RC24165 (W0701-132) January 26, 2007
Computer Science

IBM Research Report

Efficient Querying and Resource Management Using Distributed Presence Information in Converged Networks

Dipanjan Chakraborty, Koustuv Dasgupta
IBM India Research Lab - Delhi
Block I, Indian Institute of Technology
Hauz Khas, New Delhi - 110016
India

Archan Misra
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Efficient Querying and Resource Management Using Distributed Presence Information in Converged Networks

Dipanjan Chakraborty*, Koustuv Dasgupta *, Archan Misra†
*IBM India Research Laboratory, New Delhi, India
†IBM T.J. Watson Research Center, New York, USA

Abstract

Next-generation converged networks shall deliver many innovative services over the standardized SIP-based IMS signaling infrastructure. Several such services exploit the joint presence information of a consumer, i.e. SIP entity requesting a service, and a vendor, i.e. SIP resource providing a service. Presence information is a collection of contextual attributes (e.g. location, availability, reputation), some of which change dynamically. Moreover, this collective presence information is distributed across multiple presence servers. While performing query matching based on joint presence information, a server usually routes each query to a locally available resource. However, skews in the spatio-temporal distribution of queries and resources may require queries to be routed to alternate servers with available resources. We propose a novel Resource-Aware Query Routing scheme, called RAQR, where each server proactively establishes gradients to suitable servers via a diffusion-based algorithm. Gradients are set up whenever a server anticipates scarcity of resources and withdrawn when the resource crunch is mitigated. We compare RAQR with alternative resource matching schemes and show that it adapts to spatio-temporal variations in resource availability, thereby leading to effective query matching with minimal control overhead.

1 Introduction

Mobile service providers will shortly deploy the SIP-based [8] Intelligent IP Multimedia Subsystem (IMS) [18] framework for next-generation services. *Presence*, defined as the network's capability to track the dynamically varying values of various contextual attributes of an individual, is a key feature of several emerging context-aware network applications. Common examples of such presence-based services include proximity-based notification (alerting a user when friends on his buddy list are nearby) or availability-aware filtering (e.g., routing a non-emergency call to voicemail when the user is in a movie theater).

Presence-based applications typically leverage upon contextual attributes such as location, availability, schedule etc. Typical presence-based applications like "RestaurantFinder" or "Buddy Finder" services, only exploit the presence attributes of requesters or of known end-points. Unlike the applications that consider only mobile requesters and *static* vendors[1] (e.g., restaurants, ATMs), we focus on new applications that target the *nomadic* vendor segment, and thus consider the joint presence-based attributes of *both* requesters and vendors. An example of such an application is an on-demand, context-aware matching service which enables a cellular service provider to effectively connect a user requesting a service ("Find-A–Plumber") to the nearest and available mobile vendor offering the service. We call the general class of such applications as **Live Resource Finder (LRF)** applications.

Such LRF applications exhibit three important properties: (1) *ephemeral* nature of presence information (e.g., location and availability); (2) *spatio-temporal distribution* of resources across presence servers; (3) *snapshot queries* as opposed to subscription–based queries (i.e., "Find me a plumber now" rather than "Continually notify me when a plumber is nearby"). In an IMS infrastructure, for reasons of scale and performance (many tier-1 cellular providers have several millions of customers/resources), presence information will be stored across a distributed set of presence servers. Hence, queries for users satisfying predicates on presence attributes must often be routed to multiple servers. Novel mechanisms are hence needed to

[1]We use the terms *vendor* and *resource* interchangeably throughout the document

efficiently query such a distributed repository.

In this paper, we first explain the query routing and reply semantics that LRF applications impose on the distributed presence environment, and then describe a Resource Aware Query Routing scheme, called **RAQR**[2], for LRF applications. One key observation from several LRF applications is that a query does not merely match to an available resource (vendor), but also *consumes* it (i.e., the vendor becomes unavailable for other requesters for an appropriate time period). Accordingly, in a distributed presence infrastructure, the routing of a LRF query should factor in resource consumption, and thus be tied to the spatio-temporal distribution of *available* resources. Standard publish-subscribe systems [10, 17] handle subscription-based queries and do not consider *consumption* of resources in matching. RAQR enables servers facing a crunch in local resources (vendors) to learn the identity of alternative servers with spare resources in the network. The identities of such alternative servers are established via a diffusion-style [11] algorithm – whereby a particular server's resource requirements percolate *minimally* through the network, until an appropriate set of servers with the corresponding amount of spare resources become available. The diffusion process avoids the overheads of broadcasts (either for queries or presence data), and increases the query success rate with minimal control overhead.

2 SIP-based Presence and LRF Queries

SIP [8] has gained widespread acceptance in both telecom and enterprise environments as the control protocol for the creation, migration, modification and termination of multimedia sessions among different endpoints. End-points for SIP-based signaling are typically identified by SIP URIs, which are similar to HTTP URIs and are of the form *sip:user@domain*.

2.1 SIP Presence Architecture

SIP has recently been extended [14] as a unified signaling substrate for services such as Instant-Messaging and Presence. Presence fundamentally denotes the ability of SIP end-points (user devices) to indicate their up-to-date network status and associated attributes (e.g., location, link bandwidth) to a designated Presence server, and the corresponding ability of other SIP end-points to receive notification of such status changes. SIP Presence can thus be viewed as a pub-sub mechanism for dynamic network-related events.

