Temporal Addressing for Mobile Context-Aware Communication

Lars Geiger, Ronald Schertle, Frank Dürr, and Kurt Rothermel
Institute of Parallel and Distributed Systems (IPVS), Universität Stuttgart
Universitätsstrasse 38, 70569 Stuttgart, Germany
{geiger, schertrd, duerr, rothermel}@ipvs.uni-stuttgart.de

***Abstract*—Mobile clients in context-aware systems benefit from the indirect addressing of users via their context (contextcast), such as addressing messages to all users in downtown Toronto whose age is below 35. There is, however, almost no support for a temporal decoupling in such a contextcast system, i.e., the addressing of users that were or will be in a certain context in the past or future, respectively. This could for instance be used to distribute the minutes of a meeting to all people who attended the meeting in room 1.138, 3 days ago, between 1 and 3 pm.**

To enable a context-aware communication system to address messages with temporal relations, especially those contexts in the past, the system needs to manage information about user context histories. This poses the risk that the system can be abused to profile users, which would most probably hinder acceptance. Therefore, privacy aspects need to be considered in the core design of such a system. We present an extension to our earlier work, which allows a temporal decoupling of messages and users and requires very little additional overhead to manage historic context information. The solution includes mechanisms to efficiently disseminate messages to both users with past and future contexts, while effectively preventing user profiling through the use of virtual identities.

I. Introduction

Context-aware communication enables mobile context-aware systems to disseminate messages according to client contexts, i.e., the addressing of receivers with a certain context. An application of this is the dissemination of event announcements for concerts or the communication with fellow students for the purpose of forming a study group. In [1], we propose our CONTEXTCAST system based on principles from content-based publish/subscribe. Event-based systems, such as subject- or content-based publish/subscribe, have been a widely studied topic in recent years. They offer a loose coupling of senders and receivers and, in particular, a spatial and temporal decoupling. User mobility (with all its problems such as changing connectivity) is a key aspect in mobile context-aware systems. The authors of [2] even call location "primary context". Therefore, it seems reasonable to exploit such a decoupling of senders and receivers in context-aware communication systems as well.

However, originally CONTEXTCAST only routes messages to receivers whose context is known to the system at the time a message is sent. It cannot address clients that were in a certain context in the past and has only limited support to address clients in the future via the message lifetime. The solution we propose in this paper extends CONTEXTCAST to use temporal predicates to address both past and future contexts of mobile users. The resulting "temporal contextcast" achieves an efficient message dissemination while protecting the users' privacy.

A. Temporal Contextcast

A temporal contextcast increases the flexibility of the system as it enables addressing of contexts that satisfy a given temporal constraint, both in the past and the future.

This has several applications: For example, the person writing the minutes of a meeting can distribute them via a temporal contextcast to all people who attended the meeting in room 0.108, 3 days ago, between 1 and 3 o'clock pm. While this is also possible using a pub/sub system, it requires that the participants explicitly subscribe to any future notifications concerning the meeting. A temporal cotextcast, however, is a sender-centric approach. It does not require the clients to explicitly register for messages they might be interested in. Another example is a fashion store, which uses CONTEXTCAST to send out a questionnaire for three weeks starting tomorrow, to all people who pass by, are female, and are between 15 and 35 years old, which is the store's target audience.

As one can see from this informal description, a temporal extension requires a couple of challenges to be addressed: It needs an efficient method to distribute messages to addressed users, for both past and future contexts. To achieve this, the system needs information about past user contexts to resolve the addressed users and deliver messages. At the same time, additional measures must ensure that the system cannot be abused for profiling users.

In the remainder of this section we discuss related work. Section II introduces the CONTEXTCAST system and discusses the requirements of a temporal extension. Our main contribution is Section III, where we present a temporal addressing, the privacy-aware archival of past contexts, the routing algorithms for both *historic* and *future* messages, and an estimate for the scalability of our approach. We show the performance of our approach in Section IV and conclude the paper in Section V with a short outlook on future topics.

B. Related Work

The concept of a contextcast, which we introduced in [1], is closely related to both publish/subscribe (or simply pub/sub) and geocast. The support for locations to address receivers comes from geocast while the use of attribute/value pairs to address and route messages is similar to content-based pub/sub.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

Existing geocast systems offer support for locations as addresses. Examples for such systems are GEO [3], which uses a geometric location model based on WGS84, [4], which uses a hierarchical symbolic location model, or "semantic geocast" [5], which uses a hybrid model. All these systems rely on current location as the sole method of addressing and routing, none of these considers any temporal relations.

Similarly, various content-based pub/sub systems have been researched and developed in recent years, e.g., JEDI [6], REBECA [7], or SIENA [8]. All of these systems allow clients to express their interest in notifications using predicates (attribute, operator, value). However, none of these systems includes any particular support to use temporal predicates in either subscriptions or notifications and subscriptions are limited to future notifications.

While regular pub/sub systems allow clients only to subscribe to future events, PADRES [9] is currently the only system that supports clients to also retrieve notifications of past events. The system achieves this by storing past events in a database and allowing subscriptions to specify temporal predicates, which are then used to query the database accordingly. However, the semantics of publish/subscribe and contextcast are essentially reversed, i.e. in pub/sub the subscribers (or clients) select the notifications while in contextcast the messages (ot their senders) select the receivers (cf. [1] for an in-depth discussion of this difference). Therefore, the solution used in PADRES cannot be easily incorporated into a context-aware communication system, since it needs information about past or future contexts in the routing process, not past events or notifications.

A newly designed set of temporal predicates allows CONTEXTCAST to address past and future user contexts (cf. Section III for details). These predicates are based on the temporal relations presented by Allen [10] but were adapted for a temporal, context-aware communication system.

Several applications can build on the contextcast paradigm to efficiently disseminate messages to the addressed receivers. For instance, semantic email addressing [11] relies on a semantic description of email recipients such as "the group developing product X at company Y" instead of explicit addresses like developersX@Y.com.

II. CONTEXTCAST

As we mentioned in Section I, a contextcast disseminates messages to users with a specific context. Currently, the system is limited to addressing only contexts that are known to the system at the time of message dissemination. This leads to a simpler design without having to access historic information, but also limits the possibilities of the system; in this paper, we present concepts to overcome these limitations.

