1. Introduction
This document is a specification for programmers who intend to make TinyOS-based applications using GPSR that has been described on Karp’s paper [3]. It describes the idea of GPSR, functionalities that are implemented, and application program interfaces. And the software architecture and the control flow between components are also described. Lastly, formats of protocol packets and limitations of this implementation are described.

References

2. Description of GPSR [3]
GPSR is a geographic routing for wireless sensor networks. Unlike traditional Internet routing (Link State, Distance Vector), each node on a GPSR network keeps only state from immediate neighbors and uses those states for packet forwarding. The state is geographic position that sensor nodes can self-configure through GPS devices or other mechanisms. A source node transmits a packet to a destination by including the destination's location. Firstly, current forwarding node tries to delivery packets through greedy-mode routing. The greedy routing intends to forward packets to a node whose distance to a destination is less than distance from current forwarding node to destination and shortest among all immediate neighbors. If there is no neighbor node whose distance to destination is greater than distance from current forwarding node to destination (in other words, local minimum), routing method is changed into perimeter mode. The perimeter routing is based on planarized graphs such as Relative Neighborhood Graph (RNG) or Gabriel Graph (GG).

When each node receives position information from immediate nodes, it initially computes unit graph and then infers RNG or GG topology. If the node receives packets and greedy routing is not available, it selects a node among immediate nodes by right-handed rule and sends packets to the neighbor. However, the edge between sending node and receiving node should not cross the edge between perimeter starting node and destination. During packet forwarding in perimeter mode, if forwarding node finds a greedy path, it returns to greedy routing.

3. Functionalities of GPSR implementation
3.1 LLC (Logical Link Control)
- It fragments a long GPSR packet into multiple TOS messages and reassembly multiple TOS messages to a long GPSR packet.
- It can compress sending packets and decompress receiving packets if they are compressed.
- Reliable packet delivery can be performed through sequencing and hop-by-hop acknowledgment.
- It provides a link-probing function that performs probing-test on a specific link and eliminates highly
noisy or asymmetric links according to result of probing-test.

3.2 GPSR
3.2.1 Neighbor List
This keeps state such as positioning information of immediate neighbors and performs the followings.
- It adds neighbors to neighbor list and delete neighbors from neighbor list.
- It updates and looks up state of neighbors.
- It finds next-hop for greedy forward.
- It finds clockwise-path next-hop for perimeter forward.
- It finds counterclockwise-path next-hop for some purposes.

3.2.2 Beaconing
- It periodically sends beacon on the radio.
  Solicit_beacon is broadcasted to neighbors to inform them of restarting when a node is restarted.
  Normal_beacon is periodically broadcasted to neighbors whenever beacon time-out event occurs.
- It receives beacons from neighbors and processes them as follows.
  If sender of beacon is new on neighbor list, information about sender is registered to neighbor list, link probing function is triggered to measure link-quality, and then planarization task is triggered to reflect topology change if result of link-probing is well.
  If sender is already registered on neighbor list, some information about sender is update to neighbor list.
  Optionally, events such as addition/deletion/update of neighbor can be sent to applications.
- It periodically checks connectivity to all nodes on neighbor list.
  If connectivity for a neighbor is failed, link probing is triggered. If result of link probing is bad, information for the neighbor is deleted from neighbor list and then planarization task is triggered to reflect topology change.

3.2.3 Planarization
- Whenever topology is changed, planarization is triggered.
  Planarization generates Gabriel Graph or Relative Neighborhood Graph from full graph.
  The planarized graph is determined when applications initialize GPSR component. Gabriel Graph
planarization is used as default.

- According to result of planarization, some links on the neighbor list are marked as routable to be used by perimeter forward and other links is marked as non-routable not to be used by perimeter forward. Optionally, before a link is marked as non-routable, Mutual Witness query can be sent to corresponding neighbor.

