Featurecast: Lightweight Data-Centric Communications for Wireless Sensor Networks

Michał Król, Franck Rousseau, and Andrzej Duda

Grenoble Institute of Technology
CNRS Grenoble Informatics Laboratory UMR 5217
`firstname.lastname@imag.fr` 681, rue de la Passerelle, 38402 Saint Martin d'Hères, France

Abstract. We introduce the concept of *Featurecast* in which addressing and routing are based on node features defined as predicates. For instance, we can send a packet to the address composed of features {`temperature and Room D`} to reach all nodes with a temperature sensor located in Room D. Each node constructs its address from the set of its features and disseminates it in the network so that intermediate nodes can build routing tables. In this way, a node can send a packet to a set of nodes matching given features. Our experiments and evaluation of this scheme show very good performance compared to Logical Neighborhoods and IP multicast with respect to the memory footprint and message overhead.

Keywords: wireless sensor networks, data-centric routing, multicast, IPv6

1 Introduction

Wireless sensor networks need to support specific traffic patterns related to sensor applications. One of their most important goals is to forward collected data to one or several sinks. They also have to support downward traffic from a sink to all or some sensor nodes. This traffic pattern results from the need for configuring nodes, querying sensors, or transmitting commands to actuators. Sensor nodes may require communication with other nodes, for instance for aggregating data or collaborating on a common reaction to local events.

In addition to the standard unicast communication, many sensor network applications may benefit from *multicasting* to forward packets to a group of nodes or report data to multiple sinks [1,2,3]. Multicasting results in a reduced number of packets forwarded in the network, which in turn limits energy consumption—compared to unicast, less packets are transmitted when using multicast because they are only replicated when needed.

Unicast and multicast are *address-centric* communication modes in which source and destination addresses identify endpoint nodes. Such modes are suitable for structured addresses that result in small routing tables. *Data or content-centric* routing focuses on the packet content instead of communication endpoints. In the context of sensor networks, Directed Diffusion was one of the first proposals for sensor data dissemination based on this approach [4,5]: sensor nodes attach *attributes* (name-value pairs) to generated data, consumers specify interests for sensor data in terms of attributes, and sensors send unicast data packets to consumers. The data-centric paradigm is appealing for sensor networks, because it fits very well their data-oriented nature, however the approach incurs significant overhead by attaching attributes to data, which is prohibitive in energy constrained networks. Directed Diffusion uses flooding to disseminate interests for sensor data, which is inefficient in wireless networks. Moreover, it does not scale well in networks with many sinks that transmit many different queries [6].

Logical Neighborhoods (LN) proposed a similar abstraction, but at the application layer: a node declaratively specifies the characteristics of its neighbors in terms of attributes and the cost of reaching them [7]. A template specifies the attributes of a node. Nodes broadcast their attributes to neighbors that store the information in a table and re-broadcast them if there is a change in the existing state created by the advertisement. The propagation of advertisements creates a state distributed over the nodes that contains the cost of reaching the closest node with a given attribute. To find a node with an attribute, a node broadcasts an application message containing the neighborhood template. The approach suffers from significant overhead of transmitting attributes and templates. The overhead in terms of the number of transmitted messages is also important compared to the ideal multicast routing based on the minimum spanning tree rooted at the sender. Finally, all solutions based on data-centric approach require a specific grammar that may be difficult to process by sensors and significantly slows down the routing process.

In this paper, we propose *Featurecast*, a network layer communication mode well suited for sensor networks. One of our main design goal was to create a system able to cooperate with already existing IPv6 networks. Unlike Directed Diffusion or LN, Featurecast is address-centric, but it uses a data-centric approach to create addresses and operate routing: addresses correspond to a set of features characterizing sensor nodes. Features are not attributes, but predicates, which allows us to represent them in a compact way in address fields of packets and in routing tables.

Nodes disseminate Featurecast addresses in the network following a structure usually constructed for routing standard unicast packets such as a Collection Tree (CT) [8] or a DODAG (Destination Oriented Directed Acyclic Graph) [9]. Intermediate nodes merge the features of nodes reachable on a given link and construct a compact routing table for further packet forwarding. Based on the routing tables, a packet can reach all nodes characterized by a given set of features. Our proposal does not define any specific grammar for features, which makes it extremely flexible and easy to use. We propose a specific compact encoding allowing for fitting a Featurecast address into the standard multicast IPv6 address field. To the best of our knowledge, Featurecast is the only protocol able to introduce a data-centric approach into the traditional IPv6 networks.

We have implemented Featurecast and the proposed scheme for routing in Contiki OS [10] and integrated them within its uIPv6 (micro Internet Protocol) stack. The implementation provides network layer Featurecast unlike application layer overlays in other proposals.