In this section, we present our CONTEXTCAST system, including the basic components, the matching of messages and contexts (i.e., which messages to forward to which receivers), and the required changes for a temporal CONTEXTCAST.

A. The CONTEXTCAST System

The system presented in this section is based on our previous work on the CONTEXTCAST system and does not support temporal addressing concepts. It serves as a basis for our extensions for a temporal addressing.

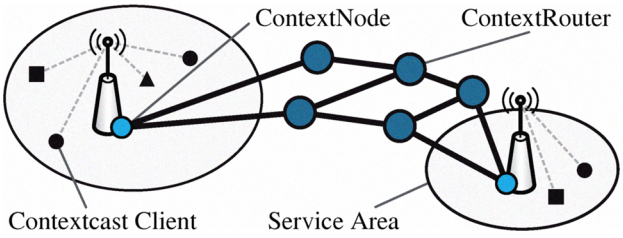


Figure 1: The CONTEXTCAST overlay

$$
\begin{aligned}
c\text{: WGS84: location} &= 48.12\text{N, } 9.10\text{E} \\
\text{string: class} &= \text{"pedestrian"} \\
\text{string: gender} &= \text{"female"} \\
\text{int: age} &= 29 \\
&\vdots
\end{aligned}
$$

Figure 2: Example of a user context

To efficiently forward messages from senders to all receivers with a matching context, we use a distributed approach with context-aware routers. We imagine an overlay network for this purpose, formed by infrastructure nodes. See Figure 1 for the overlay network and its components (in addition to *location*, the clients have a context attribute *activity* that is depicted in the figure by the shape of the clients).

The overlay nodes basically have two roles: routing and access. The mobile clients connect to an access node (or ContextNode) and thus to the system and transmit their context information to it. The ContextNodes propagate this information into the network of contextcast routers (or ContextRouters), which build routing tables based on where clients with certain contexts are currently located. A client in the system periodically refreshes the connection via heartbeat messages. This is necessary since a user might leave a service area, lose its connectivity, and can no longer deregister its context.

Each CN covers a certain service area. A new client selects a CN based on its own location and these service areas. The assignment of service areas can follow different criteria. Since we assume a system of wired or wireless LANs containing the clients, it seems natural to assign service areas coinciding with the area covered by the LAN, i.e., a context node's service area can vary in size from a single floor of a building covered by a wireless LAN to a set of LANs, e.g., a campus network of a university.

B. Matching Messages and Contexts

A user context in the system contains a number of context attributes α. Each attribute α is a tuple *(type, name, value)*. The location attribute α_{loc} is given as a geometric location based on WGS84, with a type of "WGS84". For the other context attributes, we currently support the typical simple types such as *integer*, *float*, and *string*. See Figure 2 for an example.

A message m addresses a set of clients by specifying constraints on context attributes that need to be fulfilled by a

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

m: WGS84: location $\in$ 48.0N–48.4N, 9.0E–9.2E
string: gender $=$ "female"
int: age $>$ 15
int: age $<$ 35
payload $=$ [questionnaire & voucher]

Figure 3: Example of a contextcast message

client's context. The constraints (or attribute filters) ϕ are tuples *(type, name, predicate, value)*, where type_ϕ is the attribute type, name_ϕ is the attribute name, and predicate_ϕ can be any binary predicate that is defined on the attribute type. In addition to the attribute filters for addressing, a message also contains a payload. Figure 3 shows a sample message m.

For a given attribute filter, the system can thus evaluate the constraint 'value_α predicate_ϕ value_ϕ', e.g. for the attribute *age*: $29 > 15 \rightarrow$ `true`. If a message contains an attribute filter and the attribute is not defined in a user context, the corresponding filter evaluates to `false` for that context. The conjunction of all attribute filters in a message determines whether it matches a user's context and thus must be delivered.

C. Requirements of a Temporal Contextcast

Obviously, if the system should have the ability to address messages with temporal constraints, it needs a corresponding attribute as well as a set of temporal predicates defined for this attribute. For application programmers and users, these predicates should be as natural as possible to use, i.e., a predicate $[x, y]$ *before* $[a, b]$ is conceptually simpler than testing on intervall boundaries such as $y < a$ (even though that is the condition that the system evaluates internally).

Depending on the contexts needed to evaluate a temporal contextcast message, we can distinguish three types of messages with different requirements: *historic messages* address only past contexts, *future messages* address only contexts in the future and *hybrid messages* address both past and future contexts. This distinction also determines the strategies to evaluate a matching for temporal contextcast messages: First, for historic messages the system needs access to past contexts, i.e. methods to efficiently store and retrieve past user contexts; this includes efficient ways to deliver messages to the corresponding entities. Such an archival of past user contexts obviously raises privacy concerns, which need to be addressed. Therefore, it is necessary to store contexts anonymously, to prevent the system from being abused to track mobile users and their context, i.e. potentially private information. Second, future messages require messages to be delayed or stored until the temporal constraints can be evaluated with the contexts at that time. This should occur very early and close to the sender, as any message forwarding without a matching receiver wastes bandwidth and places load on the overlay nodes and links. Third, *hybrid messages* can be split into a historic and future part, which can be processed separately.

Since the scalability of the approach depends on the fast lookup of matching contexts, the system also requires to index the contexts via one or more attributes. As we are going to show in Section III, we have chosen *location* to index user contexts, as this is one attribute every context contains and location is considered to be primary context information. Therefore, to allow for a faster lookup, every message *must* contain a target location. This can potentially be rather large, but it needs to be present. Without a target location, the system would have to fall back to a different, less efficient method to reach all access networks with potential matches, possibly a broadcast.

Section III introduces the necessary changes and additions to the original CONTEXTCAST system of Section II-A to build a temporal context-aware communication system.

III. TEMPORAL CONTEXTCAST

A temporal context-aware communication system consists of several related aspects. In this Section, we present the main contributions of this paper: the temporal addressing, the archival of past contexts together with the means to protect the users' privacy, and the routing of both historic and future messages.

