## **1. Requirement Tính Năng (Functional Requirements)**

### **1.1 Quản lý kết nối**

1. Xử lý nhiều kết nối đồng thời:
   1. Hệ thống phải hỗ trợ việc xử lý nhiều kết nối TCP từ các peer đồng thời.
   2. Sử dụng cơ chế multiplexing I/O (epoll, select, hoặc kqueue) hoặc asynchronous I/O để quản lý kết nối hiệu quả.
2. Quản lý kết nối theo phiên (session):
   1. Mỗi kết nối phải được quản lý theo một session riêng biệt với thông tin về:
      1. ID kết nối (connection ID).
      2. Địa chỉ IP và port của peer.
      3. Thời gian hoạt động của kết nối (để hỗ trợ timeout).
3. Danh sách kết nối:
   1. Hệ thống phải duy trì danh sách các kết nối đang hoạt động, cho phép truy cập nhanh để thực hiện các thao tác như gửi tin nhắn, ngắt kết nối, hoặc kiểm tra trạng thái.
4. Timeout cho kết nối không hoạt động:
   1. Tự động ngắt kết nối các sensor không gửi bất kỳ dữ liệu nào trong một khoảng thời gian nhất định (timeout).

### **1.2 Quản lý trạng thái hệ thống**

* Hiển thị thông tin hệ thống:
  + status: Hiển thị trạng thái hiện tại của hệ thống, bao gồm:
    - Số lượng kết nối đang hoạt động.
    - Tài nguyên hệ thống đang sử dụng (CPU, RAM).
* Thống kê kết nối:
  + stats: Hiển thị thông tin thống kê chi tiết về từng kết nối:
    - Địa chỉ IP và port.
    - Thời gian kết nối.
* Log hoạt động:
  + Ghi lại tất cả các sự kiện (kết nối, ngắt kết nối, lỗi, tin nhắn gửi/nhận) vào file log để phục vụ việc kiểm tra và phân tích.

### **1.4 Bảo mật**

* Xác thực kết nối:
  + Yêu cầu các peer gửi thông tin xác thực (username/password hoặc token) khi kết nối.
* Mã hóa dữ liệu:
  + Sử dụng giao thức TLS/SSL để mã hóa dữ liệu trao đổi giữa các peer.
* Giới hạn số lượng kết nối:
  + Đặt giới hạn số lượng kết nối tối đa từ một peer (để tránh tấn công DoS).

### **1.5 Thoát và phục hồi**

* Thoát chương trình:
  + exit: Ngắt tất cả các kết nối đang hoạt động và ghi lại trạng thái hệ thống trước khi thoát.
* Phục hồi từ lỗi:
  + Hệ thống phải có khả năng tự động khôi phục trạng thái nếu gặp lỗi (crash hoặc mất kết nối).

## **2. Requirement Phi Tính Năng (Non-Functional Requirements)**

### **2.1 Hiệu năng**

* Tối ưu xử lý đồng thời:
  + Sử dụng event-driven architecture (như epoll, libuv) hoặc multi-threading để xử lý hàng nghìn kết nối đồng thời mà không bị treo.
* Sử dụng tài nguyên hiệu quả:
  + Hạn chế sử dụng CPU và RAM quá mức.
  + Giải phóng tài nguyên ngay sau khi kết nối bị ngắt.

### **2.2 Độ tin cậy**

* Hỗ trợ lỗi (Fault Tolerance):
  + Hệ thống phải tiếp tục hoạt động bình thường nếu một số peer bị ngắt kết nối hoặc gửi dữ liệu không hợp lệ.
* Cơ chế Retry:
  + Tự động thử lại khi kết nối bị ngắt tạm thời.
* Kiểm tra và phục hồi kết nối:
  + Thường xuyên kiểm tra trạng thái kết nối và tự động phục hồi nếu cần.

### **2.3 Khả năng mở rộng (Scalability)**

* Phân phối tải (Load Balancing):
  + Sử dụng một load balancer để phân phối kết nối đồng đều giữa các instance.

### **2.4 Bảo mật**

* Ngăn chặn tấn công DoS/DDoS:
  + Giới hạn số lượng kết nối từ một địa chỉ IP cụ thể.
  + Sử dụng tường lửa hoặc các công cụ bảo mật mạng (optional).
* Mã hóa dữ liệu:
  + Sử dụng các giao thức an toàn như TLS/SSL để bảo vệ dữ liệu trong quá trình truyền tải.

### **2.5 Mã nguồn sạch và bảo trì**

* Thiết kế module hóa:
  + Chia hệ thống thành các module độc lập (quản lý kết nối, xử lý tin nhắn, ghi log, v.v.) để dễ bảo trì.

**2.6. Không bị rò rỉ bộ nhớ (Memory Leak)**

* 1. Đảm bảo giải phóng tài nguyên (socket, bộ nhớ) khi kết nối bị ngắt hoặc khi chương trình thoát.
  2. Sử dụng các công cụ kiểm tra rò rỉ bộ nhớ (như Valgrind) nếu lập trình bằng C/C++.