Exploiting Epidemic Data Dissemination for Consistent Lookup Operations in Mobile Applications*

Christoph Lindemann and Oliver P. Waldhorst
http://mobicom.cs.uni-dortmund.de
Department of Computer Science, University of Dortmund, 44227 Dortmund, Germany

This paper presents a general-purpose distributed lookup service, denoted Passive Distributed Indexing (PDI). PDI stores entries in form of (key, value) pairs in index caches located at mobile devices. Index caches are filled by epidemic dissemination of popular index entries. By exploiting node mobility, PDI can resolve most queries locally without sending messages outside the radio coverage of the inquiring node. For keeping index caches coherent, configurable value timeouts implementing implicit invalidation and lazy invalidation caches implementing explicit invalidation are introduced. Inconsistency in index caches due to weak connectivity or node failure is handled by value timeouts. Lazy invalidation caches reduce the fraction of stale index entries due to modified data at the origin node. Similar to index caches, invalidation caches are filled by epidemic distributions of invalidation messages. We evaluate the performance of PDI for a mobile P2P file sharing a mobile instant messaging application. Simulation results show that with the suitable integration of both invalidation mechanisms, up to 80% of the lookup operations return correct results and more than 90% of results delivered by PDI index caches are up-to-date.

I. Introduction

Many distributed applications require global resolution of application-specific keys to application-specific values, a functionality provided by a lookup service. Perhaps the most prominent lookup service is the Domain Name System (DNS, [11]), which resolves each host name to the corresponding IP address. Similarly, instant messaging systems require a one-to-one mapping of a user ID to the current user's terminal or presence state. Further examples of lookup services include Internet search engines and distributed information retrieval systems, which provide for each query consisting of keywords a one-to-many mapping to matching documents. Recent research efforts in peer-to-peer technology [14], [15] aim at building Internet-scale distributed hash tables, which provide a general-purpose approach for mapping keys to values.

In mobile and wireless environments, weak connectivity or even disconnected operation hampers the employment of a centralized lookup service. The smart collaboration of mobile devices in an ad hoc fashion constitutes an attractive alternative for implementing an effective distributed lookup service for such scenarios. Suitable methods for implementing such collaboration constitute epidemic algorithms. Such algorithms transmit information when nodes get in direct contact, similar to the transmission of an infectious disease between individuals. Mathematical models for the spread of epidemic diseases have been widely studied. There exist applications of epidemic algorithms in various fields of computer science, e.g., for the maintenance of replicated databases [2]. Papadopouli and Schulzrinne introduced seven degrees of separation (7DS), a system for mobile Internet access based on web document dissemination between mobile users [12], [13]. To locate a web document, a 7DS node broadcasts a query message to all mobile nodes currently located inside its radio coverage. Recipients of the query send response messages that contain file descriptors of matching web documents stored in their local file caches. Subsequently, such documents can be downloaded with HTTP by the inquiring mobile node. Web documents may be distributed to other nodes that move into radio coverage, implementing an epidemic spread of information.

Using a related approach, Goel, Singh, Xu and Li proposed broadcasting segments of shared files using redundant tornado encoding [4]. Their approach enables nodes to restore a file, if a sufficient number of different segments have been received from one or more sources. Khelil, Becker, Tian, and Rothermel presented an epidemic model for a simple information diffusion algorithm [8] inspired by the SPIN-1 protocol [5]. Both systems implement a push model for information dissemination. That is, shared data is advertised or even actively broadcasted without a node requesting it. Recently, Hanna, Levine, and Mamatha proposed a fault-tolerant distributed information retrieval system for peer-to-peer document sharing in mobile ad hoc environments [6]. Their approach distributes the index of a new document to a random set of nodes when the document is added to the system. The complete index of a document, i.e., all keywords

* This paper is an extended version of the paper "Consistency Mechanisms for a Distributed Lookup Service supporting Mobile Applications" that appeared in the *3rd Int. ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 2003).*

matching it, constitutes the smallest unit of disseminated information.

In this paper, we introduce a general-purpose distributed lookup service, denoted Passive Distributed Indexing (PDI). Building upon [9], PDI stores index entries in form of (key, value) pairs in index caches located at each mobile device. Index caches are filled by epidemic dissemination of popular index entries. By exploiting node mobility, PDI can resolve most queries locally without sending messages outside the radio coverage of the inquiring node. Thus, PDI effectively reduces network traffic for the resolution of keys to values for applications possessing a sufficiently high degree of temporal locality in their query streams as known for web content, Internet search engines and P2P file sharing systems [16], [18]. Beyond [9], we generalize the semantics of key-to-value matching beyond the simple matching of keywords to document-sources. Subsequently, PDI supports arbitrary application-specific keys to be matched to application-specific values. For keeping index caches coherent, we introduce two novel consistency mechanisms for PDI index caches: (1) configurable value timeouts implementing implicit invalidation and (2) lazy invalidation caches implementing explicit invalidation. Inconsistencies in index caches due to weak connectivity or node failure are handled by value timeouts. Lazy invalidation caches reduce the fraction of stale index entries due to modified data at the origin node. Similar to the epidemic distribution of index entries, invalidation caches are filled by epidemic distributions of invalidation messages.

As for PDI, a sufficiently high degree of locality in user queries is essential for all related approaches [4], [6], [8], [12], [13]. Opposed to 7DS [12], [13], PDI implements document search based on a distributed index rather than immediate document sharing. In fact, PDI disseminates individual index entries where as 7DS disseminates complete documents. Opposed to [4] and [8], PDI implements a pull model and does not advertise data yielding a substantial reduction of traffic over wireless links. In contrast to [6], PDI disseminates just a single key and one or more matching values rather than the complete index of a document. Furthermore, opposed to all related work, PDI provides effective means for coping with stale index entries due to weak connectivity, node failure, and modified data. Moreover, opposed to all related work, PDI cannot only be employed for P2P file sharing, but also for numerous other mobile applications like mobile instant messaging or a mobile information portal.

In a comprehensive simulation study using ns-2, [3], we investigate the performance of PDI for two selected mobile applications. As workload model for the simulation studies, we present a general analytical framework to capture the workload generated by mobile applications. We customize the framework for a mobile peer-to-peer (P2P) file sharing application and a mobile instant messaging (IM) application, and apply the customized framework subsequently for workload generation in the simulation studies. The presented performance curves show that for both applications the presented invalidation messages reduce stale hits by up to 80%. As a consequence, PDI resolves up to 80% of the requests for key-to-value resolutions correctly. Furthermore, the number of stale values returned on a request is below 10%.

The remainder of this paper is organized as follows. Section II summarizes the basic concepts of PDI. Section III introduces the two novel consistency mechanisms for PDI, value timeouts and lazy invalidation caches. In Section IV, an analytical framework for characterizing mobile applications is derived and customized for a mobile P2P file sharing and a mobile IM application. Based on the customized framework, performance results for PDI and each of the applications are presented in Section V. Finally, concluding remarks are given.