3.2.4 Mutual Witness Protocol

- It makes mutual witness query and sends it to specific neighbor.
  - Query message includes all nodes that are placed within area (GG/RNG) created by local node and specific neighbor.
- It receives mutual witness query, processes it, and sends reply to source node.
  - If all nodes in the query are not witnessed by receiving node, receiving node sends reply with no. Otherwise, receiver responses reply with yes.
- If source of mutual witness query receives reply with yes, corresponding neighbor on neighbor list is marked as un-routable. If source of mutual witness query receives reply with no, corresponding neighbor on neighbor list is kept as routable.

3.2.5 Packet reception

- It receives 3 sorts of packets from LLC: greedy-type packet, perimeter-type packet, or broadcast packet.
- If receiving one is broadcast packet, the broadcast packet is changed into greedy-type packet, and then the greedy-type packet is transferred to application.
- If packet reaches destination node or application runs as tightly coupled, receiving packet is transferred to application without any action.
- If receiving node is not destination and application runs as loosely coupled, receiving packet is taken over GPSR forward task in order to keep forward to destination.

3.2.6 GPSR Forward (Greedy and Perimeter)

- It firstly tries to find a greedy neighbor to destination
  - If a greedy neighbor is found,
    - If it is performed on intermediate node, distance from greedy neighbor to destination and one from Lf to destination is compared.
      - If first one is shorter than second one, forward mode for packet is change to greedy mode.
  - If a greedy neighbor is not found, forward mode for packet is changed to perimeter mode.
    - It tries to select a clockwise neighbor by right-handed rule and face-change rule.
      - If selected clockwise neighbor is same as Lf in the packet, the packet is dropped (transfer it to application)
    - After a neighbor to which packet is sent is found, packet is transferred to LLC to be sent to a next hop.

3.2.7 Broadcast Forward and UART communication

- A gspsr packet from application is changed into broadcast packet.
- Broadcast packets are transferred to LLC to be sent to all neighbors.
- UART communication is optional function for applications that need communication with Linux Box. GPSRBaseStation that is provided as an example application uses this function.
4. Application Programming Interface (API)

4.1 GPSRForwarder interface [Mandatory]

- **command result_t init(uint8_t running_mode, position_t *local_pos);**
- **command result_t set_position(position_t *new_pos);**
- **command result_t send_msg(gpsr_packet_t *packet);**
- **event result_t send_msg_done(gpsr_packet_t *packet, char success);**
- **event result_t receive_msg(gpsr_packet_t *packet, uint8_t recv_mode);**

/* definitions for running mode */
#define TIGHT_COUPLE 0x01 /* LOOSE_COUPLE as default mode */
#define RNG PG 0x02 /* GG_PG as default mode */
#define LZ77 COMP 0x04 /* NO_COMPRESSION as default mode */
/* definitions for position */
typedef struct {uint16_t x, uint16_t y} position_t;
/* definitions for receive mode */
#define UNICAST_DATA 0
#define BROADCAST_DATA 1
#define DROP_DATA 2
#define UART_DATA 3
/* definitions for success */
// if success is greater than 0, sending message to radio is successful. Otherwise, success represents error code.

4.2 GPSREvent interface [Optional]

- **event result_t add_neighbor(uint16_t node_id, position_t *node_pos);**
- **event result_t del_neighbor(uint16_t node_id, position_t *node_pos);**
- **event result_t chg_neighbor(uint16_t node_id, position_t *pre_pos, position_t *new_pos);**

4.3 How to use API

- Setting an application as **tightly couple** (e.g. Index query processing, In-network processing)
call init(TIGHT_COUPLE | RNG, &my_pos); /* set TIGHT_COUPLE, RNG, No Compression */

- Setting an application as **loosely couple**
call init(LOOSE_COUPLE, &my_pos); /* set LOOSE_COUPLE, GG, No compression */

- When application sends BROADCAST packet or UART packet
gpsr_packet_t tx_packet; // this should be global variable.
tx_packet.hdr.type = BROADCAST_PACKET;
tx_packet.hdr.payload_len = data_len; // length of data that application intends to send
memcpy(tx_packet.payload, (uint8_t *)data, data_len); // data that application intends to send
send_msg(&tx_packet); // data length should be less than BROADCAST_PAYLOAD_SIZE