To evaluate Featurecast, we have simulated in Cooja an application scenario developed for CoAP group communications [11] with several sensors placed across buildings, wings, and rooms. We have compared Featurecast with LN and IP multicast with respect to the memory footprint and message overhead. Featurecast results in a significantly smaller memory footprint and a lower average number of messages for updating routing tables compared to other schemes.

2 Featurecast

We want to provide a new communication mode for wireless sensor networks, in which we usually want to designate relevant sensor nodes or data destinations by means of their characteristics and not with some low level identifiers or node addresses. For instance, we may want to get the "*average temperature on the 1st floor*" or "*turn off all the lights in the building*". Such reasoning is close to applications that take advantage of sensors and actuators. Obviously, we could support such messages by associating a multicast group with each query, however, the number of such groups may quickly become too large, because of all possible combinations of characteristics.

We introduce below the notion of Featurecast addresses, present the construction of routing tables, and the forwarding process.

2.1 Featurecast Addresses

We assume that each sensor defines a set of its features, for instance its capability of sensing the environment (`temperature`, `humidity`), location (`sector 5`, `1st floor`), state (`low-energy`), or some other custom features (`my favorite nodes`). Features are *predicates*, i.e., statements that may be true or false (in the previous examples, we explicitly state features that are true). Predicates are commonly used to represent the properties of objects and we use them here to represent the properties of sensors: if f is a predicate on sensor X, we say that f is a property of sensor X. Note that features are not attributes (i.e., `name:value` pairs), which allows us to represent them in a much more compact way without loosing any flexibility. We assume that there is no coordination in defining features, but all features are known and each node can define its features at will.

A sensor node derives its Featurecast address from its features—more formally, a node address is the set:

$$A = \{f_1, f_2, ..., f_n\}, f_i \in \mathcal{F}, \tag{1}$$

where f_i is a feature predicate and $\mathcal{F}$ is the set of all possible features with cardinality of N. Features in the network may evolve in time and nodes may change their features, for instance the location of a node may change when it moves or a sensor may define a state of high temperature when exceeding a given threshold. Note that N, the total number of features in the network does not depend on the number of nodes, but rather on applications that define node characteristics.

The destination address may contain a subset of features—we say that it *matches* a node address, if the node address contains the destination address:

$$D = \{f_1, f_2, ..., f_k\}, f_i \in \mathcal{F}, D \textbf{ matches } A, \text{ if } D \subset A$$

For instance, a packet to `temperature, 1st floor` will match nodes defining both `temperature` and `1st floor` in their addresses. The conjunction seems the most useful way of representing nodes of interest for most sensor network applications. In the real world, somebody can describe an object with a set observed features. Such an approach is thus a very natural way of designating objects.

We can consider the node address as a representative of all possible multicast groups that would be created based on the node features to make it reachable for any combination of features using the traditional multicast groups, which gives $\sum_{k=0}^{n} C_n^k = 2^n$ addresses for n features.

Note that such an addressing schemes allows additional communication modes, especially for machine-to-machine communication. For example, a node addressing a packet using its own location can reach all sensor in the same room/floor/-building without creating any dedicated multicast group.

2.2 Constructing Routing Tables

Forwarding packets based on Featurecast addresses requires the construction of routing tables. Featurecast does not store the information about nodes, but just the features reachable through a given neighbor. Constructing routing tables involves the creation of a DODAG or a Collection Tree and then advertising features along the routing structure. In our implementation, we have used our proper way of constructing a DODAG described below, but Featurecast can also operate along any protocol that creates such a structure (e.g. RPL).

Creating a routing structure. Featurecast needs a DODAG or a Collection Tree to construct routing tables and forward data packets. However, using only one routing structure may be inefficient, because two nodes on different branches need to communicate by passing through the root. We can alleviate this problem by deploying multiple DODAGs or Collection Trees in the network (cf. Figure 1). Each node stores the information about all DODAGs present in the network, but to send a packet, it uses only one DODAG, the one with the root closest to the node. Multiple DODAGs deployed in the network result in nodes that are close to any root, which improves communication efficiency.

We propose to construct a DODAG that takes Featurecast into account: the root starts the DODAG construction process by broadcasting route advertisements with the distance set to 0. Every node receiving such a message checks

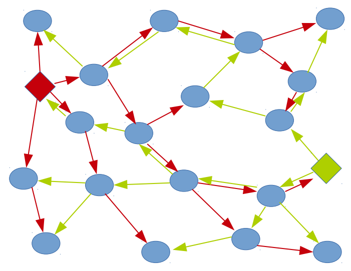

Fig. 1. Multiple DODAGs deployed in the same network for better connectivity.