II. Basic Concepts of PDI

We assume a system consisting of several mobile nodes, e.g., mobile users equipped with notebooks or PDAs and wireless network interfaces as illustrated in Figure 1. All mobile nodes collaborate via a shared application that uses a distributed lookup service. Radio coverage is small compared to the area covered by all nodes, so that most nodes cannot contact each other directly. Thus, communication may be performed using several intermediate hops as in mobile ad hoc networks (MANET, [10]). Subsequently, we assume IEEE 802.11x as underlying radio technology [7]. However, we would like to point out that PDI could be employed on any radio technology that enables broadcast transmissions inside a node's radio coverage.

PDI implements a general-purpose lookup service for mobile applications. In general, PDI stores index entries in the form of pairs (k,v). Keys k and values v are both defined by the mobile application. In case of file sharing, keys are given by keywords derived from the file name or associated meta-data. Values are given by references to files in form of URIs. Opposed to distributed hash-table systems [14], [15], that require a one-to-one matching of keys and values, PDI does neither limit the number of keys matching a value nor the number of values matched by a key. Thus, PDI can implement more sophisticated query semantics, e.g., logical AND and OR concatenations of keys. However,

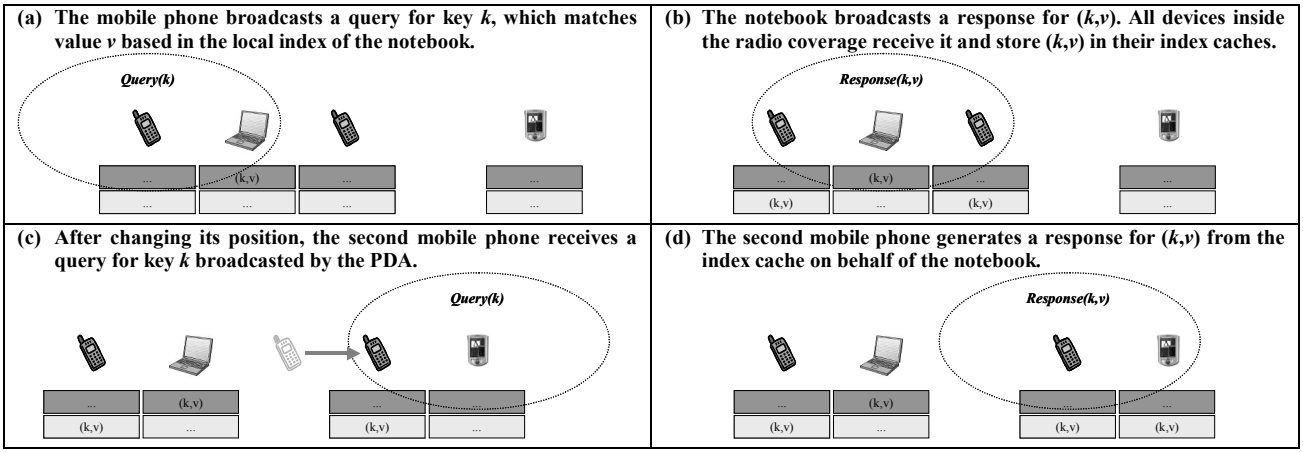

Figure 1. Illustration of epidemic information dissemination with PDI

some mechanisms introduced in Section III require that a value is unique in the system, i.e., it is only added to the system by a single node. This can be easily achieved by extending the application specific value v by a unique node identifier i_n for node n. For example, the node identifier i_n may be derived from the node's IP address or the MAC address of the radio interface. For ease of exposition, we will abbreviate the unique value given by (v, i_n) pairs just by v.

A node n may contribute index entries of the form (k,v) to the system by inserting them in a *local index*. In Figure 1, the local index is drawn as the first box below each mobile device. We refer to an index entry in the local index as supplied. The node n is called the origin node of the index entry. For example, the notebook shown in Figure 1 is the origin node of the index entry (k,v). A key k matches a value v, if (k,v) is currently supplied to the PDI system. Each node in the system may issue queries in order to resolve a key k to all matching values v_i (see Figure 1a). We refer to a node issuing a query as the inquiring node. We restrict ourselves to a query semantic given by conjunctions of keys, i.e., implementing Boolean AND. That is, for a query comprising of more than one key, a value matches the query, if it matches all keys. Note that the query semantic can be easily extended to additionally support disjunctions of keys, i.e., implementing Boolean OR, by including a bit vector indicating the matched keys for each value in the response message.

Query messages are sent to the IP limited broadcast address 255.255.255.255 and a well-defined port. Thus, all nodes located inside the radio coverage of the inquiring node receive a query message. These nodes may generate a response message. A response message contains the query and all matching values from either the local index or a second data structure called *index cache*. To enable epidemic data dissemination, PDI response messages are sent to the IP limited broadcast address 255.255.255.255 and a well-defined port, too. Thus, all mobile nodes within the radio coverage of the responding node will overhear the message (Figure 1b). Not only the inquiring node but also all other mobile nodes that receive a response message extract all index entries and store them in the index cache (see Figure 1b). In Figure 1, index caches are drawn as the second box below mobile devices. Index entries from the index cache are used to locally resolve queries, if the origin nodes of matching values reside outside the radio coverage of the inquiring node (see Figures 1c and 1d). Obviously, the index cache size is limited to a maximum number of entries adjusted to the capabilities of the mobile device. The replacement policy least-recently-used (LRU) is employed when newly received index entry will see a full cache. Note that information is disseminated to all other nodes that are in direct contact, similar to the spread of an infectious disease. Due to the movement of nodes and overhearing response messages of neighboring nodes, index entries are disseminated within the network without costly global communication. In fact, PDI builds and maintains an index distributed among mobile node of the MANET in a passive way.

Recall that responses derived from the index caches only contain index entries from remote hosts. Such responses may be out of date when the origin node either has left the system or has withdrawn the index entry from its local index. Thus, effective consistency mechanisms as introduced in Section III are the key for the effective design of PDI.

To extend information dissemination beyond the radio coverage of inquiring nodes, PDI includes a message forwarding mechanism. Queries may be relayed a certain number of hops specified by the inquiring node in a time-to-live field (TTL_{query}). Similarly, response messages may be forwarded TTL_{query} hops. Before forwarding a response message, a mobile node removes all values found in the index cache from the message.

For a detailed description of this concept, denoted as selective forwarding, we refer to [9].

Note that beside inconsistent results, PDI might even return no results to a query for key k at all. This may occur, if the corresponding query neither reaches the origin node nor any other node storing an index entry for value v in its index cache. We refer to such unresolved query as false misses. For the application running on top of PDI, there are two ways to resolve false misses: First, the application can tolerate false misses and simply ignore them. Second, the application has to resort to a centralized lookup service via the cellular infrastructure of a mobile network as fallback. Since query streams for P2P file sharing and Internet search engines possess a high degree of temporal locality [16], [18], PDI will produce only few false misses for such applications.