- When application on source node sends GPSR packet
gpsr_packet_t tx_packet; // this should be global variable.
tx_packet.hdr.type = GREEDY_PACKET;
tx_packet.dst_pos.x = dest_x; // x,y coordinate to which this packet will be sent
tx_packet.dst_pos.y = dest_y;
tx_packet.hdr.payload_len = data_len; // length of data that application intends to send
tx_packet.hdr.priority = 0; // don’t need hop-by-hop reliable transfer
memcpy(tx_packet.payload, (uint8_t *)data, data_len); // data that application intends to send
send_msg(&tx_packet);

- When **tightly coupled** application on intermediate nodes sends GPSR packet
gpsr_packet_t tx_packet; // this should be global variable.
tx_packet.dst_pos.x = dest_x; // x,y coordinate to which this packet will be sent
5. Overall of Implementation

5.1 Characteristics

Because GPSR is implemented on TinyOS programming platform, this has following characteristics.

1) **Component-base**: GPSR software is a component program using TinyOS system components.

   GPSR program consists of three components: GPSRForwarder, GPSRRouter, and LLC component.

2) **Event-driven**: when GPSR data packets from neighbor or from applications are received, corresponding event handlers or command handlers are invoked. Also, event handlers are invoked by time-out events.

3) **Concurrency-intensive**: Because jobs like forwarding a GPSR data packet spend much time for processing, event handler or command handler is inadequate for long-term jobs. Hence, these event/command handlers fork long-term tasks and are promptly exited. According to this action, multiple tasks to forward a GPSR data packet can exist by a sequence of events or commands.

![Figure 2. Simplified components layering for GPSR software](image)

5.2 Hardware platforms on which TinyOS-base GPSR is runnable

<table>
<thead>
<tr>
<th>Hardware Platform</th>
<th>Crossbow Mica</th>
<th>Crossbow Mica2/Mica2Dot</th>
<th>Intel Mote</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>ATmega128L</td>
<td>ATmega128L (4MHz)</td>
<td>ARM core (12MHz)</td>
</tr>
<tr>
<td>Program Memory</td>
<td>128k Flash memory</td>
<td>128k Flash memory</td>
<td>512k Flash memory</td>
</tr>
<tr>
<td>Data Memory</td>
<td>4k SRAM</td>
<td>4k SRAM</td>
<td>64k SRAM</td>
</tr>
<tr>
<td>Radio</td>
<td>RFM TR1000</td>
<td>ChipCon CC1000</td>
<td>Bluetooth</td>
</tr>
<tr>
<td>Data Rate</td>
<td>40 kbps</td>
<td>76.8 kbps</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>Frequency</td>
<td>916.50 MHz</td>
<td>315/433/868/915MHz</td>
<td>2.4GHz</td>
</tr>
</tbody>
</table>

5.3 Memory Usage

<table>
<thead>
<tr>
<th>Component</th>
<th>Volume of data memory</th>
<th>Example (N=10, T=54, GP=50, FQ=5, GC=25, RQ=5, LQ=10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPSRForwarder</td>
<td>19 + 2FQ + GPxFQ</td>
<td>19 + 2x5 + 50x5 = 279B</td>
</tr>
<tr>
<td>GPSRRouter</td>
<td>23 + 10N + 6RQ + GCxRQ</td>
<td>23 + 10x10 + 6x5 + 25x5 = 278B</td>
</tr>
<tr>
<td>LLC</td>
<td>36 + 9N + 3T + 3L</td>
<td>36 + 9x10 + 3x54 + 3x10 = 318B</td>
</tr>
<tr>
<td><strong>Total Usage</strong></td>
<td></td>
<td><strong>865B</strong></td>
</tr>
</tbody>
</table>

