Aggregation Based Information Collection for Mobile Environments

Qi Han and Nalini Venkatasubramanian
Department of Information and Computer Science
University of California-Irvine, Irvine, CA 92697-3425
{qhan,nalini}@ics.uci.edu

Abstract: In the future, we are likely to see a tremendous rise in mobile computing and communications as ubiquitous applications incorporate multimedia information. These mobile multimedia applications will have QoS requirements; resource provisioning algorithms utilize current system resource availability information to ensure that these applications meet their QoS requirements. Information collection algorithms collect and maintain current system resource information and are vital in performing efficient resource provisioning. In this paper, we present a novel information collection technique for mobile environments - the ABIC (aggregation-based information collection) algorithm. This algorithm derives aggregate mobility from individual user mobility, then utilizes the aggregation to drive the information collection process and adjust the related parameters in the process. A feedback loop allows the information collection process to utilize feedback from the resource provisioning process in further decision making. We compare the ABIC approach to other proposed information collection algorithms under different workloads and mobility patterns. Our experimental results show that the aggregation based information collection algorithm exhibits superior performance under most mobility criteria and request patterns.

1 Introduction

Wireless communication has exhibited tremendous growth in recent years. This has enabled ubiquitous data transfer; mobile multimedia applications such as multimedia conferencing, video streaming and image transfer are becoming increasingly popular. Achieving such an advanced level of tetherless mobile multimedia services requires (1)the development of a wireless network that supports the integrated multimedia services; and (2) the development of agile network management middleware services that can ensure Quality of Service (QoS) to mobile multimedia applications. Prior work in mobile networking has focused on protocols for mobility management for various types of wireless architectures such as Public Land Mobile Networks (PLMN), Mobile Internet Protocol (Mobile IP) networks, Wireless ATM (WATM) networks and Low Earth Orbit (LEO) Satellite networks [20]. Our objective is to develop a middleware infrastructure that can provide mobility and QoS to applications independent of the underlying specific network architecture. Such a middleware infrastructure will provide efficient resource provisioning that will optimally allocate resources to applications under dynamically varying system conditions. In this paper, we develop an information collection technique for mobile environments that will support increasing number of mobile hosts with QoS and decreasing management overhead.

There exists an inherent tradeoff between information accuracy and system performance. More accurate information leads to better QoS provisioning, however, higher overhead is introduced to maintain the accuracy. Several solutions have been suggested for information collection in non-mobile environments [6, 9]. However, these techniques might not be appropriate for mobile environments for the following reasons. Firstly, in mobile environments, changes in network states are likely to occur more frequently due to increased dynamicity. Furthermore, in mobile environments, as clients keep moving around, their access points to the wired networks might change. The middleware framework must keep track of current locations of clients so that it can deliver the services needed at appropriate quality levels.

In this paper, we take advantage of the mobility patterns of end hosts to better serve information collection, and hence resource provisioning. Traditionally, location information from each user is used to determine the mobility patterns. However, this entails significant overhead to maintain accurate location information for mobile hosts. We propose a novel technique for information collection in mobile environments called Aggregation Based Information Collection (ABIC). This technique uses information about the population of mobile hosts in a region, i.e., the aggregation status to drive the information collection process. We expect the ABIC process to provide reasonable level of accuracy with significantly less overhead, enabling efficient QoS support.

The rest of this paper is organized as follows. Section 2 describes the overall architecture of the system with a focus on the middleware components. In Section 3, we describe the basic issues in information collection for dynamic environments and highlight proposed solutions. We propose the aggregation-based information collection (ABIC) algorithm in Section 4 and describe the two phases of the algorithm. We present a comparative performance evaluation of the ABIC algorithm with other proposed solutions in Section 5 and analyze the obtained results. We discuss related work in Section 6 and conclude with future research directions.

2 System Architecture

Figure 1 depicts an information collection architecture suited for highly dynamic environments. The information collection framework consists of three components:

- *Information Source:* This corresponds to the managed entity, such as the server, link or mobile host. In our system, we use the directory service to hold system state information about information sources. This state information includes network parameters(such as residual link bandwidth, end-to-end delay on links etc.), server parameters (such as CPU utilization, buffer capacity, disk bandwidth, etc), and mobile host parameters (such as mobile host location, connectivity, power level etc.).

- *Information Consumer:* This module consumes data collected from the information sources (stored in the directory service) for application and system level tasks. For instance, resource provisioning modules consume information about network and system status to perform admission

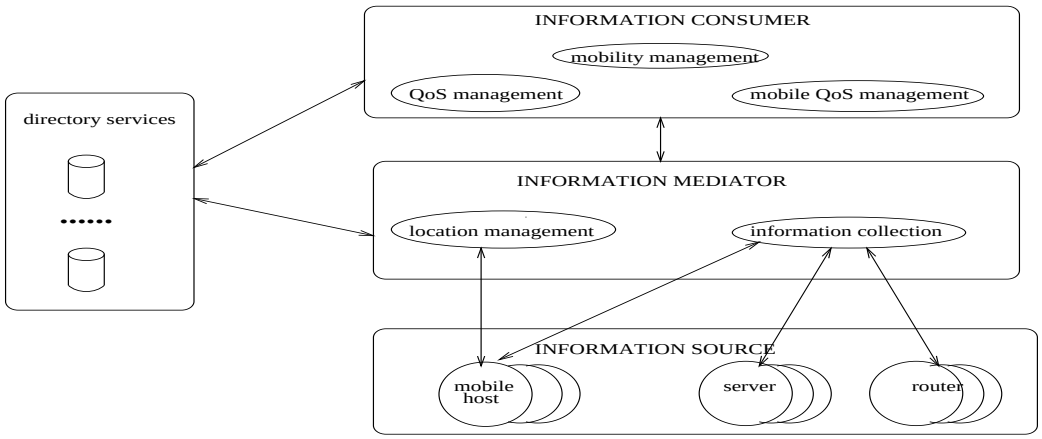

Figure 1: Information Collection System Architecture

control and resource allocation.

- *Information Mediator:* This module serves as the decision point of the information collection. It listens to notifications from sources or consumers and invokes suitable actions so that the directory service maintains information at a suitable level of accuracy satisfactory to the consumers.

In order to make decisions about the collection process, the information mediator frequently probes the managed entities (i.e., information source). There exists a tradeoff between the accuracy of information maintained in the directory service and the overhead required to collect the information. The objective of information collection algorithms implemented within the information mediator is to balance this tradeoff, i.e., maintain the highest possible accuracy with the least possible overhead. In highly mobile environments, constant movement of information sources causes the current system status to change rapidly; hence information is quickly outdated; it also causes certain regions to become hot spots. In mobile environments, we can characterize two types of hot spots: (1) services/data hot spots, i.e., data centers/servers which contain popular data that is frequently accessed. (2) mobile host hot spots where there is an aggregation of mobile hosts. e.g. due to the occurrence of periodic trends and seasonal variations. We define the first category of hot spots as request aggregation and the second category as mobile host aggregation. Request aggregation exists in both mobile and non-mobile environments. Solutions to manage request aggregation in non-mobile environments have been studied in the context of load-balancing. In this paper, we focus on using mobile host aggregation to derive efficient information collection algorithms for mobile environments.

To represent mobile host aggregation, we must choose a suitable mobility model. In general, the mobility models characterize host mobility using the speed, direction, or movement history of the mobile users. There has been extensive work on mobility models used to describe individual user

movement behavior, these models do not consider the collective motion of all the mobiles relative to a geographical area (region) over time. Our objective is to provide an aggregate representation that quantifies how crowded a region is at any point in time. The gravity model [12] has been used extensively in transportation research to model human movement behavior. In this model, the traffic volume between two regions are symmetric, this implies that the population in each region remains constant. This is not characteristic of existing mobile environments. In this paper, we use the incremental mobility model [8] to describe individual host mobility. We assume mobile hosts are distributed in a closed coverage area which is divided into many non-overlapping regions . Such region has a collection point (e.g. base station) that serves as the wired network access point for all the mobile hosts in its controlled region. Mobile hosts either move or stop in this area, the population in each region changes all the time and can be captured. This aggregation is what we are interested in.

For our performance study, we focus on a specific resource provisioning technique- CPSS (Combined Path and Server Selection). CPSS deals with path and server selection in a unified way, allows load balancing not only among replicated servers, but also among network links to maximize the request success ratio and system throughput [7]. However, our algorithm is not limited to CPSS and it can also be applied to other resource provisioning processes in a similar way.

3 Information Collection Approaches

In this section, we describe previous approaches to the information collection problem. These approaches maintain system state information within a directory service using either an instantaneous value or a range-based representation. The policies also differ in the degree of dynamicity with which current state information is obtained and updated. Much of this work has been developed in the context of system and network management, i.e., monitoring network and server status. Prior solutions can be classified into one of the following three categories:

- *Instantaneous Snapshot Based Information Collection(SS) [16]*: In this policy, information about the desired parameters (e.g. residue capacity of network nodes and server nodes) is based on an absolute value obtained from a periodic snapshot. During each sampling period, probing is initiated to gather the current information from managed entities (e.g. router nodes); the information repository (e.g. the directory service) is updated with the collected values.

