ISSN: 2582-7219

International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Impact Factor: 8.206 **Volume 8, Issue 7, July 2025**

© 2025 IJMRSET | Volume 8, Issue 7, July 2025| DOI:10.15680/IJMRSET.2025.0807005

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Live Video Streaming using the Secure Reliable Transport (SRT) Protocol: Architecture and Performance Insights

Pramod Negi[1], Vibha Negi[1], Srihari Kumar Pendyala[2]

Independent Researcher, USA[1]

T-Mobile, USA[2]

ABSTRACT: The demand for high-quality, low-latency video streaming has surged across various domains, including live broadcasting, telemedicine, remote education, and real-time surveillance. Traditional streaming protocols such as RTMP and HLS are often constrained by high latency and limited resilience in unreliable network environments. Secure Reliable Transport (SRT) is an emerging UDP-based protocol designed to address these limitations through features such as adaptive packet retransmission, forward error correction, and stream-level encryption. This paper presents a detailed end-to-end architecture for live video streaming using the SRT protocol, detailing the components in the encoding, transmission, and decoding pipeline. We discuss implementation strategies using open-source tools, such as FFmpeg and OBS, alongside practical considerations, including NAT traversal and jitter buffer tuning. Furthermore, performance evaluations demonstrate SRT's effectiveness in maintaining stream integrity and low latency under variable network conditions. The study concludes with a discussion of current limitations, deployment recommendations, and future directions for SRT-based streaming systems.

KEYWORDS: Secure Reliable Transport (SRT), Live Video Streaming, Low-Latency, Packet Loss Recovery, Jitter Buffering, NAT Traversal, FFmpeg, Open Broadcaster Software (OBS), Real-Time Communication, Video Contribution, UDP Streaming, Stream Reliability, AES Encryption, RTMP, HLS, Cloud Streaming

I. INTRODUCTION

Live video streaming is an integral component of modern digital communication, enabling real-time interaction across geographical boundaries. In applications ranging from professional broadcast workflows to mission-critical remote operations, the ability to transmit video with minimal delay and maximal reliability is essential. However, conventional protocols like Real-Time Messaging Protocol (RTMP), HTTP Live Streaming (HLS), and Dynamic Adaptive Streaming over HTTP (DASH) exhibit significant drawbacks for real-time use cases. They often suffer from high end-to-end latency, poor adaptability to network fluctuations, and limited native support for secure transmission.

The Secure Reliable Transport (SRT) protocol, initially developed by Haivision and now managed by the open-source SRT Alliance, was engineered to overcome these challenges. It uniquely combines the low-overhead speed of UDP with reliability mechanisms traditionally associated with TCP. By incorporating intelligent packet loss recovery, dynamic jitter buffering, and optional AES encryption, SRT enables low-latency, high-fidelity video transmission even over unpredictable public networks.

This paper presents a comprehensive examination of the implementation of a complete end-to-end video streaming solution utilizing the SRT protocol. It describes the architectural components, software tools, and configuration strategies necessary to deploy a robust streaming system, with a focus on real-world cloud deployment scenarios. Additionally, it evaluates the protocol's performance under simulated network impairments to provide insights into its limitations and future potential.

© 2025 IJMRSET | Volume 8, Issue 7, July 2025| DOI:10.15680/IJMRSET.2025.0807005

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

II. BACKGROUND AND RELATED WORK

2.1Overview of Streaming Protocol

A typical live video workflow involves capturing video, compressing it via a codec (e.g., H.264, HEVC), and transmitting it over a network for decoding and playback. The transport protocol is a critical choice that dictates the system's latency, reliability, and scalability.

- **RTMP (Real-Time Messaging Protocol)**: Once dominant, RTMP's reliance on TCP introduces retransmission delays (Head-of-Line blocking), making it less suitable for low-latency applications.
- **HLS (HTTP Live Streaming) & DASH (Dynamic Adaptive Streaming over HTTP)**: These HTTP-based protocols are well-suited for large-scale distribution and video-on-demand, as they utilize standard web servers. However, their segment-based nature inherently introduces several seconds of latency, rendering them unsuitable for real-time interaction.

2.2Emergence of SRT

SRT (Secure Reliable Transport) is a UDP-based protocol explicitly designed to address the drawbacks of its predecessors for live video contribution. Its key features include:

- **ARQ (Automatic Repeat reQuest):** An intelligent packet loss recovery mechanism that retransmits only the lost packets.
- **Selective Acknowledgments (SACK):** Used alongside ARQ to avoid redundant retransmissions.
- **Timestamp-Based Packet Delivery:** Enables precise timing reconstruction to mitigate network jitter.
- **Configurable Jitter Buffering:** A receiver-side buffer absorbs network timing variations, preventing stream disruption.
- **AES-128/256 Encryption:** Secures content delivery from source to destination.