III. Means for Maintaining Index Consistency

III.A. Consistency Issues for Mobile Systems

As mentioned above, caching index entries may introduce inconsistency. For instance, when a node stops supplying an index entry (i.e., the pair (k,v) is removed from the local index of the node), copies of the index entry will remain in the index caches of other nodes. We refer to an index entry contained in a response message as *fresh* or *up-to-date*, if it is currently stored in the local index of the origin node. Otherwise, the index entry is denoted as *stale*. Stale index entries obviously yield stale search results and even disseminate in the system. As shown in Section V.B, hits for stale index entries constitute a significant fraction of all results received in response to a query when no invalidation mechanism is used. In this section, we discuss invalidation mechanisms that can cope with both weak connectivity and modification of information. Since PDI has to deal with both sources of inconsistency, PDI implements an integrated approach that combines both invalidation mechanisms.

Timeouts constitute a common concept in several areas of distributed applications, as they can assure cache consistency without the need to contact the source of the cached information. Examples include the invalidation of cached DNS records in the domain name system or the invalidation of cached Web documents in WWW caches. To achieve both maximum consistency and a sufficient number cache hits, it is crucial for a timeout-based invalidation mechanism that the timeout durations are chosen appropriately. In DNS, the origin DNS server specifies the timeout duration for each entry. In web caching, the adaptive TTL algorithm calculates the timeout duration from the last modification time of a document at the origin server. Note that both approaches rely on information directly received from the origin server. In contrast, due to the epidemic dissemination, most index cache entries are extracted from responses comprising of entries of other index caches, i.e., without direct contact to the origin node. Thus, PDI defines the concept of value timeouts to approximate the most recent information about the state of an index entry at the origin node.

Examples of explicit invalidation schemes include the invalidation of cached memory blocks in distributed shared memory (DSM) systems, or the invalidation of documents in web caches. To achieve consistency, the origin node of an item sends invalidation messages to exactly those nodes that are caching this item. In DSM systems, the origin node of a shared page sends invalidation messages to all nodes sharing this page. In web caching systems, the origin server of a web document sends invalidation messages to each web cache that holds a copy of the document. Note that both mechanisms require that the origin node knows where all copies of an item reside and that all sharers are reachable. In contrast, in a mobile environment consisting of nodes with limited resources, connectivity of nodes cannot be guaranteed nor directories for all cached copies of a shared item can be maintained. To address these constraints in mobile systems, PDI defines the concept of lazy invalidation caches implementing explicit invalidation of values.

III.B. Configurable Value Timeouts for Dealing with Weak Connectivity and Node Failures

The basic concepts of PDI as described in Section II do not take into account low connectivity and spontaneous departures of nodes; circumstances under which all information previously supplied become stale. Examples of these cases include node failure or nodes leaving the area covered by the system.

Value timeouts limit the time for which any index entry (k,v) of a given value v will be stored in an index cache. By receiving a response from the origin node of (k,v), the corresponding value timeout will be reset. Let $a_{(k,v)}$ be the time elapsed since (k,v) has been extracted from a response message generated by its origin node. We define the age a_v of value v as $a_v = \min_k \left(a_{(k,v)} \right)$, i.e., the time elapsed since the most recent response message of this kind was received. When at a node holds $a_v > T$ for the given timeout value T, all pairs (k,v) are removed from its index cache. PDI implements only one timeout per value v rather than an individual timeout for each index entry (k,v). This is because in most applications the fact that one index entry (k,v) for a given v expires indicates a substantial change of the value. Subsequently, all other index entries (k',v) are likely to be influenced. For example, in a file sharing

system, a pair (keyword_i, URI) is removed when the file specified by URI is withdrawn from the system. Thus, all other pairs (keyword_j, URI) also become stale. Note that depending on the application the concept of value timeouts can be easily extended to individual timeout durations T_v for each value v. Such duration may be included in a response message generated by the origin node. For ease of exposition, we assume in the remainder of this paper a global timeout value T for all values in the system.

To determine the current age of a value, an age field is included in the response message for each value. This age field is set to zero in each response from the origin node. When receiving a response message, a node n extracts the age of each value and calculates the supply time s_v. That is the time at which a response for this value was generated by the origin node. Assume that the response message contains age a_v, then s_v is determined by $s_v = c_n - a_v$, where c_n denotes the local time of node n. s_v is stored in the index cache together with v. Note that v might already be present in the index cache with supply time s'_v. The copy the index cache might result from a more recent response by the origin node, i.e., $s_v < s'_v$. Thus, in order to relate the age of a value to the most current response from the origin node, the supply time is updated only if $s_v > s'_v$. When a node generates a response for a cached index entry (k,v), it sets the age field for each value v to $a_v = c_n - s_v$. Note that only time differences are transmitted in PDI messages, eliminating the need for synchronizing clocks of all participating nodes.

III.C. Lazy Invalidation Caches for Dealing with Data Modification at Origin Node

Additional to the scenarios described above, a node produces stale index entries by modifying information. That is the case when an index entry is removed from the local index. In a worst-case scenario, a node suddenly leaves the system and all index entries supplied by the node expire at the same time. One way to handle such modification of information is to wait until the timeouts of the values in the stale index entries elapse. Depending on the application and the timeout value T, this straightforward solution may cause severe inconsistency, especially if T is large. A more effective way to handle information modification in distributed applications constitutes the explicit invalidation by control messages.

As basic idea of the explicit invalidation mechanism, a node removes all index entries (k,v) from the index cache when it receives an invalidation message for value v. Flooding is a straightforward way to propagate invalidation messages. To flood an invalidation message for value v, the node removing an index entry sends the message to the limited broadcast address. All mobile nodes that receive the message will relay it exactly once, so that the message is propagated to each node that is connected to the initial node via one or more hops. Unfortunately, in mobile systems even a multi-hop connection between two nodes frequently does not exist. Subsequently, stale index entries are still contained in the index caches of nodes that are not reached by the invalidation message. Note that these index entries will be redistributed in the system due to the epidemic dissemination. We will show in Section V.C that even repeated flooding of invalidation messages does not significantly reduce the number of hits for index entries.

This observation is consistent with [2], which reports that deleted database items "resurrect" in a replicated database environment due to epidemic data dissemination. In [2], a solution is proposed that uses a special message to testify the deletion of an item, referred to as death certificate. Death certificates are actively disseminated along with ordinary data and deleted after a certain time. In contrast, we propose a mostly passive (or "lazy") approach for the epidemic propagation of invalidation messages, which is illustrated in Figure 2. For the initial propagation of an invalidation message by the origin node, we rely on flooding as described above (Figure 2a). Each node maintains a data structure called *lazy invalidation cache*, which is drawn as a third box below the mobile devices in Figure 2. When a node receives an invalidation message for a value v it does not only relay it, but stores v in the invalidation cache (Figure 2b). Note that an entry for v is stored in the invalidation cache, regardless if the node stores any index entry (k,v) for v in the index cache. Thus, every node will contribute to the propagation of invalidation messages, so that distribution of information and invalidation messages is separated. To enable the epidemic propagation of the invalidation message, a node scans the invalidation cache for all values contained in an overheard response message (Figure 2c). If a value v' is found, the node will generate an invalidation message for v' itself, because the hit in the invalidation cache indicates that the index cache of a nearby node contains a stale entry (Figure 2d). The invalidation message is not flooded through the complete network, but only with a certain scope similar to forwarding query and response messages as described in Section II. A node that receives a cached invalidation message will store the included value v in the invalidation cache, and remove all index entries (k,v) from the index cache.