[2]pronounced "raker"

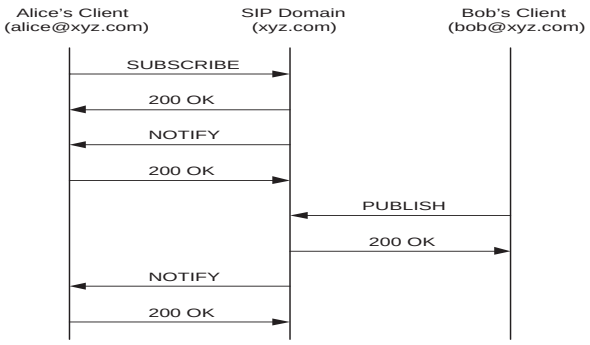

Figure 1. Basic Event Publish/Subscribe for SIP-based Presence

Figure 1 establishes the fundamental message sequence in SIP for supporting presence-based applications. In SIP, the presentity (Bob in Figure 1) refers to the source of presence information. Similarly, the subscriber (Alice in Figure 1) refers to the other end-point interested in tracking the dynamic changes in a presentity's user status. A presentity transmits its status to the intermediate SIP server via a PUBLISH message, while the subscriber issues a SUBSCRIBE message specifying the events on which it requests notification. When Bob's PUBLISH message satisfies the predicate in Alice's subscription, the Presence server will use a NOTIFY message to inform Alice of the updated state. The presence information is published in an XML-based extensible format, and can thus include an arbitrary number of presence attributes. When the Presence servers are distributed (for example, when different users PUBLISH to different servers), additional intelligence is needed to match published information to existing subscriptions.

2.2 LRF Application Overview

LRF applications match a consumer's request for a particular type of Live Resource (vendor) to a nearby and available resource (vendor). As a detailed description of LRF applications is not possible, we limit our exposition to the features relevant to our resource management problem. The main novelty in LRF applications is the incorporation of the resource's current location and other contextual attributes in the matching process, making the service relevant to nomadic vendors such as plumbers, electricians or auto mechanics, as captured through the following usage scenario.

Alice, currently located at point X, issues an SMS for "plumber" indicating her desire to locate a nearby

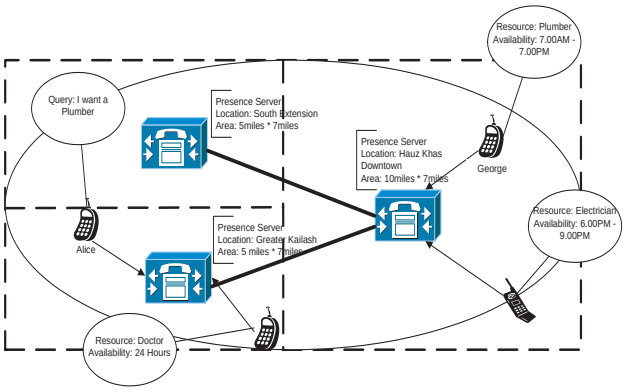

Figure 2. LRF Query Support on Distributed Presence Architecture

plumber. The telecom service provider uses localization technology to track both her location, as well as the location of cellphones of registered plumbers. Her location is then matched with that of a nearby available *vendor (say George), and George's number is returned to Alice. Availability may be expressed via either profiles (e.g., plumber George indicates that he's available from 7am-7pm on weekdays) or via dynamic messages (e.g., George sends an SMS indicating that he's free to take another job). Moreover, the LRF application also collects user feedback to rank vendors, and can use this ranking in the matching process.*

Figure 2 outlines the architectural interaction of an LRF application with the distributed Presence infrastructure. We assume that each presence server is responsible for a certain partition of the overall geographic space. Hence, all vendors currently located within the zone of a presence server dynamically update it with their changing presence attributes. In the figure, Hauz Khas, South Extension and Greater Kailash refer to three popular residential areas located in New Delhi, India. For the sample LRF application, the presence attributes include a vendor's location, availability and the type of service offered (e.g. plumber).

The LRF application runs in a distributed manner. Each application instance is associated (logically) with a specific presence server and tracks the available resources (vendors) within the corresponding region. Each customer requesting the matching service issues a "query" for a vendor to the application instance associated with his current location. This application instance first tries to return a "locally available" vendor from the local presence server–the matching process on the local server may use well-known spatial matching techniques (e.g., spatial databases [9]). This local matching is, however, not the focus of this paper. Instead, we focus on the case where the local presence server is unable to provide any available resources, implying that the query must be routed to alternative presence servers.

3 Architectural Considerations for Routing of LRF Queries

A routing architecture for LRF queries, invoked whenever a request for a resource cannot be satisfied locally, should exhibit the following characteristics:

- *Handle snapshot nature of queries*: Traditional pub-sub systems, which set up data routing trees for long-lived subscriptions, are ill-suited for the 'snapshot' queries issued by LRF applications.

