

TinyDiffusion Design:

Detailed Project description:

Packaging TinyDiffusion (Direct Diffusion for sensors) into a software module that can be easily used by researchers who want to quickly put together applications involving

- (1) Various types of sensors and sensor data;
- (2) Customizable in-network aggregation and processing (filtering);
- (3) Dynamic control of the sensor network (if time permits);
- (4) GUI interface for users of the network (if time permits).

Supported Types

- int16 - Default 16bit data type currently in use by most applications.
- blob [Length, Data] - Uninterpreted binary data allows for User Defined data type

Attribute:

Key	Operator	Value
1 byte	1 byte	2 bytes

Both data requests and data responses are composed of data attributes that describe the data. Each piece of the subscription (an Attribute) is described via a key-operator-value triplet.

key indicates the semantics of the attribute (latitude, frequency, etc.). Keys are simply constants (integers) that are either defined in the network routing header or in the application header. Allocation of new key numbers will be done with an external procedure to be determined. Keys in the range 0 - 99 are reserved and should not be used by an application.

Operator describes how the attribute will match when two attributes are compared.

Available operators are: IS, EQ, NE, GT, GE, LT, LE, EQ_ANY.

The IS operator indicates that this attribute specifies a literal (known) value (the LATITUDE KEY IS 30:456). Other operators (GE, LE, NE, etc.) mean that this value must match against an IS attribute. Note however that for blobs the API doesn't know how the information is encoded and will perform a bit wise comparison only (i.e. IS can be used to specify a literal blob value that can only be matched with the EQ and NE operators).

Value has some type and contents. Some values also have a length (if it's not implicit from the type).

Matching Rules

Data is exchanged when there are matching between subscriptions and publications. Since diffusion is based on the core concept of subject-based routing, it is very important to make sure attributes in publications, subscriptions and filters match. A user subscribe an interest and publish data. For both Publish/Subscribe and Filters API, matches are determined by one way match applying the following rules between the attributes associated with the publish (P) and subscribe (S):

For each attribute Sa in S, where the operator Sa.op is something other than IS
Look for a matching attribute Pa in P where Pa.key = Sa.key and Pa.op = IS
If none exists, exit (no match)
else use Sa.op to compare Pa and Sa
If all attributes where matched – then S matches P

For example,

A user might look for TELs by subscribing with the attribute:

TEMP_KEY EQ 32 or
TEMP_KEY LE 40 or
TEMP_KEY GE 25

A sensor would publish this set of attributes:

LATITUDE_KEY IS 30.455
LONGITUDE_KEY IS 104.1
TEMP_KEY IS 32

A user might also look for anything in a particular region with:

TARGET_KEY EQ ANY
LATITUDE_KEY GE 30
LATITUDE_KEY LE 31
LONGITUDE_KEY GE 104
LONGITUDE_KEY LE 104.5

Basic Diffusion Description

Subscribe

Each subscription causes TinyDiffusion to send an Interest message to the network. These interest messages are flooded throughout the network. Every interest is matched against cached interest list. Duplicate interests are dropped. Overlapping interests are aggregated. When interests encounter publishers with matching data, simple (non-reinforced) gradients are set up from the publisher to the subscriber.

Publish

A publisher sends a Data message in reply for an interest or reinforcement. Data messages are sent only through reinforced gradients. The reinforced gradients are determined by the information stored in the Interest Cache.

TinyDiffusion implements one way pull. On a new arriving interest data is published based on the gradient of the first arriving interest.

Periodically, in order to discover new paths or repair broken paths, diffusion marks a data message as exploratory and sends it to all nodes. When one of these messages reaches a Subscriber, it causes a positive reinforcement to be sent to the Publisher. This can make regular gradients become reinforced gradients.

Reinforcement

Reinforcement is on Application Type, Unicast, through neighboring nodes providing distinct data. Specific sources or location box can be reinforced using the attribute triple matching mechanism. In order to establish reinforcement gradient the reinforcement is matched against the cached interest list instead of the data it self. Caching the data it self gives more accurate information but it is not scalable on motes with limited memory.

The matching rules for the reinforcement are relaxed to support a match to an Interest. Basically, reinforcement with different expiration time and interval attributes match a cached interest with the same other attributes. The cached interest holds a link to the information required for the path selection. This link is updated at each match between a data packet and the interest.

Filter API

Filters API is the same as Subscribe/Publish API. The filters are mapped with a unique key.

Keys: 100 – 120 are reserved for filters. The filters have to be precompiled with TinyDiffusion, in a filters.h file before deployment. On data packet arrival it first matched against all subscribed filters. On a match a copy of the data is forwarded to the filter. The data than is matched against other interests.

