Greedy Perimeter Stateless Routing (GPSR)

USER Specification

Version 0.1
(July 6 2003)

Young Jin Kim (Youngjki@usc.edu)
Ramesh Govidan (Ramesh@usc.edu)

Embedded Network Lab
Department of Computer Science
University of Southern California
1. Introduction
This document is a specification for users who intend to make application-software running on Greedy Perimeter Stateless Routing (GPSR) that is described on Karp’s paper[3]. It provides the brief description of GPSR, the structure of software, application programming interfaces, and the format of protocol packet.

References:

2. Description of GPSR
GPSR is a geographic routing for wireless sensor networks. Unlike traditional Internet routing (DV, LS), each node keep states from immediate neighbors and uses only those states for data forwarding. The state is geographic position that all sensor nodes can self-configure through GPS devices or others. Source node propagates data with the position of destination to wireless sensor network. In normal case, forwarding node runs greedy mode routing. The greedy routing firstly selects a node whose distance to a destination is less than distance from forwarding node to destination and shortest among all immediate neighbors. Then, data are forwarded to it. If there is no neighbor whose distance to destination is greater than distance from forwarding node to destination and shortest among all immediate neighbors. Then, data are forwarded to it. If there is no neighbor whose distance to destination is greater than distance from forwarding node to destination, forwarding node runs perimeter mode routing. 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 makes unit graph and then determine RNG or GG. If the node receives data and it can’t perform greedy routing, it selects a node among immediate nodes by right-handed rule and sends data to the neighbor. However, the edge between sending node and receiving neighbor should not cross the edge between origin node and destination. During data forwarding in perimeter mode, if forwarding node run greedy routing, it returns to perimeter mode into greedy mode.

3. Structure of software

![Software block diagram](image)

Figure 1. Software block diagram
3.1 States of Neighbors
This keeps states based on positioning information from all immediate neighbors and provides the following functions for other modules.
- Add and delete neighbor
- Update and look-up the state of neighbor
- Update RNG topology and GG topology
- Find shortest-path for greedy forward or clockwise-path for perimeter forward

3.2 Beaconing
- Receive beacon packet and beacon_solicit packet
- Periodically broadcast beacon packet to neighbor nodes
- Periodically check connections to neighbor nodes.

3.3 Greedy Forwarding
- Receive greedy-mode data packet
- Send the pure application data except for GPSR protocol header to upper application if the destination of packet is same as the position of local node.
- Send the receiving whole packet to upper application if GPSR daemon runs on loosely-coupled mod.
- Forward the packet to shortest neighbor nodes if GPSR daemon runs on tightly-coupled mod.

3.4 Perimeter Forwarding
- Receive perimeter-mode data packet
- Send the pure application data except for GPSR protocol header to upper application if the destination of packet is same as the position of local node.
- Send the receiving whole packet to upper application if GPSR daemon runs on loosely-coupled mod.
- Forward from on perimeter mode to on greedy mode if distance from local to destination is shorter than one from previous node to destination.
- Drop (send to application) the pure data except for protocol header if receiving packet is the previous sent packet to neighbor.
- Perform “face_change” algorithm.
- Forward the packet to counterclockwise neighbor nodes.

3.5 Upper interface
- Send data to GPSR_API who is a communication agent for applications.
- Trigger greedy-mode or perimeter-mode forward if data from GPSR_API are received.
- Create a thread per a receiving data from GPSR_API.

3.6 Lower interface
- Unicast data packet to neighbor node or broadcast beacon to all neighbors via communication socket.
- Trigger greedy-mode or perimeter-mode reception if data from neighbor nodes are received.
- Create a thread per a receiving data from neighbor nodes.
4. Application Programming Interfaces (API)

4.1 gpsr_open

```c
#include "gpsr_api.h"
gpsr_id_t gpsr_open(u_int16_t my_port, u_char flag, u_char mode);
```

- **my_port**: `DEFAULT_MY_PORT` or any ports except 4000, 5001, and 5002
- **flag**: `IO_ASYNC` or `IO_SYNC`
- **mode**: `LOOSE_MODE` or `TIGHT_MODE`

* gpsr_open is used to establish a channel to GPSR daemon as two ways.
  1. **Tightly-mode**: when GPSR packet is received from neighbors, GPSR daemon unconditionally sends the original GPSR packet to application (e.g., Index query processing, In-network processing)
     ```c
gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_ASYNC, TIGHT_MODE);
```
  2. **Loosely-mode**: when GPSR packet is received from neighbors, only if packet destination is own node or packet is dropped due to some reason, GPSR daemon sends the pure application excluding GPSR header to application.
     ```c
gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_ASYNC, LOOSE_MODE);
```

4.2 gpsr_close

```c
#include "gpsr_api.h"
int32_t      gpsr_close(gpsr_id_t gpsr_id) ;
```

* gpsr_close is used to close a channel between application and GPSR daemon. (e.g.)
  ```c
gpsr_close(gpsr_id); // gpsr_id is the value that is returned by gpsr_open.
```

4.3 gpsr_signal

```c
#include "gpsr_api.h"
void          gpsr_signal(upcall_t *handler);
```

* gpsr_signal should be used if application open an asynchronous channel to GPSR daemon (4.1-examples). (e.g.)
  ```c
  upcall_t  handler;
  handler.gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_ASYNC, LOOSE_MODE);
  handler.func = upcall_recv   // function invoked by GPSR_API to receive message from GPSR daemon
  gpsr_signal(&handler);
  ```

