Aggregation of User Contexts in Context-based Communication

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

***Abstract*—A context-based communication system enables the indirect addressing and routing of messages according to the users' contexts. This provides, for example, the means to send a message to all students on campus who attend a certain class, with information about an upcoming exam. However, for a targeted forwarding of messages towards users, the routers need information about the context of connected users. Global knowledge, i.e., each router knowing about every user, is not scalable, though, because of the necessary update messages to keep this information up-to-date.**

To address this challenge, a router can aggregate similar contexts and only provide such an aggregated view to neighboring routers. In this paper, we present an approach to aggregate similar contexts, based on a similarity measure for user contexts. The algorithm can be adjusted according to the observed messages and user contexts in the system by specifying a similarity threshold to determine when contexts are aggregated.

The aggregation of user contexts improves the scalability of our approach by significantly reducing the load of context updates by up to 30%, depending on the usage of the system. This improvement comes at the cost of a negligible increase in *false positive* messages due to the loss of information used for forwarding messages.

I. Introduction

Context-aware communication (contextcast) enables clients to disseminate messages to receivers whose context matches a given set of constraints or filters. Possible applications for such a technique include the dissemination of concert information for people interested in a certain musical style or invitations to study groups for students attending the same university class (cf. [1]).

In order to distribute contextcast messages efficiently, overlay networks and context-based routing algorithms are used. In these networks, contextcast routers make forwarding decisions by comparing the addressed context to the context of users located in the access networks. In a distributed system of context-aware routers, messages need to be either broadcast to reach all matching receivers or the routers need to maintain user contexts and their position in the network. However, it is obviously not scalable to maintain complete knowledge of all users on every router: that way, every change needs to be replicated to every router in the network, resulting in a broadcast of every update message. Instead, we propose that routers maintain an aggregated view of reachable contexts for each link. Such an aggregation can hide updates from parts of the network. For example, if a router has the aggregated information that it can reach users with an *age* between 15 and 23 over a certain link, it does not affect the aggregation when a user with the age 19 disconnects from the system. This effectively hides the disconnect update from neighboring routers.

The aggregations also allow us to exploit a locality principle in our design: Users at a location usually share certain characteristics, e.g., many users on a university campus are of type student and within a certain age range.

However, while an aggregation lowers the number of updates, it also brings a loss of information: in the example above, there might not be any contexts with an *age* between 20 and 22, even though the aggregation describes an age between 15 and 23. A message addressed to an *age* 21 would still have to be forwarded in this direction until more detailed knowledge is available, which a router can then use to prune the dissemination tree. Such messages that the system forwards without a matching receiver due to the information loss are called "false positives". Thus, the aggregation algorithm needs to find a good balance between reducing the information, and thus saving update messages, and eliminating false positives early in the network. The approach we present is able to reduce update messages by almost 30%, depending on the scenario, at a negligible increase in false positives.

In Section II, we give a brief overview of our system model, before presenting our aggregation approach to lower the update load in Section III. This comprises an algorithm to aggregate two or more contexts, a similarity measure to select which contexts to aggregate, as well as algorithms to achieve a continuous addition and removal of contexts from our system. Section IV presents the results of our evaluation of a prototype implementation of these concepts, before we conclude in Section V with a summary and a brief outlook on future work.

A. Related Work

The concept of context-based routing is similar to content-based routing. As we have discussed in [1], the concepts "cover" and "merge" in pub/sub systems such as Siena [2] or REBECA [3] cannot simply be transferred to CONTEXTCAST. "Cover" is intended for subscriptions containing predicates on *ranges*, where one covers the others. The user contexts, however, are *points* in a context space, defined by the attributes and their values. Hence, two contexts are either identical or differ in one or more attributes, without overlap in values. Thus,

Published in 2010 6th EURO-NF Conference on Next Generation Internet (NGI). Paris, France, 2-4 June 2010.
© IEEE 2010
http://dx.doi.org/10.1109/NGI.2010.5534466

CONTEXTCAST requires a more general approach similar to "merge". However, it also needs to take the type of data, i.e., attribute type, into account. Especially a *type*-attribute based on a class hierarchy, which is important in our system, needs to be handled differently than, for instance, a quantitative attribute such as *age*.

Aggregating similar contexts is also closely related to the clustering of data items (cf. [4] for an overview). Clustering algorithms with a predefined number k of clusters, such as the k-Means algorithm [5] are not useful for CONTEXTCAST. Since the optimum number k of clusters is not known a priori, a reasonably good clustering dynamically depends on the messages, or rather which contexts they address.

Furthermore, users can join and leave the CONTEXTCAST system at any time. This eliminates clustering algorithms that require random access to the data being clustered, e.g., the ISODATA algorithm [6]. Similarly, algorithms that use a random sampling of the data such as CLARA [7] cannot reliably determine a representative sampling of the data due to continuous addition and removal of contexts. A continuous addition of new elements is possible with algorithms such as BIRCH [8] or more specialized stream clustering algorithms. Neither can, however, remove old elements from the data. CluStream [9] exploits a *subtractive* property of the data to generate a clustering for an arbitrary time window. However, such a subtractive property does not hold for the user contexts in our system.