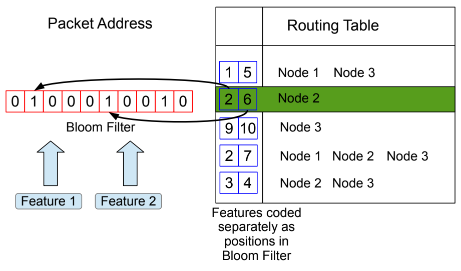

Fig. 2. Proposed compact representation of features: Bloom filter in the destination address and a bit position list in the routing table.

if it knows a node closer to the root. If not, it sets the message sender as its preferred parent and rebroadcast the message with a modified distance d that takes into account the similarity of nodes—a node receiving a route advertisement from a neighbor compares a set of features with its own adds the result to the advertised metric:

$$d = h - (|F_n \cap F_s|)/(|F_n| + 1), \tag{2}$$

where h is the hop count (original metric of RPL OF0), F_n is the set of node features, and F_s is the set of the sender features. Note that $h + 1 > d$.

By grouping similar sensors, we decrease the overall cost of forwarding Featurecast messages, because a packet addressed to a given group of nodes will be duplicated less often. Moreover, nodes are much more likely to find a common ancestor thus reducing communication overhead. In the rest of the paper, we will refer to this routing structure as the Featurecast DODAG.

Advertising Features The process of advertising features starts at leaf nodes that send their features to their preferred parent. Parents obtain the features from their children nodes, add their own features, and forward the list of features reachable through them to their own parent. The process continues up to the root of the DODAG. Finally, the root node obtains the list of all features in the network and it can use it to forward packets to relevant neighbors. The sink can also initialize the process in the reverse direction by sending its features to children nodes, which speeds up machine-to-machine communication.

When a node receives a feature already in its routing table, it does not forward it to its neighbors and ignores subsequent advertisements, so most of the changes in features will only result in localized transmissions, as shown in Section 4. Even if a single node fails, other nodes may define the same feature and the routing tables may remain valid.

2.3 Forwarding

When features have propagated in the network and filled in routing tables, nodes can send packets that will take advantage of the routing information. Their destination address contains a set of features that intermediate nodes match against the routing tables—they interpret the address as a conjunction of all features, as they all need to appear in the matching routing entry. In this case, the intermediate node transmits the packet to all neighbors having the matching entry. As a result, the destination will receive a given packet if its address contains all features in the destination address.

2.4 Topology Maintenance

It is possible that some neighbors of a sensor node disconnect due to topology changes, node failures, or battery depletion. For detecting disconnected peers and maintain a valid topology, Featurecast relies on hello messages and RPL local and global repair mechanisms. In case of neighbor disconnection, the node checks the set of features advertised by other connected neighbors—if they provide all the features advertised by the disconnected node, there is no need for an update. Otherwise, the node informs its parent node about the absence of the features available through the disconnected neighbor. The parent node will do the same with respect to its neighbors and the process continues until the root node if necessary.

It is also possible to delay sending the advertisement about missing features until the node receives a packet using them. A node changing its parent node or changing its set of features, advertises the change as explained in Sect. 2.2.

3 Compact Representation of Features

We have followed several design guidelines for the compact representation. First, we want an open network able to accept any feature defined on the nodes. Second, the addressing scheme should not depend on the number of features defined in the network—we do not want to force the user to define a hierarchy of features. Most of data-centric approaches use a grammar exchanged in a text form. Such an approach is often a problem while integrating such solutions into real life scenarios. We want our solution to still use user-friendly addresses, while being easily stored and processed by nodes. We then need fixed-size addresses for efficient forwarding and possibility to integrate Featurecast within the standard IPv6 addressing scheme with 112 bits in the multicast IPv6 address. Such integration will show that a data-centric approach may have the same overhead as address-centric solutions and lead to easy integration with existing networks. A part of such an IPv6 address can be used for a global prefix and routed in the Internet. Finally, we want to take into account resource constraints (memory size) of sensor nodes for storing routing tables.

We also want to avoid global synchronization mechanisms disseminating a mapping between features and their binary representation. Such a solution would result in a significantly higher volume of communications and could delay packet forwarding during the feature update. For these reasons, we have decided to use hash functions and a structure allowing to efficiently store many hashes—a Bloom filter.

3.1 Bloom Filters

A Bloom filter is a probabilistic structure allowing for efficient storage of a set of elements. A typical filter contains an array of m bits. At the beginning all bits are set to 0. There are k hash functions that map an element to a bit position in the array. When inserting an element into a filter, we compute k hash functions on the element and set all the resulting bits to 1. If a bit was already set to 1, we do not change it. To check whether an element belongs to a set, we compute the same hash functions on the element and check if all corresponding bit positions are set to 1. If not, we are sure that the element does not belong to the set.

False positives may occur in Bloom filters: it is possible that all bits corresponding to the hash functions on a tested element are set to 1 by other stored elements, even if the element does not belong to the set. The probability of false positives depends on the number of stored elements n and the size of the filter (m and k):

$$p \approx (1 - e^{-km/n})^k. \tag{3}$$

To maintain the same false positive rate with a growing number of elements, we need to increase the number of bits and hash functions, which results in larger memory consumption and increased computational overhead.