4.4 gpsr_recv

```c
#include "gpsr_api.h"
int32_t      gpsr_recv(gpsr_id_t gpsr_id, u_char *buff, int32_t buff_len);
```

* content of buff: notify_msg or data_msg

```c
typedef struct {
  api_t           type;              // _NOTIFY
  notify_t      notify_type;   // _INIT_INFO, _ADD_NEIGH, _DEL_NEIGH
  length_t     data_len;
  u_char       data[MAX_BUFF_SIZE];      // sematic: list of neighbor's position
} notify_msg_t;
```

* gpsr_recv is used to read messages(notify or GPSR-data) from GPSR daemon as Synchronous IO. (e.g.)
  ```c
gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_SYNC, LOOSE_MODE);
```
....

u_char buff[200];
int32_t msg_length,
api_t api;
msg_length = gpsr_recv(gpsr_id, buff, 200);
memcpy(&api, buff, sizeof(api_t));
if (api == _DATA) {
    data_msg_t *data_msg = buff;
    ...
}
elif (api == _NOTIFY) {
    notify_msg_t *notify_msg = buff;
    ...
}

In order to message from GPSR daemon as Asynchronous I/ O,
(e.g.) upcall_t handler;
    static void upcall_recv(void *arg) {
        ...
        api_t *api = (apt_t *)arg;
        if (api == _DATA) {
            ...
        } else if (api == _NOTIFY) {
            ...
        }
    }
    ...

int main() {
    ...
    handler.gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_ASYNC, LOOSE_MODE);
    handler.func = upcall_recv
    gpsr_signal(&handler);
    ...
}

4.5 gpsr_send

```c
#include "gpsr_api.h"
int32_t gpsr_send(gpsr_id_t gpsr_id, void *data, data_info_t *data_info)

typedef struct {
    api_t type;            // _DATA
    data_info_t data_info;
    union {
        data_packet_t raw_packet;
        u_char app_data[MAX_BUFF_SIZE - DATA_MSG_HDR_SIZE];
    } data;
} data_msg_t;

typedef struct {
    data_t data_type;      // _SEND_PACKET, _RELAY_PACKET,
    length_t data_len;
    position_t dst_pos;
    position_t src_pos;
} data_info_t;
```
gpsr_send is used to send data (pure data or GPSR packet) to GPSR daemon as two ways

1) _SEND_PACKET
   position_t my_pos;
data_msg_t data_msg;
data_info_t data_info;
data_info.data_type = _SEND_PACKET;
gpsr_getloc(gpsr_id, &my_pos);
memcpy(&data_info.src_pos, &my_pos, sizeof(position_t));
memcpy(data_msg.app_data, "I love you.", 11);
....
gpsr_send(gpsr_id, &data_msg, &data_info);

2) _RELAY_PACKET: it should be used on only tightly-mode application.
gpsr_id = gpsr_open(DEFAULT_MY_PORT, IO_ASYNC, TIGHT_MODE);
....
msg_length = gpsr_recv(gpsr_id, buff, 200);
memcpy(&api, buff, sizeof(api_t));
if (api == _DATA) {
data_msg_t  *data_msg = buff;
data_info_t  *data_info = data_msg->data_info;
if (data_info->data_type == _RELAY_PACKET) {
  memcpy(pure_data, data_msg->data.raw_packet.payload, data_len);
  ....
gpsr_send(gpsr_id, data_msg, data_info);
  ....
}
}

4.6 gpsr_getloc

#include "gpsr_api.h"
int32_t      gpsr_getloc(gpsr_id_t gpsr_id, position_t *my_pos);

if target is PC104:           otherwise:
typedef struct {
  u_char   x;
  u_char   y;
} position_t;
typedef struct {
  u_int32_t   x;
  u_int32_t   y;
} position_t;

gpsr_getloc is used to intend to get own location.
5. Format of GPSR Data Packet

Data_packet_t is a format that GPSR daemon uses to communicate with neighbor nodes. It is defined at “gpsr_packet.h” of GPSR daemon directory.

```c
typedef struct {
    packet_hdr_t   hdr;
    position_t    Lp;
    position_t    Lf;
    edge_t         e0;
    position_t    dst_pos;  // position where GPSR packet finally reaches.
    position_t    src_pos;  // position of node where a source application places.
    u_int16_t     app_port;  // to discriminate one among multiple applications
    u_int16_t     reserved;  // don’t use it at application-level.
    u_char   payload[MAX PAYLOAD_SIZE];  // space for application data
} data_packet_t;
```

Figure 2. Format of GPSR data packet

hdr : refer to figure 3.
Lp : location packet entered perimeter mode
Lf : point on the line of xV packet entered current face
E0 : first edge traversed on current face
dest_post: position(x,y) of destination, 2 four bytes

typedef struct {
    u_int16_t    p:1,
        type:3,
        ttl:12;
    u_int16_t    payload_len;
    node_id_t    src_node;
    position_t    src_pos;
} packet_hdr_t;

Figure 3. Format of header in GPSR packet [don’t write-access this at application-level]

p : Flag for piggyback, 1 bit
- 0: no piggyback
- 1: piggyback (in other words, src_node and src_pos in packet header are effective)

type : packet type for GPSR, 3 bits
- 000: beacon solicitation
- 001: beacon
- 002: greedy data
- 003: perimeter data
- others: spare for other messages

payload_len: the length measured in octets, length of pure application data, 2 bytes
src_node: effective if P is set. It is used to perform piggy backing beaconing.
src_pos: effective if P is set. It is used to perform piggy backing beaconing.