# A distributed compression service based on SenZip and CTP

Sundeep Pattem
Ying Chen

February 8, 2010

## Contents

## 1 Introduction

Sensor networks are aiding the evolution of monitoring systems for earth and space science applications. Frequently, these systems require continuous gathering of correlated data. The correlation in data could be spatial (proximity of sensor nodes) or temporal (slow evolution of physical phenomena) or both. These correlations can be exploited for efficient and long-lived operation via in-network compression.

This is a manual for a sensor networking system for data gathering with distributed, spatio-temporal compression. The design is based on SenZip, an architectural view of compression as a service that interacts with standard networking components [1]. The current implementation provides a SenZip compression service built over the Compression Tree Protocol [CTP, TinyOS Enhancement Proposal (TEP) 123] [2]. Even though the implementation requires only small additions to a few files in CTP, this release includes all CTP files for completeness. The hope is that these changes will soon be incorporated into CTP. The current release is more suited for in-lab deployment and testing. Primarily, limitations of the current reconstruction code need to be addressed for use in field deployment.

The SenZip architecture is designed to work for different compression schemes. Currently, compression is based on a spatial DPCM transform followed by fixed quantization encoding. Future releases of code will include (a) the option of 'temporal only' encoding in case sensor nodes have little or no spatial correlation, (b) more efficient temporal encoding based on Golomb-Rice codes, (c) spatial encoding based on 2D wavelets.

Section. 2 describes how to run a SenZip data gathering system. Section. 3 has a brief overview of how the system operates. Sections 4, 5 and 6 provide details of the code.

## 2 Data gathering with SenZip

This section describes how to setup and run a SenZip-based data gathering sensor network, along with the assumptions and limitations of the current implementation.

### 2.1 Code

1. TinyOS code

- SenZipApp: This is a sample application using SenZip. Copy to $TOSROOT/apps/
- senzip: This contains SenZip and CTP components. Copy to $TOSROOT/tos/lib/net/

2. MATLAB code

   - SenZipReconstruct: This contains the code for reconstruction of the gathered compressed data.

### 2.2 How to setup and run

1. Program motes

   - Compile SenZipApp and program motes with ids 0, 1, 2....
     – mote with id 0 is the sink (or root of routing tree)
     – currently relaying data over the air from node 0 to laptop/pc
     – check with assumptions in Section 2.3
   - Program a base station with id 127 (match frequency with senzip nodes).

2. Setup java controller

   - compile SenZipController.java in SenZipApp directory
     – set the number of nodes in network (eg. static short num_nodes = 4;)
     – currently assuming number of nodes is fixed and known in advance

3. Setup reconstruction

   - Set file path to SenZipApp directory in SenZipProcessing.m
   - choose normalization constant for sensor
   - set range of values for plot axis

4. Run

   - Setup SerialForwarder (eg. java net.tinyos.sf.SerialForwarder -port 9001 -comm serial@COM1:telosb)
   - Send a start message from SenZipController
     – eg. java SenZipController -c START -d 1 (command START, destination node 1)

- Run SenZipProcessing.m (reconstruction in MATLAB)

5. Result

    - In SenZipApp folder, files 'coeff_data_#.txt' and 'raw_data_#.txt' will be populated as the corresponding packets are received for each node.
    - Matlab will plot the raw data signal and the reconstructed version for each node.
    - The default setting is to encode by quantizing 16 bit DPCM coefficients to 4 bits each. The nominal cost is reduced by a factor of 4. Due to the overheads involved, the cost reduction factor is closer to 2.

## 2.3 Assumptions and limitations

The following are the assumptions and known limitations in the current implementation.

- Application written for Tmote Sky motes with on-board temperature sensor [in SenZipAppC.nc - components new SensirionSht11C() as Sensor;]
- In defining packet format (senzip_comp_msg in Compression.h ), sensor measurements are assumed to be 16 bits each.
- Currently, reconstruction code assumes:
  – node ids are in sequence (0, 1, 2... n)
  – size of network is fixed throughout operation
  – packet delivery to sink is 100% and there are no repeats
  – base station (node 0) acts only as relay and not a source of data
- Note that raw data is being collected for testing purposes.

# 3 How the system works

The following is a sequence of events and operations in SenZip-based data gathering with distributed spatio-temporal compression:

- Routing tree: When nodes are switched on, the (CTP) routing algorithm builds a tree rooted at the sink. Each node chooses a parent in the tree for which the ETX metric is minimized.

- Local aggregation tree: Once the parent is known, beacons are exchanged to build local aggregation tree. For the DPCM transform being used currently, for each node this tree consists of all 1-hop children in tree. When nodes choose a parent, they send it a beacon for inclusion in the parent's aggregation tree.

- Start gathering: When a START message is sent to any node in the network, it broadcasts this message and initiates sensor measurements. When other nodes receive this START message, they re-broadcast it and initiate sensing.