3.2 Naive Bloom Filter Solution

A possible solution is to use Bloom filters of the same size to represent a set of features in the destination address and in the routing table entry for each neighbor. To decide where to forward the packet, we only have to verify if all bits set in the address are also set in the routing entry for a given neighbor. However, such a solution limits the number of possible features to store in the routing tables. If n is the number of elements in the filter (features in our case) and p is the required probability of false positives, the minimum number of bits m for the filter is $m \geq n \log_2(e) \times \log_2(1/p)$. To achieve the probability of 2%, we need 5 bits per feature. To fit 112 bits available in IPv6 address, we would be able to store only 22 features with 2% of false positives, which can be good for the packet destination address (as we store features for only one sensor or a group of sensor), but insufficient for routing tables in which we would need to store all features defined in the network in the worst case.

3.3 Bloom Filter in Addresses and a Bit Position List in the Routing Table

We describe the proposed compact representation of features that satisfies our requirements: being able to represent around 10 features in a destination address limited to 112 bits with a small false positive probability and representing potentially all features in the network in routing tables with a small memory footprint.

The proposed solution consists of using a different feature representation in the routing tables while using a Bloom filter in the address field as described above. However, a single feature in the routing table is represented by the positions of bits set in the Bloom filter. For example, a feature that sets bits on positions 5 and 76 in the Bloom Filter, will be represented by these two numbers in the routing table (cf. Fig. 2).

The probability of two different features having the same representation is: $p_N = N/m^k$, where N is the number of features in the network and m is the size in bits of the Bloom filter. The size of each represented feature in an address depends on the Bloom filter size and the number of hash functions: $s = k \log_2(m)$.

Taking into account Eq. 3, this solution allows supporting 200 different features in the routing table with the probability of false positives less than 2%, which satisfies our requirements. As we want to use a 112 bit long Bloom filter and 2 hash functions, we only need 2 bytes to store a feature in the routing table, which results in the routing table of only 400 bytes for 200 features.

Routing Entry Aggregation. As Featurecast may operate over multiple DODAGs, we aggregate the same routing entries from multiple DODAGs: for features defined on the same set of neighbors in different DODAGs, we keep only a single entry, which results in a reduced amount of memory without introducing any computational overhead during forwarding.

Computational Overhead. Our representation of routing tables requires iterating through all present features to forward a packet, which makes the operation limited by $O(n)$, where n is the number of features. However, with n features, we are able to construct $g = 2^n$ groups, which means that in a well constructed system, the computational complexity in terms of the number of groups g is $O(\log(g))$. As features are stored already in a hashed form, each comparison only requires few bitwise operations to check the corresponding bit in the source address Bloom filter, which does not introduce a significant computational overhead, especially by contrast with text comparisons used in many data centric solutions.

To further speed up the forwarding process, we have developed several optimization techniques. First of all, we do not have to iterate through features present at every neighbor. This modification significantly reduces the overhead especially at nodes close to the root, which have many such features. Secondly we start the forwarding process from features being present at only one neighbor. If any of them is present in the source address, we just need to check if this neighbor defines all required features without iterating through the whole table.

4 Implementation and Evaluation

We have implemented Featurecast in Contiki OS (ver. 2.6) [10]. For performance evaluation, we have run simulations in Cooja, a simulator that emulates both the software and hardware of sensor nodes. As an execution platform, we have considered Sky Motes with CC2420 2.4 GHz radio and ContikiMAC as Layer 2.

Contiki supports the RPL routing protocol to builds a DODAG that takes into account the distance to the sink in terms of the number of hops, the metric defined by Objective Function Zero (OF0). We have modified the metric for constructing the Featurecast DODAG (cf. Sect. 2.2).

4.1 Evaluation Setup

We have compared Featurecast with Logical Neighborhoods (LN) [7] and the traditional IP Multicast as it is the recommended solution for group communications in WSN [11]. We have set the parameters of LN (exploration parameter E and the number of credits) to the values used in the LN evaluation [7]. Note that LN does not guarantee packet delivery for a small amount of credits, so we have used the LN recommended values [7].

As there is no implementation of any multicast routing protocol in Contiki (ver. 2.6), we have implemented a simple routing protocol in which nodes willing to join a multicast group just send a message towards their parents in the RPL DODAG using UDP. Each sensor, after receiving the message, waits for an advertisement from its children, adds its own advertisement, and sends it up through the DODAG. We use the number of control messages exchanged for maintaining Featurecast or multicast routing as the main comparison index. They directly influence the energy consumption of nodes and the network lifetime.

4.2 Scenarios

We consider two scenarios: i) the building control application developed for CoAP group communication [11] and ii) a random topology of nodes with random features.