N:MAX_NEIGHBORS, T:TOS_MSG_LEN, GP:GPSR_PACKET_SIZE, GC:GPSR_CONTROL_SIZE
FQ:GPSRForwarder_QUEUE_SIZE, RQ:GPSRRouter_QUEUE_SIZE, LQ:LLC_QUEUE_SIZE
6. Software architecture of implementation

![Software structure of implementation diagram]

Figure 3. Software structure of implementation

<table>
<thead>
<tr>
<th>Components</th>
<th>Interfaces</th>
<th>Modules</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPSRForwarder</td>
<td>GPSRForwarder.nc</td>
<td>GPSR_ForwarderM.nc</td>
<td>GPSR_Forwarder.h</td>
</tr>
<tr>
<td>GPSRRouter</td>
<td>GPSREvent.nc NeighborList.nc</td>
<td>GPSR_RouterM.nc</td>
<td>GPSR_Router.h</td>
</tr>
<tr>
<td>LLC</td>
<td>LLCForwarder.nc LLCRouter.nc LLControl.nc</td>
<td>LLCM.nc</td>
<td>LLC.h</td>
</tr>
</tbody>
</table>

6.1 Link Probing

Simple algorithm:
- Periodically send probe frame to a testing link until sending counter reaches predefined number.
- Calculate link quality whenever probe frame is received and then send reply frame to source.
- Calculate link quality whenever reply frame is received.
- After probing on a testing link is done, inform GPSRRouter of calculated metric.

![State transition diagram of neighbor-link with link probing]

Figure 4. State transition diagram of neighbor-link with link probing
Link probing is triggered by GPSRRouter component or LLC component under following situations.

1) [GPSR] when receiving new beacon from a neighbor
2) [GPSR] when connectivity to certain neighbor is failed
3) [LLC] when sequence gap between receiving GPSR data packets is large
4) [LLC] when hop-by-hop acknowledgement in response of GPSR data packet is not received

6.2 Mutual Witness Protocol

Why do we need mutual witness protocol in the GPSR implementation?

To fix problems occurred by planarization when nodes running GPSR are deployed in real environments.

Simple Algorithm:

```
receive_beacon (peer_id, peer_location) :
  check if peer_id is witnessed by existing neighbors.
  if yes, send Query (including all witnesses) to peer_id
  for existing neighbor \( \in \) neighbor_list
    check if existing neighbor is witnessed by peer_id
    if yes, send Query(peer_id) to existing_neighbor
```

Figure 7. When beaconing sequence is 1:B 2:C 3:D, node A sends Queries to neighbors whenever new beacon is received from neighbors.
7. Sequence diagram
This chapter illustrates normal procedures of GPSR implementation.

- **GPSRRouterM**: Beaconing, Planarization, and Mutual Witness Protocol

### NeighborList Interface that is implemented at the GPSRRouterM.

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPSRReceiveBeacon</td>
<td>neighbor_add(neighbor_id, neighbor_pos)</td>
</tr>
<tr>
<td>GPSRUpdateNeighbor(neighbor_id, neighbor_pos)</td>
<td>neighbor_add(neighbor_id, neighbor_pos)</td>
</tr>
<tr>
<td>GPSRNotify(neighbor_id, neighbor_pos)</td>
<td>signal_add_neighbor(neighbor_id)</td>
</tr>
<tr>
<td>GPSRDelNeighbor(neighbor_id)</td>
<td>signal_del_neighbor(neighbor_id)</td>
</tr>
<tr>
<td>GPSRNotify(neighbor_id, neighbor_pos, mw_reply)</td>
<td>signal_mw_reply(neighbor_id, neighbor_pos)</td>
</tr>
<tr>
<td>GPSRNotify(neighbor_id, neighbor_pos, mw_query)</td>
<td>signal_mw_query(neighbor_id, neighbor_pos)</td>
</tr>
<tr>
<td>GPSRSendMwQuery(neighbor_id)</td>
<td>send_mw_query(neighbor_id)</td>
</tr>
<tr>
<td>GPSRReceiveMwReply(mw_reply)</td>
<td>send_mw_reply()</td>
</tr>
<tr>
<td>GPSRReceiveMwQuery(mw_query)</td>
<td>send_mw_reply()</td>
</tr>
<tr>
<td>Timer.fire()</td>
<td>TASK(GPSR_CONNECT_TIMER)</td>
</tr>
<tr>
<td>probe_req(neighbor_id)</td>
<td>probe_resp(neighbor_id, metric)</td>
</tr>
<tr>
<td>metric &lt; threshold</td>
<td>gprs_del_neighbor(neighbor_id)</td>
</tr>
<tr>
<td>gprs_notify(neighbor_id, neighbor_pos_del)</td>
<td>send_mw_reply()</td>
</tr>
<tr>
<td>gprs_notify(neighbor_id, neighbor_pos_add)</td>
<td>send_mw_reply()</td>
</tr>
<tr>
<td>gprs_send_beacon(normal, broadcast)</td>
<td>BeaconTimer.start()</td>
</tr>
</tbody>
</table>