A. Temporal Addressing for Contextcast

In a temporal contextcast, suitable attribute filters need to be defined to restrict the set of receivers according to a temporal relation. Informally, such a constraint could be similar to "Address all users who pass through the shopping mall between January 14th, 9 o'clock a.m., and January 28th, 5 o'clock p.m., and whose age is below 30", i.e. whose context matched the temporal attribute filter as well as the attribute filters for *location* and *age*. To use temporal constraints as attribute filters, a new temporal attribute *time* is introduced. For a user context, the *time* attribute is implicitly defined by the temporal interval $[t_{\text{register}}, t_{\text{deregister}}]$ (or $[t_\text{r}, t_\text{d}]$ for short), where $t_{\text{register}} < t_{\text{deregister}}$. t_{register} is the time the context was registered with the system and $t_{\text{deregister}}$ is the time it was deregistered, either explicitly or by missing heartbeat messages. While t_r is always known for any given user context in the system, t_d can still be unknown as long as a context has not been deregistered yet.

The CONTEXTCAST system uses UTC internally, so, e.g., the timestamp for the first date mentioned above is written as 2009-01-14T09:00:00Z. When sending messages, a concrete application can of course offer the user an interface with the more intuitive local time but must then transform those times into UTC when sending a message.

Based on [10], we have defined 11 temporal predicates for CONTEXTCAST. Figure 4 shows an overview of all 11 predicates together with the equivalent comparisons on the interval boundaries. It also contains the expiration condition, which we will explain in detail in Section III-D. It includes, for instance, a predicate *before*, a predicate *during*, or a predicate *equals*. These allow comparison of the context interval $[t_\text{r}, t_\text{d}]$ with an interval $[a, b]$ given in the message (i.e., *time* $< [a, b]$). The predicate *before* ($[t_\text{r}, t_\text{d}] < [a, b]$) specifies that a user context had to be registered and deregistered before the interval $[a, b]$, which is equvalent to $t_\text{d} < a$ when comparing interaval boundaries. Another example is the predicate *during* to test whether a given context was valid at some point during the interval $[a, b]$. It is equivalent to $a \leq t_\text{r} \wedge t_\text{d} \leq b$. The third

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

Predicate		Evaluation	Expiration
before:	$[t_r, t_d] < [a, b]$	$\Leftrightarrow t_d < a$	NOW $> a$
after:	$[t_r, t_d] > [a, b]$	$\Leftrightarrow b < t_r$	—
starts-only:	$[t_r, t_d] <^{so} [a, b]$	$\Leftrightarrow a \leq t_r \wedge t_r \leq b \wedge b \leq t_d$	NOW $> b$
ends-only:	$[t_r, t_d] >^{eo} [a, b]$	$\Leftrightarrow t_r \leq a \wedge a \leq t_d \wedge t_d \leq b$	NOW $> b \vee$ (NOW $> a \wedge \forall c \in \mathbf{C} : t_\mathrm{r} > a$)
start-by:	$[t_r, t_d] <^{s} [a, b]$	$\Leftrightarrow a \leq t_r \wedge t_r \leq b$	NOW $> b$
end-by:	$[t_r, t_d] >^{e} [a, b]$	$\Leftrightarrow a \leq t_d \wedge t_d \leq b$	NOW $> b$
overlaps:	$[t_r, t_d] \cap [a, b]$	$\Leftrightarrow t_r \leq b \wedge a \leq t_d$	NOW $> b$
excludes:	$[t_r, t_d] \neq [a, b]$	$\Leftrightarrow t_d < a \wedge b < t_r$	—
equals:	$[t_r, t_d] \approx [a, b, z]$	$\Leftrightarrow \lvert t_r - a \rvert \leq z \wedge \lvert t_d - b \rvert \leq z$	NOW $> (b + z) \vee$ (NOW $> (a + z) \wedge \forall c \in \mathbf{C} : \lvert t_\mathrm{r} - a \rvert > z$)
during:	$[t_r, t_d] \subseteq [a, b]$	$\Leftrightarrow a \leq t_r \wedge t_d \leq b$	NOW $> b$
contains:	$[t_r, t_d] \supseteq [a, b]$	$\Leftrightarrow t_r \leq a \wedge b \leq t_d$	NOW $> b \vee$ (NOW $> a \wedge \forall c \in \mathbf{C} : t_\mathrm{r} > a$)

Figure 4: Predicates for temporal relations, their evaluation, and expiration conditions

example is the predicate *equals*. It tests two intervals $[t_\mathrm{r}, t_\mathrm{d}]$ and $[a, b, z]$ for equality. Since usually no two time intervals are exactly the same, the comparison interval contains an additional tolerance, z, which applications can specify. This results in an equivalent comparison on interval boundaries as $|t_\mathrm{r} - a| \leq z \wedge |t_\mathrm{d} - b| \leq z$.

B. Archival of Historic Contexts protecting User Privacy

Addressing past contexts requires that the system can match messages with past contexts and can deliver these messages to the corresponding entities. However, the entities may have changed their context or might not be connected to the system at the time of message sending. Thus, addressing past contexts not only requires knowledge about the historic contexts, but also a mapping from contexts to the corresponding physical entity, i.e., resolving entities from a given context. The straight-forward solution, tagging all user contexts with a unique user ID, guarantees that. However, with such an unambiguous mapping from user contexts to entities it is possible to create complete movement and context profiles of users.

To improve user privacy, our system uses (globally unique) virtual identities (VIDs) [12] for the user contexts. Such a VID can be constructed as [timestamp]@[fully qualified domainname (FQDN)]. The advantage of such a construction will become clearer in the following paragraphs. With these VIDs, archival of contexts simply means storing a context and the corresponding unambiguous mapping context $\mapsto$ VID, e.g., in a database. N.b., these VIDs were not necessary in the original system, as messages were simply delivered to anonymously connected users with a matching context. Because the users were still registered with the system, the network had an implicit mapping from contexts to the entities.

Users can potentially choose a new VID every time they register their context with the system. This achieves two things: First, it becomes very hard to obtain a complete history of a single entity's context since no pair of contexts can reliably be determined to relate to the same entity. Second, no other entity except the one choosing a certain VID can reliably map an archived context with this VID to the corresponding entity. Unfortunately, to deliver historic messages to users, who may be offline or have changed their VID, it is still necessary to have such a mapping from VIDs to entities.

