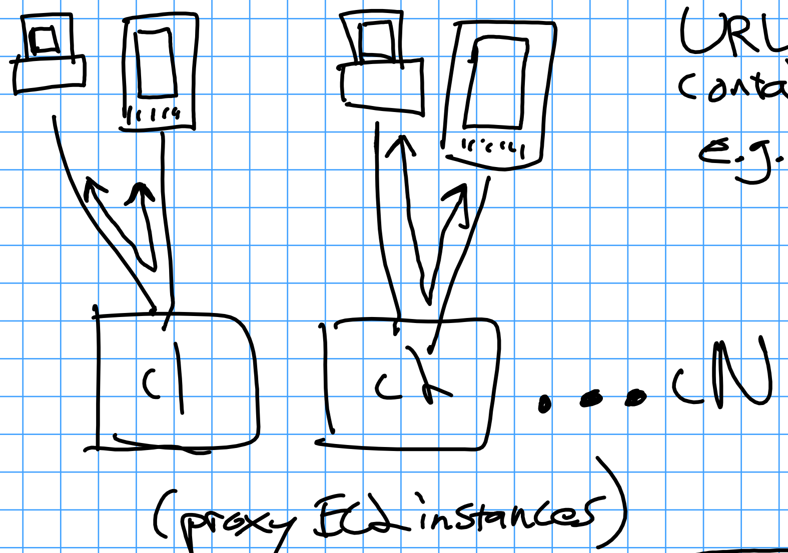
SwiFTP proxy internal architecture

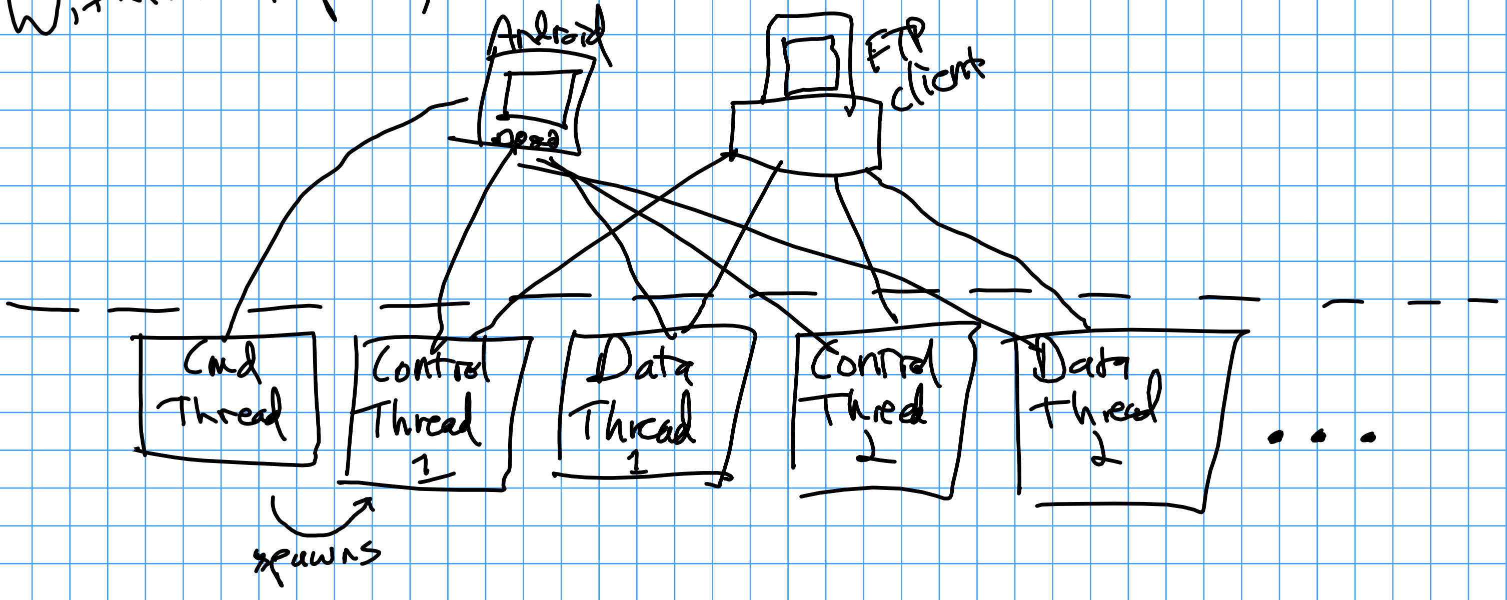


URL swiftp.s3 .... /servers contains a list of proxy server URLs
e.g. c1.swiftp.org
c2.swiftp.org
⋮

DNS names are manually mapped to EC2 elastic IPs.

- Devices (FTP Servers) will connect to a random server on startup. We rely on this randomness for load balancing.
- For any Android device/FTP server, all connections to/from that device are handled by the same proxy. That avoids complicated load balancing and inter-proxy communication.
- If any proxy server fails, all command, control, and data sessions through that proxy will be broken. The Android device will establish a new command session with a ~~different~~ proxy.

Within a proxy node:



Android device comes online:
- Picks proxy at random, establishes control connection
- Device and proxy wait for anything to happen

FTP client connects on FTP control port
- Proxy spawns thread to accept()
- thread looks up prefix to find command thread, starts listening on arbitrary port, sends message to cmd thread with port number (to be passed to device)
- Device should connect momentarily, and JSON authenticate
- Control connection established, proxying begins

Client sends PASV to device via control connection
- Device opens new socket to proxy, authenticates, sends pasv_listen
- Server begins listening on a random port, replies to device with port number
- Device sends port to client (device already knows hostname)
- In either order, device sends pasv_accept and client connects
- As soon as both ↑ happen, proxying begins, data session OK

Client sends PORT <ip> <port>
- Device remembers arguments, and waits
- When it's time for data transfer, device:
    - opens a new socket to the proxy, authenticates
    - sends port_connect with host and ip
- Socket proxying begins

Logging, stats, quotas, throughput limits