Also, clustering algorithms for objects in $\mathbb{R}^n$ cannot be used for user contexts with their various different attributes, such as a structured variable based on a type hierarchy. Gowda et al. [10] have researched clustering of such "symbolic objects", which are very similar to the contexts in our system. We use their results as a basis to derive a similarity measure for our contexts in Section III.

II. SYSTEM MODEL AND MESSAGE MATCHING

CONTEXTCAST's main function is the dissemination of messages to users matching a given addressed context. To achieve this, in [1] we presented a contextcast overlay network for routing and defined the semantics of messages, user contexts, and the matching between them. The next Sections give a brief overview of this work.

A. System Model

CONTEXTCAST uses a distributed system of context-aware routers, or ContextRouters, which form an overlay network, as shown in Figure 1. The links in the overlay network are set up to form an acyclic undirected graph; for an arbitrary overlay, a routing algorithm can ensure the acyclic property. The connections between the routers follow a locality principle, i.e., connections between close routers are more likely than between ones that are far apart. This design is used to exploit an existing locality with regard to user contexts, e.g., a clustering of students on a campus. In addition to routing functionality, some routers serve also as access nodes for clients, covering a certain service area. When the distinction of the added access functionality is significant, we call the access nodes ContextNodes.

The ContextNodes maintain information about connected users and their contexts in their respective service area. They propagate the information into the network where the ContextRouters maintain tables of contexts that can be reached via each of their links.

B. Message Matching

A user context c in the system consists of any number of context attributes α. $A(c)$ denotes the set of all the attributes that make up a context c. 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, CONTEXTCAST currently supports the typical numerical types such as *integer* or *float*, as well as more complex types such as an enumeration *gender* or a *class* attribute based on a class hierarchy. The system can be extended to support arbitrary attribute types with associated matching predicates. Figure 2 shows an example of such a context c.

A (context) message m addresses clients by specifying constraints on their context attributes. These constraints need to be fulfilled for a user (or rather their context) to actually receive a message. A constraint (or *attribute filter*) ϕ is a tuple *(type, name, predicate, value)*, where type$_\phi$ is the attribute type, name$_\phi$ is the attribute name, and predicate$_\phi$ can be any predicate that is defined on the attribute type. The attribute filters in a contextcast message serve purely to address users. Thus, the messages have an additional payload, which is the actual message content. Figure 3 shows such a message m.

For any given attribute filter, the system can thus evaluate the constraint 'value$_\alpha$ predicate$_\phi$ value$_\phi$', e.g., for the attribute *age*: $29 > 15 \rightarrow \texttt{true}$. 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 m determines

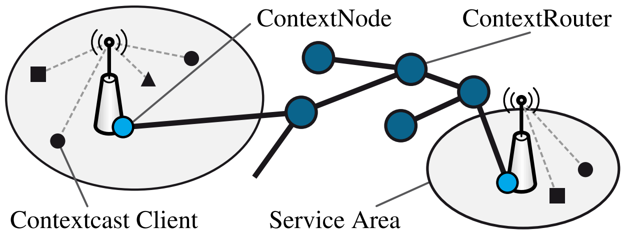


Figure 1: The CONTEXTCAST system

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

Figure 2: Example of a user context

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

Figure 3: Example of a contextcast message

whether it *matches* a user's context c and thus needs to be forwarded and eventually delivered to that user.

III. Continuous Context Aggregation

In the following Sections, we first present the idea of aggregating user contexts, together with a formal aggregation condition and an algorithm that fulfills this condition. After that, we derive a similarity metric for the contexts in our system. We then use this metric in a continuous algorithm to aggregate new contexts in our system and remove invalidated ones.

A. Context Aggregation

Without knowledge of user contexts, the system must broadcast messages (or geocast, with knowledge about access networks' service areas) to disseminate a message to all potential receivers. Information about reachable contexts on the routers allows for a more directed forwarding. However, as discussed in Section I, maintaining complete knowledge of all users on the routers impairs scalability due to the necessary updates. Therefore, we propose an aggregation of user contexts, i.e., propagating only a summary view of reachable contexts.

Definition 1 (Aggregation of user contexts)**.** *Let C be a set of User Contexts $c_1, \dots, c_n$. An* Aggregation *C' of these contexts is a set of contexts $c'_1, \dots, c'_k$ (where typically $k \ll n$) for which the following* aggregation condition *holds.*

Definition 2 (Aggregation Condition)**.** *Let C and C' be two sets of contexts. C' is an aggregation of C* iff *every message m that matches at least one context in C also matches at least one context in C'.*

Definition 1 abstracts from the actual aggregation method; all that is required is that the above mentioned condition holds for a given aggregation approach. The aggregation condition ensures that every client receives at least all the messages they would have received without the aggregation of client contexts. That is, no client with a matching context falsely misses a message because of the aggregation ("false negative").