- *Utilize spatio-temporal distribution of resources*: A server's interest in the presence data of another server is load-dependent. Hence, server A is interested in the available resources of server B only during a resource crunch. Traditional pub-sub systems are not equipped to handle such spatio-temporal variations. A routing infrastructure for LRF applications should, however, utilize the skews in resource availability to intelligently forward queries.

- *Exploit the application semantics of LRF queries*: LRF applications usually do not require stringent guarantees–practically speaking, we do not need to match to the "guaranteed closest" available vendor. Rather, a "reasonably close" but "available" vendor would usually suffice.

- *Enable Quality-of-Response (QoR) driven routing to nearby servers*: Since LRF applications aim to match to vendors in the vicinity, the matching process must factor in a Quality of Response (QoR) metric associated with the available resources, where QoR may be determined by a combination of attributes like the vendor's location, his availability and/or reputation. In this work, we explain how such a QoR value is used in the query routing/matching process, rather than focus on the specifics of QoR computation.

- *Avoid query flooding*: A naive approach is to simply route a query to the next "geographically" closest presence server - i.e., the presence server managing the region closest to the requester. Unfortunately, this approach ignores the *consumption*

of LRF queries. Approaches such as flooding or expanding ring search, which ignore the spatio-temporal distribution of resources, are less useful and waste bandwidth (e.g., if the "closest" server has no available plumbers).

- *Avoid presence data flooding*: Alternatively, each server may flood its presence data (the state of all associated vendors) to all other (nearby or remote) servers, thus effectively providing each server with global (or semi-global) knowledge of the presence state. This approach is not scalable as presence documents are highly *ephemeral*. Moreover, this approach requires complex resource-reservation protocols to ensure consistency across distributed queries (e.g., to ensure that the same plumber isn't surfaced to two requesters).

  LRF applications thus impose some new constraints on previous load-aware resource discovery approaches. For example, the Intentional Naming System (INS) [1] describes a hierarchical (attribute, value) based naming scheme where state updates for individual resources are esentially *flooded* over a resolver tree. To reduce the flooding load in environments with high update rates (as would be the case with presence data for different vendor classes), INS proposes the use of virtual namespaces to partition the set of servers that need to maintain (weak) consistency. Such namespace partitioning is adequate for queries with equality predicates (e.g., "find me a printer in this room"), but not for LRF queries which may have a "nearest neighbor" semantic defined over the entire geographical space.

4 The RAQR Algorithm

Based on the above observations, we have designed the Resource Aware Query Routing (RAQR) algorithm for LRF applications. RAQR utilizes the spatio-temporal distribution and consumption of resources, in forwarding LRF queries to alternate presence servers. It avoids query and presence information flooding, and utilizes QoR metrics to select the right presence server to match a query. The main feature of RAQR is the on-demand formation of the *gradients*, that point to servers having a surplus of resources. Gradients are created for a particular server only when it is projected to face a crunch in its local resources (relative to the request arrival load) and are destroyed whenever the crunch condition ceases to exist (either due to an increase in the arrival rate of new vendors or a reduction in the request arrival load). Note that the individual vendors themselves are mobile, and thus associate with distinct presence servers dynamically. Since servers maintain presence information as soft-state (with expiration times specified in the presence document), the vendor's information at the previous presence server is automatically erased.

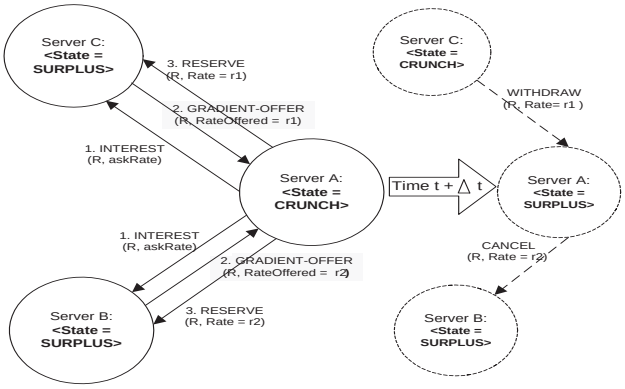

Figure 3. Basic Steps and Messages in RAQR

For creating gradients, RAQR employs a diffusion-based technique where the originating server (the one facing crunch) injects an INTEREST message (informing it's need for a certain resource), which is propagated over the network. Nodes receiving the INTEREST message respond directly to the originating server with a GRADIENT-OFFER message, offering their services, only if they have an available pool of resources. An intermediate node relays an INTEREST only if it is unable to completely satisfy the original request rate–the parameters of this relayed message are adjusted to reflect the residual demand. On receiving the GRADIENT-OFFER responses, the originating server uses the QoR values of the servers to determine the best set of servers that can satisfy its request load and then sets up direct **gradients** to these servers (using a RESERVE message). Of course, as the resource levels (of available vendors) fluctuates, nodes that had earlier offered their services may WITHDRAW their offer, thereby requiring the originating server to issue new INTEREST messages. Similarly, if the originating server feels that the resource crunch is over, it may issue CANCEL messages to servers on which it had previously reserved resources.

Figure 3 shows the basic operations of RAQR. At time t, server A is resource crunched, and sends INTEREST messages (with $askRate$ denoting the additional resources required by A) to B and C. Since B and C have a surplus of resources, they respond with

GRADIENT-OFFER messages (rates $r1$ and $r2$)[3]. Server A reserves the corresponding rates along these gradients, and uses these to forward queries. At time t+$\triangle t$, server A has surplus while C faces a crunch. In this case, C issues a WITHDRAW to A. At the same time, server A sends a cancel to B as it no longer requires these resources.