Therefore, in CONTEXTCAST, we rely on trusted third parties, which we call Trusted Node or TN, to actually create and resolve virtual identities for entities. For instance, this can be a user's cell phone provider or internet service provider, which are usually trusted implicitly. The Trusted Nodes only store a mapping VID $\mapsto$ entity, they do not know a user's context.

If the VIDs follow the schematic [timestamp]@[FQDN], the FQDN determines the TN to resolve this VID while the timestamp, which the TN hands out, ensures that all of the VIDs of a single trusted node are locally unique. These two properties together ensure that VIDs are globally unique.

Nevertheless, none of these considerations actually designate any particular location in the network for the archival of contexts. The system only requires an efficient method to store and lookup contexts, no matter where the physical location of the data is. However, for the reasons outlined below, in our opinion the simplest and most efficient solution to archive contexts is at the access nodes. First, the access network is where each context of a user actually originates. The routers in the network usually aggregate user contexts to lower the amount of state information and thus improve the system's scalability. Therefore, if the archival of contexts does not take place locally, the system must forward a verbatim copy of each user context to the storage location. And once a context is deregistered, an update must be sent, containing the time y when the context was deregistered. All this information is available in the access network without additional communication. One drawback of the approach is that archiving contexts locally at the ContextNodes requires additional storage. This is not a problem, though, since historic contexts can be placed in a dedicated database co-located with the CN. Second, all access networks have a well-known service area and the network of ContextRouters can be exploited to act as a spatial index over these areas. Therefore, location forms a readily available index over historic contexts as well. It is possible that the lookup of historic contexts improves with additional distributed indexes over other context attributes. However, since this aspect is outside of the scope of this paper, we do not consider it further.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

C. Routing of Historic Messages

The basic approach to route historic messages consists of four phases: First, in the *Spatial Context Lookup* phase, the ContextRouters forward a message to all access networks whose service area intersects with the target location of the message. These are the access networks with historic contexts of potentially matching users. Second, in the *Local Context Lookup*, the access nodes lookup the set of VIDs of contexts matching this message in their local context database. Third, for each of the VIDs, the ContextNode forwards a copy of the message to the corresponding Trusted Nodes to resolve the actual user during the *VID Resolution*. And in the *Historic Message Delivery* phase, each trusted node either delivers the messages to the users directly or, if a user is not connected to the system, to a mailbox, which users poll regularly.

For the example of the minutes of a meeting (cf. Section I-A) addressed to the *location* and the *time* of the meeting, the routing takes place as follows: In the *Spatial Context Lookup*, the system routes the message to all access networks whose location intersects the target location, i.e., the meeting room. In the *Local Context Lookup*, the access nodes find the matching contexts and their VIDs and forward a copy of the message to each responsible Trusted Node during *VID Resolution*. For the *Historic Message Delivery*, the Trusted Nodes can then either find the corresponding entity, possibly connected from home, or store the message in the user's mailbox.

In the following paragraphs, we present these phases in detail and also show optimizations to both the *Local Context Lookup* and the *VID Resolution* to further reduce network load.

Spatial Context Lookup. During the Spatial Context Lookup phase, the overlay routes a message along a distribution tree defined by a spatial index to all the access networks where potentially matching contexts are stored. This is very similar to a geocast. Thus, this phase can directly benefit from any advances in the field of geographic communication such as more efficient routing algorithms.

Algorithm 1 shows this phase in pseudo code. It requires that each router maintains a geographic routing table, i.e., for each link the information what areas are reached via that link. This can be achieved, e.g., by running a modified link-state algorithm with geographic extension, where each node periodically broadcasts its service area, and then computes shortest paths and bounding areas for this information. CONTEXTCAST already maintains this information for regular contextcast messages. In the following algorithms, service area(n) denotes the service area of a ContextNode n, while service area(l) denotes the cumulative area that can be reached via a link l.

When routing a historic message, each router checks first whether this message was received and thus forwarded before. Then it forwards the message via each link for which the service area of the link intersects with the message's target location. If a router is also responsible for an access network and its service area intersects m's target location, it continues with the next phase, the Local Context Lookup.

Local Context Lookup. This phase finds all locally stored contexts that match m and creates a set of the VIDs of these contexts. This is basically a database lookup, thus any improvements in the area of databases and indexing will also benefit this phase.

Algorithm 1 Spatial Context Lookup

Require: A ContextRouter CR and a historic contextcast message m.
Ensure: m forwarded over all links whose service area intersects with m's target location.
  if m was not received before **then**
    for all $l \in$ {links of CR} **do**
      if service area(l) $\cap$ target location(m) $\neq \emptyset$ **then**
        forward m via l
      end if
    end for
    if CR is also an AccessNode and service area(CR) $\cap$ target location(m) $\neq \emptyset$ **then**
      Local Context Lookup(m)
    end if
  end if

Algorithm 2 shows the pseudo code of the algorithm. First, it looks up all the locally archived contexts that match m. Then, it collects all the VIDs of these contexts in a set. It is possible and desirable to optimize the local lookup by using additional indexes over context attributes other than location. However, this is outside the scope of this paper.

Algorithm 2 Local Context Lookup

Require: A ContextNode CN and a historic contextcast message m.
Ensure: mVID = {Set of all VIDs matching m}.
  mVID $\leftarrow \emptyset$
  for all $c \in$ {locally stored contexts} **do**
    if c matches m **then**
      mVID $\leftarrow$ mVID $\cup\, c$
    end if
  end for

VID Resolution. The VID Resolution phase sends a copy of the message to each of the Trusted Nodes that are responsible for the VIDs from the previous phase. The pseudo code for this step is shown in Algorithm 3. It creates a set of all Trusted Nodes that are responsible for subsets of the given VIDs. For each of these TNs, the CN tags m with the subset of VIDs that TN is responsible for and forwards m to the respective TN. This way, only a single message is sent from a given CN to each of the TNs with VIDs for matching contexts.