Based on the semantics we presented in Subsection II-A and Definition 1 and 2, we propose an algorithm to aggregate two contexts into one: It aggregates all attributes that occur only in either c_1 or c_2 directly into c_{aggreg}. For all attributes that occur in both contexts, it merges the values and then aggregates the resulting new attribute into c_{aggreg}. This merge of attribute values needs to be defined for the different attribute types in our system. For example, for WGS84 coordinates, a merge of two values is the bounding rectangle of both positions. Or for ordered, numerical types, the merge of values can be either

Algorithm 1 Pairwise Context Aggregation

```
Require: Two user contexts $c_1$ and $c_2$ (singleton contexts or already aggregated ones).
Ensure: An aggregated context $c_{\text{aggreg}}$.
  $c_{\text{aggreg}} \leftarrow \emptyset$
  for all $\alpha_{\text{i,1}} \in A(c_1)$ do
    if $\exists \alpha_{\text{j,2}} \in A(c_2) : \text{name}(\alpha_{\text{j,2}}) = \text{name}(\alpha_{\text{i,1}})$ then
      $\alpha_{\text{merged}} \leftarrow \alpha_{\text{i,1}}$
      $\text{value}(\alpha_{\text{merged}}) \leftarrow \text{value}(\alpha_{\text{i,1}}) \cup \text{value}(\alpha_{\text{j,2}})$
      $c_{\text{aggreg}} \leftarrow c_{\text{aggreg}} \cup \alpha_{\text{merged}}$
    else
      $c_{\text{aggreg}} \leftarrow c_{\text{aggreg}} \cup \alpha_{\text{i,1}}$
    end if
  end for
  for all $\alpha_{\text{j,2}} \in A(c_2)$ do
    if $\alpha_{\text{j,2}} \notin c_{\text{aggreg}}$ then
      $c_{\text{aggreg}} \leftarrow c_{\text{aggreg}} \cup \alpha_{\text{j,2}}$
    end if
  end for
```

the set resulting from the union of all values or it can be a closed interval from the smallest to the largest value. The definition of such a merging of values must take care that it works for both single values and values that resulted from a previous aggregation. In addition, to ensure that the aggregation condition holds, the predicates used in attribute filters must be adapted to this definition. In detail, we have to check now whether the addressed attribute values occur in the values of aggregated attributes, which may now contain ranges or sets instead of single values. This is also rather straight-forward, for example for a numerical type aggregated to an interval the equality predicate must check whether the interval contains the specified value.

Any context resulting from the (repeated) application of this algorithm fulfills Definition 2: Obviously, the aggregation condition holds for an aggregation of two contexts, since the aggregated context contains the union of all attributes and supersets of the attribute values. Also, the algorithm can be applied to both singleton contexts or already aggregated ones. Thus, by repeated execution for pairs of user contexts, it can aggregate any given set of contexts $C = \{c_1, \dots, c_n\}$ into a single context c', for which the aggregation condition holds. □

B. Context Similarity Metrics

While the algorithm from the previous section aggregates a given set of contexts into a single one, it obviously needs to aggregate similar contexts: For example, if we aggregate a context with an *age* of 8 with one with an *age* of 80, the resulting information loss might be enormous (a context with *age* $= [8; 80]$). This would lead to many "false positives", e.g., all messages addressed to an *age* between 8 and 80. Therefore, the question remains "how to select a good set of similar contexts for aggregation?" This requires a metric to measure the similarity between user contexts.

As one can see from the algorithm in the previous section, an aggregation introduces two types of uncertainties into the system: First, *new attribute combinations* from attributes that only occurred in individual contexts: Attributes that previously did not occur in one context together may end up together in an aggregated context. However, all filters in a message must match for a message to be delivered. Therefore, these additional combinations can lead to aggregations matched by messages that would not have matched the individual contexts. Second, *uncertain values* resulting from the value aggregation of attributes that occur in two or more contexts: This may create new values for attributes, which were not present in any of the individual contexts. Again, this can lead to more filters matching an aggregated attribute value, which would otherwise not have matched the individual, unaggregated attributes.

We therefore propose to measure the overall similarity of user contexts by two components, a structural similarity for the attribute sets of two contexts and a value similarity for the attributes occurring in both contexts.

1) Structural Similarity: The structural similarity of two contexts can be measured by the relative overlap in attribute sets between them: With $A(c)$ denoting the set of all attributes making up c (without values), we can express the structural similarity $S_{\text{structural}}$ between c_1 and c_2 formally as:

$$S_{\text{structural}}(c_1, c_2) = \frac{|A(c_1) \cap A(c_2)|}{|A(c_1) \cup A(c_2)|} \quad (1)$$

2) Value Similarity: The similarity of attribute values depends on the respective type of an attribute. We have adapted the similarity of quantitative and qualitative attributes by Gowda et al. [10], [11]. However, we changed the formulae slightly to normalize the similarity measure to $[0;1]$. This change was done to eliminate the influence of the number of overlapping attributes on the overall similarity since that is already part of the structural similarity.