  The sampling period solely determines the accuracy of the information stored in the repository. In highly dynamic traffic, the monitoring module has to sample at a very high frequency to prevent information from being outdated. Obviously, a shorter sampling period causes information in the directory service to be more accurate, resulting in better resource provisioning performance. However, a high overhead is incurred due to the frequent sampling. Our studies indicate that for a variety of traffic conditions, a sampling period of 10 seconds yields the best overall price-performance tradeoff.

- *Static Range Based Information Collection (SR) [1]*: In this policy, we define a fixed interval B

which is used to partition the capacity of the collected information into a fixed number (say n) of equal size classes: $(0, B), (B, 2B), (2B, 3B), ..., ((n-2)B, (n-1)B)$. The classes are represented by corresponding indices $0, 1, 2, ...(n-1)$. A probe is initiated at each sampling interval to obtain current information from the managed entities. If the obtained value is out of the range indicated by current index, the repository is updated with another index, otherwise no update is needed.

Studies show that with frequent sampling, a smaller interval brings better resource provisioning performance than a bigger interval. For a large sampling period, the interval based policy is attractive because it is natural to represent a residual value using a range (instead of an instantaneous value that remains constant over a long period of time). For a larger sampling period, a bigger interval brings better provisioning performance. The reason is that when the sampling period is short, representing resource availability using a big interval introduces information inaccuracy, a smaller interval is better. However, when the sampling period is very long, resource availability represented using a small interval is quickly out-dated, resulting in more inaccurate system state information.

- *Dynamic Range Based Information Collection (DR) [9]*: In this policy, the information repository holds the the monitored parameter using a range with an upper bound U and a lower bound L; the range may be modified dynamically based on the sampled information. The monitoring module sends out probes periodically according to an initial probing period to collect status information. If the sampled value falls within the current range for a specified time period, we tighten the range thereby enhancing accuracy. Otherwise, we relax the current range to hold the enlarged values observed during the previous sampling period. In previous work, we have developed dynamic range based policies using (a) a simple throttle based [9] and (b) a more complex analytical time series based technique [6]. In the throttle based algorithm, we use the average value of samples in previous monitoring window to decide whether a range adjustment is needed or not. If deemed necessary, the range may be increased or decreased exponentially using a pre-specified throttle factor. Our work shows that a throttle factor of 0.25 works best under most cases. In the time series based approach, we first use statistical analysis techniques based on time-series to derive a range such that the deviation between the predicted and observed values remains in the range with a given confidence level. Based on the size of the range and the confidence level, we determine a bound on the sampling rate. We then dynamically adjust the range as well as the sampling rate based on the burstiness of the incoming traffic.

  Studies show that DR yields better range accuracy as compared to SR, brings uniformly better provisioning performance and introduces lower update overhead. In comparing the two DR policies, we observe that although the time series based approach analytically shows better accuracy than the throttle based approach, it does not lead to better provisioning performance. The reason is that although the network traffic exhibits self-similarity, there still exist bursts, hence it is almost impossible to have a very accurate model to characterize and forecast network traffic. The time series approach uses the inaccurate model to derive the sampling period and

residual link bandwidth, thereby aggravating the inaccuracy. Using prediction is always risky, so we believe the time series based approach could be superior to the other approaches sometimes, but not always. In contrast, the throttle based approach adapts reasonably well to the constantly changing environment.

Our prior work has also studied the impact of combining the information collection policies with resource provisioning techniques under varying network/server conditions and different application workloads [9]. The performance results indicate that the dynamic range based techniques (especially the throttle based approach) outperforms the others in most cases. However, all these approaches do not explicitly model host mobility. A good solution to managing resources in mobile environments must consider host mobility information. Location management techniques have been proposed to monitor mobile host movement. An efficient QoS-provisioning technique for mobile environments must compose location management with traditional information collection techniques in a cost-effective manner. This makes the information collection problem more complex.

4 Aggregation-Based Information Collection (ABIC)

Solutions proposed for information collection in non-mobile environments do not perform well when applied to mobile environments. First, the traditional approaches to network monitoring use periodic probes. The sampling frequency is the same for all the managed entities and is kept constant during the whole monitoring process. This is impractical for mobile environments because users are constantly moving, some regions are more crowded than others, thus causing significant variation in the resource availability. Second, range-based approaches (where each monitored parameters is represented by an interval with a lower bound and an upper bound) do not consider consumer accuracy needs. The range is adjusted only based on the network status. Third, no user mobility is considered. In this section, we propose a novel approach to information collection in the presence of mobile hosts called ABIC (Aggregation Based Information Collection). As the name suggests, ABIC uses mobile host aggregation to determine the frequency and accuracy of the collected data.

Before going into the details of the algorithm, we first define the terminology and notation used in describing the ABIC algorithm.

- Aggregation $A(t)$: It is the number of mobile hosts in the specified region at certain time period t.

- Range-based Representation: A parameter V is bounded by a range R with an upper bound U and a lower bound L with V uniformly distributed in the range $U-L$. Each monitored parameter is stored in the directory service using a range-based representation.

- Range Precision: Given the range $R < L, U >$, the precision of the range is inversely proportional to the range size, i.e., $Prec(R) \propto \frac{1}{R}$.

- Sampling Frequency (SF): It represents the periodicity with which probes are sent out to managed entities to query the current status of the collection parameters.
- Utilization Factor (UF): It is the percentage of occupied resources, i.e., the ratio of resources used to the resource capacity. The capacity of a server can be specified using four parameters [17]: CPU cycles, memory buffers, I/O bandwidth and network transfer bandwidth

$$< CPU_{max}, BUF_{max}, DB_{max}, X_{max} >.$$

The occupied server resources at given time t can be modeled as

$$< CPU^s_{used}(t), BUF^s_{used}(t), DB^s_{used}(t), X^s_{used}(t) >.$$

The utilization factor (UF) for a server s, at given time t, is defined as

$$UF^s(t) = max(\frac{CPU^s_{used}(t)}{CPU^s_{max}}, \frac{BUF^s_{used}(t)}{BUF^s_{max}}, \frac{DB^s_{used}(t)}{DB^s_{max}}, \frac{X^s_{used}(t)}{X^s_{max}}).$$

The UF for a link l at given time t with link capacity BW^l_{max} is defined as

$$UF^l(t) = \frac{BW^l_{used}(t)}{BW^l_{max}}.$$

The goal of the ABIC algorithm is to find an efficient way to adjust collection parameters so that (1) the collection process will not cause network congestion and (2) desired information accuracy is maintained. In other words, the algorithm should minimize the information collection cost (or network traffic caused by information collection process) while still maintaining reasonable information precision that can satisfy user requirements. ABIC combines periodic probing with *source and consumer-initiated triggers* that allow for further customization of the information collection process. Every source (managed entity) maintains the current exact value and the approximation held in the directory service. When the current value changes unexpectedly and falls outside the current range, a *source-initiated trigger* is issued to the information mediator for furthur action. The feedback from the information consumer may indicate that the current range is not accurate enough to make decisions, this causes a *consumer-initiated trigger*. The information mediator responds by collecting more accurate information.

To begin with, we partition the underlying topology into non-overlapping regions. Each region is equipped with a collection point that accumulates all the state information of the mobile hosts, servers and links for that region. The region collection point maintains the number of mobile hosts in that region at any point in time.

Aggregate Mobility Model: We describe how to derive aggregate mobility from individual host mobility model. We use the incremental mobility model [8] to characterize individual host mobility. In this model, mobile hosts are distributed randomly and move freely in a closed coverage area. The movement of the mobile host is represented by its velocity vector $\vec{v} = (v, \theta)$, where v is the speed and θ is its direction. The location of the mobile host (x, y) and its velocity $\vec{v}$ are updated periodically every δt time units as follows:

$$\begin{gathered} v(t+\delta t) = min[max(v(t)+\delta v, 0), V_{max}] \\ \theta(t+\delta t) = \theta(t) + \delta\theta \\ x(t+\delta t) = x(t) + v(t) \cdot cos(\theta(t)) \\ y(t+\delta t) = y(t) + v(t) \cdot sin(\theta(t)) \end{gathered}$$

where V_{max} is the maximal mobile velocity; δv, the velocity change is uniformly distributed within $(-A_{max} \cdot \delta t, A_{max} \cdot \delta t)$, A_{max} is the maximum acceleration/deceleration of the mobile host. $\delta\theta$ is the change in the mobile host's direction and uniformly distributed in $(-\alpha \cdot \delta t, \alpha \cdot \delta t)$, where α is the maximal angular change of the mobile host's direction per unit time.