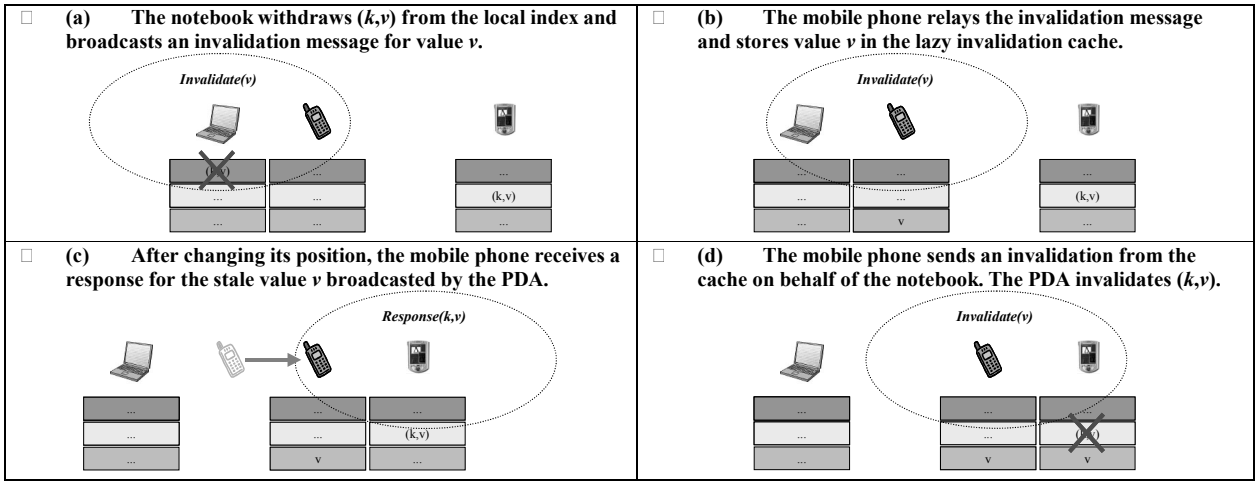

Figure 2. Epidemic dissemination of invalidation messages using lazy invalidation caches

Additionally, the node checks whether it has recently received hits for v in response to an own query, which must also be invalidated and may not be passed to the application using PDI.

As the index cache size, the invalidation cache size is limited to a fixed number of values and LRU replacement is employed. In Section V.C, we show that setting the invalidation cache size to a fraction below 20% of the index cache size achieves sufficient reduction of false hits assuming a reasonable rate of data modification. Note that LRU replacement does neither guarantee that an invalidation cache entry is kept until all stale index entries are invalidated, nor that it is removed after a certain time, inhibiting a node indefinitely from restoring a value it has once invalidated. Increasing the invalidation cache size solves the first problem, though, doing so amplifies the second problem. To avoid this tradeoff, storing the supply time of invalidation messages similar to the supply time of values as described by Section III.B yields an efficient mechanism to decide whether a result for a value is more recent than an invalidation message.

IV. Characterization of Mobile Applications

IV.A. Lookup Requirements of Mobile Applications

In this section, we discuss how we can characterize the way a mobile application makes use of a lookup service for simulation purpose. That is, we describe the *lookup requirements* of the mobile application. We assume N_m users participate with mobile devices. The application defines a set of keys $\mathbb{K}$ with cardinality $|\mathbb{K}| = N_K$. We associate each key with its popularity rank, i.e., $\mathbb{K} = \{1,\ldots,N_K\}$, where key 1 is the most frequently requested key. Additionally, the application defines a set of values $\mathbb{V}$ with cardinality $|\mathbb{V}| = N_V$. We assume an arbitrary numbering of values, i.e., $\mathbb{V} = \{1,\ldots,N_V\}$. We describe the lookup requirements of the mobile application by five basic functions denoted as *workload functions*:

Query function $w_{query} : \mathbb{K} \to [0,1]$. The function $w_{query}(k)$ denotes the probability that a query is for a given key k.

Selection function $w_{select} : \mathbb{K} \times \mathbb{V} \to [0,1]$. The function $w_{select}(k,v)$ denotes the probability that a key k matches a value v.

Pause function $w_{pause} : \mathbb{R}^+ \to [0,1]$. The function $w_{pause}(t)$ denotes the probability that a node pauses for the time t between two successive queries.

Expiration function $w_{expire} : \mathbb{R}^+ \to [0,1]$. The function $w_{expire}(t)$ denotes the probability that a value expires after a time t.

Arrival function $w_{arrive} : \mathbb{R}^+ \to [0,1]$. The function $w_{arrive}(t)$ denotes the probability for the time between two consecutive arrivals of nodes is equal to t. Since we assume that the system is in steady state, the departure function is equal to the arrival function.

Subsequently, we show how to define workload functions for characterizing two selected mobile applications, mobile P2P file sharing and mobile IM. Note that the customized workload functions are sufficient for generating synthetic workloads as input for the simulation studies presented in Section V.

IV.B. Characterization of Mobile P2P File Sharing

The operation of a P2P file sharing systems requires two basic mechanisms: (1) a keyword-based search algorithm for files shared by remote peers, and (2) a mechanism for downloading files from the search results. PDI can effectively implement the search algorithm of a mobile P2P file sharing system; subsequent file downloads may be performed using standard MANET routing protocols or cellular network infrastructure. In P2P file sharing, keys are given by keywords from file name or some meta-data associated

with a file. Values denote a location from which a file can be downloaded, e.g., given by a URI. Note that this definition implies a many-to-many matching between keys and values, i.e., may keys may match a given value, and a given key may match many values. Using PDI for mobile P2P file sharing obviously requires invalidation mechanisms: First, a mobile device may stop sharing a file, e.g., by deleting it from the local disk. In this case, the value denoted by this file expires. Second, a mobile user may depart from the system, e.g., due to node failure. In this case, all files shared by the user are removed from the system, and all corresponding values expire.