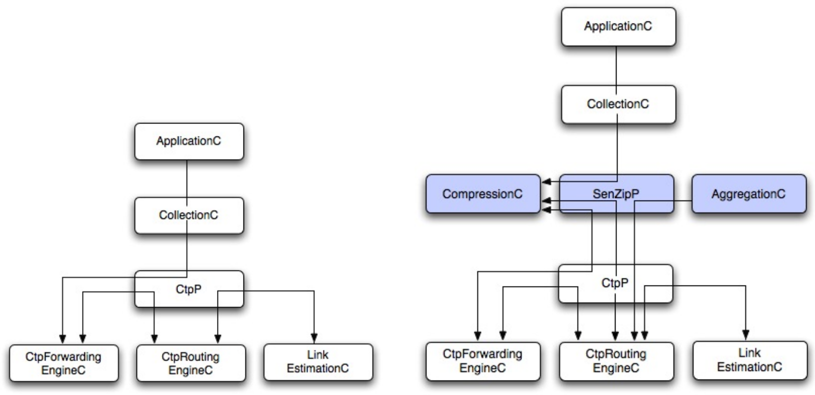
- Compression: Once started, nodes collect sensor measurements in packets and send to the parent in tree. The parent applies the DPCM transform for each child and generates coefficients. This is how spatial de-correlation is achieved. These coefficients are passed through a fixed quantization encoder for compression. When there are enough bits from compressed coefficients to fill a packet, they are packetized and sent to the sink.

- Reconfiguration: When changes in the routing tree and hence in local aggregation trees occur, beacons are exchanged locally to update this information. These changes are handled gracefully in the compression operations.

## 4 TinyOS code

The system provides a distributed compression service built over the Compression Tree Protocol (nesC/TinyOS, TEP 123). Figure. 1 is an illustration of the component wiring for CTP and the proposed extension.

SenZip introduces two new components - aggregation and compression. The aggregation component is for constructing and maintaining the local routing structure around each node over which data will be aggregated and processed. Currently, data is gathered over a dynamic tree topology and compression is based on DPCM transform followed by fixed quantization encoding. DPCM requires the aggregation component to maintain only 1-hop local neighborhood information. The compression component receives



(a) (b)

Figure 1: (a) CTP and (b) SenZip compression service over CTP

data packets from the forwarding engine, performs the compression operations and packetization of compressed data. This is illustrated in Figure. 2.

## 4.1 Components

### 4.1.1 AggregationP

The aggregation component maintains the local aggregation tree. The routing component signals changes in parent in routing tree. At this point, the aggregation component sends an ADD beacon to the new parent and a DELETE beacon to the old parent. The old and new parents update their aggregation tables accordingly.

**(a) Events:**

- Routing.parentChange: Signalled from the Routing engine to indicate a change in the parent in routing tree.
    1. Send ADD beacon to new parent and DELETE message to the old parent.
    2. Signal beacon to Compression component.
- AggBeaconReceive.receive:
    1. Update table for ADD/DELETE beacons from neighbors.

**(b) Commands:**

- Table.contactDescendant: Called by the Compression component to directly contact neighbors in the table from which expected packets have not been received.

This component is currently setup and tested for one hop neighborhood information only. The local aggregation table is defined in a way that can extended for more hops.

#### 4.1.2 CompressionP

When the aggregation component signals changes in the aggregation table, the compression component allocates/de-allocates memory for storing the data of children in the aggregation tree. When the forwarding engine presents packets with data arriving from children, the data is stored, transformed, compressed and packetized to be handed back to the forwarding engine to transport it to the sink. Figure 2(a) shows the sequence of operations for compression at each node. Currently the DPCM transform is applied and fixed quantization encoding is used for compression. Given the overheads, per packet payload available for compressed data is 10 bytes or five 16-bit measurements.

**(a) Events:**

- Table.tablePointer: Signalled from the Aggregation component to provide a pointer to the table for the local aggregation neighborhood.
- Table.change: Signalled from the Aggregation component to inform of changes in the local aggregation neighborhood.
- Intercept.forward: Signalled from the forwarding engine to filter out packets meant for in-network processing i.e. compression.
- AllRxTimer.fired: Internal timer setup to check if all expected packets from the local aggregation neighborhood were received. If not, currently use the readings from the previous epoch.

**(b) Commands:**

- StartGathering.isStarted(): Called from application to check if compression has already started.
- StartGathering.getStarted(): Called from application to get compression started.