4.1 RAQR Design Details

RAQR assigns different states to different servers. For ease of explanation, we focus on only one type of vendor. Let N_i denote the number of available vendors at server i (denoted by S_i). The actual determination of "crunch" or "surplus" states includes the use of both N_i, the current rate of change in N_i, as well as lower and upper thresholds T_L and T_H as follows:

$$\text{if } N_i < T_L \ \& \ \frac{dN_i}{dt} < 0, \quad \text{then} \quad \text{state= CRUNCH}$$

$$\text{if } N_i > T_H \ \& \ \frac{dN_i}{dt} > 0, \quad \text{then} \quad \text{state= SURPLUS}$$

T_L and T_H are threshold values below and above which a server *may* declare itself in CRUNCH and SURPLUS states respectively. For each resource type, each server maintains two separate lists: *suppliers*-listing other servers to which it has established gradients, and *dependents*-listing other servers which have established gradients to it.

4.1.1 Resource Distribution and Gradient Setup

In RAQR, each presence server periodically performs a *health* check (determined by a time period $T_{monitor}$) to determine any changes to its current state, and then initiates the relevant associated actions. If S_i identifies itself in the CRUNCH state, it propagates an INTEREST (resourceID, askRate) to its known neighbors. *askRate* is the rate at which the available resources at S_i are being consumed. In response to such a request, other servers reply with GRADIENT-OFFER messages, that include an average *Quality-of-Response (QoR)* value of the *supplier* (or responding) server (say S_j) and the *RateOffered* offered by S_j. After waiting to collect enough such GRADIENT-OFFERs, S_i sorts the offer in the ascending order of QoR values and sends out RESERVE (resourceID, providedRate) messages to the first n servers that cumulatively meet the askRate. It also adds these servers to its *suppliers* list. The RESERVE message essentially earmarks resources for the requester and is needed due to the consumptive nature of LRF queries–an absence of an explicit RESERVE response allows a SURPLUS server to offer its resources for subsequent requesters facing a resource crunch.

A server S_j on receiving an INTEREST message takes actions depending on the state that it currently is in. It simply relays the INTEREST message unchanged to its one-hop neighbors if either:

1. $S_j.suppliers.size \neq 0$ i.e., if S_j itself has a current shortage of vendors; OR
2. $N_j < T_L$, i.e., if its available supplier pool isn't large enough; OR
3. $\frac{dN_j}{dt} < 0$ i.e. if it's own pool of resources is dwindling with time, indicating resource shortage.

S_j replies back with a GRADIENT-OFFER with *RateOffered=askRate* (i.e., indicating its ability to fully support the deficit of S_i) if either:

1. $\frac{dN_j}{dt}(N_j) - askRate > 0$ i.e., S_j's pool of available vendors is increasing more rapidly than the *askRate*; OR
2. $\frac{N_j}{|\frac{dN_j}{dt} - askRate|} \geq T_{depl}$, i.e., S_j does not anticipate its resource pool running out until T_{Depl} time has elapsed.

Here, T_{depl} is an algorithm parameter, indicating the minimum acceptable time until complete exhaustion of resources. Intuitively, T_{depl} is a large value within which S_j can expect the underlying rates of supply and demand to change so that prediction of system state beyond this time is unnecessary.

If S_j cannot provide the entire *askRate*, but satisfy only part of it, then it replies back with a GRADIENT-OFFER with the maximum rate it can provide. Additionally, it forwards the INTEREST message to it's nearby servers with a *newAskRate=askRate - RateOffered*. Server S_j on receiving a RESERVE message from S_i adds it to it's *dependents* list. In this way, the INTEREST message is diffused minimally through the network, until it visits a set of servers who can collectively satisfy the vendor deficit at the requester. Note that the use of neighbor broadcast implies that S_i typically receives several GRADIENT-OFFERs, with cumulative rates higher than its advertised need; the use of the subsequent RESERVE message avoids over-reservation of resources for S_i.

4.1.2 Gradient Cancellation

Temporal variations in resource availability and query rates will change conditions, whereby CRUNCH servers

[3]For simplicity, assume that $askRate = r1 + r2$.

become SURPLUS servers, or vice versa. During the periodic *health* check, if S_i discovers that it is in the CRUNCH state AND still has a set of *dependent* servers, it sends out WITHDRAW (resourceID, RateOffered) to the respective servers and removes them from the list. Similarly, a server discovering finding itself in the SURPLUS state AND having a set of *suppliers* sends out CANCEL (resourceID, RateOffered) message to the servers in the *suppliers* list.