Historic Message Delivery. Once a message reaches a Trusted Node, the TN can deliver the message to all matching receivers. Algorithm 4 shows the message delivery algorithm. First, it looks up the entity represented by a given VID. If the corresponding client is currently connected to the system (under a potentially different VID), the TN can deliver the message directly to that user. This requires that TNs know

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

Algorithm 3 VID Resolution

Require: A historic contextcast message m and set of VIDs mVID.
Ensure: m sent to all Trusted Nodes for all VIDs in mVID.
  TNs $\leftarrow \emptyset$
  for all VID $\in$ mVID **do**
    TNs $\leftarrow$ TNs $\cup$ Trusted Node for VID
  end for
  for all TN $\in$ TNs **do**
    Tag a copy m' of m with VIDs that TN can resolve
    Send m' to TN
  end for

about their clients' status. If the user is not connected to the system or the TN is not aware of it, the message is delivered to the user's mailbox, which every user needs to poll in regular intervals. In the absence of failures, the algorithm also ensures that a message is delivered to a single user at most once, even though several VIDs of a physical entity may have matched m.

Algorithm 4 Historic Message Delivery

Require: A historic contextcast message m and set of VIDs.
Ensure: m delivered to all matching receivers.
  for all VID $\in$ VIDs **do**
    R $\leftarrow$ lookup physical entity for VID
    if R has not already received m **then**
      if R is currently connected **then**
        Deliver m directly to R.
      else
        Deliver m to R's mailbox.
      end if
    end if
  end for

Unicast Messages and Duplicates during VID Resolution. While the Spatial Context Lookup employs a distribution tree over the geographic index, the naive Local Context Lookup and VID Resolution are less efficient for two reasons:

First, since the user contexts usually represent mobile users, a user's context can be archived in several access networks. Thus, a message with a target location spanning multiple access nodes may match the same VID in each of these access networks and must thus be sent to the same TN for resolution. The naive historic (NH) algorithm uses the local results from every CN and starts the VID Resolution without removing duplicate VIDs. This results in several ContextNodes trying to send a given message to the same TN. To avoid these duplicates, the Full Backtracking Historic algorithm (FBTH) fully collects the matching VIDs for a given message from all access networks where matching contexts are archived, reconciles them and removes obvious duplicates, i.e., identical VIDs. After this step, the algorithm starts the VID Resolution from the node where the results were collected, without obvious duplicates.

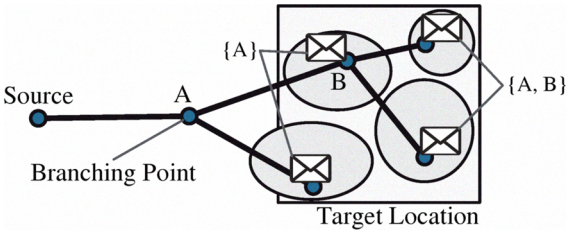


Figure 5: A historic message's distribution tree, with branching points recorded in the message along the route

To this end, we need to record the distribution tree for the messages so the results can take the same route back: Every time a ContextRouter forwards a message to more than one neighbor (or children in the distribution tree), it records its own ID in the message. This sequence marks all the branching points in the distribution tree down to the leaves, i.e., all the nodes where results from two or more subtrees can be collected. See Figure 5 for an example of a message and its distribution tree, with branching points recorded in the message.

With the information about the distribution tree, the access nodes can return their local results to the previous node where the distribution tree branched. This router then consolidates the results, i.e., unifies the sets of VIDs from all its subtrees, and sends the result back to the previous branching point in the tree. Again, this router consolidates the VID lists and so on until it reaches the backtracking point, depending on the algorithm. (For succinctness, we omitted any discussion of lost results from subtrees. Obviously, the system needs a mechanism, e.g., timeout and/or retransmit, to prevent these situations.) Algorithm 5 shows the pseudo code to collect the results from the subtrees until we reach a backtracking point. While collecting the resulting VIDs suppresses duplicates from different CNs, it also causes an increase in end-to-end message delay because of the time it takes to collect and reconcile the results from the different subtrees. Between these two extremes

Algorithm 5 Optimized Local Context Lookup

Require: A ContextNode CN and a historic contextcast message m.
Ensure: mVID = {Set of all VIDs matching m} returned to backtracking point.
  mVID $\leftarrow$ Collect and unify results from all subtrees
  if CN $\neq$ backtracking point **then**
    return mVID to previous branching point
  end if

NH and FBTH, one can also imagine several approaches that collect the results but limit the level up to which to collect the results. We call these algorithms Backtracking Historic (BTH) <n>, $n = 2, 3, \ldots$, where n is the number of the branching point where we collect the results. The nodes that collect results from their subtrees are called "backtracking points". Thus, BTH1 would be identical to the FBTH algorithm, as the backtracking point is at the first branching point, BTH2

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

collects the results at the second branching points (node B in Figure 5), BTH3 at the third, and so on.

Second, each access node sends a single copy of a temporal message via unicast to each Trusted Node, neglecting any shared links between the different unicast paths. A reasonable solution to reduce the system load from these messages is to employ a form of multicasting. However, for any given message the receivers are an element of the power set of all Trusted Nodes $\mathcal{P}(\{\text{TN}_i\})$. This would require us to establish in the order of $2^{|\{\text{TN}_i\}|}$ separate multicast groups for every combination of Trusted Nodes. Such a large number of groups poses a scalability problem, so, instead of relying on a network layer multicast, we route these messages along the overlay network to the Trusted Nodes using an explicit multicast (cf. [13]). An explicit multicast message contains a set of all the receivers of the message and the routers construct the distribution tree on the fly, usually based on available unicast routing information. In the process, they duplicate the message and split the receiver sets as needed. This scheme avoids the large amount of multicast groups at the expense of a more complex message forwarding and larger messages.

Algorithm 6 shows how the backtracking point collects all the local results from the Context Lookup into a set and then determines the TNs that are responsible for the VIDs from step one. After that, it tags m with all the TNs and also with the VIDs that matched this message and sends it via an explicit multicast to the TNs.