Additionally, user contexts in our system are classified according to a *type* attribute with an underlying class hierarchy. While Gowda et al. mention structural attributes with such an underlying hierarchy, they do not detail how to compute the similarity of these attributes. We obtain the similarity of such values by their proximity in the hierarchy over the total size of the hierarchy. Formally, let n_1, n_2 be two nodes that correspond to attribute k in contexts c_1 and c_2 in a given tree structure and h the height of this tree. Let $l(n)$ denote the level of a node n in this tree. Then, the similarity between n_1 and n_2 – and thus between attributes D_k and E_k – is computed using the first common ancestor a for n_1 and n_2:

$$\begin{aligned} S(c_{1,k}, c_{2,k}) &= S_p(n_1, n_2) \\ &= 1 - \frac{(l(n_1) - l(a)) + (l(n_2) - l(a))}{2 \cdot h} \\ &= 1 - \frac{l(n_1) + l(n_2) - 2l(a)}{2 \cdot h} \end{aligned} \quad (2)$$

Generally, the system we present can support arbitrary types, for which it is possible to define both a similarity metric and an aggregation scheme for the attribute values.

3) Overall Similarity: For the overall similarity of two contexts, we combine the structural and the value similarity as follows: multiply the arithmetic mean of the attribute similarity with the structural similarity of two contexts. Formally, with attributes $1, ..., k$ shared between c_1 and c_2 and n attributes in an aggregated context, this leads to

$$\begin{aligned} S(c_1, c_2) &= S_{\text{structural}}(c_1, c_2) \cdot \frac{1}{k} \sum_{i=1}^{k} S_{\text{value}}(c_{1,i}, c_{2,i}) \\ &= \frac{k}{n} \frac{\sum_{i=1}^{k} S(c_{1,i}, c_{2,i})}{k} \\ &= \frac{1}{n} \sum_{i=1}^{k} S(c_{1,i}, c_{2,i}) \end{aligned} \quad (3)$$

This multiplication of the two components matches the intuitive notion that for contexts with identical attribute sets, thus $S_{\text{structural}}(c_1, c_2) = 1$, only the value similarity is important. For contexts with disjoint attribute sets, and thus $S_{\text{structural}}(c_1, c_2) = 0$, the overall similarity is 0, independent of the value similarity (which is also 0, since there are no overlapping attributes for which to compute the value similarity).

C. Continuous Aggregation Algorithms

Based on the previously defined similarity metric and the pairwise aggregation we propose the following algorithms. They provide a continuous aggregation of new, similar contexts and a disaggregation of removed contexts.

Context Addition. Using the similarity measure from the previous section, the system can compute the similarity between pairs of contexts, either singleton ones or aggregations. We can then aggregate a newly added context with an existing one as follows: When a router receives a newly added context c_{add} from one of its neighbors, it computes the similarity between c_{add} and all other contexts associated with that link. If the highest similarity between c_{add} and an existing context c_{ex} exceeds a configurable threshold similarity, s_{th}, the system aggregates this new context with the existing one. This similarity threshold serves as a parameter for the operation of Algorithm 2. It determines how similar two contexts must be before they are aggregated and thus whether the system has a very fine or a rather coarse aggregation. By adjusting its value, either manually by a human administrator or autonomously, one can optimize the system for the observed contexts and messages. We evaluate reasonable values for s_{th} under different scenarios in Section IV. If the resulting context c_{new} differs from c_{ex}, c_{new} must be propagated as an update for c_{ex}. In this case, we call c_{add} a "defining context" for c_{new} (this is important for the context removal). When a defining context is removed from an aggregation, we need to recompute the aggregation from the remaining parts. For this purpose, a router stores individual contexts that it aggregates but only propagates the aggregations to its neighbors. Algorithm 2 shows this sequence formally.

Context Removal. The removal algorithm for contexts (e.g., when a user disconnects from the system) is rather similar to the addition algorithm. Basically, when a context c_{rem} is supposed

Algorithm 2 Context Addition

```
Require: A newly added context $c_{\text{add}}$, received via a link $l$.
Ensure: $c_{\text{add}}$ is attached to the link $l$.
  local context store $\cup\, c_{\text{add}}$
  $s_{\max} \leftarrow 0$
  for all $c_i \in$ {contexts already attached to $l$} do
    $s_i \leftarrow s(c_{\text{add}}, c_i)$
    if $s_i > s_{\max}$ then
      $s_{\max} \leftarrow s_i$
      max $\leftarrow i$
    end if
  end for
  if $s_{\max} > s_{\text{th}}$ then
    $c_{\text{new}} \leftarrow$ merge $c_i$ and $c_{\text{add}}$
    if $c_{\text{new}}$ differs from $c_i$ then
      Mark $c_{\text{add}}$ as defining context for $c_{\text{new}}$.
      Replace $c_i$ on $l$ with $c_{\text{new}}$.
      Propagate $c_{\text{new}}$ as update/replacement for $c_i$ on all links
      except $l$.
    end if
  else
    Attach $c_{\text{add}}$ as singleton context to $l$.
    Propagate $c_{\text{add}}$ on all links except $l$.
  end if
```