For defining the workload functions for P2P file sharing, we assume that both $\mathbb{K}$ and $\mathbb{V}$ are finite in an observation period of finite length T. Consistent with [16], [17], assume that the query function w_{query}^{P2P} for a P2P file sharing system can be approximated by Zipf-like distributions with parameters α (In a Zipf-like distribution, for the access probability $P(i)$ to an item with rank i holds $P(i) \cong i^{-\alpha}$). Furthermore, we assume that the selection function w_{select}^{P2P} is given by a Zipf-like distribution with parameter β. Thus, these workload functions are given by:

$$w_{query}^{P2P}(k) = \frac{1}{\sum_{j \in \mathrm{K}} j^{-\beta}} k^{-\alpha} \text{ and } w_{select}^{P2P}(k,v) = \frac{h}{\sum_{j \in \mathrm{K}} j^{-\alpha}} k^{-\beta} \quad (1)$$

Here, h denotes the average number of keys matching a given value, with $h \le \sum_{j \in \mathrm{K}} j^{-\beta}$. For defining the request function, we assume that the intervals between two successive queries by a node are exponential distributed with parameter λ. Thus, the pause function for P2P file sharing w_{pause}^{P2P} is given by:

$$w_{pause}^{P2P}(t) = \lambda e^{-\lambda t} \quad (2)$$

For definition of the expiration function, we assume that each value expires exactly once in an observation period of length T. Thus, the expiration function for P2P file sharing w_{expire}^{P2P} is given by:

$$w_{expire}^{P2P}(t) = \frac{1}{T} \quad (3)$$

In an observation period of length T, we assume that dN_n nodes depart from the system with $0 \le d \le 1$. Subsequently, we assume that the arrival function $w_{arrival}^{P2P}(t)$ is given by an exponential distribution with parameter dN_n / T. Then, the number of arrivals in the observation period is Poisson distributed with parameter $dN_n / T \cdot T = dN_n$ and mean dN_n. That is the arrival function for P2P file sharing $w_{arrival}^{P2P}(t)$ is given by:

$$w_{arrival}^{P2P}(t) = \frac{dN_n}{T} e^{-\frac{dN_n}{T}t} \quad (4)$$

Throughout the simulation studies, we use the parameter setting given by Tables 1 and 2, if not stated otherwise.

IV.C. Characterization of Mobile Instant Messaging

A key feature of an IM system is the ability to track the present state of a user. That is, a user can determine whether another user is free for chat, busy, away, or offline. In a wire-line IM system, a dedicated IM server maintains presence information, and returns presence state when queried for a user's unique IM identifier. Since such centralized component is not available in a self-organizing mobile network, the IM server must be replaced by a distributed approach. PDI can be effectively employed to maintain presence information in a mobile IM system; additional features of IM systems such as message-based communication may be implemented using standard MANET routing protocols or cellular network infrastructure. In a mobile IM system based on PDI, each mobile user contributes his current presence state to the system as only value, which is matched by a key given by the users IM identifier. Subsequently, this results in a one-to-one matching of keys and values. Using PDI in a mobile IM obviously requires invalidation mechanisms, since a user occasionally changes his presence state or departs from the system. That is, the value contributed by the user expires and is replaced by a new value.

For definition of the workload functions for a mobile IM application, we define $\mathbb{K} = \{1,\ldots,N_m\}$, i.e., the user's IM identifiers provide the keys. Furthermore, define $\mathbb{V} = \{(u,s)\}$ with $1 \le u \le N_m$ and $s \in \{1,2,\ldots\}$. When user u that changes his current presence state, he sends an invalidation messages for (u,s) and subsequently contributes $(u,s+1)$ to the system. We assume that each mobile user maintains a buddy list, i.e., a list of his favorite contacts. Each user u is on the contact list of each other user with probability Cu^{-1}, where C is a constant. I.e., the social contacts are described by a Zipf distribution. An IM client periodically polls the presence state of each contact on a users contact list and displays whether the user is busy, free for chat, etc. Based on this model, some calculus shows that the query function w_{query}^{IM} is given by a Zipf distribution:

$$w_{query}^{IM}(k) = Ck^{-1} \quad (5)$$

Given that each contact is polled once within a period T_p, the pause function w_{pause}^{IM} can be calculated using the

probability mass function $l(n)$=P{length of buddy list is n}:

$$w_{pause}^{IM}(t) = \begin{cases} l(n) & t = T_p / n, n \in \mathbb{N} \\ 0 & \text{else} \end{cases} \quad (6)$$

For the expiration function, we assume that users change presence state in exponentially distributed intervals with parameter γ. Thus, the expiration function $w_{\exp ire}^{IM}$ for mobile IM is given by:

$$w_{\exp ire}^{IM}(n) = \gamma e^{-\gamma t} \quad (7)$$

Furthermore, for the arrival function we assume exponentially distributed arrivals similar to the P2P file sharing application. Thus, the arrival function w_{arival}^{IM} is given by:

$$w_{arrive}^{IM}(t) = \frac{d \cdot N_n}{T} e^{-\frac{d \cdot N_n}{T} t} \quad (8)$$

V. Performance Studies

V.A. Simulation Environment

To evaluate the performance of PDI, we conduct simulation experiments using the network simulator ns-2 [3]. We developed an ns-2 application implementing the basic concepts of PDI as described in Section II as well as the two consistency mechanisms described in Section III. An instance of the PDI application is attached to each simulated mobile device, using the UDP/IP protocol stack and a MAC layer according to the IEEE 802.11 standard for wireless communication [7]. All MAC layer parameters were configured to provide a radio-coverage with a radius of 115m. We assume that the N_m mobile nodes move in an area of 1000 m × 1000 m according to the random waypoint mobility model [1]. Maximum node speed is 1.5 m/s and a pause time between two movement epochs is 50 s. The random waypoint model with this configuration is commonly used to mimic the movement of pedestrians.

Table 1: Common parameters for both applications

Para-meter	Description	Mobile P2P File Sharing	Mobile Instant Messaging
N_m	Number of mobile nodes	100	100
N_K	Number of keys	10.000	N_m
N_V	Number of values	$16 \cdot N_n$	N_m
T	Simulation time	7200 s	7200 s

Table 2: Application-specific parameters

Mobile P2P File Sharing	
Parameter	Value
α_{P2P}	0,9
β_{P2P}	1,2
h	3
D	0.3
λ_{P2P}	1/120

Mobile Instant Messaging	
Parameter	Value
T_{P}	0,05
γ_{IM}	0,9
D	0.3

For both applications, we considered four different sizes for PDI index caches, i.e., 32, 128, 512 and 2048 entries for mobile P2P file sharing and 16, 32, 64, and 128 entries for mobile IM. In all experiments, we set the TTL for selective forwarding $\text{TTL}_{\text{query}} = 4$ hops, and for forwarding of invalidation messages $\text{TTL}_{\text{inv}} = 2$ hops. A synthetic workload for each mobile application was generated using the customized analytical framework presented in Section IV.

We choose performance measures to evaluate the accuracy and the coherence of the results delivered by PDI and the impact of the introduced invalidation mechanisms. Accuracy is measured by the hit rate *HR*, i.e., $HR = H_F / K_F$ for H_F denoting the number of up-to-date hits and K_F the total number of all up-to-date matching values currently in the system. Note that hit rate can be compared to the information retrieval measure *recall*. Coherence is measured by the stale hit rate *SHR*, i.e., $SHR = H_s / (H_s + H_F)$, where H_S denotes the number of stale hits returned on a query. Note that stale hit rate is related to the information retrieval measure *precision* by $precision = 1 - SHR$. As last measure, reduction of stale hits is measured by the coherence efficiency *EF*, i.e., $EF = 1 - (H_s / \hat{H}_s)$, where $\hat{H}_s$ denotes the number of hits for stale index entries without employing an invalidation mechanism.