Algorithm 6 Optimized VID Resolution

Require: A historic contextcast message m and the first CR where the distribution tree branched.
Ensure: m sent to all Trusted Nodes responsible for all VIDs that matched m.
mVID $\leftarrow \emptyset$
for all $\text{mVID}_{\text{local}} \in$ {resulting VIDs from subtrees} **do**
  mVID $\leftarrow$ mVID $\cup$ $\text{mVID}_{\text{local}}$
end for
TNs $\leftarrow \emptyset$
for all VID $\in$ mVID **do**
  TNs $\leftarrow$ TNs $\cup$ Trusted Node for VID
end for
Tag a copy m' of m with set TNs and mVID
Send m' via an explicit multicast to all elements in TNs

Even with these optimizations, it is impossible to avoid all duplicate messages: Since a user can change VIDs between CNs, not all duplicate VIDs can be recognized as such. This is a disadvantage of the chosen level of privacy. But as each Trusted Node receives only one copy for all matching VIDs, this only increases the message size. The Trusted Nodes are able to discover and reconcile all the different VIDs for every physical entity.

D. Routing of Future Messages

When routing future messages, it is usually not possible to fully evaluate the addressing based on current contexts alone at the time of sending the message. Similar to historic messages, in a naive approach one can use the target location of future messages to proactively forward a future message to all access networks, whose service area intersects the message's target location. The access nodes then store the future messages and match them with future contexts as soon as clients register them. This makes routing future messages conceptually simpler than historic messages since the matching can happen continuously.

However, if a message is forwarded to an access network before a matching context exists there, the system has wasted network bandwidth if no future context ever matches the message. In CONTEXTCAST, we can exploit a property of the system to delay forwarding of a message until a matching context exists in the system: CONTEXTCAST ensures that all newly registered and updated contexts get propagated in the network as current contexts (possibly aggregated with other contexts but still any message matching a context c must also match an aggregated, more general context c'). Thus, it is possible to cache messages very close to message senders instead of at the access network level. This caching could even take place directly on the mobile device of the sender. However, the sender should be able to disconnect from the network, e.g., to save energy. Therefore, in our approach, the first infrastructure node caches the message. As an optimization, to avoid repeatedly forwarding the message from the first caching node, any intermediate node also caches a message if it has at least one other potential downstream router to which it has not forwarded the message yet.

The routers that have cached messages must then evaluate new contexts against their cached messages and forward a message if it matches a new context and it hasn't been forwarded over a link before. This caching is a trade-off between unnecessary forwarding and increased delay for the first receiver when a message finally matches a context. To avoid that routers need to store messages indefinitely, it is possible to define a maximum lifetime for which to cache future messages.

Algorithm 7 shows the caching and routing for future messages. For each neighbor it determines whether one of the current contexts matches the messages and forwards it accordingly. If this isn't the case (or none of the current contexts *can* match the message because of the temporal constraint), it determines whether (part of) the message's target location intersects with the area reachable via that neighbor; in this case, a node may have to forward the message later, when a matching context is registered. Furthermore, if the node is responsible for an access network and it intersects with the message's target location, it must also cache it for potential future contexts from local entities.

Figure 6 shows an example for this approach with a number of nodes and the distribution tree for a given message. The message is first cached by the sender directly (or rather the sender's access node). Once the two matching contexts are registered and propagated through the network (shown for one of the contexts by the arrows in the Figure), the reactive message forwarding delivers the message to the two necessary

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

Algorithm 7 Future Message Caching and Forwarding

Require: A ContextRouter CR and a future contextcast message m.
Ensure: m forwarded to all access nodes with matching receivers and possibly cached on CR.

```
if m was not received before then
  children ← ∅
  for all l ∈ {links of CR} do
    if ∃c ∈ {contexts reachable via l}: c matches m then
      Forward m over l
    else if service area(l) ∩ target location(m) ≠ ∅ then
      children ← children ∪ l
    end if
  end for
  if children ≠ ∅ or
      service area(CR) ∩ target location(m) ≠ ∅ then
    Cache m for potential contexts, local or subtrees.
  end if
end if
```

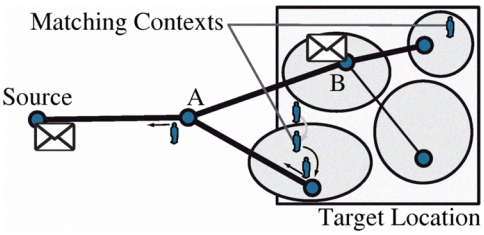


Figure 6: Caching of Future Messages: Initial caching at the source; at B, when the two matching contexts are registered

access network. Node A on the delivery tree does not need to cache a copy of the message since it already forwarded it to all its children from the location-based distribution-tree. Node B, however, only forwarded it to one of its children, thus caching the message for its remaining child. Whenever a new context or an update of an existing one gets propagated in the network, the routers must also match the context against the cached messages and forward some if necessary.

Obviously, with downstream routers also caching messages, once a router has forwarded a message to all its children in the distribution tree and/or when no longer any newer contexts (either local or from its children) can match a given message, the message can be removed from the cache. E.g., if a messages is addressed to some contexts *before* an interval $[a, b]$ in the future, once the current time NOW is past the beginning of the interval, a, no newer contexts can match the temporal predicate. Similar conditions apply to the other predicates, some of which, in addition to the current time NOW, also depend on the set C of contexts that are registered at that time. Figure 4 also shows these expiration conditions. Messages with the predicates *after* and *excludes* do not have an expiration condition and router administrators should define a maximum message lifetime for these cases.

E. Storage Requirements

Our approach to enable historic and future messages requires the storage of certain information. In the next two paragraphs, we discuss the storage requirements for our infrastructure nodes and derive an estimate for the number of users.

ContextNodes. The ContextNodes need to locally store the historic contexts of their users. The actual storage requirements depend on the number of users and the update rate of contexts. We assume that a co-located database is used for the purpose of context storage that is capable of 50,000 inserts of user contexts (i.e., newly registered or changed user contexts) per day. With an average size of 2 KB per insert, this amounts to 100 MB of context data every day. This is well within reach of today's storage and database technology. Thus, the system should scale to up to 10,000 users with an average of 5 updates per day in a single access network.

