# TinyDiffusion Application Programmer's Interface (API) 0.1

Deepak Ganesan (deepak@cs.ucla.edu)

May 12, 2001

## 1 Introduction

This document describes the programming interface to the TinyDiffusion implementation in TinyOS[2]. TinyDiffusion is based on the concept of data-centric or subject-based routing as is the SCADDS data diffusion implementation[3]. Stringent resource constraints on the motes restrict the ability to provide all capabilities currently supported by the implementation on the PC104 testbed.

The TinyDiffusion implementation abstracts the underlying communication primitives, providing an interface to access sensor data by naming attributes.

## 2 TinyDiffusion API Overview

The current TinyDiffusion API supports a preliminary cut at the data naming mechanism. The NR API[1] describes an extensive framework to support attribute naming and to describe matching rules. The current TinyDiffusion implementation provides a very rudimentary attribute naming framework, essentially reducing attributes to a single byte, and focusses on getting the publish/subscribe routing infrastructure in place. Building a more elaborate attribute naming hierarchy, and defining a language to describe rules for TinyDiffusion is future work. While we expect to draw heavily from the NR Routing API, the TinyDiffusion version will provide much less functionality and will be more resource-optimized.

The following is an overview of the approach taken and a brief description of the available interface. In the following sections, the terms SRC and publisher are used interchangably, as are SINK and subscriber.

### 2.1 TinyDiffusion Architecture

#### 2.1.1 Architecture Description

Figure 1 shows a schematic of interconnects between Filters, TinyDiffusion, AM, Timers, and the Photo components. The schematic does not show all hooks between the components, and highlights only salient interconnects. The filters are connected at different ports to the TIMERS component, which exports alarms for periodic events. Note that SRC is connected to both port 0, and port 1 of TIMERS, since it wants to sample the PHOTO sensor at a periodicity different from the periodicity of sending data to a subscriber. Similarly, the filters connect at different ports to TinyDiffusion to send and receive packets. They specify which attribute types they are intersted in and when the attribute type field in the packet matches the specified type, the packet is sent to the corresponding filter. TinyDiffusion registers four kinds of packet types with Active Messaging Layer, INTEREST, REINFORCEMENT, EXPLORATORY and DATA messages.

Figure 1 shows 3 types of Filters:

- Publisher of data (SRC)
- Subscriber of Data (SINK)

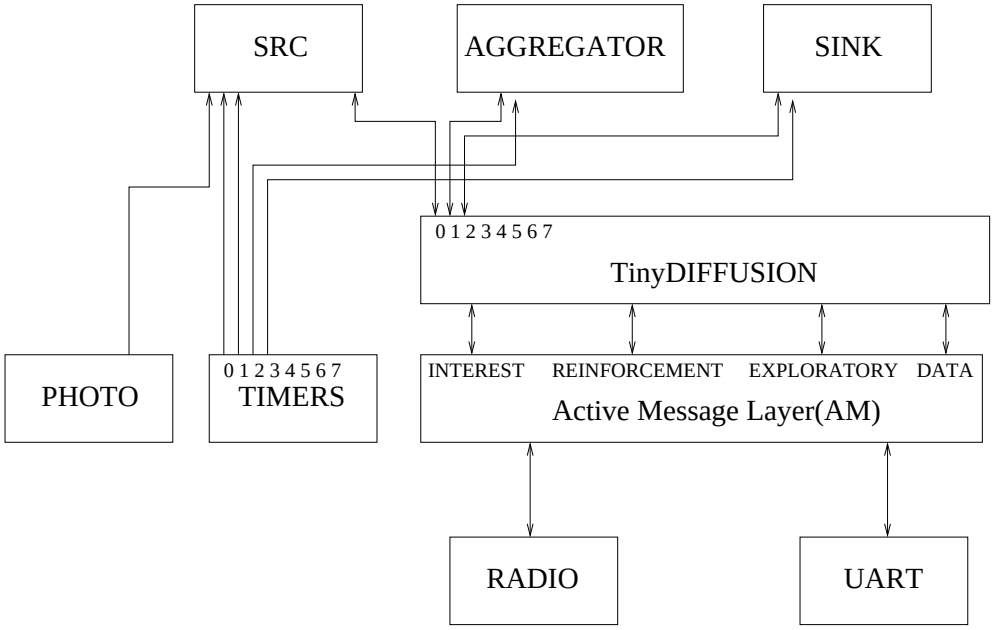


Figure 1: TinyDiffusion Architecture

- Opportunistic Aggregator (AGGREGATOR)

These components are described in greater detail in the rest of this document.

