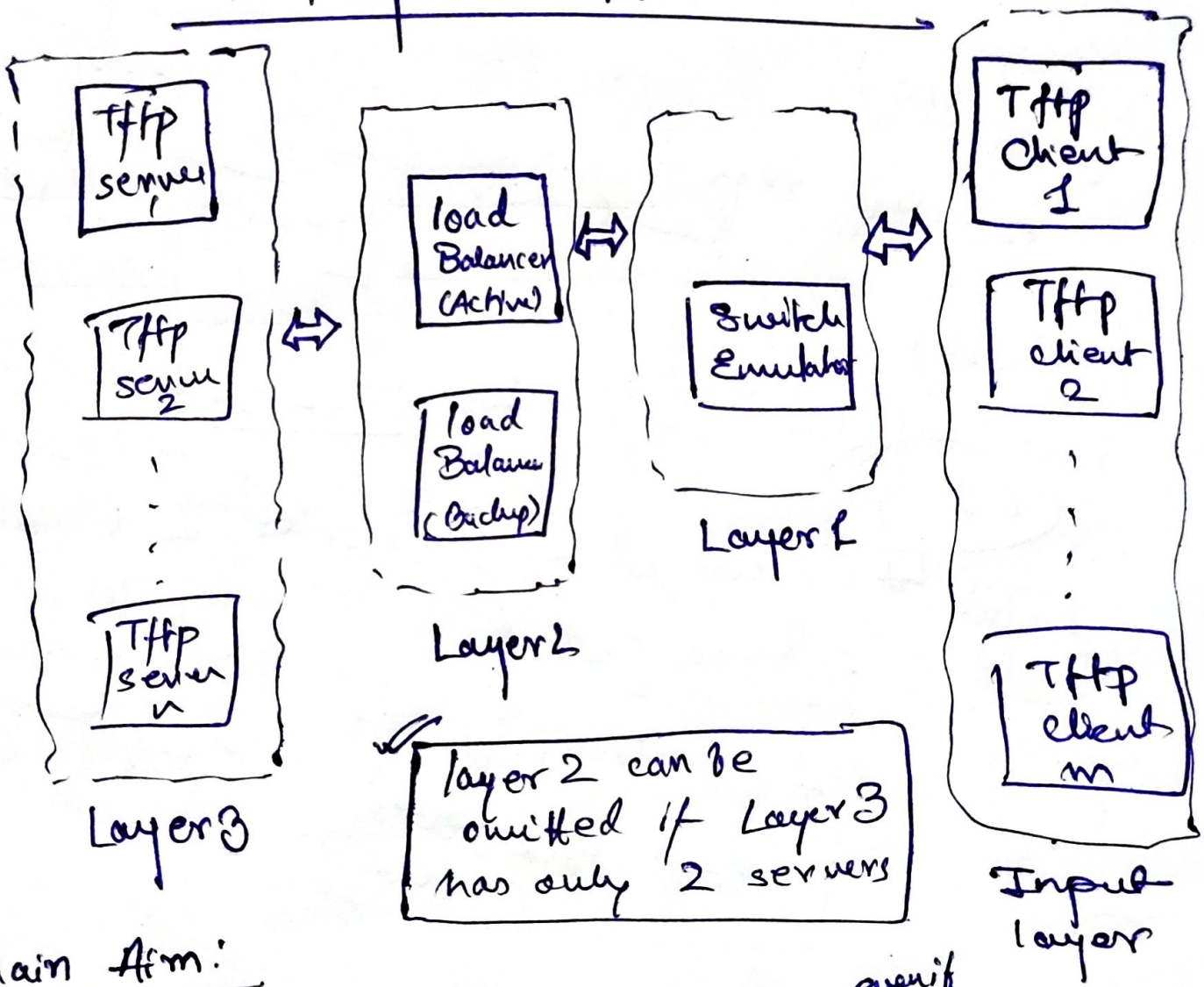
Highly Available Distributed File Transfer System