To increase scalability further, the system can reduce older context information, e.g., by only keeping *location* and *time* information.

ContextRouters. In contrast to ContextNodes, ContextRouters store only future messages. Since our network of overlay routers consists of regular computer hardware, a ContextRouter with current technology probably has 100 GB or more of persistent storage space. With average messages of 100 KB (depending on the payload, probably less), each router could cache $1,000,000$ messages. If the maximum message lifetime is limited to 60 days, this equates to more than 5500 new future messages per day that each router can cache.

Should a ContextRouter at any time run out of cache space, it can always "push down" (i.e., forward) a message in the distribution tree until it reaches either a router that can still cache new messages or an access network, where the previously mentioned database provides additional storage space.

IV. Evaluation

To show the efficiency of our approach, we implemented a prototype with support for temporal contextcast in the simulator PeerSim and used it to evaluate our algorithms. As a basis for the simulation, we set up an overlay topology of $10,000$ routers, which were uniformly distributed over a normalized area $[0, 1[\times[0, 1[$. The links between the routers follow an Internet-like power-law distribution [14], i.e., a new node i connects to an existing node j that minimizes the weighted sum $\alpha \cdot d_{ij} + h_j$, where d_{ij} is the Euclidian distance from the root and h_j is a measure for the centrality such as the hop count from the source. We choose $\alpha = 20$, which is significantly less than $\sqrt{10,000} = 100$ and therefore leads to pronounced clusters of routers. We expect to see similar clusters in real ContextCast systems, corresponding to the local networks of different network providers.

From these $10,000$ routers we then selected $7,000$ as access nodes. Usually, the access nodes are closer to the edge of the network, i.e., nodes with few neigbors are more likely to be access nodes. To achieve this distribution of access nodes, we sorted the nodes by node degree and used a Zipf distribution such that 80% of the $7,000$ access nodes were selected from

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836

the 20% of nodes with the lowest degree. For historic messages, 300 overlay nodes were also selected uniformly as TNs.

In the following two Sections, we present our evaluation of historic and future messages. We compare the message load for several algorithms to disseminate historic messages in our simulation scenario. For future messages, all simulation results would directly mirror the parameters we input into the simulation. We therefore present an intuitive analysis of the reduction of message load we can expect with the presented routing algorithm instead of a simulation.

A. Historic Messages

The simulation creates random historic contexts for the participants of the system. The simulated users randomly select one of the 300 Trusted Nodes to be responsible for its VID resolution. This selection again follows a Zipf distribution to account for the fact that usually only a few providers (20%) are very popular, i.e., serve 80% of all clients. We evaluated our approach in several simulations, with random historic messages where the target locations were squares, with edges ranging in length from 0.05 to 0.25.

We do not evaluate the Spatial Context Lookup or the Local Context Lookup because they are based on existing mechanisms for geocast routing and database lookup and indexing, which are outside the scope of this paper. For the simulation of our approach, we therefore focus on the phase of VID Resolution, especially on the topic of duplicate suppression. We compare the different algorithms which we introduced in Section III-C:

1) The naive historic algorithm (NH) does not suppress duplicates and forwards an explicit multicast message for the local results from each CN.
2) The full backtracking historic (FBTH) algorithm provides the other extreme. It collects the results from all subtrees at the first branching point. With the complete result set, this router can easily eliminate obvious duplicates from the result set and then send only a single explicit multicast message to the Trusted Nodes.
3) A number of algorithms BTH<n> ($n = 2, 3, 4$) provide different levels between these two extremes by specifying the backtracking points for the local result sets. These algorithms cannot fully eliminate duplicates and several branching points send multicast messages to the TNs. (For $n = 1$ this degenerates to the FBTH algorithm, while for $n >$ maximum depth of the distribution tree this is identical to the NH algorithm.)

Figure 7 presents the results from our simulation of these algorithms. As our focus is on the effects of duplicates in the system, we show the message load for the different algorithms. The number of messages is shown as the arithmetic mean of ten simulation runs, with errorbars indicating the minimum and maximum number of messages during these runs.

Figure 7a shows the number of messages resulting from the collection of the local results for the different backtracking levels and for different target area sizes. Obviously, the FBTH algorithm produces the highest message load, which is lower for the algorithms BTH2 through BTH4, since these do not return the local results all the way to the first branching point. E.g., with a target location of edge length 0.25, BTH4 saves approximately 60% of the backtracking messages compared to FBTH.

Despite the FBTH algorithm requiring more messages to collect the matching VIDs from all access networks, Figure 7b indicates that the total message load (i.e., collecting the results *and* VID Resolution) is lowest for the FBTH algorithm. It increases with the limit on how far back we return the local results. The NH algorithm places the highest load on the system as it is unable to recognize and suppress duplicates from different CNs. These results indicate that collecting the results is dominated by the effect of VID Resolution and the explicit multicast messages in the overall message load.

However, our simulation also showed that FBTH in fact increases the average delay for historic messages by 5 overlay hops. We do not think this to be a problem, though, as the increased delay is largely irrelevant since a historic message addresses past contexts, possibly several hours old or more. An additional delay in the order of seconds is not going to affect the system.

Another potentially problematic aspect is the size of the historic messages in the FBTH algorithm. When collecting all matching VIDs, the explicit multicast message contains the TNs for all these VIDs. Therefore, the size of a header of such messages with many receivers increases. However, this increase is limited by the total number of TNs in the system, which we expect to be several hundred in reality. Thus, the overhead of the message header is still small compared to the size of the message data.

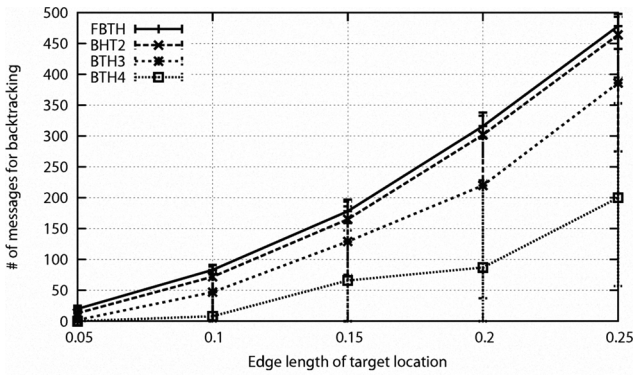
Therefore, we conclude that in most scenarios the FBTH algorithms performs better than the other alternatives.