Mobile hosts are distributed randomly and move freely in a closed coverage area with the size of (X_{max}, Y_{max}) which is divided into many non-overlapping equal sized regions with the size of (X_{region}, Y_{region}). Each region has a collection point (e.g. base station) that serves as the wired network access point for all the mobile hosts in its controlled region. If we let $X_{dim} = \lceil \frac{X_{max}}{X_{region}} \rceil$ and $Y_{dim} = \lceil \frac{Y_{max}}{Y_{region}} \rceil$, then there are $N_{region} = X_{dim} \cdot Y_{dim}$ regions in the area. Mobile hosts either move or stop in this area, the population/aggregation in each region changes all the time and can be captured. At a certain time t, a mobile host located at $(x(t), y(t))$ is in region $\lfloor \frac{x(t)}{X_{region}} \rfloor + X_{dim} \cdot \lfloor \frac{y(t)}{Y_{region}} \rfloor$. At time t, the aggregation of region i is the number of mobile hosts located in region i, i.e., the cardinality of the set of the mobile hosts who are in this region.

$$A_i(t) = \| \{ j | \lfloor \frac{x_j(t)}{X_{region}} \rfloor = i \mod X_{dim}, \lfloor \frac{y_j(t)}{Y_{region}} \rfloor = \frac{i}{X_{dim}} \} \|.$$

The ABIC Algorithm: The ABIC algorithm consists of two phases: Phase 1 derives the aggregate mobility patterns from individual user mobility and utilizes the aggregation to adjust the collection parameters such as sampling frequency and range size; Phase 2 utilizes feedback from the consumers (the resource provisioning process in our case) to further adjust range size. In the following subsections, we describe the two phases of ABIC algorithm in more detail. Figure 2 gives the outline of the ABIC algorithm and Figure 3 presents the state diagram of the information collection process.

4.1 Phase 1 - Aggregation Driven Coarse-grained Information Collection

In this phase, we use the combination of mobile host aggregation and resource(server/link) utilization to decide SF and R. The overall strategy is to start with a certain SF and R, then adjust it periodically based on the changes in aggregation status or the real resource utilization.

The ABIC Algorithm:

//phase 1: aggregation driven coarse-grained adjustment of collection parameters
//invoked periodically
switch(resource utilization)
{
case high: set SF and R to be minimum;
case low: set SF and R to be minimum;
case medium: increase/decrease SF and R based on the aggregation;
}

//phase 2: fine-grained adjustment of range size
switch (type of messages received by information collection process)
{
case source-initiated trigger:
{
if (a significant change has been confirmed)
expand R;
}
case consumer-initiated trigger:
{
if (the number of triggers reaches a threshold within specified duration)
shrink R;
}
}

Figure 2: The Aggregation Based Information Collection Algorithm

We classify the aggregation level $AL(t)$ of region i as being one of the three levels high, medium and low as follows:

$$AL_i(t) = \begin{cases} High & \text{if } A_i(t) \geq N_{mh}/8 \\ Medium & \text{if } N_{mh}/N_{region} \leq A_i(t) < N_{mh}/8 \\ Low & \text{if } A_i(t) < N_{mh}/N_{region} \end{cases}$$

where N_{mh} is the total number of mobile hosts in the system. Similarly, we classify resource utilization level into level of high, medium and low as follows:

$$UL(t) = \begin{cases} High & \text{if } UF(t) \geq 0.9 \\ Medium & \text{if } 0.4 \leq UF(t) < 0.9 \\ Low & \text{if } UF(t) < 0.4 \end{cases}$$

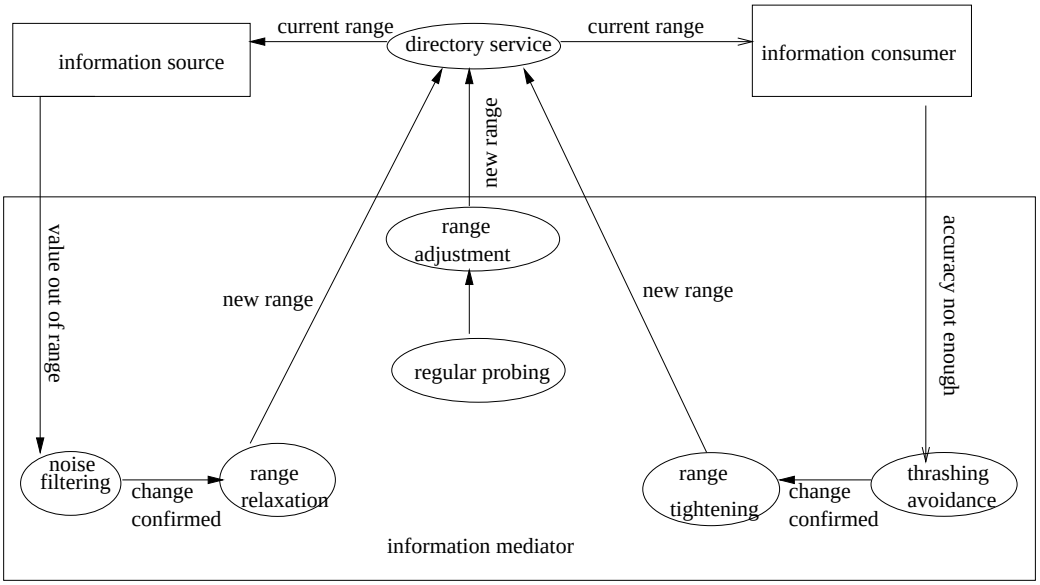

Figure 3: State Diagram of the Information Collection Process

The threshold utilization values that demarcate the boundaries between high, medium and low are implementation specifics and dependent on the traffic characteristics and system capacities. The impact of measured aggregation and resource utilization on the policy of adjusting SF and R can be summarized in Table 1.

	Mobile Host Aggregation Level AL		
Resource Utilization Level UL	High	Medium	Low
High	SF_{min}, R_{min}		
Medium	$\uparrow_{\alpha_H}$ SF $\uparrow_{\beta_H}$ R	$\uparrow_{\alpha_M}$ SF $\uparrow_{\beta_M}$ R	$\uparrow_{\alpha_L}$ SF $\uparrow_{\beta_L}$ R
Low	SF_{min}, R_{min}		

Table 1: Coarse Adjustment of Information Collection Parameters

- when resource utilization is *high*: it does not really matter what the aggregation is, since most of the requests are rejected due to the high resource utilization. Using a smaller range to describe the available resource can make the resource provisioning process make the right decision earlier just by looking at the directory service. This avoids the large overhead of attempting to reserve

resources all the way between the mobile host and server. This is especially true under conditions of high load where the network is already congested or the server is heavily loaded; we do not want to aggravate the congestion by injecting more traffic into the system. This implies the choice of a small SF and a small R.

- when resource utilization is *low*: it does not really matter what the aggregation is since requests will most likely be accepted because of sufficient resources. It is not necessary to have a high SF; keeping a smaller range can result in fewer consumer-initiated triggers. Again, this implies the choice of a small SF and a small R.

- when resource utilization is *medium*: This is the place where aggregation comes into play. Basically, we adjust both SF and R based on a change in aggregation status or resource utilization to allow for graceful aggregation management. We increase SF when a change in aggregation is noticed and reduce SF when the aggregation status is stable. The following equations describe how SF and R change in response to the change in either aggregation status or resource utilization. α_H,α_M,α_L are sampling frequency adjustment parameters for high, medium and low aggregation; and similarly, β_H,β_M,β_L are range size adjustment parameters for high, medium and low aggregation respectively. Here, $\alpha_H < \alpha_M < \alpha_L$ and $\beta_H > \beta_M > \beta_L$.

$$SF(t) = \begin{cases} min\{(1+\alpha_H) * SF(t-1), SF_{max}\} & \text{if } AL(t) = High \\ min\{(1+\alpha_M) * SF(t-1), SF_{max}\} & \text{if } AL(t) = Medium \\ min\{(1+\alpha_L) * SF(t-1), SF_{max}\} & \text{if } AL(t) = Low \end{cases}$$

$$R(t) = \begin{cases} min\{(1+\beta_H) * R(t-1), R_{max}\} & \text{if } AL(t) = High \\ min\{(1+\beta_M) * R(t-1), R_{max}\} & \text{if } AL(t) = Medium \\ min\{(1+\beta_L) * R(t-1), R_{max}\} & \text{if } AL(t) = Low \end{cases}$$

The collection process is now initiated using the above parameters. Initially, the collection process (i.e., the mediator) is in the regular probing status. It probes the source periodically and stores a range for the current sample in the directory service. The directory service holds an active range R with upper and lower bounds $U(R)$ and $L(R)$ of the collected information.

4.2 Phase 2: Selective Push/Pull based Information Fine Tuning

In this phase, we further fine tune the collection parameters to accommodate (a) unexpected changes in system load and (b) consumer specific accuracy needs. The collection process implements a push-pull based technique where information is (a) selectively pushed from the source to the mediator or (b) pulled by the mediator from the consumer as system conditions change significantly. Furthermore, the information consumer may demand information at a specific accuracy. This in turn may trigger an information collection update.

The managed entity (source) keeps the exact information value and a current interval that approximates this value; the approximate value is held in the directory service for middleware services

implemented in the upper layer(consumer) to use. Initially, the sampling frequency and range size are decided based on the aggregation status. The information in the directory service will now be fine tuned and updated in phase two based on triggers initiated by the source and consumer.

We now describe the tasks performed by the three components in the architecture, i.e., mediator, consumer and source, during phase two of the information collection process.

Information Mediator: Basically, information mediator responds to notifications from the source and the consumer in the phase 2 of the collection process:

- responding to source-initiated trigger: When the mediator gets the notification from the source about a change in the source value (i.e., source-initiated trigger), the mediator determines if the range size must be changed. It proceeds to look into it further by entering the noise filtering state to verify the change of the source value.