to be removed from a link, a node first checks whether c_{rem} is a singleton context or part of an aggregation. A singleton context can simply be removed and its removal propagated to neighbors. If the context is part of an aggregation, it depends if it is a defining context for this aggregation. If it is not, its removal does not affect this context. If it is a defining context, the algorithm needs to recalculate the aggregation from the stored contexts it contains, and potentially propagate an update for this context. See Algorithm 3 for a formal description.

Algorithm 3 Context Removal

```
Require: A context $c_{\text{rem}}$ to remove from a link $l$.
Ensure: $c_{\text{rem}}$ is removed from the link $l$.
  if $c_{\text{rem}}$ is a singleton context then
    Remove $c_{\text{rem}}$ from $l$.
  else {$c_{\text{rem}}$ is part of an aggregation $c_{\text{aggreg}}$}
    if $c_{\text{rem}}$ is defining context for $c_{\text{aggreg}}$ then
      Recalculate $c_{\text{aggreg}}$ from local context store.
      Propagate an update of $c_{\text{aggreg}}$ on all links except $l$.
    else
      Do nothing.
    end if
  end if
```

D. Optimized Aggregation Candidate Selection

For a newly added context, a node must calculate its similarity with all other contexts that are attached to the same link. Especially when there are a large number of contexts attached to a link, this calculation can cause a rather large amount of load for the node. However, it is possible to simplify this calculation by first selecting a suitable set of aggregation candidates.

The similarity of two contexts consists of the product of the structural and the value similarity. By employing the concept of Bloom filters [12], it is possible to calculate an upper bound of the structural similarity very efficiently: Let b_i be a bit string, which represents the structure of a context c_i as follows: Every attribute in c_i is hashed to a position in b_i, which is set to 1.

Also, let $|b_i|$ be Hamming weight of b_i, i.e., the number of 1 bits in b_i. Then, the structural similarity between c_{ex} and c_{add} can be expressed as

$$S_{\text{structural}}(c_{\text{ex}}, c_{\text{add}}) = \frac{|A(c_{\text{ex}}) \cap A(c_{\text{add}})|}{|A(c_{\text{ex}}) \cup A(c_{\text{add}})|} = \frac{|b_{c_{\text{ex}}} \wedge b_{c_{\text{add}}}|}{|b_{c_{\text{ex}}} \vee b_{c_{\text{add}}}|} \quad (4)$$

Binary AND and OR can be computed very fast on conventional computer systems; for counting the 1 bits, efficient algorithms exist and some processors even offer a dedicated machine instruction such as POPCNT. The hashing of attribute names to bit positions can be done once and cached.

From this structural similarity, we can now derive a set of candidate contexts as follows: for a given similarity threshold S_{th}, we exclude a context from the candidate set (and thus skip the calculation of the value similarity) if the structural similarity is smaller than S_{th}: From the condition $S = S_{\text{structural}} \cdot S_{\text{value}} > S_{\text{th}}$, we can derive that also $S_{\text{structual}} > S_{\text{th}}$ (for $S_{\text{value}} = 1$). However, the hashing of attribute names can lead to collisions and thus a higher structural similarity value. Thus, for two contexts with a similarity above the threshold, the algorithm needs to recalculate the structural similarity based on the actual attribute sets instead of their bit string representations.

IV. Evaluation

We have implemented our method for aggregating contexts, described in Algorithms 1, 2, and 3 in a prototype implementation for a network simulator to evaluate its performance. Also, the simulations serve to find sensible values for the parameter of the algorithm, the *similarity threshold* s_{th}. Since our approach is designed to reduce update messages in the overlay, we are mainly interested in message overhead in the overlay and less interested in things such as underlay latency, Therefore, we implemented the prototype in the Peersim [13] network simulator.

For the simulation, the network topology consists of 400 ContextRouters, which are connected to form an undirected acyclic graph. Links are constructed according to the Heuristically Optimized Trade-off model (cf. [14]): the nodes are sequentially added and uniformly distributed over an area of size $[0, 1] \times [0, 1]$. Each new node is connected to an existing node such that a weighted distance metric of network and Euclidian distance is minimized. The parameter α of this weighted distance was selected as $\alpha = \sqrt{n}$ for a network of n nodes. According to [14], this leads to an Internet-like power-law distribution for the node degrees.

A fraction of 60% of these nodes are then selected as access nodes for receivers of contextcast messages and assigned a rectangular service area. The edge lengths of these service areas are uniformly distributed between 0.04 and 0.06. Each service area is then placed such that the corresponding node is in its center. Any service area that extends beyond the $[0, 1]$ interval in either direction is cut off to remain within the boundaries of the simulation.