Building Control. The building control scenario uses a deployment scheme in which sensor nodes are placed in several buildings across multiple floors, wings, and rooms. The scenario considers sensor nodes of multiple types (e.g. measuring temperature, humidity, luminosity, etc.). CoAP clients communicate with sensor nodes by means of URLs with a hierarchical structure that encodes the node location and its capabilities using the following format: `node_type.room.wing.-floor.building`. If q_i is a number of elements on each level, then to be able to access any set of nodes, we need to define a label for each feature at each level (u being the number of levels in the URL): $\sum_{i=1}^{u} q_i$.

We need the same amount of features for LN expressed in the form of attributes. If we use IP multicast in the same scenario, we have to define a multicast group for each combination, which results in $\prod_{i=1}^{u} q_i$. If we want to use the URLs that do not contain all the defined levels (e.g. `bldg1.all_nodes`), the number of multicast groups is even higher: $\prod_{i=1}^{u}(q_i + 1)$.

Random Topology. In the second scenario, we evaluate communication performance in a random topology. Each node chooses its address as a set of 10 random features. After establishing the routing infrastructure, we choose a random node to send a packet to a randomly chosen group. We vary the network size from 50 to 500 nodes and average the results from 100 different runs. A UDP packet with 100B payload is generated every 30s.

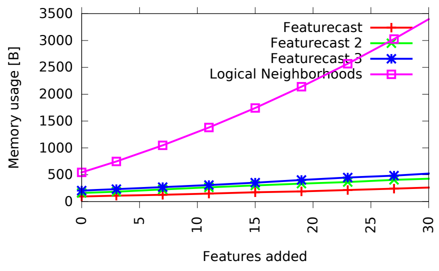

Fig. 3. Memory usage for Featurecast (1, 2, 3 DODAGs) and LN.

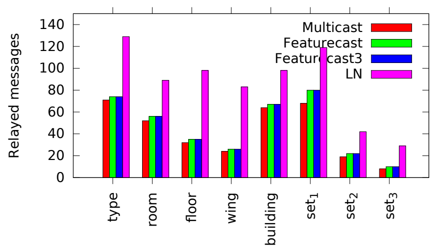

Fig. 4. Number of relayed messages needed by the sink to access all nodes in a given group.

4.3 Results: Memory Footprint in Building Control Scenario

First, we perform our evaluation in the building control scenario with 128 sensor nodes across 2 buildings (Building 1 and 2), 2 floors in each building (Floor 1 and 2), 2 wings (East, West), 4 rooms in each wing (Room 1 to 4), and 2 sensor types (light, temperature). We place 2 temperature and 2 light sensors in each room. We place nodes at regular intervals on a 16x8 matrix and assign the right features simulating the given scenario. Featurecast and LN require 12 features or attributes to realize such scenario, while with IP multicast, we need 405 groups. We place the sink in the center of the network. We also evaluate Featurecast with 2 and 3 DODAGs.

Note that in this scenario, Featurecast is extremely scalable. If we want to connect another building with a similar infrastructure, we need to add only one new feature (e.g. Building 3), while with IP multicast, we need to add 135 new groups. LN maintains associations between attributes, so with every new added attribute, the amount of memory per item increases. Fig. 3 presents the routing table memory usage for Featurecast and LN. We reduce x axis to 30 new features for better readability. We also omit the results for IP Multicast: because of an extremely large number of the required groups and high memory usage per address, IP Multicast needs 6480 B (over 67 times more then Featurecast) with only 12 unique features.

Then, we add features at each level of the hierarchy defined in the scenario [11] (one building, one floor etc.). Featurecast performs more then 5 times better (96 B vs. 544 B) than in our original scenario. Each new item in LN adds some new information to all existing entries, which requires much larger amount of memory per item. With 100 new features added to the network, Featurecast requires more then 26 times less memory (654B vs 17044B). Note that even the topologies with multiple DODAGs (Featurecast 2, Featurecast 3) consume much less memory than LN due to entry aggregation (1064B and 1323B, respectively, for 100 added features).

4.4 Results: Message Overhead in Building Control Scenario

To establish the forwarding topology and guarantee connectivity, Featurecast needed to exchange only 248 messages per DODAG. In comparison, IP Multi-

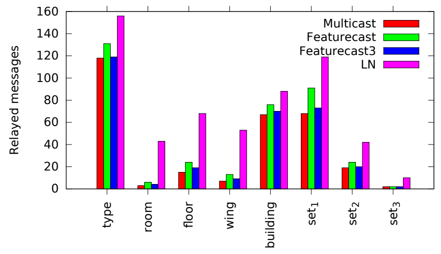

Fig. 5. Number of relayed messages needed by a member node to access all nodes in a given group.

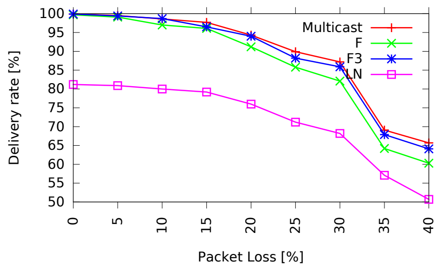

Fig. 6. Delivery rate for different packet loss rates.

cast used 4992 messages to construct a DODAG for each multicast group. LN requires 226 messages, which is slightly better then Featurecast. However, the LN messages are on the average 5 times bigger than the ones of Featurecast, so even for 3 DODAGs, our system requires 2 times less bandwidth.