Proposed Architecture



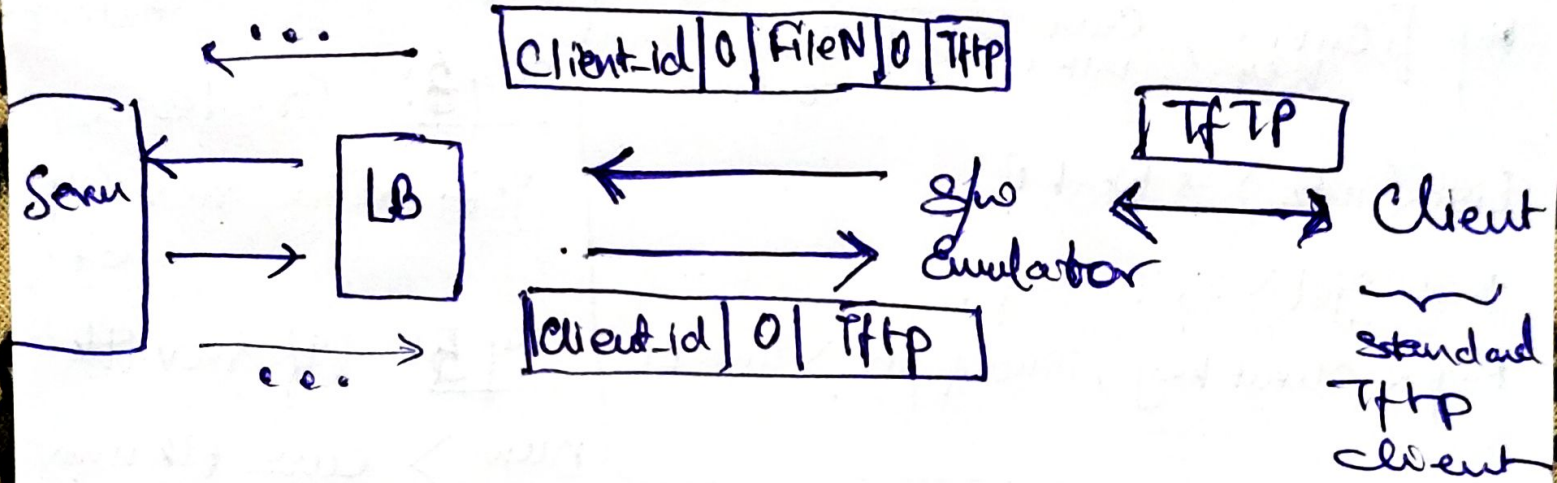
Main Aim:

↳ If File Transfer done x% [even if then either Layer-2/3 any one instance goes down] the remaining (100-x%) will have no effect.

No need to restart full-retransfer.

Internal Protocol

Client_id = IP + "_" + Port



Need of Client_id?

↳ Else s/w will not know which client to send the tftp response.

Need of FileN?

↳ Else maintain Context of Client_id v/s FileN on Layer-3 Server. ⇒ This Context has to shareable to all servers of Layer-3.

Working Example (How Server Reads)

Suppose C_1, C_2 are doing Read on F_1 & F_2

C_1 sends Ack 4 & C_2 sends Ack 17.

Then Server does go to F_1 → fseek 512x(4) → extract next 512 & build data Packet.

Similarly goto F_2 → fseek 512x(17) ...

Switch Emulator

Has 2 threads ① Read ② Frwd.

Has following Global DS

Map < client-key, fileName > client-counter.

watchdogCnt {0}

List <job> workQ.

Server_list [2] = { Active IP_Port, Back IP_Port }.

Read thread → **Step1**: Recv from()

Step2: Make job as per sender of the recv packet → client / active LB

Step3: workQ.push(job).

if(packet recv from active LB) → watchdogCnt = 0 //.

Frwd thread → **Step1**: Extract job from workQ.

Step2: Serve the job (Add/Remove prefix and Send packet to its destination).

Step3: If the destn is Active LB (watchdogCnt ++)

if(watchdogCnt / client_num > threshold)

- Swap (Active, Backup).
- Traverse workQ (& update destn of the active LB/server)

Assumption: UDP will have retransmission cnt — if the retransmission per client > threshold then the destn is as good as dead.

Load Balancer (working).

* Has 3 threads ① Read ② Frwd ③ Timer
* Has the following Global Data Structures.

Map <Server-id, INT> WD_cnt

Map <Server, Set <Clients>> server Cli Map.

List <Job> Work Q Set of Active & Deactive Servers

Read Thread → Step1 ① does Recv from() Step2 ② Create job.

Step3 ③ As per Sender of the Recv-packet()

if(Sender == switch-emulator)

↳ Extract Client-id from recv-packet

Assign Client → to server with least load.

else set server's WD_cnt to 0.

Step4 ④ Push job to work Q.

Frwd Thread: Step1 Extract job from work Q.

Step2 Serve the job.

Step3 if the job_destn is Tftp-server then ↑ its WD_cnt by 1 & if WD_cnt > threshold then the server is down.

Step4 Remove it from Active Server List and Add to deactive Server List & Remove from Server Cli Map.

Timer Thread: Periodically Wake Up & Move Contents of Deactive Server List to Active Server List.

Tftp server (working).

* 3-thread ① Read ② Frwd ③ Timer

Read thread :→

Step1: Recv from & stores IP_Port of sender (LB/svr)

Step2: Extract prefix from recv data buffer → Client_key → Filename

Step3: init client context

Step4: Create a job as per recv tftp packet

Step5: If recv blk num > curr blk num

⇒ New packet recv ⇒ { ① Push Job to Work Q. ② Erase Job from Timer Q.

Global DS (shared by 3 threads)

Map [Client key, Curr Blk num] Client context

List<Jobs> → Work Q

List<Job> → Timer Q.

Map<Client key, TimerQ ptr> TimerM

Frwd thread :-

Step1: Extract job from WorkQ.

Step2: Perform the job

Step3: Push the job to TimerQ.

Timer thread :-

→ Periodically wakes up.

→ Traverses the TimerQ. & puts the expired jobs in the WorkQ.

Need of TimerM? → Clear Timer (by Read Thread) in O(1) time. avg.

TimerQ → List [as jobs are inserted in order of timestamp

⇒ in TimerThread [No need to traverse full TimerQ.]