Video Sharing Protocol (VSP/1.0)

Application-Layer Protocol Specification

Waqar Ali (B23F0126AI080)
Najam ali
Faiq ali
Date: 18 Nov 2025

Video Sharing Protocol

1) Overview

(Video Sharing Protocol) is a simple, text-based, application-layer protocol over **TCP** (optionally **TLS**) for a video-sharing app. It supports:

- User authentication
- Video creation & resumable chunked uploads
- Metadata handling
- Listing and streaming/downloading with range requests- Basic error handling, versioning, and extensibility

This protocol intentionally uses readable, HTTP-like lines and headers so students can implement it with raw sockets while keeping the message formats explicit (no reliance on HTTP libraries).

---

2) Use Cases & Scope

In scope (MVP):
- User logs in to obtain a bearer token
- Create a new upload session
- Upload video in chunks with checksums
- Commit/abort upload
- List available videos
- Fetch video bytes (support `Range` for streaming) - Fetch/update video metadata (optional for MVP)

Out of scope (for future versions):
- Real-time live streaming
- Adaptive bitrate / DASH/HLS manifests
- Comments/likes/subscriptions
- Thumbnails/transcoding pipeline

---

3) Functional Requirements

FR-1: Users can authenticate and receive a time-bound token.
FR-2: Client can create a new upload session for a video (title, size, format).
FR-3: Client uploads video using **fixed-size chunks** (except final chunk).
FR-4: Client can **resume interrupted uploads** using `Upload-Id` and byte offsets.
FR-5: Client can **commit** an upload to finalize the video or **abort** it.
FR-6: Client can **list** videos it has access to, with pagination.
FR-7: Client can **stream or download** video bytes with `Range` support.
FR-8: Basic **metadata** operations (get/update title, description).
FR-9: Protocol **versioning** and simple feature negotiation.
FR-10: **Error handling** with numeric status codes and machine-readable messages.
---

4) Non-Functional Requirements (Network-Focused)

- **Reliability**: Built on **TCP** for ordered, reliable delivery.
- **Security**: Support **TLS 1.2+** in production; bearer tokens for auth.
- **Performance:** Chunked upload with client-tunable `Chunk-Size` (e.g., 512 KB–4 MB).
- **Resilience:** Resume via `Content-Range` and `Upload-Id`.
- **Scalability:** Stateless request handling (except upload state), file-backed or object storage.
- **Observability:** `Request-Id` header, server-side logs, and clear error codes.
- **Interop:** Text-based framing, UTF-8, CRLF line endings.
- **Backward Compatibility:** `VSP-Version` and negotiation; unknown headers must be ignored.
- **Rate Limiting (optional):** `429 Too Many Requests` with `Retry-After`.
- **Portability:** Implementable in any language with TCP sockets.

---

5) Transport & Port

- **Transport:** TCP (recommended: TLS over TCP for production)
- **Default Dev Ports:**
- Plain TCP: `9080`- TLS: `9443`
- **Keep-Alive:** Either single request per connection OR multiple requests per connection (bothallowed).
- **Line endings:** CRLF (`\r\n`) for all protocol lines.

---

6) Protocol Message Model

6.1 Start-Line & Headers

Client Request Start-Line:
```
VSP/1.0\r\n
```

Server Response Start-Line:
```
VSP/1.0 \r\n
```

Headers:
```
Header-Name: value\r\n
...\r\n
[optional body]
```

Character set: UTF-8 for headers and any text bodies.
Binary body: Raw bytes (e.g., chunk uploads, video responses).

6.2 Core Headers

Header	Direction	Required When	Description
Content-Length	C↔S	When body present	Size in bytes of body.
Content-Type	C↔S	If body is not binary chunk	e.g., `application/json`, `video/mp4`
Authorization	C→S	After login	`Bearer `
VSP-Version	C↔S	Optional	e.g., `1.0`
Request-Id	C↔S	Optional	Client-generated UUID for tracing
Upload-Id	C→S	For CHUNK/COMMIT/ABORT	Upload session id
Video-Id	C↔S	When referencing a video	Stable video identifier
Content-Range	C→S	CHUNK	`bytes -/`
Range	C→S	GETVID (optional)	`bytes=-`
Chunk-Checksum	C→S	CHUNK (recommended)	`sha256=`
Video-Checksum	C→S	COMMIT (recommended)	`sha256=`
Chunk-Size	S→C	NEWVID response	Server-accepted chunk size
Accept-Codecs	C→S	NEWVID (optional)	e.g., `video/mp4; codecs="avc1.42E01E"`
Retry-After	S→C	429	Seconds to wait
Location	S→C	201	Resource URI
Error-Code	S→C	4xx/5xx	Machine code, e.g., `UPLOAD_OFFSET_MISMATCH`

---

7) Methods (Message Types)

1. HELLO — Version / Liveness
2. LOGIN — Authentication
3. NEWVID — Create Upload Session
4. CHUNK — Upload Data Chunk (Resumable)
5. COMMIT — Finalize Upload
6. ABORT — Cancel Upload
7. LIST — List Videos
8. GETVID — Stream/Download Video Bytes
9. META — Metadata Get/Set
10. DELETE — Remove Video

