Compléxité Cyclomatique :

damemon.c :

fonction create_daemon : complixité cyclomatique de 4

fonction ping_request : complixité cyclomatique de 1

fonction send_check: complixité cyclomatique de 3

fonction check_keep_working : complixité cyclomatique de 4

fonction daemon_work : complixité cyclomatique de 3

db-sqlite.c :

fonction db_connect : complixité cyclomatique de 1

fonction db_disconnect : complixité cyclomatique de 1

fonction insert_hourly_report : complixité cyclomatique de 1

ping-report.c :

fonction main.c : complixité cyclomatique de 4

stats.c :

fonction get_ping_from_temp_log : complixité cyclomatique de 11 **(Voir pdf dans le git pour le diagramme)**

fonction write_ping_log : complixité cyclomatique de 4

fonction set_stats_ping : complixité cyclomatique de 13 **(Voir pdf dans le git pour le diagramme)**

utils.c :

fonction write_pid_file : complixité cyclomatique de 2

fonction remove_file: complixité cyclomatique de 1

-

Nouvelles Fonctions :

get_ping_from_temp_log :

```
static FILE* open_ping_log(void) {
    return fopen("/var/log/ping-report/last-ping.log", "r");
}

static regex_t* compile_ping_regex(void) {
    regex_t* p_reg = malloc(sizeof(*p_reg));
    if (!p_reg) return NULL;
    if (regcomp(p_reg, "time=(.*) ms", REG_EXTENDED) != 0) {
        free(p_reg);
        return NULL;
    }
    return p_reg;
}

static char* extract_ping(const char* line, regmatch_t* pmatch) {
    int start = (int)pmatch[1].rm_so;
    int end = (int)pmatch[1].rm_eo;
    size_t size_ping = (size_t)(end - start);

    char* ping = malloc(sizeof(char) * (size_ping + 2));
    if (!ping) return NULL;

    strncpy(ping, &line[start], size_ping);
    ping[size_ping] = '\n';
    ping[size_ping + 1] = '\0';
    return ping;
}

static char* process_line(const char* line, regex_t* p_reg, regmatch_t* pmatch, size_t nmatch) {
    if (regexec(p_reg, line, nmatch, pmatch, 0) == 0) {
        return extract_ping(line, pmatch);
    }
    return NULL;
}

char* get_ping_from_temp_log(void) {
    FILE* fd = NULL;
    char* read_line = NULL;
    size_t n = 0;
    size_t nmatch = 2;
    regex_t* p_reg = NULL;
    regmatch_t* pmatch = NULL;
    char* ping = NULL;
```

```
    fd = open_ping_log();
    if (!fd) return NULL;

    p_reg = compile_ping_regex();
    if (!p_reg) {
        fclose(fd);
        return NULL;
    }

    pmatch = malloc(sizeof(*pmatch) * nmatch);
    if (!pmatch) {
        regfree(p_reg);
        free(p_reg);
        fclose(fd);
        return NULL;
    }

    while (getline(&read_line, &n, fd) != -1) {
        if (!read_line) break;

        ping = process_line(read_line, p_reg, pmatch, nmatch);
        free(read_line);
        read_line = NULL;
        n = 0;

        if (ping) break;
    }

    regfree(p_reg);
    free(p_reg);
    free(pmatch);
    if (read_line) free(read_line);
    fclose(fd);

}
```

set_stats_ping :

```
static FILE* open_all_ping_log(void) {
   return fopen("/var/log/ping-report/all-ping.log", "r");
}

static void update_ping_stats(const char* line, double* sum, double* max, double* min,
                              int* nb_high, int* nb_loss, int* nb_ping) {
   if (strcmp(line, "LOSS") == 0) {
      (*nb_loss)++;
      return;
   }

   double ping = strtod(line, NULL);
   if (ping < 0.1) {
      return; /* Ignore les pings nuls */
   }

   (*nb_ping)++;
   if (ping > *max) *max = ping;
   if (ping < *min) *min = ping;
   if (ping > 100.0) (*nb_high)++;
   *sum += ping;
}

static void compute_and_store_stats(double sum, double max, double min,
                                    int nb_high, int nb_loss, int nb_ping) {
   double mean = (nb_ping > 0) ? sum / (double)nb_ping : 0.0;
   insert_hourly_report(mean, max, min, nb_high, nb_loss, nb_ping);
}

void set_stats_ping(void) {
   FILE* fd = open_all_ping_log();
   if (!fd) {
      perror("stats : ");
      return;
   }

   double sum = 0.0, max = 0.0, min = 100.0;
   int nb_high = 0, nb_loss = 0, nb_ping = 0;
   char* read_line = NULL;
   size_t n = 0;

   while (getline(&read_line, &n, fd) != -1) {
      if (!read_line) break;
      update_ping_stats(read_line, &sum, &max, &min, &nb_high, &nb_loss, &nb_ping);
      free(read_line);
      read_line = NULL;
```

```
        n = 0;
    }

    fclose(fd);
    compute_and_store_stats(sum, max, min, nb_high, nb_loss, nb_ping);

    if (read_line) free(read_line);
}
```