Greedy Perimeter Stateless Routing (GPSR)
Detailed-level Design Specification (DDS)

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 detailed-level design specification for Greedy Perimeter Stateless Routing (GPSR) implementation. Our implementation is based on Karp’s paper[3] and run-able on LINUX systems:

1) PCs/WSs running Red-hat Linux 8.0/ 9.0
2) PC-104 running Linux + Real mote (Mica)
3) HP IPAQ running familiar Linux -http:// www.handhelds.org

This document provides the description of GPSR, the functionalities of GPSR implementation, the structure of software, sequence diagrams for normal functionalities and format of all GPSR protocol packets.

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, 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. Functions of GPSR implementation

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
- Find 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. Structure of software
GPSR is implemented as Daemon process (In initial design, we would provide it as a Library module. However, it was known that that way is inadequate on wireless sensor networks). GPSR implementation consists of two modules: 1) GPSR daemon that performs original GPSR functions, and 2) Application Programming Interface(API) library that provides the access to GPSR for various applications.
The characteristics of GPSR implementation are as follows:
1) **Event-driven:** when GPSR protocol packets from neighbor nodes or application messages from applications are received, polling is not used because it can unnecessarily consume power resources during idle periods and also inadequate for real-time attributes that wireless applications can own.
2) **Multi-thread:** a thread per an event(reception of packet, reception of application data, timeout) is created.
3) **Multiplexing:** GPSR daemon can establish multiple channels for multiple applications. It is intended for multiple applications concurrently use GPSR daemon. To discriminate a destination application when a GPSR data packet is received, GPSR daemon uses multiplexing table and app_port within GPSR packet header.
4) **Use of UDP socket**: UDP socket is used as communication with applications.

5) **Support of Ethernet, 802.11b, mote radio**: they can be used for communication with neighbor nodes. Lower interface module abstracts and hides specific underlying network technologies.

---

**Figure 1. Software block diagram (Linux System)**

```
typedef struct {
    node_id_t   node_id;
    position_t   loc;
    status_t       status;
    #define NEIGHBOR_STATUS_RECENT (1 << 0)
    #define NEIGHBOR_STATUS_PG      (1 << 1)           /* RNG or GG */
    void            *next;
}neighbor_info_t;
```

**Figure 2. An element of neighbor table**

GPSR daemon is organized into 14 header files and 12 C files that is based on GNU C.

- **gpsr_type.h**: define types that are commonly used at all files.

```
typedef u_int16_t length_t;
typedef pthread_t thread_id_t;
#ifdef _MOTENIC
typedef u_int16_t node_id_t;
typedef struct {
    u_char   x;
    u_char   y;
}position_t;
#else
typedef u_int32_t node_id_t;
typedef struct {
    u_int32_t   x;
    u_int32_t   y;
}position_t;
#endif

typedef struct {
    position_t   p1;
    position_t   p2;
}edge_t;
```

- **gpsr_packet.h**: define format of GPSR protocol packets (beacon, data, …)

Refer to section 6.
- **gpsr_link_if**: Functions described at section 3.6 are performed.

```c
link_if_t gpsr_linkif_init(void);
static void *gpsr_packet_recv(void *arg);
int32_t gpsr_packet_send(u_char *, length_t, next_hop, unicastorbroadcast);
```

- **gpsr_upper_if**: Functions described at section 3.5 are performed.

```c
upper_if_t gpsr_upper_if_init(void);
static void *gpsr_upper_recv(void *arg);
int32_t gpsr_upper_send(data_t, data_packet_t *, length_t);
int32_t gpsr_upper_notify(notify_t type, u_char *, length_t);
```

- **gpsr_greedy**: Functions described at section 3.4 are performed.

```c
void      gpsr_greedy_recv(data_packet_t *packet);
int32_t  gpsr_greedy_forward(data_packet_t *packet);
```

- **gpsr_perimeter**: Functions described at section 3.3 are performed.

```c
void      gpsr_perimeter_recv(data_packet_t *packet);
int32_t  gpsr_perimeter_forward(data_packet_t *packet);
int32_t  face_change(data_packet_t *packet, position_t *next_loc, node_id_t *next_hop);
```

- **gpsr_beacon**: Functions described at section 3.2 are performed.

```c
void  gpsr_beacon_send(void)
void  gpsr_beacon_recv(beacon_packet_t *packet);
void  gpsr_beacon_solicit_recv(beacon_solicit_t *packet);
static void gpsr_neighborcheck_timeout(void)
```