## 2.2 Specifications

### 2.2.1 Code Size:

- TinyDiffusion (stand-alone):
  - Program Memory: 2790b
  - Data Memory: 240b
  - Primary Data Structures:
    - Cache: 10 * 3b = 30b
    - Gradient: 8 * 17b = 136b
    - Neighbor List: 15b
  - Max Number of concurrent flows: 8
  - Max Number of Sources per flow: 5
  - Max Number of Sinks per flow: 1
  - Attribute Values: 0 - 255
- Publisher
  - Total Size (TinyOS + PHOTO + TIMERS + TinyDiffusion + Publisher): 6976b PROGRAM, 371b DATA
  - Program Memory: 428b
  - Data Memory: 8b

- Subscriber
  - Total Size (TinyOS + TinyDiffusion + Subscriber): 6690b PROGRAM, 355 DATA
  - Program Memory: 350b
  - Data Memory: 16b

The above figures indicate that there is only a small margin to add more components, since we are close to 80% capacity of PROGRAM memory and around 65% capacity of DATA memory. Reducing the DATA memory usage can be done by changing the number of concurrent gradients supported from 8 to a lower value. Reducing the PROGRAM memory usage would involve some optimization, such as collapsing the AM and TinyDiffusion layers.

# 3 Diffusion Interface

## 3.1 Introduction

The TinyDiffusion interface to Filters tries to provide a clean division between functionality implemented by the Filter and by TinyDiffusion. The following are seen as potential ways in which Filters would need to interface with TinyDiffusion.

- Filters should be able to register with TinyDiffusion, to receive named data matching certain attributes. In the current version, which defines attr_type by a single byte, filters should be able to register for the attr_type.
- A subscriber filter would need to ability to
  - request subscription to a certain kind of data
  - be called back when data matching its subscription is received at the node.
  - notify TinyDiffusion when it has received good data, so that diffusion can setup data-dissemination paths to the sender of the data.
- A publisher filter would need the ability to
  - tell diffusion of its ability to publish data matching certain attributes.
  - be called back when a subscription interest matching the attributes is received.
  - setup data dissemination paths.
- An aggregation filter should be able to
  - register for data matching certain attributes.
  - suppress the forwarding of data if it is not useful
  - aggregate data from many sources, and forward the aggregate result. This results in a O(n) savings if n publishers send data to an aggregator, and only one data gets forwarded.
  - Opportunistic Aggregation: Delay data in anticipation of the arrival of potentially aggregatable data.

## 3.2 Publish/Subscribe Interface

### 3.2.1 Registering Attribute Types

A filter can publish or subscribe to an attribute type by registering with TinyDiffusion using the DIFFUSION_REGISTER function.
**char DIFFUSION_REGISTER(unsigned char port, unsigned char attr_type)**
*port:* specifies the port on which the Filter is connected to diffusion.
*attr_type:* attribute type between 0-7
For example, if component PHOTO_SRC registers with Diffusion to receive data fpr port=0, attr_type=0, then the following line in the description file would enable the function PHOTO_SRC_RECEIVE_MSG to be called when attr_type=0 is received.
PHOTO_SRC:PHOTO_SRC_RECEIVE_MSG DIFFUSION:DIFFUSION_AGENTS_PORT_0
**TOS_MsgPtr DIFFUSION_AGENTS_PORT_0(TOS_MsgPtr msg, char pkt_type);**

## 3.3 Buffer Management

With the intent of performing more efficient resource management, TinyDiffusion provides an interface to request and release message buffers.
The current implementation allocates a single TOS_Msg in the Diffusion component, and provides the following interface:
**TOS_MsgPtr DIFFUSION_GET_MSG_BUFFER()**
**char DIFFUSION_RELEASE_MSG_BUFFER(TOS_MsgPtr)**

This however, does not preclude each filter from allocating its own message buffer. In fact, this could be preferred in certain cases such as aggregation filters where a packet is not immediately sent out but is delayed in anticipation of aggregation.

