Rapport du Projet de réseau :
Dazibao

Ansari Edwin – Yaniv Benichou

L3 Double Licence Mathématiques-informatique

Partie implémenté du projet :

- Tout le projet avec fragmentation des paquets, sans aucune extension.

Type et structure utilise dans le projet :

```c
typedef uint64_t node_id;
typedef uint16_t seq_n;


typedef struct node {
    node_id id;
    seq_n seqno;
    int data_len;
    uint8_t data[MAX_DATA_LEN];
} node_t;

typedef struct neighbour {
    bool is_permanent;
    time_t last_data;
    struct sockaddr_in6 *addr;
} neighbour_t;

typedef struct pair {
    node_id id;
    size_t nb_neighbours;
    neighbour_t neighbours[MAX_NBR];
    size_t nb_nodes;
    size_t nodes_len;
    node_t *nodes;
} pair_t;
```

- tout suite d’octet est représenté comme uint8_t *p, donc un pointeur vers unsigned char.

- le node_id et seq_n sont des entier de 64 et 16 bit.

- **node** : triplet (i, s, d) est représenté comme un struct qui contient un tableau d’octet donc aussi uint8_t *p qu’on initialise de taille max = 192 et qu’on remplie que les premier case correspondant en gardant la taille actuelle data_len.

- **voisin** (neighbour_t) : contient un bool pour savoir si il est permanent ou transitoire, un time_t pour connaître la dernier date de modification de sont donnée et un pointeur d’adresse de socket *ipv6|IPv4-Mapped* pour pouvoir les indexées.

- **pair :** exactement comme decrit dans le sujet; la *liste de voisin* est représenté comme un tableau de taille max = 15, et la *table de données publiées* est un tableau dynamique qu'on garde toujours trier lors d'ajout d'un node et qu'on stock la capacité nodes_len et le nb de node actuelle nb_nodes , si le nb de node commence a dépasser la capacité on multiplie la cap par 2. pour chaque ajout-suppression de node on décale le tableau avec les fonctions nécessaire (*realloc*, *memmove*, *memcpy*).

Gestion des TLV :

- deux fichiers tlv_makers et tlv_handlers , la premier pour remplir un buffer a partir des paramètres passé en argument a fin d'envoyer ce buffer a un voisin.

la deuxième pour la gestion des paquets entrante via la fonction handle_packet qui gère tout type de paquet en appelant le handler correspondant (ex. handle_tlv_neighbour()).

Librairies utilisé dans le projet:

- pour la hashage on a tout simplement utiliser l’rfc5234 :

https://github.com/massar/rfc6234

(les fichiers sha224-256.c et sha.h et sha-private.h)

Boucle principale :

- utilisation d’une boucle à événements basée sur select avec un timeout de 20s dans le fichier core.c .

Modularité de code :

- Notre concentration était toujours de respecter le « clean code » donc on essaie de faire des #define out des macro des qu'on remarque la nécessite (voir le fichier src/include/network.h) d'autre part les fonctions de sémantique différente sont repartioner dans plusieurs fichier. Les standard de Naming en C sont aussi respecté ainsi qu'un makefile approprier et Une paramètre de ligne de commande -d pour passer en mode de débogage .