To evaluate routing performance after constructing the forwarding structure, we consider two cases: i) the sink sends packets to a given group of sensors, ii) a node communicates with another node. Fig. 4 presents the results of the first case: the average number of relayed messages (how many times a message is forwarded by intermediate nodes before reaching the destination). We also present the results for 3 different sets of features: Set1 (type, floor), Set2 (building, wing, floor), and Set3 (building, wing, floor, room, type). IP Multicast creates a minimal spanning tree for every destination group, which gives a bound for this type of traffic. Featurecast only creates one common Featurecast DODAG for all possible groups, but performs only slightly worse. The version with 3 DODAGs achieves almost the same performance as the optimal solution. LN requires however much more messages on the average to reach all destination nodes. It explores routes not present in the routing tables trying to quit local minima, which introduces an additional overhead.

Fig. 5 shows the results of the second case (node-to-node communication). Multicast IP exhibits the best performance that sets a theoretical bound. We can observe that Featurecast also requires a small number of messages. The Featurecast DODAG connects similar nodes thus allowing to find a common nearby ancestor. Introducing additional DODAGs decreases the gap even more. A LN node is never sure if a minimum is local or global, so even after reaching all target nodes, it performs a search of external paths thus increasing the number of messages.

To evaluate the cost of maintaining routing tables, we progressively disconnect random nodes from the network and compare the performance of Featurecast, IP Multicast, and LN. A LN sensor broadcasts a complete node description every 15s. However, if the underlying MAC layer is duty cycled such as ContikiMAC, the node needs to transfer the broadcast message separately to every neighbor. In both Featurecast and IP Multicast, we rely on small hello messages to check the connectivity between neighbors and send the required route update

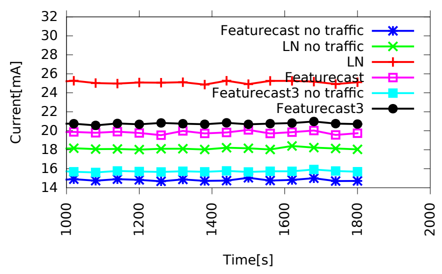

Fig. 7. Energy consumption, with and without traffic.

Discon-nected	FC	FC 3	Mcast	LN
type	3	4	240	4
room	6	7	672	6
wing	18	18	1239	19
floor	12	15	2991	14
building	13	13	2721	13

Fig. 8. Number of relayed control messages after disconnecting different groups of nodes.

only if it is necessary. IP Multicast and Featurecast try to repair the topology only when detecting a neighbor failure. Without any topology changes, LN sends a constant amount of 507 messages every 15s with the average size of 106B. Our implementation of IP Multicast and Featurecast sends on the average 384 hello messages of 4B each. The lower number of messages results from maintaining connectivity only with neighbors in the DODAG. In total, LN transfers 53742B while Featurecast and IP Multicast only 1536B, which is more than 34 times less.

We analyze the behaviour of all solutions for different scenarios. At the beginning, we shut down a single node placed further from the sink, then 2 nodes of the same type in the same room, a group of nodes in one room, all nodes in a wing, all nodes on a floor, and finally all nodes in a building. Table 8 presents the average number of additional messages needed to update the routing tables. When disconnecting single nodes, all approaches do not send any messages, because there is another node belonging to the same group that allows maintaining the DODAG. Disconnecting both nodes of a given type in a room only causes a small number of message exchanges in both Featurecast and LN, as there are other nodes defining the same features in the neighborhood. In IP Multicast, disconnecting the same nodes causes changes in several multicast groups (`bldg1.floor1.west.room4.temperature`, `bldg1.floor1.west.room4.*`, `bldg1.floor1.west.*.temperature`, etc.), and some part of this information needs to be transmitted to the sink causing a lot of traffic. Disconnecting a larger number of nodes causes more multicast group deletions and more control traffic. Shutting down the whole floor or building deletes a lot of multicast groups, but nodes responsible for sending the updates are directly connected to the sink, which lowers the number of exchanged messages.

In all cases, IP Multicast results in a large amount of control traffic due to a much larger number of groups and no group aggregation, which makes it unsuitable for implementation in sensor networks. Featurecast and LN send a much smaller number of messages in all considered scenarios. However, Feature-

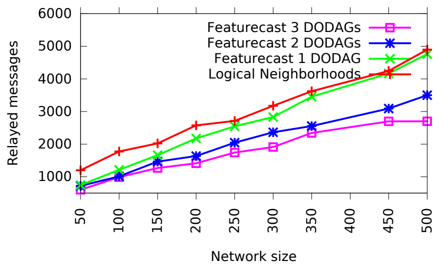

Fig. 9. Number of relayed messages for random communications.)