## 3.4 Timers

The TIMERS component provides software timers using the single clock so that each included component can use its own timer at a desired resolution. A component can request a timer that is an integral multiple(between 1-256) of the clock rate. So, if the clock is set to run at 1 tick per second, a timer can be set for between 1 and 256 seconds in steps of 1 sec.
The TIMERS interface is similar to the one provided to register with DIFFUSION and currently supports 8 ports.
To start a periodic alarm at a specified port with periodicity intervalclock_rate:
**char TIMERS_REGISTER(char port, char interval)**
For example, the source component uses two TIMERS, one to signal data collection from the Photo component, and another to signal periodic data being sent out (with some processed photo data).
SRC:SRC_TIMERS_PHOTO_EVENT TIMERS:TIMERS_FIRE_EVENT_PORT_0
SRC:SRC_TIMERS_DATA_EVENT TIMERS:TIMERS_FIRE_EVENT_PORT_1
The interval at which an alarm is generated can be dynamically changed by calling the TIMER_REGISTER function with a different interval, but the same port. To stop a periodic timer, use
**TIMERS_DEREGISTER(char port)**

## 3.5 Neighbor List

TinyDiffusion maintains a NeighborList structure that maintains addresses of all neighbors heard from. An interface to this list is currently not provided although it would be a useful interface for smart filters that

may wish to turn themselves on/off depending on the neighbors heard from.

### 3.6 TinyDiffusion Suite

We intend to provide a tinydiffusion suite of components, with different capabilities so that an application designer can pick up the right component for his/her purpose. The current implementation does not provide any hooks to modify the gradient tables or the routing mechanism. However, one can think of cases when such interfaces would be useful. For example, there might be need for greater robustness for an application, which could pick up a TinyDiffusion component that supported the setup of multiple paths. A similar approach would be taken to adding support for load-balancing, geo-routing, etc to TinyDiffusion. Defining the right set of interfaces to the support different routing mechanisms is deferred to future work.

## 4 TinyDiffusion WalkThrough

### 4.1 Data Publisher Filter

Designing a Data Publisher filter is fairly straightforward. The following pseudocode describes the main logic for the filter.

```
At startup {
        // Register with Diffusion for port 0, attr_type 0
        DIFFUSION_REGISTER(0,0);
        // Start a periodic timer with periodicity 8 clock ticks
        // for sampling PHOTO data
        TIMERS_REGISTER(0,8);
}

When a message is received of type pkt_type {
  switch (pkt_type) {

        // If an interest packet is received, set mode to exploratory
        // and send low frequency data packets
  case INTEREST_TYPE:
    TIMERS_REGISTER(1,10);
    Set mode to EXPLORATORY
    break;

        // If a reinforcement is received for me, set mode to DATA and
        // send out high frequency data packets
  case REINFORCEMENT_TYPE:
    if (Reinforcement is intended for me) {
      TIMERS_REGISTER(1,1);
      Set mode to DATA
    }
    break;
  }
}
```

The Publisher registers a (port,attr_type) with DIFFUSION. When it receives an INTEREST msg from a subscriber, it registers a periodic alarm for to send out low-frequency exploratory packets. When it receives a REINFORCEMENT msg, it sees that the subscriber has responded to the INTEREST msg by setting up a dissemination path, and sets the periodic alarm to a higher rate.