4.1.3 Choice of RAQR Parameters

The performance of RAQR will clearly depend on the appropriate choice of T_L and T_H. Distinct values of T_L and T_H introduce hysteresis in RAQR's behavior, thus avoiding rapid oscillations between CRUNCH and SURPLUS states. If T_L is too low, a server will mark itself as resource-constrained unnecessarily. RAQR's current implementation monitors the fluctuating resource levels at a vendor periodically (with time period $T_{monitor}$). Accordingly, T_L should be set to be a relatively small multiple, κ (for example, $\kappa = 5$ or 6), of the "smoothed" (average) resource depletion rate $\frac{dN_i}{dT}$– allowing a truly underloaded server enough time to set up gradients to other servers in advance. Of course, $\kappa \ll T_{depl}$, as we have implicitly assumed that the demand and supply workloads for LRF applications are not predictable beyond time-periods longer than T_{depl} with acceptable accuracy. Thus, we assume that an LRF application exhibits *piecewise-constant* query load rates, with T_{depl} denoting the interval over which a particular rate remains constant. Similarly, T_H should be greater than $2T_L$ (to prevent oscillations), but smaller than the "smoothed" number of vendors that typically register with server S_i (otherwise, it will never be in the SURPLUS state). While RAQR parameters should ideally be tuned adaptively at each server, we plan to investigate the optimal tuning of parameters in response to real-life workload fluctuations in later work.

4.2 QoR-driven Query Propagation

LRF queries coming to servers from clients are first satisfied by local pool of resources. Gradients are used when the local resource pool is over. A server looks up its list of *suppliers* and forwards the query to the server promising the best average QoR, working its way down the list until a match is found. While we currently follow the policy of terminating at the first matched reply, other variations (e.g., probing all (or a select set of) servers in the *suppliers* list and sending back the best reply obtained) are certainly possible. The query fails if no available resources are obtained from a server in the *suppliers* list.

Parameter	**Value**
Number of presence servers	25
Number of queries per server	20000
Total number of resources	500
T_L	10
T_H	20
T_{depl}	$5T$
$T_{monitor}$	$10T$
$T_{disseminate}$	$100T$

Table 1. Parameters for experimental setup. All periods are expressed as a factor of T, where T corresponds to the length of a clock tick in the simulation.

5 Experimental Results

To evaluate the performance of RAQR relative to alternative dynamic resource discovery protocols, we performed extensive simulation-based studies. For the results reported here, an overlay network of 25 presence servers were connected to one another using GT-ITM [16], a well–known tool for generating representative internetwork topologies. The entire service provider coverage area was represented by a (25000x25000) grid, with each of the presence servers managing an identically-sized partition of the grid. The vendor population consists of 500 users, whose movement is governed by the Random Waypoint Mobility model[2]. Whenever a particular vendor is matched to a user request, its availability status changes to "unavailable" for the duration $t_{service}$. At the end of the duration, the vendor is made "available" at the presence server which originated the query, i.e. in the region where the vendor provides the requested service. XXXProvide numbers for the parameters chosen for the Random Waypoint model.XXX In our initial performance studies, we assume that all responses have identical QoR values–in subsequent work, we shall explore how surplus servers may compactly advertise the QoR of their resource pool, and how this may be used by a server facing potential resource depletion.

Competitive protocols: We studied the following additional representative protocols that perform query propagation or resource (information) dissemination. We summarize these protocols below:-

- *Broadcast Query Propagation (BCast)*: If a resource is locally unavailable at server S_i, a query is broadcast to all presence servers in the network. A server S_j responds to the query with the best available resource (if one is available). While S_i can potentially receive multiple responses, our BCast implementation matches the query to the first available resource returned by any server.

- *Expanding Ring Search (ERS)*: Since network-level broadcast can be expensive, ERS employs a query propagation strategy where a locally failed query is first forwarded to all the one–hop presence servers. If none of the one–hop neighbors have available resources, it is forwarded to the two–hop neighbors and so on. As in broadcast, the original query is matched to the earliest response from a server with an available resource.

- *Selective Resource Dissemination (1 Hop) (SRD-1)*: In this scheme, each server S_i periodically disseminates information about its available resources to its one–hop neighbors. Specifically, S_i advertises its locally available resource count N_i with a periodicity $T_{disseminate}$. A neighboring server S_j updates its supplier list with the value received. When S_j has no local resources available for a query, it forwards the query to all suppliers in its one-hop neighborhood and matches to the earliest response with an available resource. If none of these one-hop suppliers have an available resource, we consider it as a *query miss*.

- *Selective Resource Dissemination (2 Hops) (SRD-2)*: This strategy is similar to SRD-1, the only difference being that a server periodically disseminates its resource information to its one–hop, as well as, two–hop neighbors. Intuitively, this increases the pool of suppliers for each server, at the cost of increased message overhead.

Evaluation Metrics: BCast and ERS result in zero query misses whenever the network has at least one available resource. However, for SRD-1, SRD-2, and RAQR, a missed query occurs when there are no available resources locally and with the list of suppliers. Accordingly, *query miss* rate is an important performance metric. Further, a resource crunched server might forward a query to a supplier and get a failed response (either because the resource(s) moved to a different region or are currently unavailable). We term this miss as a *supplier miss* and measure the number of supplier misses for SRD-1, SRD-2 and RAQR. Additionally, we measure the number of *control messages* and *query messages* that are generated for each protocol. While control messages are incurred by the protocols that exchange resource information, i.e., SRD-1, SRD-2, and RAQR, query messages represent the overhead of routing requests to additional presence servers. Finally, for each successful query, we measure the QoR as the physical (geographic) distance between the server that originates the query and the server providing the resource; a lower QoR implies a better (closer) match.