We conduct transient simulations starting with initially empty caches. For each run, the total simulation time was 2 hours. To avoid inaccuracy due to initial warm-up, we reset all statistic counters after a warm-up period of 10 minutes simulation time. For each point in all performance curves, we performed 100 independent simulation runs and calculated corresponding performance measures at the end of the simulation. In all curves 99% confidence intervals determined by independent replicates are included.

V.B. Performance of PDI in a Mobile P2P File Sharing Application

V.B.1. Performance of PDI without Invalidations

In a first experiment, we investigate the coherence of index caches maintained by PDI without the invalidation mechanisms presented in Section III. These performance curves are shown in Figures 3 and 4. Figure 3 plots hit rates as a function of system size for different sizes of the index cache. The results reveal that hit rate increases with growing system size because an increasing number of nodes increase the dissemination of information. Furthermore, the increase slows down with increasing number of nodes because the total number of values in the overall system increases. In these cases, the hit rate is clearly limited by the overall index cache size. Note that increasing the

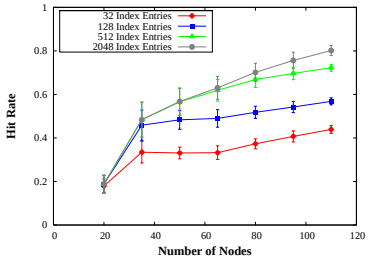

Figure 3. Mobile P2P file sharing without invalidation: system size vs. hit rate

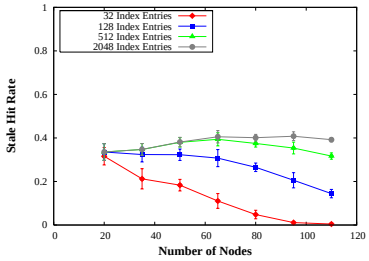

Figure 4. Mobile P2P file sharing without invalidation: system size vs. stale hit rate

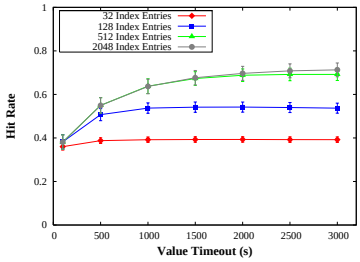

Figure 5. Mobile P2P file sharing with value timeouts: duration vs. hit rate

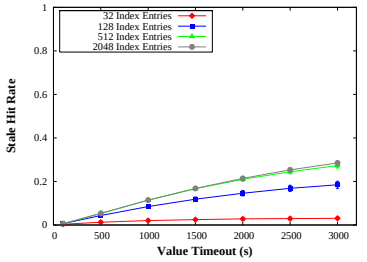

Figure 6. Mobile P2P file sharing with value timeouts: duration vs. stale hit rate

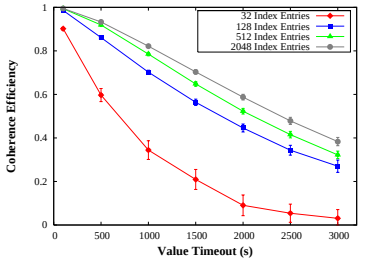

Figure 7. Mobile P2P file sharing with value timeouts: duration vs. coherence efficiency

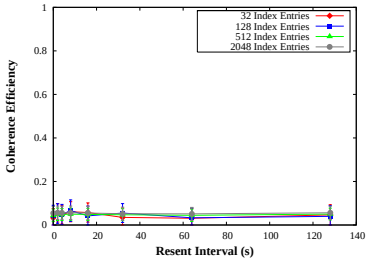

Figure 8. Mobile P2P file sharing with flooded invalidations: resent-time vs. coherence efficiency

index cache size from 512 to 2048 entries, i.e., by factor 4, only increases the hit rate about 10% for systems comprising of many nodes.

Stale hit rates as a function of system size is plotted in Figure 4. We find that without invalidation the stale hit rate is at most 0.4. For smaller index cache sizes, the stale hit rate decreases with system size. Jointly considering Figures 3 and 4 reveals that for increasing system sizes the stale hit rate drops rapidly at the point when the growth of the hit rate slows down. Looking closer at the index caches in these scenarios, we find that the cache content is highly variable. Therefore, stale index entries are removed early from the caches. We conclude from Figure 3 that large caches yield a high amount of stale hits when no invalidation mechanism is used. In contrary, small index caches naturally reduce stale hits, while they fail to provide high hit rates. This evidently illustrates the need for invalidation mechanisms in order to achieve both high hit rates and low stale hit rates.

V.B.2. Impact of Configurable Value Timeouts

For the following experiment, we fix system size to 80 nodes and investigate the performance of basic PDI extended by value timeouts as implicit invalidation mechanism. Figure 5 plots hit rates versus timeout durations for cache entries. As value timeouts invalidate both stale and up-to-date index entries, the hit rate increases with increasing timeout duration. Thus, invalidations occur less frequently. Unfortunately, as Figure 6 reveals, the stale hit rate increases, too. However, comparing Figures 5 and 6 illustrates that the stale hit rate grows almost linear with an increasing timeout duration while hit rate grows in a log-like fashion. Based on this observation, we choose low timeout duration in order to limit the decrease in hit rate. For example, given an index cache with 2048 entries, a timeout of 1000 seconds decreases the hit rate by about 0.07, while the stale hit rate is decreased from 0.5 to 0.11. That is about 75% improvement compared to the corresponding scenario without invalidation as shown in Figures 3 and 4, respectively. Note that the optimal timeout duration clearly depends on the rate of modification of the information as well as on the arrival and departure rates for nodes. Thus, a timeout of 1000s may not be the best choice for all application scenarios. To gain further inside into the behavior of value timeouts, Figure 7 plots the coherence efficiency versus timeout duration. We find that the coherence efficiency rapidly drops with increasing timeout duration due to more infrequently occurring invalidations. Surprisingly, we find that value timeouts are less efficient for small cache sizes than for large ones. Again, this is a confirmation that small caches naturally reduce hits for stale index entries by frequent replacements, shortening the room for improvements by a timeout-based invalidation mechanism. We conclude from Figures 5 to 7 that value timeouts provide an efficient mechanism for implementing implicit invalidation, especially for large index caches.