Not all routers with degree 1 are assigned an access network. However, even though this means no user is ever connected to such a node, they can still be used to send messages. For example, a provider of commercial messages might operate its own overlay node to send its messages but no client ever connects to such a pure message source.

The clients we simulate are created in the following way: we simulate an average of ten user contexts per access network, corresponding to about 2400 in total. Each of these contexts has an associated location, assigned such that it is identical its access network's service area. With probability 0.5, a context has one of three different structural attributes; the hierarchy for the structural attributes is generated randomly with a uniform height distribution between two and five and between zero and three children per node. Additionally, each contains between two and three quantitative attributes, uniformly chosen out of a set of seven different ones. The value for a structural attribute is chosen by descending into the associated hierarchy, at each node uniformly choosing one of the children; the probability to further descend into the hierarchy is chosen such that it favors deep nodes, or in the case of a type hierarchy, a detailed type specification over a more general one. The values of the quantitative attributes follow normal distributions with different means and standard deviations.

The network load is determined by the rates of messages and update during the simulation. For the simulation, we vary the ratio of update and message per cycle such that the sum of both remains constant, i.e., $5 : 1$ (message-dominated scenario), $3 : 3$ (balanced scenario), and $1 : 5$ (update-dominated scenario).

These updates and messages are generated randomly, according to the following instructions: for a fraction of 70% of all updates, the update results from a change of a single attribute. If the attribute that is being changed is the location, a node uniformly selects one of its ten nearest neighboring ContextNodes and connects to it, updating its location; for the other attributes, the values are changed in the same manner as they are created for new contexts. For the remaining 30% of updates, a uniformly selected context is removed and a different random context – created in the previously described way – added. This models users changing there contexts as well as leaving and new ones joining the system, respectively.

Context messages are also sent randomly and created similar to context updates: they contain a target location, with $p = 0.5$ a structured attribute, and between zero and two quantitative attributes.

For our evaluation, we run five simulations of this setup with different random seeds; we then compute the arithmetic mean of the simulation runs.

To quantify the performance of our approach, we measure the message overhead, i.e., messages that are not directly necessary for the correct transmission of messages to the matching recipients. In particular, we measure context updates and false positive messages.

Update Messages. A context aggregation conceals some of the users' updates from the router. Therefore, we measure the number of updates that a router actually forwards to its neighbors, both with and without an aggregation of user contexts. We denote u be the number of updates a router propagates in the unaggregated and u_{a} be the number of updates it must propagate with the aggregation.

False Positives. Besides the load of update messages, an aggregation of contexts causes load in the overlay network by transmitting messages without matching receivers, i.e., false positives. False positive messages can always be filtered closer to the receiver with more detailed knowledge, however, in the worst case, this may only be possible at the access node or on a client's mobile device. We therefore measure the amount of false positive messages, m_{fp}, caused by the loss of routing information when aggregating contexts.

A. Effect of the Similarity Threshold

To evaluate the impact of an aggregation of user contexts, we compare the amount of false positives caused by the aggregation to the reduction in updates. We carry out simulations with similarity thresholds s_{th} of 0.5, 0.6, and 0.7 to adjust the granularity of the aggregation (from coarse routing information to detailed, respectively).

The simulations cover a time of 3000 cycles; however, for the results, we limit the simulation time to 2500 cycles, leaving out the first 500 cycles. The simulation contains a warm-up phase, during which the contexts are first entered into the system; this generates a significant amount of updates, however, no messages are sent during that time. Limiting the evaluation to the cycles 500 to 3000 eliminates the warm-up phase from the measurements.

The results for the three different ratios of update to message rate is shown in Figure 4. As one can see, for each scenario, from the update-dominated on the left to the message-dominated on the right, lowering the similarity threshold causes an increase in false positives: going from $s_{\mathrm{th}} = 0.7$ to 0.6 causes an increase in false positives between 220% and a little over 330%, while lowering it from 0.6 to 0.5 causes only a further increase between 36% and 70%. The smaller increase when lowering s_{th} from 0.6 to 0.5 can easily be explained: there is an upper limit on the amount of false positives for any given message, no matter how coarse the aggregation of the contexts. The upper limit results from the target location of every message. As soon as a message is delivered to all access network that intersect a message's target location, the delivery is complete. Thus, the maximum number of false positives any message causes is simply the amount of messages required to deliver it to all access networks that overlap the target location.

Overall, the increase in false positives fits the intuition that a coarser aggregation results in fuzzier knowledge about user contexts on the routers. Therefore, the dissemination cannot be pruned as early as with complete, unaggregated knowledge on each router. As such, messages get forwarded to routers with better, less aggregated knowledge or even to the access networks before it can be determined that they have no matching receiver.