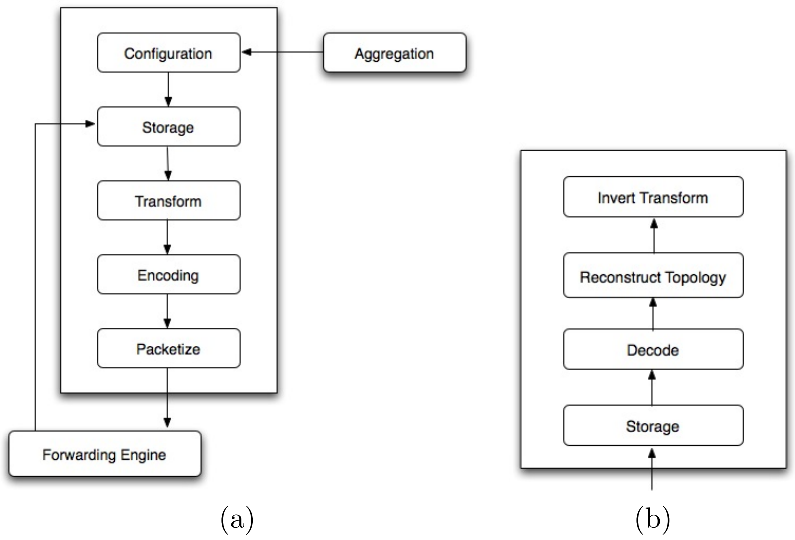


Figure 2: (a) Distributed compression and (b) Centralized reconstruction

- Measurements.set : Called from the application to transfer sensor measurements.

**(c) Tasks:**

- changesTask: Posted to update internal table according to changes signalled by Aggregation component.
- encodeCoefficientsTask: To encode and compress the coefficients generated by the transform. Currently using fixed quantization encoding.

**(d)Functions:**

- computeTransform: To apply the transform on data received from the local aggregation neighborhood. Currently DPCM or differential computation.

### 4.1.3 FixedRoutingP

For controlled experiments in the lab, the Aggregation component can be wired to a FixedRouting component which receives the topology from a central base station. However, the same information can be obtained from the CtpRoutingEngine in practical deployments. This only needs a bit of rewiring in SenZipP.nc

## 4.2 Interfaces

The following new interfaces have been defined for the interactions of the new components with other parts of the system.

### 4.2.1 AggregationInformation

For the interaction between routing and aggregation components.

### 4.2.2 AggregationTable

For the interaction between aggregation and compression components.

### 4.2.3 StartGathering

For the interaction between compression component and application.

## 4.3 Wrappers

A new wrapper SenZipP is introduced and the CollectionC wrapper defined by CTP is modified.

### 4.3.1 SenZipP

For wiring components via the interfaces as described above, roughly illustrated in Figure 1(b).

### 4.3.2 CollectionC

1. Additionally provide "interface Set<uint16_t> as Measurements;"
2. Additionally provide "interface StartGathering"
3. All interfaces equated to those in SenZipP

## 4.4 Changes to CTP

Some (small) changes are introduced in CTP components to account for and aid in-network compression.

### 4.4.1 Routing

Include AggregationInformation interface and inform Aggregation component of changes in parent.

### 4.4.2 Forwarding

Obtain the next hop for forwarding packets from Compression component rather than Routing.

## 4.5 Application

The current application is written for a TMote Sky mote with an on-board temperature sensor.

**(a) Events:**

- StartGathering.startDone(): Signal from Compression component to begin sensor measurements.
- SubReceive.receive(): At the sink node, the Compression component transfers all packets to application. They are then sent over the air to the base station attached to a pc/laptop.

## 4.6 Parameter settings

These parameters allow control over system performance.

- SAMPLE_PERIOD: The sampling period can be set in the application (SenZipApp.h).
- BIT_ALLOCATION: The system performs lossy compression using fixed quantization encoding. The bit-reduction obtained (and quality of reconstruction) depend on the number of bits allocated for quantization of each sample. This parameter can be set in the compression component (Compression.h) for encoding and needs to be matched in SenZipProcessing.m for decoding in MATLAB. Currently assuming bit allocation is same for all nodes.
- BIT_WIDTH: The current setting (Compression.h) is that sensor readings are 16 bits each.
- MAX_NUM_CHILDREN: This parameter (in Senzip.h) denotes the maximum number of children per node in the tree. It defines the size of the aggregation table. The default is set to 10.

## 5 Java code

- SenZipController.java: This code enables the pc/laptop to
  - send the START message to the network from the base station
    – eg. java StartController -c START -d 1
  - receive and store the packets sent by the sink.
    – raw and compressed data packets stored in files named coeff_data_#.txt' and 'raw_data_#.txt'

## 6 MATLAB code

- SenZipProcessing.m: This is a script which reads the data stored into files in the application folder and performs all the processing for reconstruction.
- FQDecode.m: This is a function called by the script to decode the encoded coefficients.
- TreeReconstruct.m: This function is called by the script to reconstruct the routing tree over which the transform was applied for each set of measurements.

## References

[1] S. Pattem, G. Shen, Y. Chen, B. Krishnamachari, and A. Ortega. Senzip: An architecture for distributed en-route compression in wireless sensor networks. In *Workshop on Sensor Networks for Earth and Space Science Applications (ESSA), International Symposium on Information Processing in Sensor Networks (IPSN)*. IEEE/ACM, April 2009.

[2] TinyOS 2.0 Network Protocol Working Group. Collection tree protocol, tinyos enhancement proposal (tep) 123. http://www.tinyos.net/tinyos-2.x/doc.