---

8) Status Codes

- **1xx**: (Reserved; not used)
- **2xx**: Success
- `200 OK`
- `201 Created`
- `202 Accepted` (chunk received)
- **4xx**: Client Errors
- `400 Bad Request`
- `401 Unauthorized`
- `403 Forbidden`
- `404 Not Found`
- `409 Conflict` (e.g., offset mismatch)
- `413 Payload Too Large`
- `416 Range Not Satisfiable`
- `429 Too Many Requests`

- `431 Request Header Fields Too Large` (optional)
- **5xx**: Server Errors
- `500 Internal Server Error`
- `501 Not Implemented`
- `503 Service Unavailable`
- `505 Version Not Supported`

---

9) Application Architecture

9.1 Logical Components

- **VSP Frontend (Server):** Parses VSP messages, handles auth, routes methods.
- **Auth Service:** Validates credentials and issues tokens (JWT or opaque).
- **Upload Manager:** Tracks `Upload-Id`, received byte ranges, checksums.
- **Storage Layer:** File system or object storage for chunks and committed videos.
- **Metadata Store:** Simple DB or JSON index for titles, sizes, mime, owners.
- **Streamer:** Serves bytes with `Range` support.
- **(Optional) Transcoder/Thumbnailer:** Future work.

9.2 Upload State Machine

```
[IDLE]
|
| NEWVID
v
[CREATED(upload_id)]
|
| CHUNK (0..N) in order (or resume)
v
[RECEIVED_PARTS]
|
| COMMIT (if all bytes received, checksum
ok) v
[COMMITTED(video_id)]
^
| ABORT (from CREATED or RECEIVED_PARTS)
[ABORTED]
```

10) Message Format (Formal)

```
+---------------------------------------------------------------------------------------------------------------+
                                        | VSP Message |
 +--------------------------------------------------------------------------------------------------------------+
 |                      request-message / response-message                                                      |
+---------------------------------------------------------------------------------------------------------------+
                                  REQUEST MESSAGE:
+---------------------------------------------------------------------------------------------------------------+
|  request-line                                                                                                 |
| METHOD SP request-target SP version CRLF |
| METHOD = HELLO | LOGIN | NEWVID | CHUNK | COMMIT |                                                            |
| ABORT | LIST | GETVID | META | DELETE                                                                         |
| request-target = "/" *(pchar)                                                                                 |
| version = "VSP/1.0"                                                                                           |
| *(header-field CRLF)                                                                                          |
| header-field = header-name ":" SP header-value                                                                |
| header-name = token                                                                                           |
| header-value = *(VCHAR / SP)                                                                                  |
| token = 1*(ALPHA / DIGIT / "-" / "_")                                                                         |
| CRLF                                                                                                          |
+---------------------------------------------------------------------------------------------------------------+
| [ message-body ]                                                                                              |
| message-body = *OCTET                                                                                         |
 +-------------------------------------------------------------------------------------------------------------+
```

RESPONSE MESSAGE:

```
+----------------------------------------------------------------------------------------------------------------+
| status-line                                                                                                    |
| version SP status-code SP reason-phrase CRLF                                                                   |
| status-code = 3DIGIT | | reason-phrase = *(VCHAR / SP)                                                         |
+----------------------------------------------------------------------------------------------------------------+
| *(header-field CRLF) | | CRLF | | [ message-body ]                                                             |
+----------------------------------------------------------------------------------------------------------------+
```

11) Security Model (optional)

- **Transport:** Use **TLS 1.2+** in production; self-signed for local dev.- **Auth:** `LOGIN` returns a token; all subsequent methods use `Authorization: Bearer `.
- **Password Storage:** Server must hash using **bcrypt/argon2** (implementation detail).
- **Authorization:** Ownership checks for `DELETE`, `META SET`, etc.
- **Input Validation:** Strict header and body validation; reject overlong headers.
- **Rate Limiting:** Return `429` for abuse; include `Retry-After`.

---

12) Error Handling (ontional)

- Always include meaningful `Reason-Phrase`.
- For JSON error bodies (recommended):
```
VSP/1.0 409 Conflict
Content-Type: application/json
Error-Code: UPLOAD_OFFSET_MISMATCH Content-
Length:

{"message":"Server expects next byte at 1048576"}
```
13) Caching, Streaming & Ranges (optional)

- **Streaming:** Use `GETVID` with `Range` to play progressively.
- **Caching:** Client may cache previously fetched ranges; server may add `ETag` (optional).
- **Ranges:** If no `Range` header, server may send full content (`200`). With `Range`, send `206`.---

16) Developer Notes

16.1 Suggested Stack (MVP)
- **Language:** Python, Java, Node, or C# (anything with TCP sockets).
- **Server Structure:**
- Accept TCP connection
- Read lines until blank line → parse headers
- If `Content-Length` present, read exact bytes for body
- Dispatch by `Method` and `Request-Target`
- Write response start-line, headers, CRLF, body

- **Storage:**
- Store chunks as temp files: `uploads//-`
- On `COMMIT`, concatenate in order to `videos/.mp4`
- Maintain a `videos.json` index for metadata
- **Auth:** Keep a simple in-memory or file-backed token store (`token -> user_id, expiry`) - **Checksum:** Use SHA-256 (`hashlib` in Python). Compare per chunk and final.