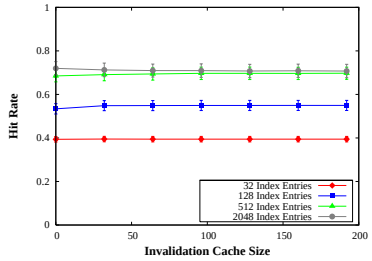

Figure 9. Mobile P2P file sharing with invalidation caches: cache size vs. hit rate

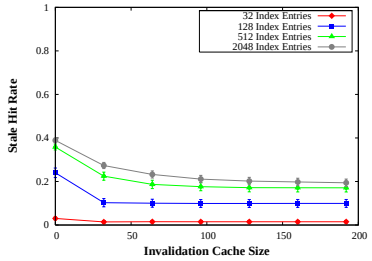

Figure 10. Mobile P2P file sharing with invalidation caches: cache size vs. stale hit rate

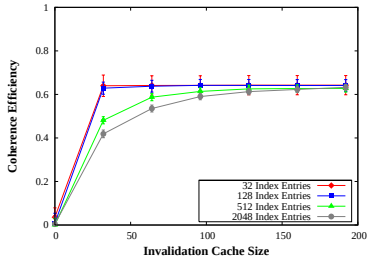

Figure 11. Mobile P2P file sharing with invalidation caches: cache size vs. coherence efficiency

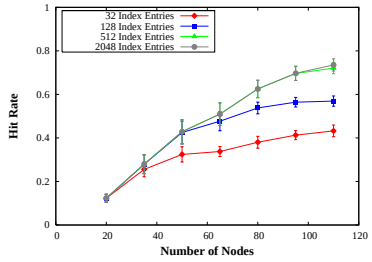

Figure 12. Mobile P2P file sharing with integrated approach: system size vs. hit rate

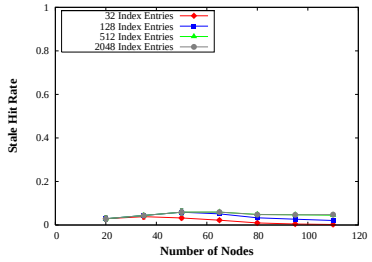

Figure 13. Mobile P2P file sharing with integrated approach: system size vs. stale hit rate

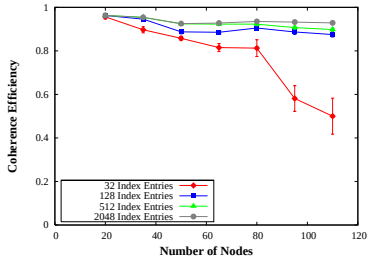

Figure 14. Mobile P2P file sharing with integrated approach: system size vs. coherence efficiency

V.B.3. Impact of Lazy Invalidation Caches

In the following experiment, we investigate the performance of lazy invalidation caches as explicit invalidation mechanisms. As above, the system size is kept fixed to 80 nodes. Figure 8 plots the coherence efficiency achieved by pure periodical flooding of invalidation messages without using invalidation caches versus re-sent time, i.e., the interval between two successive transmissions of an invalidation message. We find that independent of the re-sent time the coherence efficiency is below 0.1 regardless of index cache size. Note that due to weak connectivity, flooding of invalidation messages cannot eliminate all stale index entries. Subsequently, the epidemic dissemination of remaining index entries leads to a redistribution of stale values. Thus, as illustrated by Figure 8, periodic flooding of invalidation messages fails to implement explicit invalidations.

Subsequently, we investigate performance of the epidemic dissemination of invalidation messages using invalidation caches. In various experiments, we investigated the sensitivity of all performance measures to the TTL_{inv}. Due to space limitations, we do not show these performance results. We found that performance does not significantly increase for $TTL_{inv} > 2$ hops. Thus, we set $TTL_{inv} = 2$ hops in all subsequent experiments. Figure 9 illustrates that lazy invalidation caches regardless of their size do not affect the hit rate. This is because opposed to value timeouts explicit invalidation messages only invalidate stale index entries.

However, lazy invalidation caches significantly reduce the stale hit rate, especially for large index cache sizes, as shown in Figure 10. We find that for large caches the stale hit rate is reduced by more than 50% compared to Figure 4. Note that if the invalidation cache size increases beyond 20% of the index cache size, no significant further reduction of the stale hit rate can be achieved. For practical applications, this means that the invalidation caches can be small compared to index caches. This observation is confirmed by the results for the coherence efficiency shown in Figure 11. Opposed to the coherence efficiency of value timeouts, the coherence efficiency of lazy invalidation caches is best for small cache sizes. Compared to Figure 7, we find that the coherence efficiency of lazy invalidation caches is smaller than for value timeouts when using large index caches. To understand this observation, recall that nodes leaving the system do not explicitly invalidate all index entries they have supplied, a worst-case scenario for each explicit invalidation mechanism. We conclude from Figures 9 to 11 that lazy invalidation caches efficiently implement explicit invalidation.

V.B.4. Performance of the Integrated Invalidation Approach

In a last experiment, we investigate the performance of an integrated approach combining both value timeouts and lazy invalidation caches to take into account both

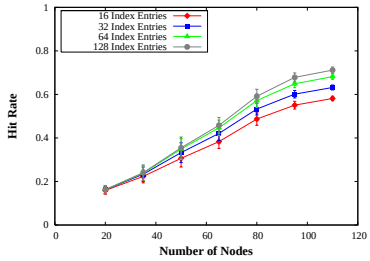

Figure 15. Mobile IM without invalidation: system size vs. hit rate

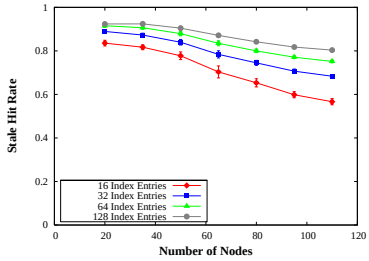

Figure 16. Mobile IM without invalidation: system size vs. stale hit rate

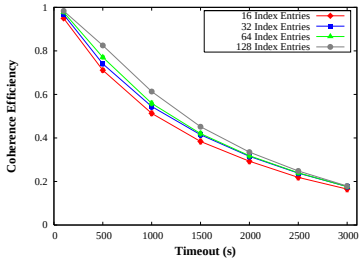

Figure 17. Mobile IM with value timeouts: duration vs. coherence efficiency

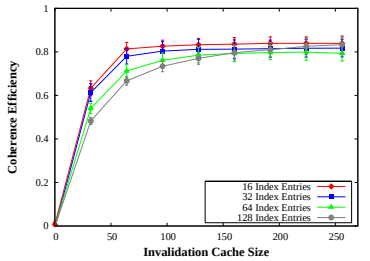

Figure 18. Mobile IM with invalidation caches: cache size vs. coherence efficiency

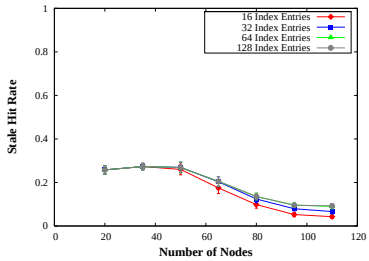

Figure 19. Mobile IM with integrated approach: system size vs. stale hit rate

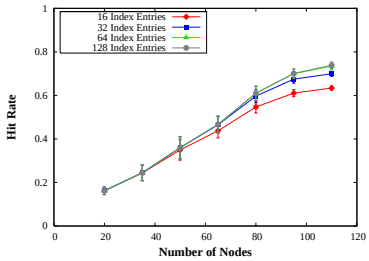

Figure 20. Mobile IM with integrated approach: system size vs. hit rate

weak connectivity and modification of information. We fix the duration of the value timeout to 1000s and the invalidation cache size to 128 entries. Figure 12 plots hit rates versus system sizes. We find that hit rate is reduced mostly for small systems due to invalidations for up-to-date index entries by value timeouts. This leads to a decrease of at most 20%. The performance of index cache sizes of both 512 and 2048 is equal because a large cache cannot benefit from long-term correlations between requests due to the short timeout. For growing system sizes, the hit rate converges towards results without an invalidation mechanism as already shown in Figure 3.