  In practice, there could be several reasons for a source-initiated trigger: measurement error, a transient burst or a significant load level change. In order to assist the underlying model to adapt to a confirmed change and filter out high frequency traffic components, we keep monitoring the source for a period T_s. We take the average of the values during this special monitoring period to see if it falls in the range R. If the average value falls with R, we keep the range unchanged. Otherwise, a significant change has been confirmed, we expand the range as follows: $R^{'} = (1 + \beta_f) * R$, i.e., $Prec(R)$ has to be lowered to accommodate the source change. As we can see, not every source-initiated trigger will cause a directory service update.

- responding to consumer-initiated trigger: When the mediator accepts a request from the information consumer, the mediator determines if the range size must be changed. Whether or not the range size is adjusted is based on how many consumer-initiated triggers exist within a predefined monitoring window T_c. To avoid thrashing caused by frequent range changes, i.e., range thrashing, we tighten the range only if the number of consumer-initiated triggers reaches a certain number N_c within a certain time period. The range is tightened as follows: $R^{'} = R/(1 + \beta_f)$, i.e., $Prec(R)$ is raised to meet consumer's requirements. After the range adjustment, the mediator sends the new range to the directory service for update. Again, not every consumer-initiated trigger will cause a directory service update.

Information Source Push: There are three reasons that may cause the exact source value to deviate from the stored interval: (1) resource reservation; (2) resource release ; (3) application load change. Application load changes are dynamic changes often caused by unmediated requests (those with no QoS requirements) that are accepted or completed. These best-effort requests are not accounted by the resource provisioning architecture since admission control is not required. When the exact value changes at the source, the source checks the validity of the range.

$$Valid([L, U], V_s) = \begin{cases} 1 & \text{if } L \leq V_s \leq U \\ 0 & \text{otherwise} \end{cases}$$

Based on the definition of $Valid()$, if the value V_s falls into the existing active range $R(Valid([L,U],V_s) = 1)$, no change is observed, the range is left unchanged. If V_s falls out of R ($Valid([L,U],V_s) = 0$), a change is observed, source notifies the mediator about the change by sending a source-initiated trigger.

Consumer-initiated Pull: Every so often, a consumer request may require information at a certain level of accuracy. The accuracy desired may or may not be satisfied by the existing range within the directory service. If the directory service accuracy is determined to be insufficient, the consumer request can initiate an information pull from the source. To maintain consistency of implementation, we translate the consumer-initiated pull to mediator initiated pull.

Given range $< L, U >$ and a client request with QoS requirements V_c, we define $Accurate([L,U],V_c)$ as

$$Accurate([L,U],V_c) = \begin{cases} 1 & \text{if } V_c \leq L \text{ or } V_c > U \\ 0 & \text{if } L \leq V_c \leq U \end{cases}$$

Our definition of the $Accurate()$ function is currently designed to fit the resource provisioning scenario (i.e., the information consumer is a resource provisioning process). [1] That is, if V_c falls outside of R , the decision is obvious: if $V_c < L$, the request can be accepted; if $V_c > U$, the request will be rejected. Otherwise, if V_c falls inside R, the approximation is not precise enough to make the admission decision, so a consumer-initiated trigger is sent to information mediator to ask for more precise information. This means that the requested resources are close to the residual capacity .The accuracy provided by the (possibly large) range size is insufficient to determine if resource reservation must be initiated. Hence, the request asks for more accurate information about the residual capacity.

A Caveat: Since the characteristics of links and servers are different, we modify the above policies slightly to allow for more efficient collection. Most networks have a large number of links, monitoring the status of each link at a fine-grained level introduces a significant overhead. We therefore eliminate source push for link entities and implement mediator pull and consumer-initiated trigger for links. In the case of server entities, we implement all the three fine tuning techniques. However, if all incoming requests have QoS requirements that are brokered via a resource provisioning module, reasonably accurate source information is available with the directory service, source push becomes unnecessary. The combinations of our policies for server and link information is listed in Table 2.

5 Performance Evaluation

In this section, we analyze the performance of the ABIC algorithm wrt. to other information collection algorithms. Our work focuses on the development of middleware frameworks for managing mobile environments. Hence, we choose QoS-enabled resource provisioning as the application built above the information collection architecture. Specifically, we choose a resource provisioning algorithm CPSS (see

[1] The system designer may develop other definitions of $Accurate()$ that better suit the application at hand.

collection process	server info.	link info.
mediator regular probe	x	x
information source push	x	
consumer-initiated pull	x	x

Table 2: Information Collection Policies Combinations

Appendix for more detail) that uses both server and network information to choose the optimal path and server for incoming requests with QoS requirements. Our objective of the performance study is to determine the information collection algorithm most suited for highly dynamic mobile environments. Of course, the performance is dependent on the applications using the information collection mechanisms, the CPSS algorithm is one such technique we use for this study. We use the following four metrics to evaluate the performance of the four information collection algorithms (SS, SR, DR, ABIC):

- Request Admission Ratio: This is the ratio of the number of admitted requests to the total number of requests. Only requests with QoS requirements smaller than available resources in both server and paths can be admitted.

- Request Completion Ratio: This is the ratio of the number of completed requests to the total number of requests. Since mobile hosts constantly move, dealing with request handoff is unavoidable. Incoming requests go through an admission control process where they may be accepted or rejected. An admitted request executing on a mobile host will require re-provisioning of resources (network links, servers etc.) as it migrates through the different regions. The intermediate re-provisioning may or may not be successful. Admitted requests may not complete due to several reasons: path failure where there is no route with sufficient resources; location failure where locating mobile hosts fails; the alternate re-scheduling server may not have sufficient resources if path to original server is not available.

- Information Collection Overhead: There are four kinds of overheads that must be accounted for: sampling, triggering, directory update and location query/update. To differentiate the cost of each overhead is difficult and application dependent, so for simplicity, we assume the cost of information collection is proportional to sum of the number of samplings, the number of triggering, the number of directory updates,and the number of location query/updates.

- Overall System Efficiency: The overall efficiency is dependent on two factors: completion ratio and overhead, so it is defined as the ratio of the number of completed requests to the overhead involved.

5.1 The Simulation Environment

Our simulator is a message driven multi-threaded simulator intended to study the dynamics of QoS sensitive network environments. It consists of three components: information sources (managed entities), information mediators (information collection decision point) and information consumers (resource provisioning).

Topology and System Configuration: In the simulation, we use the following typical ISP network topology with 18 nodes and 30 links shown in Figure 4 . We assume that each node is a network router, and that the clients and servers are distributed in the network and are directly behind the router nodes (not shown in the graph). The topology is chosen such that there are a large number of alternative paths between source and destinations nodes.

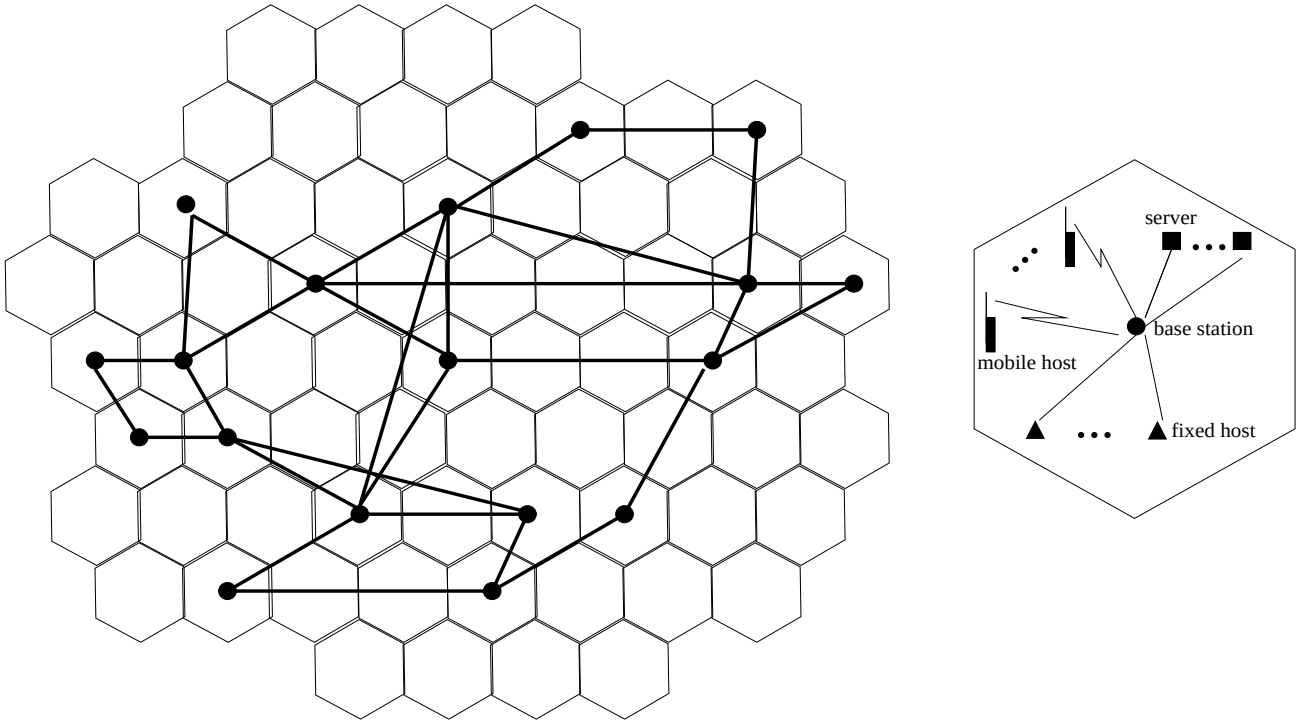

Figure 4: Simulation Topology

To better emulate the real network, the capacities of network links are selected from various widely used link types from 1.5Mbps to 155Mbps, with the mean value being 64M. When defining the capacity of the server nodes, we calibrate CPU units using a basic 64Kbit voice processing application, memory units to be 64Kbytes, and disk bandwidth units to be 8Kbytes/s. The server capacities are also selected from popular models of multimedia servers and web servers, with the CPU, memory, disk bandwidth

	Fast Mobility	Slow Mobility
Uniform Traffic	FM-UT	SM-UT
Non-uniform Traffic	FM-NUT	SM-NUT

Table 3: Four Scenarios Considered

mean to be 1845, 6871 and 5770 calibrated units respectively.