### 4.2 Data Subscriber Filter

The subscriber would execute the following pseudocode:

```
At startup {
        // Register with Diffusion for port 1, attr_type 0
        DIFFUSION_REGISTER(1,0);
        // Start a periodic timer with periodicity 200 clock ticks
        // for sending INTERESTS for photo data publishers
        TIMERS_REGISTER(2,200);
}

When a message is received of type pkt_type {
        // If DATA message is received.
        if (DATA msg) accept;

        // If INTEREST message is recvd
        // send reinforcement
        if (INTEREST msg) {
                GET_DIFFUSION_MSG_BUFFER()
                send REINFORCEMENT;
        }
}
```

### 4.3 Data Aggregator Filter

Aggregation Filters can take different forms from combinations of publish/subscribe functionality to opportunistic aggregators. A couple of examples for creating aggregation filters are described below:

#### 4.3.1 Example 1: Multi-Level Query

A Multi-Level query mechanism can be supported fairly easily using the TinyDiffusion API. Consider a 2-level query in which a query request for attr_type 1 triggers a request for attr_type 2.

The following model can be used to handle a 2-level query

- Subscriber S1 subscribes to attr_type 1
- Publisher P1 publishes attr_type 1. When it receives a subscription from S1, it spawns a subscriber S2 for attr_type 2, and subscribes to attr_type 2.

Note: The implementation in TinyOS would involve a Filter that both acts as P1 and S2. The terms subscriber and publisher do not neccessarily correspond to separate filters.

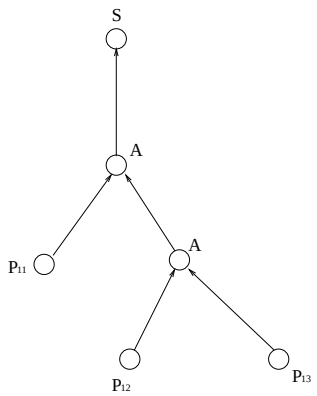


Figure 2: Example: Opportunistic Aggregation

#### 4.3.2 Example 2: Opportunistic Aggregation-Evaluating MAX

Consider the example of a filter that does opportunistic aggregation. In Figure 2, nodes $P_{11}$, $P_{12}$ and $P_{13}$ publish attr_type 1. Subscriber S requests subscribes to attribute type 1. Now assume that attribute type 1 corresponds to a function that evaluates MAX(light-readings) every 10 secs (there is an obvious bootstrapping problem here which will be handled by adding better data naming, and rule matching mechanisms). Obviously, only one of $P_{11}$, $P_{12}$ and $P_{13}$ readings needs to be sent to $S$. However, if no aggregation were performed, all readings would be sent to S. An opportunistic aggregator($A$) would see if it has more than one downstream publishers, and forward only the maximum of the readings received in every 10 sec interval. Thus in the figure, only 5 packets would be transmitted every 10secs instead of 8 packets in the absense of aggregation.

An aggregator $A$ can be deployed at each node that executes the following pseudocode:

```
// If I am on multiple data dissemination paths
If (recvd more than one REINFORCEMENT) {
        TIMERS_REGISTER(1,10sec);
}

For Each Packet Recvd in current epoch {
        If MAX(all seen packets), STORE packet
        else drop packet.
}

When TIMER_FIRES {
        Forward stored packet.
}
```

## 5 Future Work

- Priority Queuing: Currently,when a Filter wants to send a packet, and if it there is already a buffer in the send pipeline, TinyDiffusion returns NULL. The onus is on the filter to attempt again. A better structure would be to implement some form of queuing (FIFO), so that the earliest components are given priority and notified in-order of request arrival..

- Hooks to Diffusion Structures: As described earlier in the document, hooks should be provided to access TinyDiffusion data structures.
- Separate component for resource management
- Separate component for cache
- Cleaner separation of diffusion header/filter header
- Integrate packet sequencing/duplicate suppression into AM layer,
- Multiple Diffusion components, that export part of functionality
- Timer hooks to increase basic clock rate

# 6 References

[1] Network Routing Application Programmer's Interface (API) and WalkThrough 8.0
[2] System Architecture Directions for Networked Sensors, J. Hill, R. Szewcyk, A. Woo, D. Culler, S. Hollar, K. Pister, To appear in ASPLOS 2000
[3] Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks - Mobicom '00