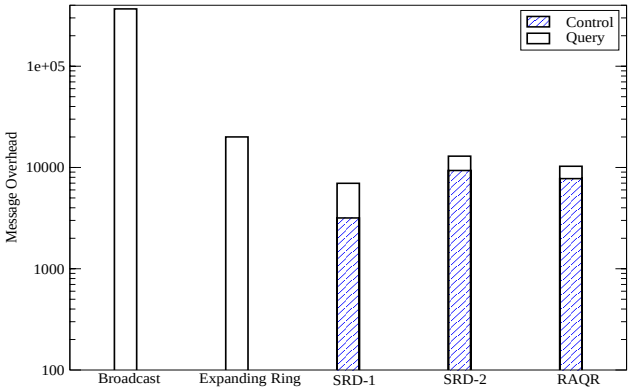

Figure 4. Message Overhead of BCast, ERS, SRD-1, SRD-2, and RAQR for $\lambda = 10req/T$.

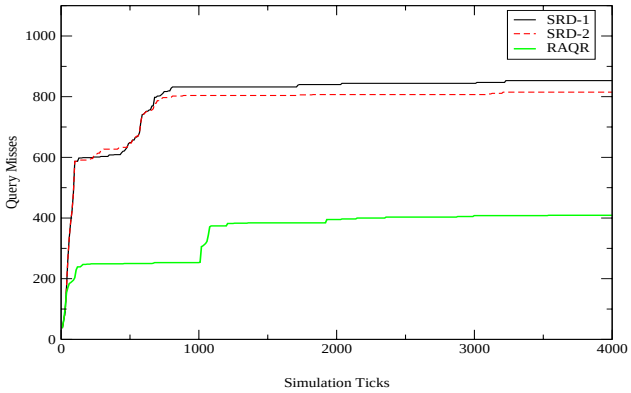

Figure 5. Query Misses of SRD-1, SRD-2, and RAQR for $\lambda = 10req/T$.

Table 1 shows the parameter settings that are used by the different protocols in our experiments. XXX For a preliminary study of RAQR's sensitivity to parameter variations, we also experiment with $T_L = 5, 20$, keeping $T_H = 2T_L$ at all times. XXX For the resources, we assume that $t_{service}$ follows an exponential distribution with a mean of $10T$.

Query Load Generation: To evaluate the perfor-

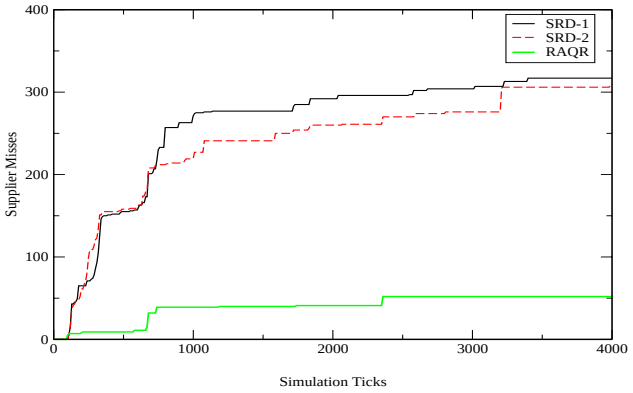

Figure 6. Supplier Misses of SRD-1, SRD-2, and RAQR for $\lambda = 10req/T$.

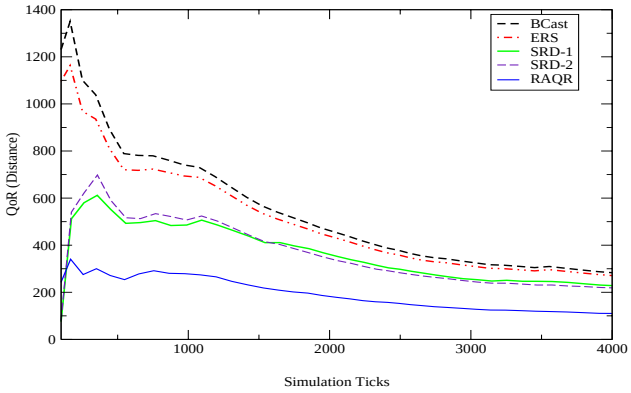

Figure 7. Average QoR of BCast, ERS, SRD-1, SRD-2, and RAQR for $\lambda = 10req/T$.

mance of the different protocols, we need to vary the spatio– temporal distribution of LRF queries generated in the network. Since the number of resources in the network is fixed, some servers can be exhibiting a CRUNCH due to a high rate of queries, while other servers are concurrently in a SURPLUS state. To capture this behavior, each server has an independent LRF query generator. We divide the entire simulation period into time intervals. During the interval T_j, queries arrive at a server S_i according to a Poisson process, i.e. exponential inter-arrival time with mean $1/\lambda_{i,j}$. Further, for different T_j's, the value of $\lambda_{i,j}$ is randomly chosen from a normal distribution, with mean λ and variance σ^2. The resulting query distribution exhibits both temporal variations at a particular server, as well as skews in the query rate (i.e. query load) across different servers. Choosing appropriate values for λ and σ^2 allows us to control both the average system query load and the load skew across servers.