Request and Traffic Generation Model: We model request arrival at the source nodes as a Poisson distribution, and the request holding time is exponentially distributed with a pre-specified average value. The request holding time is the time for which the requested network and server resources such as link bandwidth, CPU, buffer, disk bandwidth etc. are reserved. We pre-define a set of client request templates to capture typical multimedia connection request patterns in terms of network bandwidth, CPU, memory, disk bandwidth and end-to-end delay. For each request generated, the requested parameters are randomly selected from the set of request templates, with the mean requested bandwidth being 2.5Mbps, mean end-to-end-delay being 400ms and CPU, memory and disk bandwidth being 150, 374 and 271 calibrated units respectively. To represent non-uniform traffic, we designate some sets of candidate destinations as being "hot", (i.e. serving popular videos, web sites etc), and they are selected by the clients more frequently than others. To reduce the effect of local and nearby requests, we choose three pairs of source-destination sets from the topology. The requests arrive to these hot pairs, as foreground traffic, at a higher rate than other background traffic. In our non-uniform traffic pattern, we set the foreground arrival rate to be 5 times higher than the background rate; while for uniform traffic pattern, we set them to be equal to each other. Specifically we set the foreground arrival rate to be 10 seconds, and the background rate to 50 seconds.

Experimental Scenarios: We study the performance of the four information collection policies under different conditions of mobility. For the purpose of this simulation, we model 2 levels of mobility (referred to as mobility degree): fast mobility and slow mobility. In this simulation, we set $V_{max} = 65mph$, $A_{max} = 0.9meter/sec^2$, $\alpha = 0.1745radian/sec$. With fast mobility, the velocity change (δv and $\delta\theta$) is twice that of slow mobility. Two request patterns described above are considered: uniform traffic and non-uniform traffic. Table 3 depicts the four different cases we studied in this paper.

5.2 Experimental Results

We analyze the impact of information collection mechanisms on the overall CPSS performance. Before comparing the ABIC algorithm with other traditional algorithms, we would like to show the benefits of using aggregation status in the information collection process. Figure 5 shows higher completion ratio and overhead of ABIC than without using aggregation.

We first analyze the performance of each collection algorithm individually and then compare the performance of the four algorithms under similar conditions. For each scenario, we present four graphs to compare the admission ratio, completion ratio, total overhead and overall efficiency of the four

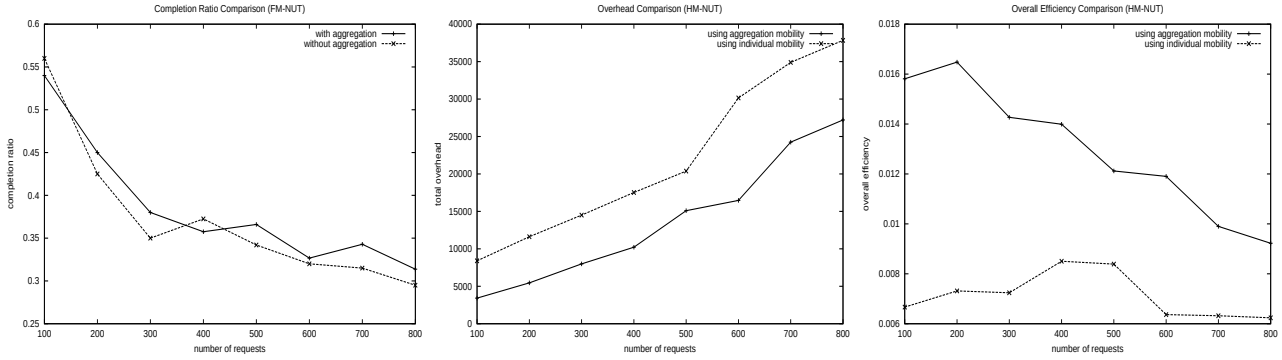

Figure 5: **Comparison of Information Collection with/without Aggregation Status under Fast Mobility and Non-uniform Traffic:**

information collection policies. To ensure a fair comparison, we pick the fine tuned parameters for the three traditional algorithms based on previous studies: we set SF to be 10 seconds for the SS algorithm, set R to be 1/4 of the source capacity for the SR algorithm, set range tightening ratio to be 0.25 for the DR algorithm [9].

Fast Mobility and Non-uniform Traffic (FM-NUT): Figure 6 shows the performance of the four information collection policies under fast mobility and non-uniform traffic. The admission ratio is uniformly higher than completion ratio, because requests admitted will require resource reallocation after the mobile host leaves its current region, the reallocation may not always be successful due to

- failure in locating the mobile host (rapid movement of mobile host makes it difficult to pinpoint current location)
- insufficient resources in the new region.

When there are fewer number of requests in the system (such as 100 in this simulation), both ratios are high; but when the number of requests increases, the ratios decrease dramatically; after certain number (such as 200 in this simulation), the ratios almost level off. This is because when only few requests are sharing the resources in the system, system can accommodate most of them; but then conflicts occur due to limitation of the resources and the number of concurrent requests decreases. The system stabilizes at some point in time where the resource requirements of incoming requests match the resources released by completing requests. Comparing the admission ratio and completion ratio of the four policies, the ABIC algorithm shows higher admission and completion ratios than the other three algorithms. We attribute this to the fact that the ABIC algorithm checks sampling frequency and range size periodically and adjusts them based on current aggregation status and resource utilization status to maintain more accurate information in the directory service. The SS, SR and DR algorithms exhibit similar admission ratio and completion ratios.

The overhead of all the four strategies increases with the increase of the number of requests because with more requests, more dynamic change is expected, the collection process will be busier with the monitoring. As expected, the instantaneous snapshot-based algorithm introduces more overhead than the other three due to frequent sampling and directory updates containing exact values. The ABIC algorithm has the least overhead, since it adjusts the sampling frequency and range size based on the current aggregation and resource utilization, fewer source-initiated and consumer-initiated triggers are introduced, which leads to less overall overhead. The other two algorithms (SR and DR) introduce almost the same amount of overhead, the results indicate that the initial range chosen for DR works very well and it was not necessary to change it very often.

The overall efficiency of ABIC is the highest among all the four algorithms studied, while that of SS is the lowest. It shows that the ABIC algorithm achieves the best completion ratio with the least overhead. The efficiencies of the dynamic range based algorithm and static range based algorithm are very close to each other. The reason is that in mobile environments, we must account for host mobility in addition to network and link changes. Not all requests admitted can be completed and range change may not keep up with the movement of mobile hosts, hence changing the range size does not always exhibit better results.

Fast Mobility and Uniform Traffic (FM-UT): Figure 7 shows the performance of the four information collection policies under fast mobility and uniform traffic.

As before, with the increase in the number of requests, the admission and completion ratios decrease and the overhead increases. This trend continues until the number of requests reaches 500 in the simulation. After that, the admission ratio and completion ratio begin to increase. Since the holding time of a multimedia request is relatively long, the system in our simulation is close to saturation when there are 500 requests in the system; but after that some requests are either completed or aborted and the system can accept more requests. In terms of admission ratio, the ABIC algorithm outperforms other algorithms when the system is highly loaded.

Note that the completion ratio of the ABIC algorithm is always the lowest, while the other three algorithms show similar completion ratios. This is because the aggregation status is changing rapidly due to fast moving mobile hosts, while the directory service value maintained by ABIC lags behind the fast changing aggregation status.

As in FM-NUT, the overhead introduced by ABIC is lowest among all the algorithms studied, while SS has the highest overhead, the overhead introduced by SR or DR lies in between. Overall, ABIC exhibits a little higher efficiency than the other algorithms, SS has the lowest efficiency, SR and DR perform similarly.

When the number of requests goes from 700 to 800, the overall efficiency of ABIC goes up instead of going down because the system is already busy dealing with the large amount of requests, it only needs a little more effort to handle some more requests, which can be seen from the almost even line in the overhead comparison.

Slow Mobility and Non-uniform Traffic (SM-NUT): Figure 8 shows the performance of the four information collection policies under slow mobility and non-uniform traffic.All the four policies follow a similar trend in admission ratio, completion ratio and overhead .

