# TinyDiffussion Design:

**Project Goal:**

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);

**Supported Data 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<br>1 byte | Operator<br>1 byte | Value<br>2 bytes |
|---|---|---|

Both data requests and data responses are composed of data attributes that describe the data. Each piece of the subscription/publish (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 species 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 using binary comparisons).

**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. Users/Applications subscribes an interest. Sources 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 Sa in S where Sa.key = Pa.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 – than S matches P**

For example,

A user might look for temperature values 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 TinyDiffusion Description

TinyDiffusion implements a one-way-pull.

**Implementation Assumption.**

1. Every node holds a symmetric connectivity list of his neighbors.
2. Links are symmetric.
3. A node maintains a Black List of neighbors of insufficient connectivity.
4. All packets from or to a Black node are dropped.
5. Every node holds an Interest Cache and a Data list.

**Subscribe**
Each subscription causes TinyDiffusion to send an Interest Message to the network. These Interest Messages are flooded (broadcast) throughout the network. On arrival of an Interest Message to a node, it is matched against the Interest Cache. Duplicate interests are dropped. Overlapping interests are aggregated. An Interest Gradient is set in the Interest Cache based on first arriving interest. When an Interest arrives to publishers with matching data, a simple hop-by-hop route is set up from the publisher to the subscriber.

**Publish**
A publisher sends a Data Message in reply to an Interest or Reinforcement. From a publisher point of view there is no different between an Interest and Reinforcement. Periodically, a publisher compares its data list to its Interest Cache. Matching data is aggregated and send in a Data Message. Data Messages are sent only through Interest Gradients of unique neighbors. On arrival of Data Message to a node, it is first matched against a Data list. Duplicate Data Messages are dropped. Later, it is matched against the Interest Cache. Matching Data Message is forwarded down stream through Interest Gradients of unique neighbors. On a match the Data Gradient list is updated.

**Reinforcement**
Reinforcement is performed 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 Interest Cache. The matching rules for the reinforcement are relaxed to support a match between Reinforcement and an Interest. Basically, Reinforcement with different EPIRATION time and INTERVAL attributes matches a cached Interest with the same other attributes. The Reinforcement can be further constraint than the Interest but not less constraint. The Interest entry in the cache holds a matching Data Gradient list. The Data Gradient list is updated at each match between a Data Message and the Interest.

**Filter API**
Filters API is the same as Subscribe/Publish API. The filters are mapped with a unique key. Keys: 200 - 250 are reserved for filters. The filters have to be precompiled with TinyDiffusion, in a filters.h file before deployment. On Data Message 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. The filter can decide if to drop a Data Message or forward it down stream modified or unmodified.

## User Interface:

From the user point of view an Interest and a 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 TinyDiffution Definition file, the number of attributes in the array, and expiration time of the interest. The subscribing opens a pipe to diffusion from which data arrives with a LAST key 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 programmer of the source put the sensed data in the attribute format and stores it in a LAST key 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 – Max size need to be determined.
Currently it limits the number of attributes to 4!

**InterestMessage**

| | | |
|---|---|---|
| Sequence Number - 4 bytes | | |
| Sink - 2 bytes | | |
| Round - 1 byte | | |
| Previous Hop - 2 bytes | | |
| TTL – time to live – 1 byte | | |
| Expiration Time - 2 bytes | | |
| AttNum – Number of attributes - 1 byte | | |
| Key | Operator | Value |
| … | … | … |
| Key | Operator | Value |

**DataMessage**

| | | |
|---|---|---|
| Sequence Number - 4 bytes | | |
| Source - 2 bytes | | |
| Previous Hop - 2 bytes | | |
| TTL – time to live – 1 byte | | |
| AttNum – Number of attributes – 1 byte | | |
| Key | Operator | Value |
| … | … | … |
| Key | Operator | Value |

**Interest Cache**

The Interest Cache provides the data structure to cache Interests, the Interest Gradients list to which the interest corresponds, and the Data Gradients list to which Reinforcements correspond. 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 structure of limited size for the Interest and Data Gradient lists.

**Interest Cache Entry**

| Interest Message |
|---|
| Interest Gradient List [Max – Degree] |
| Data Gradients List [Max – Degree] |

**Interest Gradient Entry**

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

**Data Gradient Entry**

| Source | Previous Hop |
|---|---|

## Data List

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

### Data Cache Entry

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

## Tiny Diffusion Definition File

- The definition file contains:
  - Pre-defined parameters for fine-tuning of TinyDiffusion.
  - Keys and Operations definition for attribute matching.
  - Application Types definitions.
  - Data Types definition.
  - Packet Types definition.

```
//  Size Definitions For Fine Tunings.

#define MAX_INTERESTS      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 Diffution 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         200 - 250
```