As compensation of the reduction of hit rate, the stale hit rate is significantly reduced compared to a system without invalidation. As shown in Figures 13 and 14, the stale hit rate is highest and coherence efficiency is worst for medium system size and sufficient large index cache sizes. The reason is that a fixed cache size of 128 entries is somewhat to large for small systems, while for large systems the natural limit of stale index entries illustrated in Section V.B increases the coherence efficiency. Again, the coherence efficiency drops rapidly for small index cache sizes due to natural reduction of stale hits. We conclude from Figures 12 to 14 that the integrated approach comprising of the introduced implicit and explicit invalidation mechanisms can effectively handle both spontaneous node departures and modification of information. In fact, for large index caches, the stale hit rate can be reduced by more than 85%.

V.C. Performance of PDI in a Mobile Instant Messaging Application

In a second set of experiments, we look at the performance of PDI in the mobile IM application. Due to space limitations, we omit a detailed description of the results and provide just a brief comparison to the curves for the mobile P2P file sharing application. When not employing any invalidation mechanism in the mobile IM application, we found that hit rate increases to more than 0.7 for systems with large node density and sufficient index cache sizes (Figure 15). Due to a significantly smaller number of values, even small index caches can provide sufficient hit rate compared to the P2P file sharing application. Nevertheless, in such systems stale hit rate is more than 0.58 even for small index caches (Figure 16), which in general tend to reduce stale hit rate due to frequently changing cache content as shown in Section V.B.1. Fortunately, for most configurations both value time-outs (Figure 17) and lazy invalidation caches (Figure 18) show higher coherence efficiency than in the P2P file sharing application. As a result, a hybrid approach integrating both invalidation mechanisms can reduce stale hit rate down to 0.1 even for large systems and large index caches (Figure 19), i.e., a reduction of about 88%. Opposed to the results obtained for mobile P2P file sharing, hybrid invalidation in the IM application even increase hit rate, because it clears index cache space wasted for stale index entries (Figure 20). We conclude from Figures 15 to 20 that PDI can be efficiently employed for tracking presence state in a mobile IM application.

Conclusion

We introduced a distributed lookup service for mobile applications denoted as Passive Distributed Indexing (PDI) and presented invalidation mechanisms for reducing inconsistency in PDI index caches. PDI stores entries in form of (key, value) pairs in index caches located in each mobile device. For implementing implicit invalidation of index entries, value timeouts were proposed. The epidemic distribution of invalidation messages based on lazy invalidation caches implemented explicit invalidation. Presented simulation studies using ns-2 showed that PDI achieves hit rates of up to 0.8 for both a mobile P2P file sharing and a mobile IM application. With the integration of both invalidation mechanisms, the number of stale values returned on a query is reduced to below 5% for the mobile P2P file sharing and below 10% for the mobile IM application. We conjecture that optimizing the duration for value timeouts and the size for lazy invalidation caches will further improve hit rate.

References

[1] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva, A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols, *Proc. 6th ACM/IEEE MobiCom 98*, Dallas, TX, 85-97, 1998.

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, Epidemic Algorithms for Replicated Database Maintenance, *Proc. 6th Symp. on Principles of Distributed Computing (PODC 1987)*, Vancouver, Canada, 1-12, 1987.

[3] *K*. Fall and *K*. Varadhan (editors), *The ns-2 manual*, Technical Report, The VINT Project, UC Berkeley, LBL, and Xerox PARC, 2003.

[4] S. Goel, M. Singh, D. Xu, and B. Li, Efficient Peer-to-Peer Data Dissemination in Mobile Ad-Hoc Networks, *Proc. Int. Workshop on Ad Hoc Networking (IWAHN 2002)*, Vancouver, BC, 2002.

[5] W. Heinzelman, J. Kulik, and H. Balakrishnan, Adaptive Protocols for Information Dissemination in Wireless Sensor Networks, *Proc 5th ACM/IEEE MobiCom 99*, Seattle, WA, 174-185, 1999.

[6] *K*. Hanna, B. Levine, and R. Manmatha, Mobile Distributed Information Retrieval For Highly-Partitioned Networks, *Proc. 11th IEEE Int. Conf. on Network Protocols (ICPN 2003)*, Atlanta, GA, 2003.

[7] IEEE Computer Society LAN MAN Standards Committee, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11-1997, New York, NY, 1997.

[8] A. Khelil, C. Becker, J. Tian, and *K*. Rothermel, An Epidemic Model for Information Diffusion in MANETs, *Proc. 5th ACM Int. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2002)*, Atlanta, Georgia, 2002.

[9] C. Lindemann and O. Waldhorst, A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications, *Proc. 2nd IEEE Conf. on Peer-to-Peer Computing (P2P 2002)*, Linköping, Sweden, 71-83, 2002.

[10] Internet Engeneering Task Force Working Group Mobile Ad hoc Networks (MANET). http://www.ietf.org/html.charters/ manet-charter.html.

[11] P. Mockapetris, *Domain Names - Concepts and Facilities*, IETF Request for Comments 1034, 1987.

[12] M. Papadopouli and H. Schulzrinne, Effects of Power Conservation, Wireless Coverage and Cooperation on Data Dissemination among Mobile Devices, *Proc. 2nd ACM MobiHoc 2001*, Long Beach, NY, 117-127, 2001.

[13] M. Papadopouli and H. Schulzrinne, *Performance of Data Dissemination and Message Relaying in Mobile Ad Hoc Networks*, Technical Report CUCS-004-02, Columbia University, 2003. Under submission.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A Scalable Content-Addressable Network, *Proc. ACM SIGCOMM 2001*, San Diego, CA., 149-160, 2001.

[15] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM 2001*, San Diego, CA, 149-160, 2001.

[16] *K*. Sripanidkulchai, The Popularity of Gnutella Queries and its Implications on Scalability. *Proc. O'Reilly Peer-to-Peer and Web Services Conf.*, 2001

[17] B. Yang and H. Garcia-Molina, Comparing Hybrid Peer-to-Peer Systems, *Proc. 27th Int. Conf. on Very Large Data Bases*, Rome, Italy, 561-570, 2001.

[18] Y. Xie and D. O'Hallaron, Locality in Search Engine Queries and Its Implications for Caching, *Proc. IEEE INFOCOM 2002*, New York, NJ, 2002.