User Interface:

From the user point of view interest and reinforcement are the same entity. The user subscribes an interest to TinyDiffusion using the following API. The user needs to be familiar only with the structure of an attribute. The subscription requires input of array of attributes of limited size as defined in TinyDiffusion Definition file, the number of attributes, and expiration time of the interest. The subscribing opens a pipe to diffusion from which data arrives with a NULL terminated attributes array. TinyDiffusion allows for aggregation of data, thus multiple attributes of the same kind can arrive at the same attributes array. The application layer is responsible to extract and verify multiple arriving data since as long as at least one match of data to an interest is attained, the data will be forwarded to the sink.

```
Attribute * subscribe(uint8_t AttNum, Attribute  attributes[MAX_ATT],
                      uint16_t expiration);

// Pre: expiration - expiration time of interest (or reinforcement)
//       AttNum - number of attributes contained in the packet
//       attributes - an array of max size containing the attribute list
//Post: Sends (broadcast) interest message up stream.
//       Opens a pipe stream of data- attributes arrays.
//       The last attribute is Null Terminated.
//       One data can contain up to MAX_ATT limit aggregated data.
```

The sources publish data and intermediate nodes forward it down stream from source to sink. The programmer needs to be familiar just with the attribute structure. The user put the sensed data in the attribute format and stores it in a NULL terminated array.

```
void  publish(uint8_t AttNum, Attribute  attributes[MAX_ATT]);

// Pre: expiration - expiration time of interest (or reinforcement)
//       AttNum - number of attributes contained in the packet
//Post: Sends data pocket down stream to sink.
```

Data Structures

General:

Pocket size is limited to 29 bytes due to TinyOS.

Currently it limits the number of attributes to 4 !

Header

The header structure is a common header for both the Interest/Reinforcement packet and a data packet.

The header size is 8 bytes.

Sequence Number 4 bytes	Source 2 bytes	Previous Hop 2 bytes	Packet Type 1 byte
----------------------------	-------------------	-------------------------	-----------------------

InterestMessage

Header		
TTL – time to live		
Expiration Time		
AttNum – Number of attributes		
Key	Operator	Value
...
Key	Operator	Value

DataMessage

Header		
Hops to Source – also serves as time to live		
AttNum – Number of attributes		
Key	Operator	Value
...
Key	Operator	Value

Interest Cache

The interest cache provides the data structure to cache interests, the gradients list to which the interest corresponds, and reinforcement list which hold the relevant data for reinforcement path selection. The cache is implemented as a FIFO structure of predetermined size. Expired entries are cleaned on every cache update.

The Interest Entry also implements a FIFO queue for interest gradients and Reinforcements of limited size.

Interest Cache Entry

Interest Message
Gradient List [Max – Degree]
Reinforcement List [Max – Degree]

Gradient Entry

Expiration	Previous Hop
------------	-----------------

Reinforcement Entry (Currently)

Source	Previous Hop	Hops to Source
--------	-----------------	-------------------

Data Cache

The sole purpose of the data cache is to eliminate data packet duplicates. . The cache is implemented as a FIFO structure of predetermined size.

Data Cache Entry

Sequence Number	Source
--------------------	--------

Tiny Diffusion Definition File

The definition file contains:

1. Pre-defined parameters for fine tuning of TinyDiffusion.
2. Keys and Operations definition for attribute matching.
3. Application Types definitions.
4. Data Types definition.
5. Packet Types definition.

```
// Size Definitions For Fine Tunings.

#define MAX_INTERESTS      10
#define MAX_LOCATIONS     10
#define MAX_GRADIENTS     10
#define MAX_REINFORCEMENTS 10
#define MAX_ATT           10
#define MAX_DATA          20
#define TTL                50    // max number of hops for packet to propagate

// Tiny Diffusion Definitions

#define ORIGINAL 0;           // indicates original data or interest
#define CHACHED  1;         // indicates cached interest
#define FORWARDED 2;        // indicates forwarded data or interest

// Packet Type Definitions

#define INTEREST 1
#define DATA    2

// Operator Definitions          0 - 200

#define IS      1
#define EQ      2
#define NE      3
#define GT      4
#define GE      5
#define LT      6
#define LE      7
#define EQ_ANY  8
```

```

// APPLICATION TYPE          TYPE_ID  0 - 49
//-----
#define  NULL_SENSOR         0
#define  Temperature         1
#define  Humidity            2
#define  Pressure            3
#define  Leaf_Wetness       4
#define  PAR                 5
#define  UV                  6
#define  Solar_Radiation     7
#define  Rain_Level          8
#define  Eye_Level_Light     9
#define  Wind_Speed         10
#define  Wind_Direction     11
#define  Soil_Moisture       12
#define  Temperature_Probe  13
#define  BatteryVoltage     14

// KEYS MAPPING              50 - 200

#define  APP                 50
#define  TEMP                51
#define  HUMIDITY            52
#define  INSTANCE            53
#define  X                   54
#define  Y                   55
#define  Z                   55
#define  INTENSITY           56
#define  CONFIDENCE          57
#define  TIME                58
#define  INTERVAL            59
#define  VELOCITY            60
#define  DIRECTION           61
#define  PRESSURE            62
#define  UV                  63
#define  RADIATION           64

// Filter Definitions        100 - 120

```