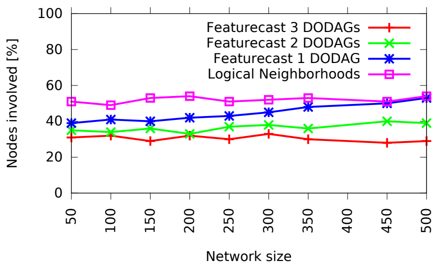

Fig. 10. Number of nodes involved in the communication process.)

cast messages are on the average 5 times smaller due to the compact feature representation.

4.5 Results: Random Topology Scenario

We have evaluated communication performance in the random topology scenario. Fig. 9a presents the number of relayed messages. Featurecast with a common DODAG has almost the same performance as LN. We have also tested a Featurecast version with 2 and 3 DODAGs spread over the network. To send packets, a node uses a DODAG with the closest root. Both versions, with 2 and 3 DODAGs, significantly outperform other approaches.

We have also evaluated the number of nodes involved in communication in the random topology scenario (cf. Fig. 9b). We consider a node involved in communication if it receives or sends a message at the MAC layer. We can observe that Featurecast with only one DODAG performs better than LN for a small number of nodes and involves the same number of nodes in larger networks. However, Featurecast with 2 or 3 DODAGs performs significantly better for all tested network sizes. Note that such a scenario is equivalent to having many sinks in the network. The results show that we do not need one DODAG per sink and several sinks can share one DODAG with only slight drop of performance.

Fig. 7 presents energy consumption measured every 60s using PowerTrace. Featurecast consumes significantly less energy, due to smaller messages and maintaining communication only with neighbors in the DODAGs and not with all nodes in the radio range. Note that Featurecast does not send hello messages separately for each DODAG, but only once for each neighbor present in any deployed DODAG.

To evaluate protocol robustness, we have measured the packet delivery rate for different packet loss rates in a network with 300 nodes. We have performed 1000 random transmissions for each rate. Figure 6 presents the results: with small packet loss rates, the MAC layer can retransmit packets if necessary, so almost all protocols are close to 100% delivery rate. LN even without packet loss cannot find all destination nodes because of the limited number of credits. Featurecast constructs slightly longer paths between destination and performs slightly worse than the optimal solution, however during the tests with 3 DODAGs, the difference is less than 1%. For packet loss rates greater than 15%, the performance of all protocols significantly decreases. Table. 1 summarizes all results.

Aspect	Featurecast	Featurecast-3	LN	Multicast
Memory (12 features)	**1x** (96B)	**2.15x** (206B)	**5.67x** (544B)	**67.5x** (6480B)
Memory (100 features)	**1x** (654B)	**2.02x** (1323B)	**26.06x** (17KB)	**1711156x** (111MB)
sink→nodes	**1x** (345)	**1x** (345)	**1.99x** (687)	**0.96x** (331)
node→node	**1x** (367)	**0.86x** (316)	**1.58x** (579)	**0.82** (299)
hello (msgs)	**1x** (384)	**1.1x** (422)	**1.32x** (507)	**1x** (384)
hello (B)	**1x** (1536B)	**1.1x** (1688B)	**34.99x** (53742B)	**1x** (1536B)
after disconnection (msgs)	**1x** (52)	**1.56x** 81	**1.08x** 56	**151.21x** (7863)
after disconnection (B)	**1x** (624B)	**1.56x** (972B)	**5.41x** (3374B)	**252x** (157KB)
energy, no traffic	**1x** (15.1mA)	**1.05x** (15.9mA)	**1.2x** (18.2mA)	—
energy, with traffic	**1x** (19.9mA)	**1.04x** (20.7mA)	**1.27x** (25.3mA)	—
random (msgs)	**1x** (23557)	**0.67x** (15668)	**1.11x** (26227)	—
random (nodes)	**1x** (401)	**0.59x** 235	**1.17x** 468	—

Table 1. Summary of results: the gain of Featurecast compared to other solutions.

4.6 Discussion of Packet Drops Due to Inexistent Addresses

Finally, we have investigated packet drops due to non-existent conjunction of features. The drops result from the aggregation of features in routing tables and not keeping more information about their compositions. If S_a is a set of features in an address, S_t^i a set of features in the routing table for neighbor i, and S_n a set of features defined by a node, the packet drop occurs when:$S_a \subset S_t^i \wedge \nexists S_n, S_a \subset S_n$.

In our scenario, the packet drop may occur if an address contains a combination of features that are not defined by any node, for example `Building 1 and Building 2`. In this case, the packet can be routed through nodes that may have both features available through the same neighbor. Eventually, it will be dropped by a sensor node that routes packets to this group through different nodes. Creating an invalid address with the location feature usually will not cause a lot of unnecessary traffic, however putting for instance only `temperature and light` into an address will cause global network flooding even if there is no node defining both features.