Looking at the decrease in updates, we can again see similar results in all three scenarios. Aggregating contexts causes a clear decrease in updates in the system. A similarity threshold $s_{\text{th}} = 0.6$ saves between 4.5 and almost 9.8 times the amount of messages than aggregating with $s_{\text{th}} = 0.7$. However, when we lower the threshold further to 0.5, we observe that this effect reverses. That means, if the system aggregates contexts too aggressively, the update load increases again, almost halving the savings it achieved with $s_{\text{th}} = 0.6$. The reason for this effect is that a lower threshold causes rather different contexts to be aggregated. Since we aggregate such dissimilar contexts, there is a high probability that an update is actually a defining context for the resulting aggregation. This changes the aggregation, and therefore results in another update being propagated further into the network.

When we compare the numbers for false positives and updates, we can see that by aggregating similar contexts, the amount of saved updates vastly outweighs the amount of false positives it introduces. The amount of additional false positives is less than 0.1% of the reduction of updates our approach achieves. It would require a system with a much higher volume of context messages or quasi-static user contexts to not benefit from an aggregation.

Compared to the update messages in the system without context aggregation, our approach lowers the amount between 18% and almost 25% on average for $s_{\text{th}} = 0.6$, with some scenarios achieving a reduction up to almost 30%.

B. Aggregation Quality over Time

Due to the dynamic of the system, new contexts continuously get added and old ones removed from aggregations. This might lead to a degeneration of the aggregations over time and thus a gradual increase in the number of false positives. (However, since the aggregations are becoming more general, this effect would also reduce update messages further.) To investigate this behavior, we simulate the three scenarios from before with a similarity threshold $s_{\text{th}} = 0.5$. The low threshold results in the most aggregations of context updates and thus also the most changes to aggregations over time.

Figure 5 shows the false positives during the simulation run time. The false positive are measured as the average number during each simulation cycle, taken over time windows of 100 cycles. For all scenarios, the Figure shows a very low amount of false positives in the beginning. However, this is due to the warm-up phases (of different lengths) in the simulation, during which no messages are sent and thus no false positives can occur.

During the remaining simulation time, the average of false positives stays relatively constant for all three scenarios. There is no noticeable increase in false positives over time. Therefore, even after 3000 cycles, i.e., between 3000 and 15000 updates in total, there is no discernible negative effect on the quality of the aggregations in our system. However, it is possible that a larger change in user contexts or a rare sequence of contexts still lead to a degeneration of the aggregation. For example, this would happen if the contexts successively covered an ever larger part of the attributes' values, thus continually increasing an aggregation in size until it matches practically every message. This is subject to future work, though.

V. Conclusion & Future Work

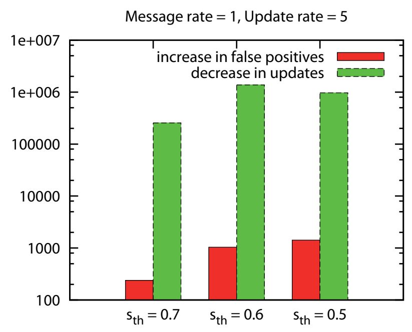
Contextcast enables an interesting group communication paradigm, which addresses users via a number of attributes instead of explicit groups. For a directed forwarding of messages, the routers require information about the users in the system. Keeping this information up-to-date poses a serious scalability challenge.

An aggregation in Contextcast can be used to lower the update load on the system by as much as 30% and thus improve the overall scalability. Such an aggregation of user contexts results in a loss of information available for routing messages. At the same time, it introduces a negligible amount of additional false positive messages into the system, which would not be necessary with unaggregated contexts.

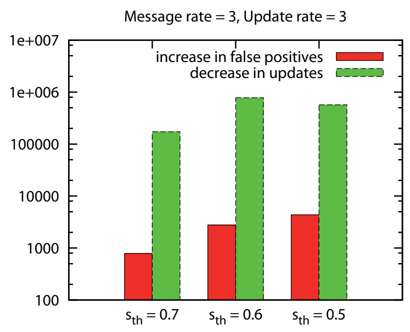
We have shown an approach to aggregate an arbitrary number of contexts into a single one, which was designed such that it fulfills the aggregation condition and thus does not cause false negatives. We have also presented two algorithms that allow us to continuously aggregate user contexts in Contextcast, while still being able to remove old contexts from the system. The algorithm uses a similarity measure we developed for user contexts to decide when a new context should be aggregated with an existing one. Our similarity measure takes into account both the structure of user contexts, i.e., which attributes are present, as well as the values of these attributes, i.e., how close are two attribute values in the context space.

A parameter *similarity threshold* s_{th} allows us to tune the aggregation algorithm to the contexts and messages that occur in the system. It controls the trade-off between update messages and false positives. The evaluation results suggest that a value for the similarity threshold around 0.6 maximizes the update reduction in our concrete scenarios. However, for less dynamic contexts and even higher message rates, we can imagine a higher value for s_{th}; in such a scenario, the false positives may dominate the trade-off, while the almost static contexts cause very little load in the network.