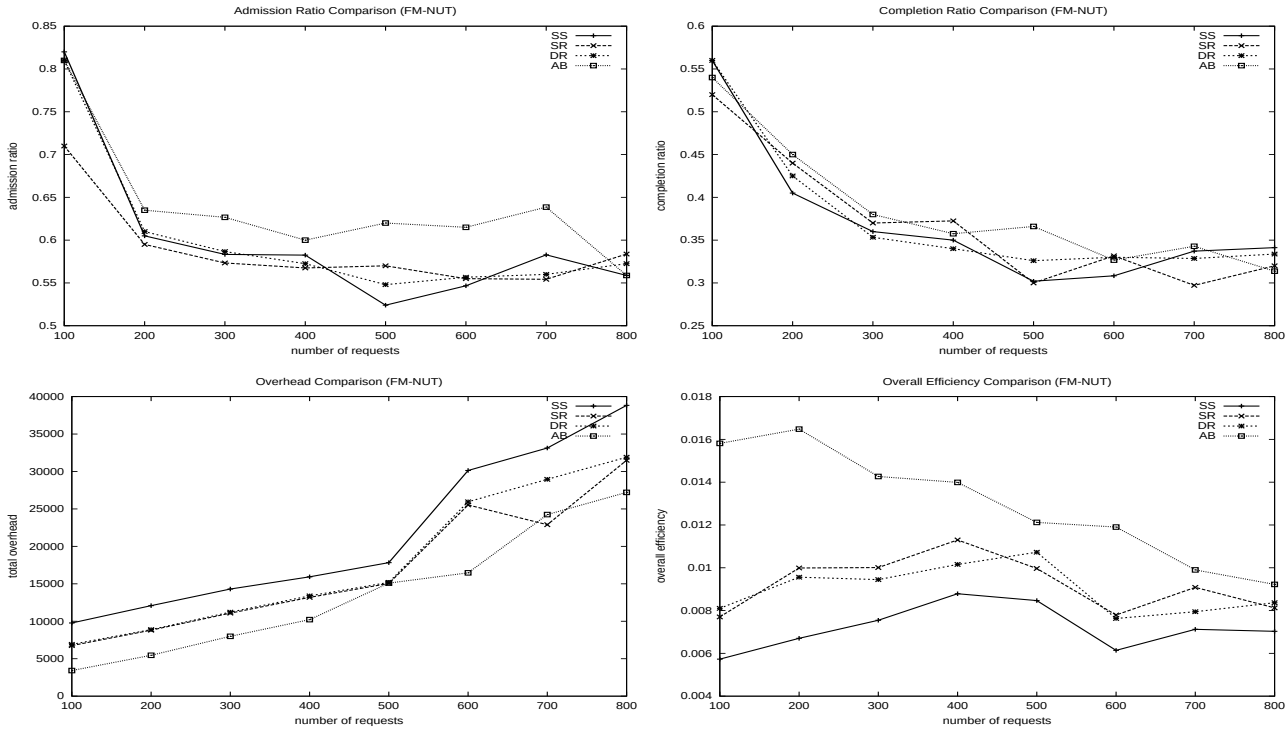

Figure 6: **Information Collection Policies under Fast Mobility and Non-uniform Traffic:** The ABIC algorithm exhibits the highest admission and completion ratios, lowest overheads, and thus highest overall efficiency. The SS algorithm introduces large overheads to achieve completion ratio similar to the other algorithms and thus leads to lowest overall efficiency. The SR and DR algorithms exhibit similar performance.

The ABIC algorithm exhibits the highest admission ratio and completion ratio, the least overhead, thus leading to uniformly highest overall efficiency. It shows that when mobile hosts move slowly, the change in aggregation status is also slow and the ABIC algorithm can easily adapt to the change. The performance of SR algorithm is next to ABIC; while DR has the lowest admission and completion ratios most of the time. It shows that the DR algorithm does not reflect the resource availability change very well here, using SR is good enough for this scenario. As always, the SS algorithm does not yield very good overall efficiency due to its high overhead.

Slow Mobility and Uniform Traffic (SM-UT): Figure 9 shows the performance of the four information collection policies under fast mobility and non-uniform traffic. In terms of admission ratio and completion ratio, with small number of requests in the system, the SS, SR works best and the ABIC works the worst; with large number of requests in the system, the SS works best, the DR works the worst. But the overhead of SS is always the highest and the overhead of ABIC is the lowest, the

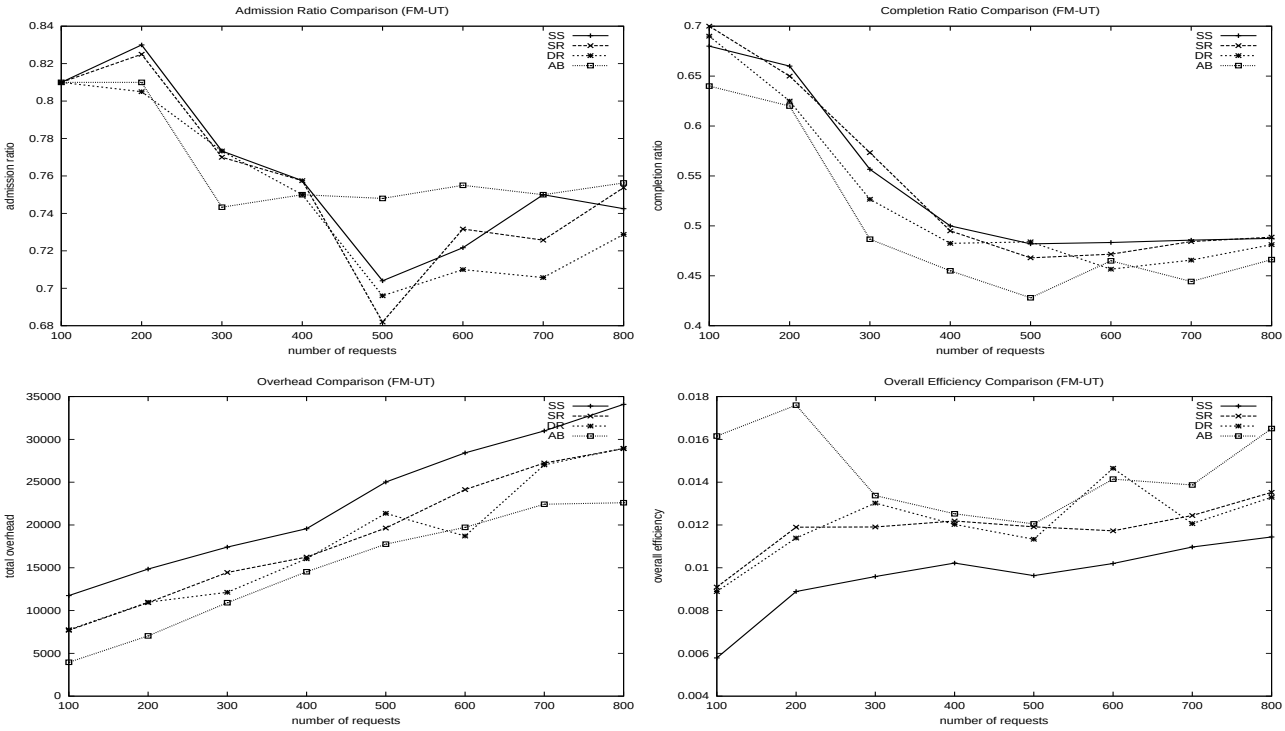

Figure 7: **Information Collection Policies under Fast Mobility and Uniform Traffic:** Even though ABIC has lowest completion ratio, due to its lowest overhead introduced, it outperforms the other three algorithms in terms of overall efficiency. SS algorithm performs the worst due to its highest overhead. SR and DR show similar overall efficiency.

overhead of SR and DR is almost the same. When traffic is uniform, resource availability is nearly consistent, thus keeping track of the availability needs less effort. Overall,the efficiency of ABIC is much better than the others and that of SS is much worse; SR and DR has similar efficiency.

Impact of Mobility Degree and Traffic Patterns: Analysis of Figures 6 and 8, Figures 7 and 9, reveals that whether mobile hosts move fast or slow does not make much difference in the performance of the four policies. When mobile hosts move fast, more request handoffs will occur, it is likely there are more requests in the system. Our results show that the performance of the four policies is not influenced by the number of requests in the system.

