0 – Contexte

2.1  Cohérence concurrente et synchronisation

* conditions de course sur les listes de clients, tables de canaux et compteurs ; ordre de diffusion incohérent ; verrous croisés pouvant mener à un deadlock.
* doublons ou omissions dans la liste des membres, notifications contradictoires, messages perdus.
* sections critiques protégées par mutex/RW‑lock, structures thread‑safe ou modèle acteur ; éliminer les verrous globaux, tester à l’aide de scénarios concurrents.

2.2  Modularité et séparation des responsabilités

* responsabilités :
  1. Acceptation des connexions et gestion du cycle de vie client
  2. Parsing/validation des commandes
  3. Logique métier : canaux, droits, routage
  4. Diffusion des messages
  5. Journalisation et, si besoin, persistance
* le réseau convertit une ligne texte en Command, reçoit une Response, puis l’envoie ; la logique métier est donc testable hors réseau.
* parseur pour la syntaxe, logique métier pour la sémantique ; la couche réseau transporte seulement.

2.3  Scalabilité et capacité à évoluer

* étendre parseur/dispatcher + module métier concerné, rien d’autre.
* passer à I/O non bloquant ou async, thread‑pool, back‑pressure, partitionnement par canal, cluster distribué.
* 1 thread bloquant par client, verrous globaux, aucune persistance ni sharding, pas de quota ou file d’attente.

2.4  Portabilité de l’architecture

* on garde la logique métier ; on remplace la couche socket et on sérialise (JSON/REST).
* gestion utilisateurs, gestion canaux, routage/broadcast, historique/logs, passerelle API.
* stock commun (base ou cache) + bus d’événements/pub‑sub pour les mises à jour.

2.5  Fiabilité, tolérance aux erreurs, robustesse

* recv == 0 ou exception ECONNRESET ; nettoyage de l’état et notification des canaux.
* EPIPE/BrokenPipe ; marquer le client mort, déclencher la même routine que pour une déconnexion.
* write‑ahead log avant diffusion, accusés de réception, ou au minimum log fiable des tentatives.

2.6  Protocole : structuration et évolutivité

* une ligne UTF‑8 par commande, préfixe /cmd + arguments ; protocole maison non normalisé.
* retourner un code d’erreur clair (ERR\_BADFORMAT, ERR\_NOCHANNEL) si arguments manquants/invalides.
* ABNF minimale + doc des codes d’erreur et séquences d’états pour les développeurs de clients.
* connexion longue, état conservé, push server→client, framing texte simple ; REST est sans état, requête/réponse, métadonnées via en‑têtes.