---

**LLCM**

- **LLCRouter Interface**
  - **NeighborList Interface**
  - **GPSREvent Interface**
  - **Application**
**GPSRForwarderM: when receiving data packet from application component and perimeter forward**

- **LLC**
  - call send_packet(packet, broadcast)
  - event send_packet_done(packet, success)

- **LLC Forwarder Interface**
  - llc_send_packet(packet, broadcast)
  - pop_queue()

- **Tx_Queue**
  - pop_queue(packet)

- **GPSR Forwarder Task**
  - gpsr_greedy_forward(packet)

- **GPSRForwarderM**
  - when receiving data packet from application

- **Application**
  - command send_msg(packet)

**Scenario:** **LLCForwarder**

**Time**

1. call send_packet(packet, broadcast)
2. pop_queue()
3. gpsr_greedy_forward(packet)
4. perimeter packet
5. next_hop = call find_closest_node(previous_hop, pos, next_pos)
6. if returned
   - next_hop = call find_greedy_node(packet, next_pos)
7. if val1 < val2
8. signal receive_msg(packet, broadcast)
9. event receive_msg(packet, broadcast)
10. signal send_msg_done(packet, success)
11. event send_packet_done(packet, success)
12. p = call find_neighbor(packet, prev_id)
13. val1 = calculate_distance(dist_pos, next_pos)
14. val2 = calculate_distance(dist_pos, Lp)
15. gpsr_greedy_forward(packet)
16. next_hop = call find_greedy_node(packet, next_pos)
17. if val1 < val2

**Neighbors List Interface**

**[Perimeter Forward Processing]**

- NeighborList Interface

- call send_packet(packet, broadcast)
- event send_packet_done(packet, success)

- gpsr_send_packet(packet, next_hop)

- pop_queue()

- signal send_msg_done(packet, success)

**GPSRForwarderM: when receiving data packet from LLC component and greedy forward**

- **LLC**
  - event receive_packet(packet, broadcast)

- **LLC Forwarder Interface**
  - llc_send_packet(packet, broadcast)

- **GPSR Forwarder Task**
  - gpsr_greedy_forward(packet)

- **GPSRForwarderM**
  - when receiving data packet from LLCM

- **Application**
  - command send_msg(packet)

**Scenario:** **LLCForwarder**

**Time**

1. event receive_packet(packet, broadcast)
2. val = calculate_distance(packet->hdr.peer_id)
3. if tight_couple
4. val = calculate_distance(packet, unicast)
5. push_queue(packet)
6. perimeter packet
7. greedy
8. gpsr_greedy_forward(packet)
9. next_hop = call find_greedy_node(packet, next_pos)
10. if packer
11. if next_hop
12. if val1 < val2
13. Drop packet
14. NeighborList Interface

**Neighbors List Interface**

**[Greedy Forward Processing]**

- NeighborList Interface

- call send_packet(packet, broadcast)
- event send_packet_done(packet, success)

- gpsr_send_packet(packet, next_hop)

- pop_queue()

- signal send_msg_done(packet, success)