Analysis of Figures 6 and 7, Figures 8 and 9 reveals that under non-uniform traffic,the admission ratio and completion ratio of ABIC is uniformly higher than the other three; while under uniform traffic, admission ratio and completion ratio of the ABIC is more random. This shows that when traffic is non-uniform, resource usage tends to be more irregular, ABIC can maintain more accurate information in the directory service than the other three policies. There is no significant difference

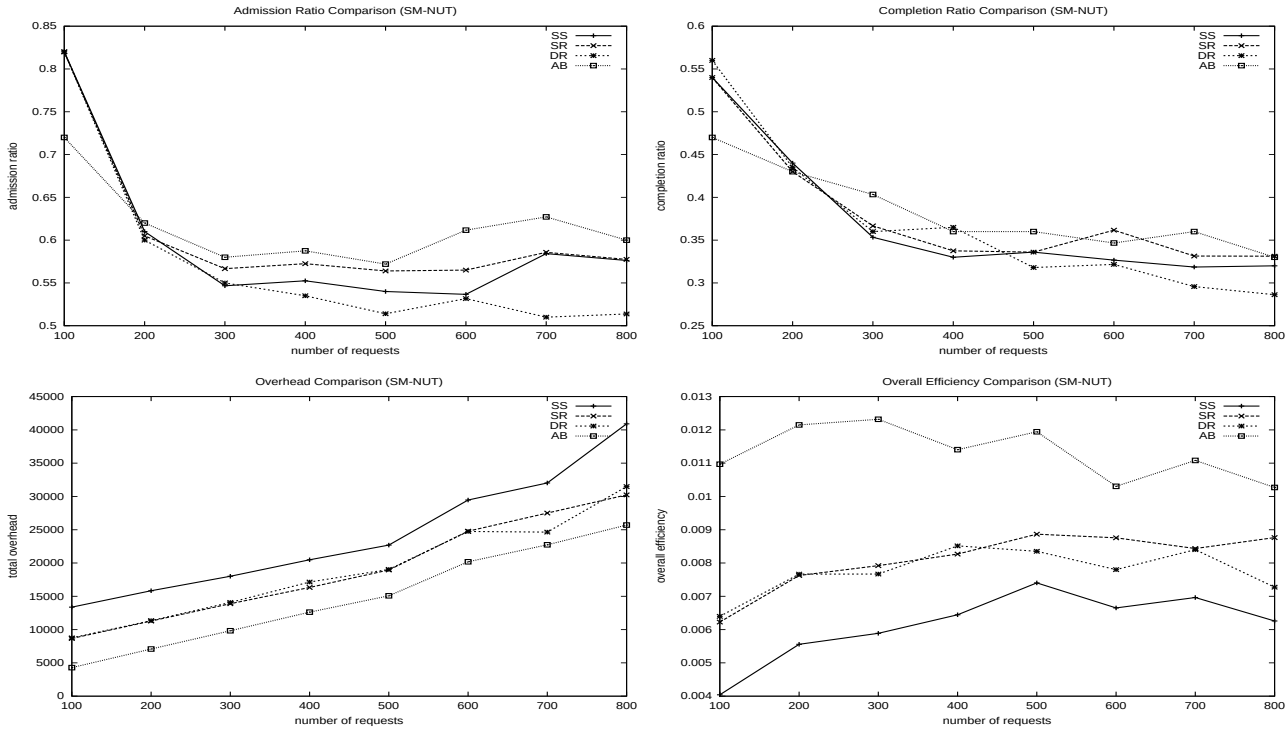

Figure 8: **Information Collection Policies under Slow Mobility and Non-Uniform Traffic:** Uniformly, the ABIC algorithm shows the highest admission ratio and completion ratio, the lowest overhead, thus the highest overall efficiency; the SR performs relatively better than DR due to its improved admission and completion ratio; the SS yields the lowest overall efficiency.

in terms of admission ratio and completion ratio among other three policies under different traffic patterns. Furthermore, the overhead of the SS algorithm always ranks the highest followed by SR and DR, ABIC has consistently the lowest overhead. The traffic patterns themselves do not have an obvious influence on the overhead.

Figure 10 shows the overall efficiency of ABIC algorithm under the four scenarios considered in this paper. We can see that the ABIC algorithm works best under slow mobility and uniform traffic. The reason is that when mobile hosts move slowly, the aggregation status also changes slowly. The ABIC process then has enough time to adapt to the change; when the traffic is uniform, each server accept similar amount of requests, then the usage of resources is more consistent.

Performance Summary: The overall efficiency of ABIC is uniformly higher than the other three algorithms for mobile environments, while the snapshot based algorithm uniformly has lowest overall efficiency due to its high overhead. In non-mobile environments, the dynamic range based algorithm always exhibits better overall efficiency than the static range based algorithm, however, we observe

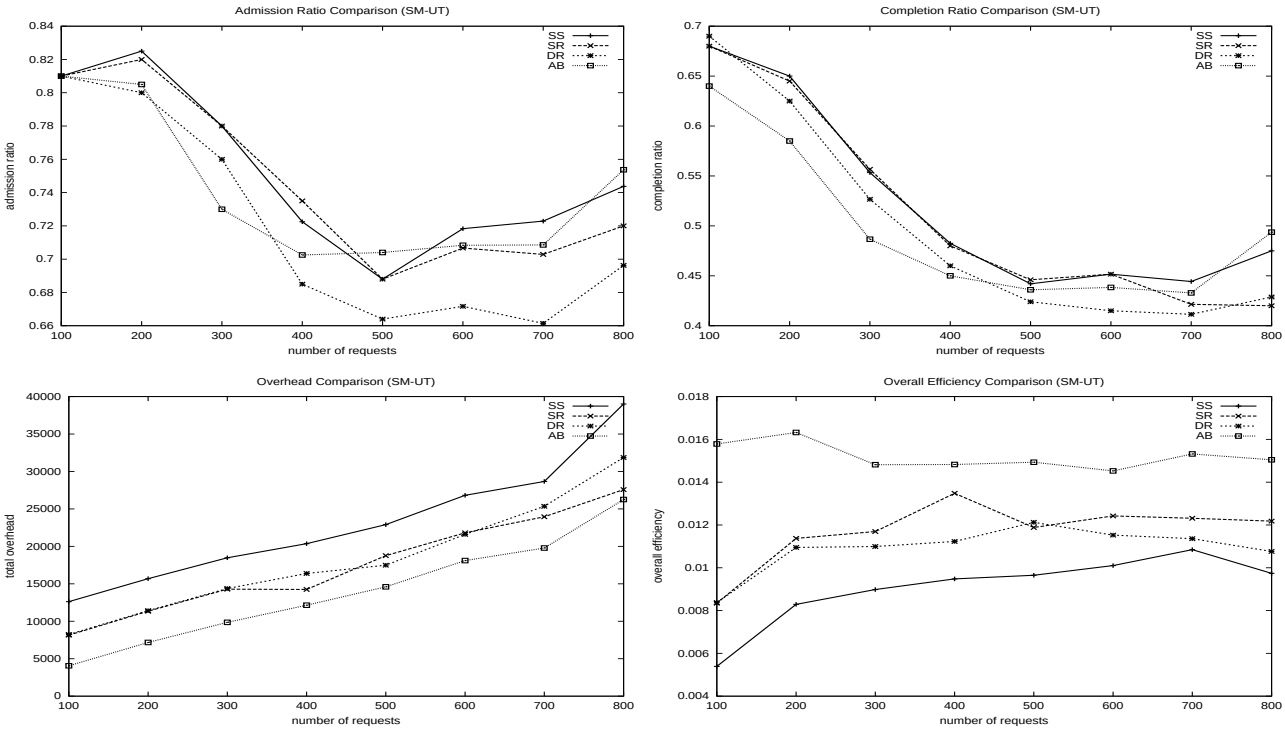

Figure 9: **Information Collection Policies under Slow Mobility and Uniform Traffic:** The ABIC algorithm does not have the highest completion ratio all the time, but its overhead is very low, thus leading to still better overall efficiency; Although SS shows very good completion ratio, this comes with high overhead, leading to low overall efficiency; the performance of SR and DR algorithms are close to each other.

that it is not always true in mobile environments. In general, the ABIC algorithm performs equally well regardless of mobility speed. When traffic is non-uniform, the ABIC algorithm outperforms the other three by a large margin than when the traffic is uniform. Since network traffic in the real world is always non-uniform, hence we conclude that the ABIC algorithm is the best candidate among the four policies studied here for information collection in mobile environments.

6 Related Work and Future Research Directions

In this section, we describe some of the related work in the area of network management, information collection and approximate caching.

Network management deals with the monitoring and analysis of network status and activities. Network monitoring tools watch network segments and provide information on data throughput , node

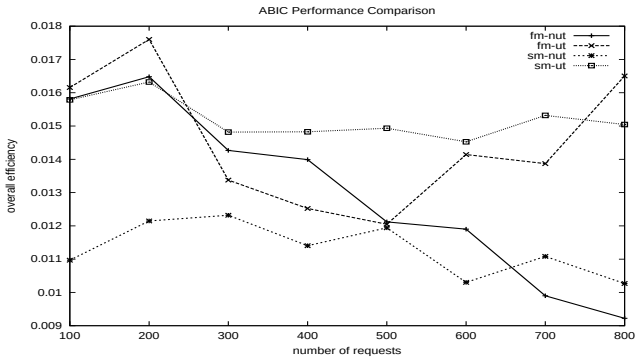

Figure 10: ABIC Performance Comparison Under Different Scenarios

and link failures, and other global occurrences on the network that may be useful in some manner to network managers. Network management projects focus on making both active and passive network statistics and information available to end-users. Recent systems that focus on the measurement of communication resources across the Internet include Network Weather Services(NWS) [19] and topology-d [11]. NWS makes resource measurements to predict future resource availability, while topology-d computes the logical topology of a set of internet computation nodes. Both of these systems actively send messages to make communication measurements between pairs of computation nodes. The Remulac project and Remos [2]is a generalizable resource monitoring system for network applications. In this system, network applications interact with a portable interface to the network that includes flow query and logical query topology abstractions. It maintains both static and dynamically changing information based on SNMP measurements on the router nodes in the network.

