Spatial video on mp-quic

Zhewen Yang

mp-quic code arch

MP-QUIC (Multipath QUIC) extends QUIC to support **multiple paths** for improved bandwidth utilization and reliability.

Key Components

- **Scheduler**: Selects the best path based on RTT or Round Robin.
- **QUIC Transport Layer (`quic-go`)**: Manages multipath connections and packet handling.
- **HTTP/2 Adaptation (`h2quic`)**: Enables HTTP/2 over MP-QUIC, used for DASH streaming.
- **DASH Client (`AStream`)**: Handles adaptive streaming over MP-QUIC.

Code Structure

- `mpquic-sbd/src/quic-go/` (MP-QUIC implementation)
- `mpquic-sbd/src/AStream/` (DASH client)
- `mpquic-sbd/src/dash/client/proxy_module/` (Python-Go QUIC binding)

mp-quic code arch (con)

The **scheduler** module manages multipath selection and load balancing.

Path Selection Strategies

- **Round Robin (`selectPathRoundRobin()`)**: Cycles through available paths.
- **Lowest RTT (`selectPathLowLatency()`)**: Selects the path with the lowest RTT.

```
func (sch *scheduler) selectPathLowLatency(s *session, ...) *path {

    return selectedPath  // Chooses the lowest RTT path

}
```

Packet Handling

- **Retransmissions (`getRetransmission()`)**: Detects lost packets and resends them.
- **Window Updates (`performPacketSending()`)**: Adjusts flow control dynamically.
- **ACK Management (`ackRemainingPaths()`)**: Ensures all paths acknowledge received data.

mp-quic code arch (con)

MP-QUIC is used for **adaptive DASH video streaming**.
The Python client (`conn.py`) interacts with the Go QUIC module (`proxy_module.so`) via `ctypes`.

Client Process

1. **DASH Client Initialization** (`dash_client.py`)
   - Calls `setupPM()` to configure QUIC.
2. **QUIC Connection Setup** (`proxy_module.go`)
   - `quic.DialAddr()` establishes MP-QUIC session.
3. **Multipath Selection** (`scheduler.go`)
   - Selects the optimal path dynamically.
4. **Segment Download** (`DownloadSegment()`)
   - Fetches video segments over MP-QUIC.
5. **HTTP Response Handling** (`handleHeaderStream()`)
   - Decodes and delivers the requested content.

```
def setupPM(useQUIC, useMP, keepAlive, schedulerName):

      scheduler = GoString(schedulerName.encode('ascii'), len(schedulerName))

        lib.ClientSetup(useQUIC, useMP, keepAlive, scheduler)
```

Code work and functional now and I made a doc for how to modify the code

Problem Statement

- Apple Vision Pro **Spatial Video** captures **stereo 3D content** with **high bandwidth requirements**.
- Traditional transmission methods (TCP, UDP) **do not optimize for multi-path or bandwidth constraints**.
- Efficient **3D video streaming** requires:
  - **Reducing redundant data transmission**
  - **Maintaining synchronization between left & right views**
  - **Preserving spatial metadata for correct 3D rendering**

Challenges

❌ **High Bandwidth Consumption** (Full-resolution stereo video doubles the data)
❌ **Network Congestion** (Limited capacity for high-quality video)
❌ **Synchronization Issues** (Delays in multi-path transmission)

Initail Idea

Splitting, Transmitting, and Reconstructing Spatial Video

Spatial Video as an input (Apple Vision Pro format) (or directly from two cameras)

Step 1: Convert to **Side-by-Side (SBS)** format

Step 2: Split into **Left Eye Video & Right Eye Video (extract the metadata)**

Step 3: **MPQUIC Transmission** (left & right sent separately)

Step 4: **DASH Streaming** for adaptive bitrate

Step 5: **Reconstruct & restore metadata** at the receiver

Proposed Approach

Key Steps

✅ **Preprocessing:** Convert **Packed Spatial Video → Side-by-Side (SBS) format**
✅ **Splitting:** Extract **Main View (High-Res) + Residual View (Low-Res or Difference Frame)** ✅ **MPQUIC Transmission:**

- **Main Path:** Sends **High-Res Main View**
- **Secondary Path:** Sends **Low-Res Residual View**
  ✅ **DASH Streaming:** Adaptive bitrate streaming for real-time playback
  ✅ **Super-Resolution Reconstruction:** Combine residuals on receiver to **restore high-quality stereo video**

Preprocessing & Splitting

Tools & Methods

- **FFmpeg**: Converts Apple Vision Pro **Packed Spatial Video → Side-by-Side (SBS) Format**
- **Stereo Decomposition**: Splits **left-eye & right-eye views**
- **Residual Calculation**:
  - **Main Path:** Sends the primary **main view** (left or right)
  - **Secondary Path:** Transmits **low-res or residual difference**

Challenges

❌ How to define the optimal **main view vs residual representation**?
❌ Compression artifacts may degrade **residual-based reconstruction**
❌ Need **real-time pre-processing** for live streaming

(For now I just splited in to right view and left view, tested worked, but still have problem in the metadata part, ffmpeg seems can not handle spatial video well in metadata, trying spatial tool right now)

MPQUIC for Efficient Transmission

How It Works

- **Path Selection Algorithm** dynamically chooses the best primary path
- **Main Path** transmits the **high-resolution view**
- **Secondary Path** carries **low-resolution view or compressed residuals**
- **MPQUIC enables congestion-aware, adaptive transmission**

Challenges

❌ How to choose the **optimal path dynamically**?
❌ Handling **packet loss & reordering in multipath QUIC**
❌ Avoiding **latency-induced desynchronization**

These can all be done by modified in the scheduler, I have already done some tests. And for now I think just use two same resolution views.

Idea: Hybrid QUIC Strategies: Combining forward error correction (FEC) with MPQUIC

DASH/HLS Streaming for Adaptive Playback

How It Works

- Uses **DASH** to enable **adaptive bitrate selection** based on network conditions
- Ensures **synchronization between left & right eye views**
- **Low-latency DASH encoding** optimizes for **real-time transmission**

Challenges

❌ Maintaining **perfect sync** between stereo streams
❌ How to handle **network fluctuations** while ensuring stable 3D perception
❌ Selecting **optimal encoding parameters** for high efficiency

Potential Solutions

Sync-aware DASH algorithms for stereo video
Low-latency encoding pipelines for real-time spatial video
Joint optimization of DASH + MPQUIC for network-adaptive transmission

Super-Resolution Reconstruction on Receiver

How It Works

- **Main View:** Received in **high quality**
- **Residual View:** Received in **low-res or compressed format**
- **Super-Resolution Model:** Uses deep learning (e.g., **SRCNN, ESRGAN**) to enhance the residual
- **Final Reconstruction:** Combines **main view + super-res residual**

Challenges

❌ Requires **low-latency super-resolution** for real-time playback
❌ How to **train a model** that generalizes well to **stereo video compression artifacts**
❌ Computational cost vs **real-time feasibility**

Results & Expected Outcomes

✅ **Significant bandwidth reduction** by **sending residual instead of full stereo views**

✅ **Better QoE** due to **higher resolution restoration** at the receiver

✅ **Lower latency transmission** with **MPQUIC multi-path optimization**

Plan: Try to fix the metadata part, and combine the pretransmission and the transmission in 1-2 weeks (I have done tested separately and they worked)