5.1 Results and Observations

We first choose $\lambda = 10$ (req/T) and $\sigma^2 = 10$, and compare the performance of the different protocols for the resulting query load. Figure 4 shows the message overhead of each protocol on a logarithmic scale. While BCast is prohibitively expensive, the number of query messages in ERS is about twice the total messages exchanged by any of SRD-1, SRD-2 and RAQR. This points to the need of proactively disseminating resource information in such environments to better support LRF queries. Figure 5 shows that RAQR achieves a 50% reduction in query misses when compared to SRD-1 and SRD-2. At the same time, the total message overhead of RAQR lies in between SRD-1 and SRD-2 – this points to the fact that RAQR does a significantly better job of successfully matching LRF queries, while avoiding the overheads incurred by BCast and ERS.

XXX I am suggesting that you simply add 2 additional points on the x-axis for RAQR in Figure 4– namely, RAQR ($T_L = 5$) and RAQR ($T_L = 10$), keeping $T_H = (20, 40)$–this will give a quick overview of RAQR's sensitivity. If you do this, then also add a line explaining the results in the previous para. XXX

Figure 6 helps us better understand the improvement delivered by RAQR in terms of supplier misses. We observe that SRD-1 and SRD-2 result in 6 times as many supplier misses as RAQR. Clearly, RAQR is better at continually tracking the available resource levels at various relevant servers than SRD-1 or SRD-2 (neither of which track the dynamic evolution of resources). Finally, in Figure 7, we plot the average QoR in terms of the geographic distance between the query and the resource that is "discovered" by each protocol. Once again, RAQR leads to better matches (a QoR that is at least 50% lower than any of the other protocols), thus demonstrating its suitability for the vicinity-based matching semantics required in LRF. The QoR of BCAST and ERS is much higher due to interaction between the network topology and policy of matching to the first server response–geographically distant servers are often directly connected on the overlay and have mutually shorter response latencies.

To understand the effect of query load on the behavior of the protocols, we next study the performance of SRD-1, SRD-2 and RAQR for increasing query loads in the network. This is achieved by varying λ for each server to take values of $2, 5, 10, 15$

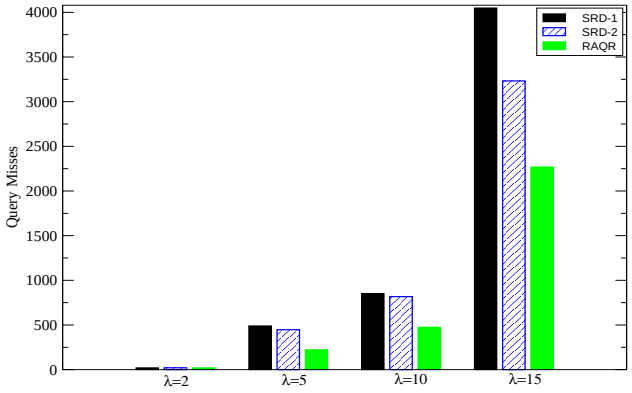

Figure 8. Query Misses of SRD-1, SRD-2, and RAQR for varying load conditions.

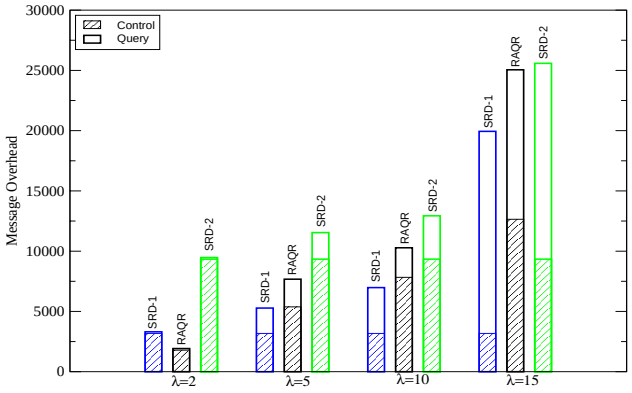

Figure 9. Message Overhead of SRD-1, SRD-2, and RAQR for varying load conditions.

req/T [4],respectively. Figures 8 and Figure 9 show the query misses and message overhead, respectively, for the three protocols. Across all load levels, RAQR continues to uniformly result in a much lower rate of query misses, while incurring comparable message overheads. RAQR thus appears to adapt very well to spatio-temporal variations in both LRF query loads and resource (vendor) distributions.

6 Related Work

There has been significant work in the area of content/context-based pub-sub queries [10, 4, 12], that focuses on delivering events from sources to interested users. However, these approaches focus only on event delivery (queries do not consume the resource) and assume relatively long-lived subscriptions. In contrast, LRF queries are essentially snapshot queries that have different requirements in terms of application semantics.

[10] presents a generic framework of an event-brokering system for content-based pub-sub in mobile environments, that covers both centralized and distributed brokers. The paper also examines the concept of "quenching" whereby the number of in-network publish messages can be reduced by performing partial matching at the brokers. For mobile environments, where maintaining the right delivery path is the major challenge problem, [5] proposes the use of Content-Based addresses. Similarly, filter-based routing has been proposed in [4, 7] for the same objective. However, these approaches do not consider the need for load-based query redirection and QoR-driven matching generated by LRF applications.