- **gpsr_neighbor_table**: Functions described at section 3.1 are performed.

```c
int16_t      update_GG (void);
int16_t      update_RNG (void);
void          gpsr_integrate_neighbor_table (void);
int16_t      gpsr_add_neighbor (node_id_t node_id, position_t *loc);
int16_t      gpsr_del_neighbor (node_id_t node_id);
int16_t      gpsr_update_neighbor (node_id_t node_id, position_t *loc);
void          gpsr_get_neighbor_table(u_char *buff, length_t *len);
void          gpsr_locations_notify (void);
node_id_t gpsr_find_shortest_path (position_t *dst_loc, position_t *next_loc);
node_id_t gpsr_find_clockwise_path (position_t *pre_loc, position_t *next_loc);
```

- **gpsr_maintenance[Optional]**: provide interface between GPSR_OAM

```c
typedef struct {
    u_char type;              /* PROBE, SET LOCATION, SET BEACON TIME,
                              SET FLAG, UNSET_FLAG, GET NEIGHBOR INFO
    u_char    data[MAX_OAM_MSG_SIZE];
} oam_packet_t;

oam_if_t    gpsr_oamif_init(void);
static void *gpsr_oam_recv(void *arg);
int32_t       gpsr_oam_send(u_char *, length_t, struct sockaddr_in *);
```

- **gpsr_mux**: provide functions to support multiplexing

```c
void         add_pcb(u_int16_t app_port, u_int16_t mode);
void         del_pcb(u_int16_t app_port);
u_int16_t find_pcb(u_int16_t app_port, u_int16_t *mode);
u_int16_t find_pcb_num(void);
u_int16_t find_pcb_list(u_int16_t *app_port);
```
• **gpsr_debug**: Print-out for debugging can be controlled at run-time.

```c
void time_print(FILE *fp);
void gpsr_log(const char *format, ...); PRINTF_ATTRIBUTE(1,2);
void gpsr_packet_dump(u_char *msg, u_int16_t len);
void gpsr_hdr_dump(packet_hdr_t *hdr);
```

• **gpsr_timer**: provide mechanisms for timeout(connection, inter-beacon) processing

```c
typedef struct {
    gpsr_timer_t timer;
    int32_t  duration;
    void (*func)(void);
} time_event_t;

int32_t  get_timer_duration(gpsr_timer_t timer);
void      timer_on(gpsr_timer_t timer, void (*func)(void), int32_t duration);
void      timer_off(gpsr_timer_t timer);
```

• **gpsr_utility**: provide Thread control, algorithms for clockwise-path search, and etc.

```c
void my_node_id(node_id_t *node_id);
void my_location(position_t *loc);
void thread_create(void *(*func)(void *), void *arg);
void suspend(pthread_mutex_t *, pthread_cond_t *);
void resume(pthread_mutex_t *, pthread_cond_t *);
uint32_t calculate_distance(position_t *p1, position_t *p2);
double _calculate_distance(double x1, double y1, double x2, double y2);
void slope(position_t *p1, position_t *p2, slope_t *s);
int32_t cmp_slope(slope_t *s1, slope_t *s2);
int32_t find_cross_point(position_t *p1, position_t *p2, position_t *p3, position_t *p4, position_t *c);
```

• **gpsr_d**: Main function for GPSR daemon

```c
typedef struct {
    node_id_t node_id;
    position_t location;
    u_int16_t daemon_port;
    u_int16_t oam_port;
    int32_t inter_beacon_time;
    #define DEFAULT_INTER_BEACON_TIME 40
    u_int32_t flag;
    #define FLAG_DEBUG (1 << 0)
    #define FLAG_RNG   (1 << 1)
} node_config_t;
```
5. Sequence Diagram

This chapter illustrates normal procedures of GPSR daemon by using 4 sequence diagrams.

Revision Date: Ver.0.1.0 Revised by: Scenario: GPSR daemon : Receiving open/ close from Application

Revision Date: Ver.0.1.0 Revised by: Scenario: GPSR daemon : Beacon processing & management of neighbor table
6. Format of GPSR Protocol Packets

6.1 Common Protocol Header

define 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 protocol packet

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.

6.2 Beacon Packet and Beacon-Solicit Packet

define struct {
    packet_hdr_t hdr;
} beacon_t;

define struct {
    packet_hdr_t hdr;
} beacon_solicit_t;

Figure 4. Format of Beacon and Beacon-Solicit

6.3 GPSR Data Packet

define 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 5. 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_pos: position(x,y) of destination, 2 four bytes