Cost-effectively collecting distributed state information still remains to be an open issue in the network management research area. Network information discovery is included in the metacomputing system Globus [4]. The maintained information includes network activity, available network interface, processor characteristics, and authentication mechanisms. Network measurement architectures have been proposed in [5, 3, 13, 16, 21]. [5] proposes a method to estimate the distance of any two points in the Internet. Linear model based information collecting and prediction infrastructure have been studied, and service APIs for upper level applications which needs such support in establishing network connections have been defined [3]. In the diffserv architecture [21], state information collected in the directory is used by the bandwidth broker to statistically guarantee the negotiated QoS agreement. Possibly, the directories of adjacent domains can exchange aggregated state information to maintain a more accurate snapshot of the system. The static range based algorithm used in this paper is proposed in [1]. In this paper, the QoS routing performance is studied in detail using different static range size and sampling intervals. They have found that smaller range size performs better if sampling with short intervals , while bigger size gets more cost-effectiveness in longer intervals. Moving object databases

deal with the modeling and tracking of a moving object within a database; information collection solutions have been proposed in this context that directly compare the cost of information imprecision with the cost of message passing and make decisions based on the difference [18].

Adaptive data caching adjusts the caching strategy dynamically as conditions change. Caching approximate values such as ranges instead of exact values has been studied extensively in database area. Divergence caching considers setting the precision of of approximate values in a caching environment [10], where precision is inversely proportional to the number of updates to the source value not reflected in the cached approximation, independent of the actual updates. This algorithm works well in its intended environment, but it is not clear that it could be generalized easily or effectively to incorporate update patterns as well as frequency. To answer a query, TRAPP systems automatically select a combination of locally cached bounds and exact master data stored remotely to deliver a bounded answer consisting of a range that is guaranteed to contain the precise answer [15, 14]. Their algorithm can also handle stale value approximations.

Future Research Directions: In this paper, we present a novel aggregation based information collection (ABIC) technique for resource provisioning in mobile environments. The ABIC algorithm utilizes individual mobility patterns to derive the aggregation mobility information and adjusts the information collection process based on the aggregation. Feedback from resource provisioning and the information source itself can further trigger the collection process.

In the AutoSeC project [9], we proposed an integrated middleware framework that can dynamically select an appropriate combination of information collection and resource provisioning policies based on current system conditions and user requirements. We are working on enhancing the AutoSeC tool for mobile environments by integrating techniques such as ABIC algorithm with the other resource provisioning algorithms. In addition to this, we are also working on other models for approximate data management such as control theoretic approaches and real-time information collection so that resource provisioning decisions can be made in real time.

The eventual goal of our work is to develop effective tools for system management in highly dynamic environments. Efficient directory service management requires effective representation of collected data in the directory service. The data schemas selected should optimize the overheads involved in querying and updating information. We are looking into efficient data schemas for the representation of dynamic information. A scalable information collection architecture suitable for wide-area environments must incorporate distributed directories. Distributed directory service management is a hot topic for future work.

Middleware techniques for adaptive service management such as those described in this paper are key to guaranteeing application QoS in highly dynamic mobile environments. This is a necessity to achieve the goal of true ubiquitous computing.

Appendix: Combined Path and Server Selection(CPSS)

Traditionally, the problem of effective resource utilization for networks and servers has been studied independently. At the network level, QoS routing techniques are used to improve the network utilization by balancing the load among the individual network links. At the server end, since data may be replicated across multiple servers, server selection policies direct the user to an optimal server that can handle the incoming requests for information. Server selection mechanisms often treat the network path leading from the client to the server as static. The two techniques(QoS routing and server selection) can independently achieve some degree of load balancing. When applications are highly sensitive to QoS parameters, high-level provisioning mechanisms are required to address the route selection and server selection in a unified way. Such integrated mechanisms can potentially achieve higher system-wide utilization and therefore allow more concurrent users.

The basic idea of CPSS algorithm is: given a client request with QoS requirements, we select the server and links that maximize the overall utilization of resources. It allows load balancing not only between replicated servers, but also among network links to maximize the request success ratio and system throughput. For detailed discussion, refer to [7].

We define *Utilization Factor (UF)* for server as follows. The capacity of a server can be specified as four parameters: CPU cycles, memory buffers, I/O bandwidth and network transfer bandwidth [17]. The server resources needed by a request r are modeled as $< CPU_r, BUF_r, DB_r, X_r >$, the available server resources can be modeled as $< CPU^s_{avail}, BUF^s_{avail}, DB^s_{avail}, X^s_{avail} >$. The UF for a server s, given a request r is defined as

$$UF(s,r) = \begin{cases} max(\frac{1}{CPU^s_{avail}-CPU_r}, \frac{1}{BUF^s_{avail}-BUF_r}, \frac{1}{DB^s_{avail}-DB_r}, \frac{1}{X^s_{avail}-X_r}) \\ \qquad \text{if available server resources are greater than those requested} \\ \infty \\ \qquad \text{otherwise} \end{cases}$$

Similarly, we define UF for network links to quantify the residue capacity of links. The UF for a link l with available link bandwidth BW^l_{avail}, given a request r with bandwidth requirement BW_r is defined as

$$UF(l,r) = \begin{cases} \frac{1}{BW^l_{avail}-BW_r} & \text{if } BW^l_{avail} > BW_r \\ \infty & \text{otherwise} \end{cases}$$

For an assignment X=$\{p,s\}$, with network path p and server s, we define the distance of the server s from the client to be $Dist(s,r) = \sum_{l \epsilon p, p \epsilon X} UF(l,r) + UF(s,r), s \epsilon X$. Given a client request with QoS requirements $r :< BW_r, CPU_r, BUF_r, DB_r, DL_r >$, an assignment X=$\{p,s\}$ is feasible if and only if the available resources in both server and links are greater than requested. We define a feasible set X_f as set of all the assignments that meet the feasibility condition. An assignment X^*=$\{p^*, s^*\}$ is optimal if and only if it satisfies the feasibility condition and a policy dependent optimality criteria. For instance, the optimality clause for the BEST UF policy is
Dist(s,r)=Min{ UF(s*,r)+UF(p*,r)}, for all s in feasible set S.*

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, and S.K. Tripathi. Quality of service based routing: A performance perspective. In *ACM SIGCOMM*, 1998.

[2] T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, and J. Subholk. Remos: A resource monitoring system for network aware applications. Technical Report CMU-CS-97-194, CMU, 1997.

[3] Peter A. Dinda and D. R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, CMU, 1999.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal on SuperComputer Applications*, 11(2), 1997.

[5] P. Francis, S. Jamin, V. Pasxon, L. Zhang, D. Gryniewica, and Y. Jin. An architecture for a global internet host distance estimation service. In *IEEE InfoCom*, 1999.

[6] Z. Fu and N. Venkatasubramanian. Adaptive parameter collection in dynamic distributed environments. In *IEEE ICDCS*, 2001.

[7] Z. Fu and N. Venkatasubramanian. Directory based composite routing and scheduling policies for dynamic multimedia environments. In *IEEE IPDPS*, 2001.

[8] Z. Haas. A new routing protocol for the reconfigurable wireless networks. In *the IEEE Int. Conf. on Universal Personal Communications*, October 1997.

[9] Q. Han and N. Venkatasubramanian. Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2(7), 2001.

[10] Y. Huang, R. Sloan, and O. Wolfson. Divergence caching in client-server architectures. In *PDIS*, 1994.

[11] K.Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. Technical Report TR97-660, USC, Dept. of CS, October 1997.

[12] D. Lam, D. Cox, and J. Widom. Teltraffic modeling for personal communications services. *IEEE Communications Magazine*, 35(2), September 1997.

[13] Nancy Miller and Peter Steenkiste. Collecting network status information for netowrk-aware applications. In *IEEE InfoCom*, 1999.

[14] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, 2001.

[15] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, 2000.

[16] M. Stemm, R. Katz, and S. Seshan. A network measurement architecture for adaptive applications. In *InfoCom*, 2000.

[17] N. Venkatasubramanian and S. Ramanathan. Load management for distributed video servers. In *IEEE ICDCS*, May 1997.

[18] O. Wolfson, S. Chamerlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *IEEE ICDE*, 1998.

[19] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forcasting system for metacomputing: The network weather service. Technical Report TR-CS97-540, UCSD, May 1997.

[20] Vincent W.-S. Wong and Victor C.M. Leung. Location management for next-generation personal communications networks. *IEEE Network*, September/October 2000.

[21] Z.L. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling qos control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *ACM SIGCOMM*, 2000.

Qi Han is a Ph.D. student at the Department of Information and Computer Science, University of California- Irvine. Her research interests include distributed systems middleware and mobile computing. She has an M.S. degree in Computer Science from Huazhong University of Science and Technology, Wuhan, Hubei, China and is a student member of the IEEE.

Nalini Venkatasubramanian is an assistant professor at the Department of Information and Computer Science, University of California, Irvine. Her research interests include distributed and parallel systems, middleware, real-time multimedia systems, mobile environments and formal reasoning of distributed systems. She is specifically interested in developing safe and flexible middleware technology for highly dynamic environments. Nalini was a member of technical staff at Hewlett-Packard Laboratories in Palo Alto, California for several years where she worked on large scale distributed systems and interactive multimedia applications. Nalini has also worked on various database management systems and on programming languages/compilers for high performance machines. She has an M.S. and Ph.D. in Computer Science from the University of Illinois, Urbana-Champaign and is a member of the IEEE and ACM.