In unstructured P2P systems, [6] introduces the concept of routing indices (which are "hints" about the kind of information that neighbors hold) to forward queries. [12] takes it a step further by using multi-level Bloom filters to summarize the information/content of nodes. However, these approaches do not provide efficient means to satisfy "nearest" or vicinity-based matching. [3, 13, 17] have presented work using DHTs to form the routing substrate for SIP messages. While interesting, these approaches focus on a different problem–efficient partition of the SIP address space among heterogeneous servers–and assume that the destination URI of each query is known. In contrast, we focus on the efficient resolution of vicinity-based matching requests, where the target URI is unknown.

Our diffusion-based interest propagation approach extends recent work (e.g., [11, 15]) on efficient data dissemination protocols for sensor networks. For example, [11] proposes to establish routes from sources to queries (sinks) by diffusing the query (interest) and subsequently reinforcing preferred paths. However, unlike [11], which assumes that queries have a well defined "zone of interest" within which an interest is diffused, RAQR only *minimally diffuses* queries to a neighborhood large enough to satisfy the current resource crunch. Similarly, [15] creates a distributed forwarding mesh (as opposed to a tree) from the sources to the sink by including a cost metric in the initial flooding of the interest, and having intermediate nodes insert themselves into potential paths only as long as the cumulative path cost does not exceed this metric. In a similar fashion, RAQR also requires intermediate nodes to modify the *askRate*, thus propagating a resource request only to the extent necessary. Unlike the

[4]Note that, there are 20 resources available on an average with each server.

event notification scenarios addressed by [11, 15], LRF applications actually *consume* resources–accordingly, RAQR uses an elaborate RESERVE-CANCEL primitive over the diffusion substrate to ensure that queries do not overbook resources on nearby nodes.

7 Conclusions

The main novelty of RAQR is that it effectively adapts to (a) the spatio-temporal distribution of mobile resources, and (b) the load skew across presence servers, to improve resource provisioning and query efficiency for LRF applications. RAQR further enables the use of a *quality of response (QoR)* metric to determine the best server(s) to which queries are forwarded, thereby making the protocol sensitive to location and other contextual parameters associated with individual resources. Finally, unlike other resource discovery protocols, control overhead in RAQR depends on the resource distribution in the environment. Simulation studies demonstrate RAQR's promise as a technique for supporting context-dependent and vicinity based queries in a distributed presence infrastructure. In the future, we plan to extend the QoR computation to include other contextual attributes like reputation (ranking) of the resources. We would also like to understand the principles behind the choice of parameters in RAQR, based on resource distribution and their consumption, and study RAQR's *sensitivity* to these parameters. Finally, we plan to implement RAQR on a SIP–based testbed that supports dynamic selection, evaluation and matchmaking in converged networks.

8 Acknowledgements

The authors would like to thank Amit Purohit for his active participation in the design and implementation of the simulation framework, and Sumit Mittal for his contributions to developing the ideas in the paper.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Liley. The designa nd implementation of an intentional naming system. In *ACM SOSP*, December 1999.

[2] C. Bettstetter and C. Wagner. The spatial node distribution of the random waypoint mobility model. In *Workshop on Mobile Ad-Hoc Networks (WMAN)*, March 2002.

[3] D.A. Bryan, B.B. Lowekamp, and C. Jennings. SOSIMPLE: A serverless, standards-based P2P SIP communication system. In *International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, June 2005.

[4] F. Cao and J.P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *IEEE INFOCOM*, March 2004.

[5] A. Carzinagi, MJ. Rutherford, and AL. Wolf. A routing scheme for content-based networking. In *IEEE INFOCOM*, 2004.

[6] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *22nd IEEE International Conference on Distributed Systems*, 2002.

[7] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, 2001.

[8] J. Rosenberg et al. SIP: Session initiation protocol. RFC 3261, IETF, June 2002.

[9] R.H. Guting. An introduction to spatial database systems. *Very Large Data Bases. Springer Verlag*, October 1994.

[10] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks Issue on Pervasive Computing and Communications*, 10(6):643–652, November 2004.

[11] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking*, 11(1):2–16, February 2002.

[12] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *9th International Conference on Extending Database Technology (EDBT), Greece*, March 2004.

[13] S. Marti, P. Ganesan, and H. Garcia-Molina. SPROUT: P2P routing with social networks. In *First International Workshop on Peer-to-Peer Computng and Databases*, March 2004.

[14] J. Rosenberg. A presence event package for the session initiation protocol (SIP). In *RFC 3856, IETF*, August 2004.

[15] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *ACM WINET (Wireless Networks)*, 11(2), March 2005.

[16] Ellen W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. *IEEE Infocom, San Francisco, CA*, March 1996.

[17] R. Zhang and Y. C. Hu. HYPER: A hybrid approach to efficient content-based publish/subscribe. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 427–436, 2005.

[18] Wei Zhuang, Yung-Sze Gan, Qing Gao, K.J. Loh, and K.C. Chua. Multi-domain policy architecture for ip multimedia subsystem in umts. In *Proceedings of the IFIP TC6 / WG6.2 and WG6.7 Conference on Network Control and Engineering for QoS, Security and Mobility*, volume 235, pages 27–38, 2002.