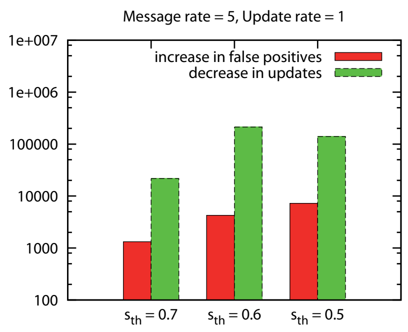
In the future, we are planning to investigate a self-tuning extension to the presented algorithm. This would allow the routers in the system to autonomously and independently adapt the similarity threshold, based on the observed contexts and messages. It could be further improved by having a feedback mechanism for the access networks to signal when they receive too many false positives. This allows the routers to verify their aggregations, potentially adjusting the threshold or certain aggregations in the process.



(a) update-dominated scenario



(b) balanced scenario



(c) message-dominated scenario

Figure 4: Effect of the similarity threshold in different load scenarios

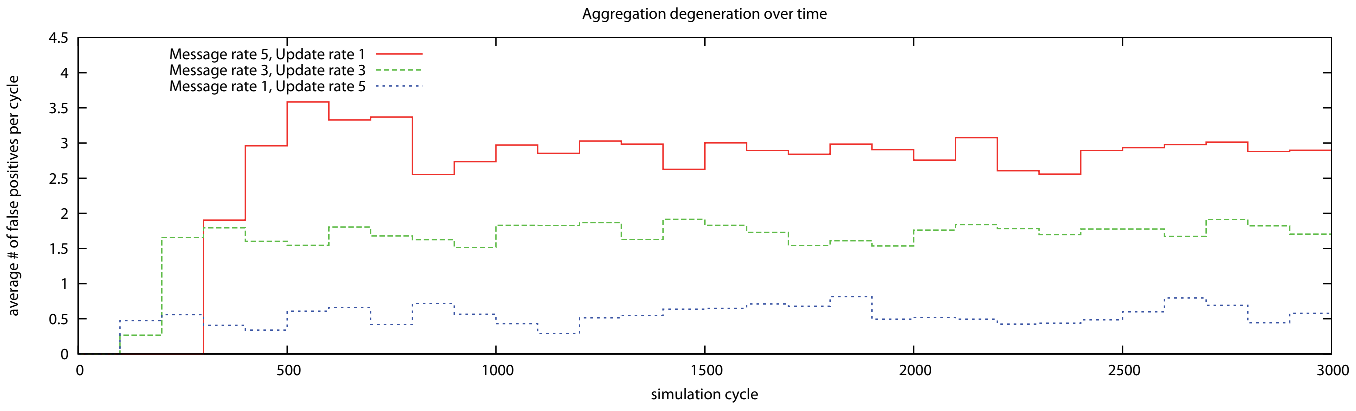


Figure 5: Degeneration of aggregations over time

Another extension we are planning is to incorporate network distance into the aggregation. This would allow the system to have coarser information the further a context is propagated into the network. For example, if two or more contexts were propagated a number of hops without being aggregated, a router could dynamically lower the aggregation threshold, thus preventing too detailed information from being propagated far into the network.

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. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-Based Addressing and Routing: A General Model and its Application," Department of Computer Science, University of Colorado, Boulder, CO 80309, USA, Tech. Rep. CU-CS-902-00, jan 2000.
[3] G. Mühl, "Generic Constraints for Content-Based Publish/Subscribe," in *Cooperative Information Systems*, ser. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, 2001, pp. 211–225.
[4] R. Xu and D. Wunsch II, "Survey of Clustering Algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.
[5] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the 5th Berkeley Symp. on Mathematics Statistics and Probability*, L. M. LeCam and J. Neyman, Eds., 1967.
[6] G. H. Ball and D. J. Hall, "A clustering technique for summarizing multivariate data," *Behaviorial Sciences*, vol. 12, no. 2, pp. 153–155, Mar. 1967.
[7] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.
[8] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1996, pp. 103–114.
[9] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *vldb'2003: Proceedings of the 29th international conference on Very large data bases*. VLDB Endowment, 2003, pp. 81–92.
[10] K. Gowda and E. Diday, "Symbolic Clustering Using a New Similarity Measure," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 2, pp. 368–378, Mar/Apr 1992.
[11] K. C. Gowda and T. V. Ravi, "Agglomerative clustering of symbolic objects using the concepts of both similarity and dissimilarity," *Pattern Recognition Letters*, vol. 16, no. 6, pp. 647–652, 1995.
[12] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[13] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim Simulator," `http://peersim.sf.net`, 2010-03-21.
[14] A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou, "Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet," in *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, 2002, pp. 110–122.

Published in 2010 6th EURO-NF Conference on Next Generation Internet (NGI). Paris, France, 2-4 June 2010.
© IEEE 2010
http://dx.doi.org/10.1109/NGI.2010.5534466