B. Future Messages

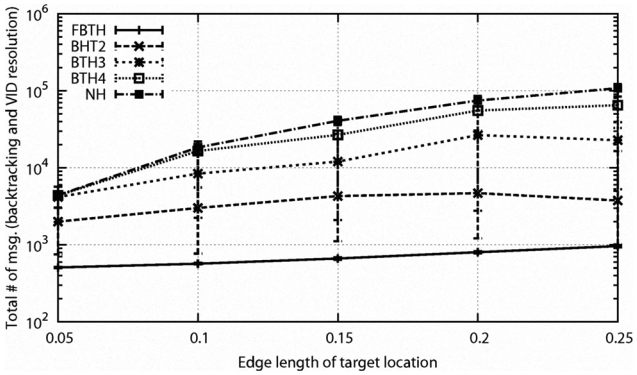
The early caching of future messages we presented in Section III-D reduces network load when no client in an access network ever registers a matching context. There are two main aspects in the routing of future messages: How much network load does the early caching save? And how many messages do the routers need to cache?

Network Load. The amount of saved messages depends on the actual addressing of messages, the user contexts, and the network. For example, if we assume random contexts, with uniformly distributed attributes (including *location*), and a message that 30% of our contexts match, then statistically the message must be forwarded to 30% of the access networks intersecting the addressed area. Or if we put it differently, an early caching in this case avoids to forward the message to 70% of the access networks compared to the naive approach, which simply forwards the message to every CN, whose service area intersects the addressed area. In addition to this percentage of matching contexts, the actual reduction in network load also depends on the network topology and the resulting distribution tree. While in the worst case this could nullify any saving from the delayed forwarding (e.g., if the topology were just a linear

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836



(a) Backtracking message load (without VID resolution



(b) Historic message load (including backtracking and VID resolution)

Figure 7: Simulation of historic messages

sequence of routers), in reality we expect the topology to be rather tree-like and thus more favorable for our approach.

Storage Space for Caching. The early caching does not increase the storage requirements in the system: An inner node in the distribution tree only caches a message when there is at least one access node with potentially matching contexts within a subtree. Therefore, if we cache a message at an inner node, we save the same space at a leaf node. And the space saving is even higher if we cache a message at an inner node for multiple leaves with potentially matching future contexts. The locality in our CONTEXTCAST overlay network supports this approach.

V. CONCLUSION & OUTLOOK

Temporal addressing for contextcast messages extends the original paradigm in two useful ways: First, it allows to send historic messages to entities who had a certain context in the past without requiring explicit addresses. Second, it enables senders to address future messages to entities with a given context in the future.

We have shown how to efficiently deliver both historic and future messages using our CONTEXTCAST overlay network. The dissemination of historic messages employs a history of user contexts, indexed via the contexts' locations. It avoids duplicate messages between the CNs and the TNs by collecting the local results and sending a single explicit multicast message to the TNs. Future messages are delayed until receivers actually enter a service area within the target location. This avoids unnecessary messages to areas without matching receivers.

In the future, we are going to improve the presented approach further. In its current version, the system uses a rather complete history of an entity's context. This obviously raises the complexity of the management and the lookup of local contexts. An aggregation of historic information similar to the aggregation of current contexts in the routers could improve the system in this regard, while incurring a certain amount of false positives because of the information lost in the aggregation.

ACKNOWLEDGMENTS

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627.

REFERENCES

[1] L. Geiger, F. Dürr, and K. Rothermel, "On Contextcast: A Context-Aware Communication Mechanism," in *Proceedings of the IEEE International Conference on Communications (ICC '09)*, 2009.

[2] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," Georgia Institute of Technology, Tech. Rep. GIT-GVU-99-22, 1999.

[3] T. Imielinski and J. C. Navas, "GPS-based geographic addressing, routing, and resource discovery," *Commun. ACM*, vol. 42, no. 4, pp. 86–92, 1999.

[4] F. Dürr, C. Becker, and K. Rothermel, "An Overlay Network for Forwarding Symbolically Addressed Geocast Messages," in *Proceedings of the 15th International Conference on Computer Communications and Networks (ICCCN'06)*, oct 2006, pp. 427–434.

[5] J. Roth, "Semantic Geocast Using a Self-Organizing Infrastructure," in *Innovative Internet Community Systems*, ser. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, 2003, pp. 216–228.

[6] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS," *IEEE Trans. Softw. Eng.*, vol. 27, no. 9, pp. 827–850, 2001.

[7] G. Mühl, "Large-Scale Content-Based Publish/Subscribe Systems," Ph.D. dissertation, TU Darmstadt, 2002.

[8] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Interfaces and Algorithms for a Wide-Area Event Notification Service," University of Colorado, Tech. Rep. CU-CS-888-99, 2000.

[9] G. Li, A. Cheung, S. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen, and S. Manovski, "Historic data access in publish/subscribe," in *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS '07)*, 2007, pp. 80–84.

[10] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[11] M. Kassoff, C. Petrie, L.-M. Zen, and M. Genesereth, "Semantic email addressing: Sending email to people, not strings," in *AAAI 2006 Fall Symposium on Integrating Reasoning into Everyday Applications.*, 2006.

[12] B. Weyl, P. Brandão, A. F. G. Skarmeta, R. M. Lopez, P. Mishra, C. Hauser, and H. Ziemek, "Protecting privacy of identities in federated operator environments," in *Proceedings of the 14th IST Mobile & Wireless Communications Summit*, jun 2005.

[13] M.-K. Shin, Y.-J. Kim, K.-S. Park, and S.-H. Kim, "Explicit multicast extension (xcast+) for efficient multicast packet delivery," *ETRI Journal*, vol. 23, no. 4, pp. 202–204, dec 2001.

[14] A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou, "Heuristically optimized tradeoffs: A new paradigm for power laws in the internet," in *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*. Springer Verlag, 2002.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6836
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6836