To alleviate this problem, a packet drop may be signaled by an ICMP packet, so that the user can avoid sending packets with the address in the future. Another problem arises if there are nodes defining for instance both `temperature and light`, but the rest of nodes defines only one of them. In such a case, a new feature `temperature_light` shall be defined allowing to efficiently query both types of nodes. However, the problem heavily depends on applications and will not occur in well configured network (as indicated above).

5 Related Work

Huang et al. proposed a spatio-temporal multicast protocol called *mobicast* that distributes a message to nodes in a delivery zone that evolves over time in some predictable manner [1]. Flury et al. focused on efficient algorithms for routing: they provide close to optimal unicasting and constant approximations to anycast and multicast with small routing tables of a bounded size [2]. Su et al. described oCast, an energy-optimal multicast routing protocol for wireless sensor networks [3]. The authors take into account networks operating under intermittent connectivity resulting from duty-cycling and consider small multicast groups. All three solutions focus more on constructing the minimal spanning tree than proposing an addressing scheme, which gives an opportunity to apply them in our future work.

As mentioned earlier, Directed Diffusion [4,5] is a data dissemination protocol in which sensor nodes attach *attributes* (name-value pairs) to generated data. Nodes interested in some attributes send interests that propagate in the network and in response, sensors send unicast packets to the interested nodes. The approach is similar to ours, but Directed Diffusion uses attributes for the sensed data and not for sensor nodes, focusing more on a publish/subscribe approach. Moreover, Directed Diffusion uses the packet payload and not addresses to convey attributes. The routing approach is also completely different from ours. To contact nodes, Directed Diffusion performs every time global flooding, later establishing a path between the sender and the recipient, which in our case would be very inefficient.

Content-Centric Networking (CCN) focuses on content considered as a communication primitive [12]. It is designed for dissemination of data objects in the Internet and basically operates as a Publish-Subscribe system: each data object has a name and are sent to a subscriber in response to interests sent to publishers. Caching content is one important aspect that contributes to speed up the delivery of objects.

6 Conclusion and Future Work

We have presented Featurecast, a one-to-many communication primitive designed for Wireless Sensor Networks. In our proposal, nodes can create addresses representing some properties of nodes without any coordination or without an external name server. To support routing to Featurecast addresses, nodes disseminate features in the network so that intermediate nodes can merge the features of nodes reachable on a given link for further packet forwarding.

Our comparisons with LN and IP Multicast show very good performance of Featurecast. The application of Featurecast to the building control scenario supports our view that such a primitive can greatly simplify the process of sensor application development—it is possible to introduce already existing applications into a completely new environment without any changes to the code or relying on external systems like DNS or DHCP.

Featurecast defines a simple abstraction based on predicates that enables the reuse of IPv6 addresses and results in efficient memory usage and simple processing at nodes. We believe that such ability can be crucial for integration with existing systems. Even if our implementation is dedicated to wireless sensor networks and the Contiki IPv6 stack, it remains generic and flexible so it may adapt to other use cases.

References

1. Huang, Q., Lu, C., Roman, G.C.: Spatiotemporal Multicast in Sensor Networks. In: Proc. ACM SenSys, New York, NY, USA, ACM (2003) 205–217
2. Flury, R., Wattenhofer, R.: Routing, Anycast, and Multicast for Mesh and Sensor Networks. In: IEEE INFOCOM. (2007)
3. Su, L., Ding, B., Yang, Y., Abdelzaher, T.F., Cao, G., Hou, J.C.: oCast: Optimal Multicast Routing Protocol for Wireless Sensor Networks. In Schulzrinne, H., Ramakrishnan, K.K., Griffin, T.G., Krishnamurthy, S.V., eds.: Proc. of ICNP. (2009) 151–160
4. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. of MOBICOM. (2000) 56–67
5. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., Silva, F.: Directed Diffusion for Wireless Sensor Networking. IEEE/ACM Trans. Netw. **11** (2003) 2–16
6. Hebden, P., Pearce, A.: Data-Centric Routing Using Bloom Filters in Wireless Sensor Networks. In: Proc. of ICISIP-06. (2006) 72–78
7. Mottola, L., Picco, G.: Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In: Proc. IEEE DCOSS. (2006)
8. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proc. ACM SenSys, Berkeley, CA, USA (2009)
9. Winter, T., et al.: RPL: IPv6 Routing Protocol for Low power and Lossy Networks. RFC 6550, IETF (2012)
10. Dunkels, A., Grönvall, B., Voigt, T.: Contiki—a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: IEEE EMNETS, Tampa, Florida, USA (2004)
11. Rahman, A., Dijk, E.: Group Communication for CoAP. IETF draft-ietf-core-groupcomm-07 (2013)
12. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.: Networking Named Content. In: Proceedings of CoNEXT '09, New York, NY, USA, ACM (2009) 1–12