**Perimeter Forward Processing**

- NeighborList Interface

- p = call find_neighbor(packet, prev_id)
- next_hop = call find_greedy_node(packet, next_pos)
- val1 = calculate_distance(dist_pos, next_pos)
- val2 = calculate_distance(dist_pos, Lp)
- gpsr_greedy_forward(packet)
- next_hop and

**Perimeter Forward Processing**
### LLCM: when sending Data frame or LinkProbe frame to radio

<table>
<thead>
<tr>
<th>Time</th>
<th>Event/Task/Interface</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>call send(TOS_msg(llc_frame))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>event sendDone(TOS_msg, success)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>call send(TOS_msg(ack))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>call seqno_filter(tos_msg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>call send_done(TOS_msg, packet, success)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>call send(TOS_msg(probe))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>call calculate_link_quality()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>call PassiveLinkProbe_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>call send(TOS_msg(probeack))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>call calculate_link_quality()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>call ActiveLinkProbe_start</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LLCM: when receiving Data frame or LinkProbe frame from radio.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event/Task/Interface</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>receive(tos_msg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>send_done(tos_msg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>receive(dataframe)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>calculate_link_quality()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>send(tos_msg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>decompress_buffer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8. Format of GPSR data packet and LLC frame

Figure 8. Fragmentation and reassembly on LLC when TOSH_DATA_LENGTH is 49 BYTES

8.1 Format of GPSR packet

```
#define GREEDY_PACKET 0x4
#define PERIMETER_PACKET 0x5
#define BROADCAST_PACKET 0x6
#define UART_PACKET 0x7

typedef struct {
    uint32_t type:3;  /* should be set */
    sub_type:2;  /* don't touch it */
    priority:1;  /* should be set */
    reserved:1;  /* don't touch it */
    ttl:11;  /* don't touch it */
    #define GPSR_DEFAULT_TTL 2047
    payload_len:14;  /* should be set */
    uint16_t peer_id;  /* don't touch it */
} __attribute__((packed)) packet_hdr_t;

typedef struct {
    packet_hdr_t hdr;
    position_t dst_pos;
    position_t src_pos;
} __attribute__((packed)) gpsr_packet_hdr_t;

typedef struct {
    position_t Lp;
    position_t Lf;
    edge_t Le;
} __attribute__((packed)) gpsr_packet_tail_t;

#define GPSR_OVERHEAD_SIZE (sizeof(gpsr_packet_hdr_t)+sizeof(gpsr_packet_tail_t)

typedef struct {
    packet_hdr_t hdr;
    position_t dst_pos;
    position_t src_pos;  /* don't touch it */
    uint8_t payload[GPSR_PAYLOAD_SIZE];  /* space for pure data */
    uint8_t reserved[4];  /* don't touch it */
} __attribute__((packed)) gpsr_packet_t;
```
8.2 Format of LLC frame

<table>
<thead>
<tr>
<th>SRC_ID</th>
<th>SEQ</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>4B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SRC_ID[2B] : node identifier that sends this LLC frame.
SEQ_NO[1B] : transmission sequence number.
INFO[1B] = T | C | A | D | M | Frag_no(3b)
T: Type={DATA_FRAME, DATA_FRAME_ACK, PROBE_FRAME, PROBE_FRAME_ACK}
C: Compressed frame
A: Need acknowledgement
D: GPSR data packet, not control packet such as BEACON, MW_QUERY/REPLY
M: Exist fragmented frames to be sent
Frag_no: fragment number within a long GPSR data packet

9. Limitations
- No interaction to localization function. Hence, all applications should determine own local position and then let GPSR know it.
- The limited size of 2-D coordinate plane: up to 65536x65536 coordinate planes.
- In-network processing application needs Carefulness. If application intends to increase the length of perimeter packet that is received from GPSR component and send updated GPSR packets, GPSR component will make errors.

[Instruction]
Before data are written to payload field in tx_packet, gpsr_tail information in received packet should be copied to other place. After data are written, kept gpsr_tail information should be written to after last data byte.