.

[18] Sun microsystems, inc.: Jinitm architecture specification. [Online]. Available: http://www.sun.com/jini/specs/

[19] S. S. Yau, D. Chandrasekar, and D. Huang, "An adaptive, lightweight and energy-efficient context discovery protocol for ubiquitous computing environments," in *Proc. of 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, 2004.

[20] A. Shehzad, N. Q. Hung, K. A. Pham, and S. Y. Lee, "Formal modeling in context aware systems," in *Proc. of First Workshop on Modeling and Retrieval of Context (MRC'04)*, vol. 114. Germany: CEUR, 2004.

[21] N. Q. Hung, A. Shehzad, S. L. Kiani, M. Riaz, and S. Y. Lee, "Developing context-aware ubiquitous computing systems with a unified middleware framework," in *Proc. of Embedded and Ubiquitous Computing(EUC 2004)*, vol. LNCS 3207. Springer-Verlag, 2004, pp. 672–681.

[22] L. Mehedy, M. K. Hasan, Y. Lee, S. Y. Lee, and S. M. Han, "Hybrid dissemination based scalable and adaptive context delivery for ubiquitous computing," in *Proc. of the 2006 IFIP International Conference on Embedded And Ubiquitous Computing (EUC 2006)*, vol. LNCS 4096. Seoul, Korea: Springer-Verlag, August 01-04 2006, pp. 987–996.

[23] J. Heer, A. Newberger, C. Beckmann, and J. I. Hong, "liquid: Context-aware distributed queries," in *Proc. of the Fifth International Conference on Ubiquitous Computing: Ubicomp 2003*. Seattle, WA: Springer-Verlag, 2003, pp. 140–148.

[24] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications," in *Proc. of the IEEE International Conference on Pervasive Computing (PERCOM)*, Dallas, March 23-25 2003.
[25] G. Chen and D. Kotz, "Context aggregation and dissemination in ubiquitous computing systems," in *Proc. of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, June 2002.
[26] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1, pp. 33–40, July-September 2002.
[27] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 8, June 2000.
[28] D. Barbara and T. Imielinski, "Sleeper and workaholics: Caching strategy in mobile environments," in *Proc. of ACM SIGMOD Conference Management of Data*, 1994, pp. 1–12.
[29] Omnet++. [Online]. Available: http://www.omnetpp.org/index.php
[30] Wentian li, references on zipf's law. [Online]. Available: http://www.nslij-genetics.org/wli/zipf/
[31] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington, "Implementations of a service-oriented architecture on top of jini, jxta and ogsa," in *Proc. of the UK e-Science Program All Hands Meeting 2003*, Nottingham, UK, September 2003.
[32] P. Rosenzweig, M. Kadansky, and S. Hanna, "The java reliable multicast service: A reliable multicast library," Sun Microsystems, Tech. Rep. SMLI TR-98-68, 1998.

Lenin Mehedy received the B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in November 2004. Since March 2005, he has been pursuing the M.S degree in Computer Engineering at Kyung Hee University, Korea. His research interests include middleware for Ubiquitous Computing, Graph Theory and Pervasive Networks.

Sungyoung Lee received his B.S. from Korea University, Seoul, Korea. He got his M.S. and PhD degrees in Computer Science from Illinois Institute of Technology (IIT), Chicago, Illinois, USA in 1987 and 1991 respectively. He has been a professor in the Department of Computer Engineering, Kyung Hee University, Korea since 2001. Before joining Kyung Hee University as an assistant professor in 1993, he was an assistant professor in the Department of Computer Science, Governors State University, Illinois, USA. His current research interests include Ubiquitous Computing Middlewares, Operating Systems, Real-Time Systems and Embedded Systems. He is a member of the ACM and IEEE

Salahuddin Muhammad Salim Zabir got his PhD and MS degrees in Information Science from Tohoku University, Japan and MSc and BSc Engineering in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET). He is currently working as a visiting professor at the Kyung Hee University (KHU), Korea. He also worked as a faculty in Tohoku University, Japan and Bangladesh University of Engineering and Technology. Prior to joining KHU, he was working for the R&D efforts of Panasonic, in Matsushita Electric Industrial Company, Japan. Prof. Zabir maintains a good record of publications and patents in the fields of computer networking and ubiquitous computing. He has been serving on the technical and/or program committees of several conferences as well as guest editing journal special issues. He is a member of the IEEE, BCS and IEB.

Young-Koo Lee got his B.S., M.S. and PhD in Computer Science from Korea Advanced Institute of Science and Technology, Korea. He is a professor in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests include ubiquitous data management, data mining, and databases. He is a member of the IEEE, the IEEE Computer Society, and the ACM.