These features enable SRT to deliver stable, high-quality streams over unpredictable networks, such as the public internet, where packet loss and jitter are common.

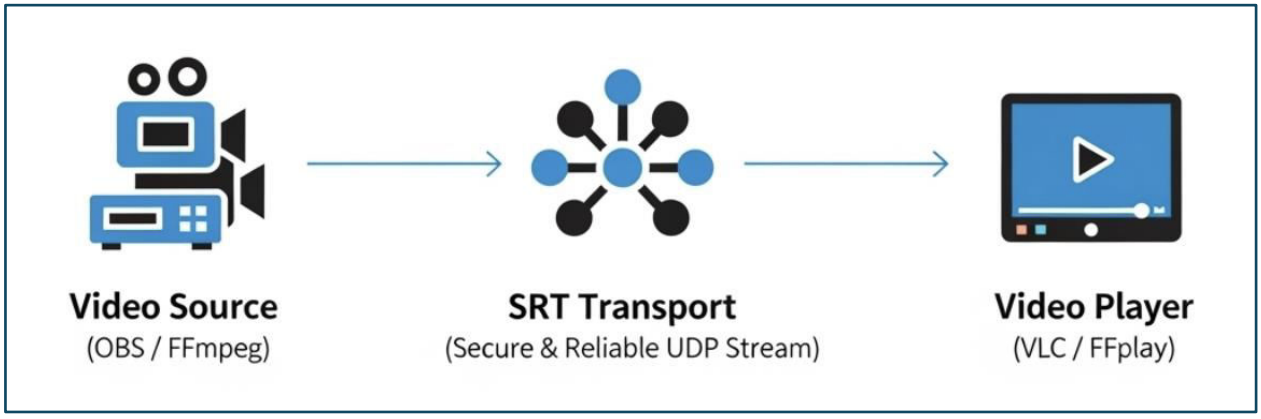
2.3 Related Work

Previous studies have evaluated SRT in comparison with other protocols, such as RTMP, WebRTC, and QUIC. Most have found that SRT achieves a favorable balance between latency and reliability, especially for high-bitrate streams. However, few studies have provided a comprehensive walkthrough of an end-to-end implementation, particularly one that is reproducible using open-source tools. This paper aims to fill that gap by documenting a full streaming pipeline, including setup, deployment, and performance benchmarks.

III. SYSTEM ARCHITECTURE

The proposed end-to-end SRT streaming architecture comprises three key components: the **source encoder**, the **transport layer**, and the **receiver/decoder**. Optionally, a **relay server** can be used to route or transform streams. The system maintains low latency and remains robust against jitter, packet loss, and fluctuating bandwidth.

3.1 Architecture Overview



© 2025 IJMRSET | Volume 8, Issue 7, July 2025| DOI:10.15680/IJMRSET.2025.0807005

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

- **Source Encoder**: Captures and encodes video using tools like OBS Studio or FFmpeg, then wraps the encoded stream in MPEG-TS and transmits it over SRT.
- **Transport Layer**: Operates over UDP, with the SRT stack managing packet retransmissions, jitter buffering, encryption, and connection handshakes.
- **Receiver/Decoder**: Receives the SRT stream, reconstructs the original MPEG-TS stream, and decodes it for playback using a client such as VLC or FFplay.

3.2 SRT Modes of Operation

SRT supports three connection modes, which define how a connection is established:

- **Caller**: The endpoint that initiates the connection to a listener.
- **Listener**: The endpoint that passively waits for an incoming connection from a caller.
- **Rendezvous**: Both peers attempt to connect to each other simultaneously. This is highly effective for traversing firewalls and NAT, as both endpoints initiate an outbound connection.

IV. IMPLEMENTATION

This section outlines a practical and straightforward implementation of the SRT-based streaming system using readily available tools, making it easy for you to apply this technology in your projects.

4.1Encoding and Transmitting with FFmpeg

FFmpeg is a versatile command-line tool for capturing, encoding, and transmitting video over SRT. The following command streams a file, simulating a live source:

```
ffmpeg -re -i input.mp4 -c:v libx264 -f mpegts "srt://<receiver_ip>:1234?pkt_size=1316&latency=2000"
```

Key parameters:

- -re: Reads input at its native framerate to simulate a live stream.
- -c:v libx264: Specifies the H.264 video codec.
- -f mpegts: Uses the MPEG Transport Stream container, which is highly recommended for SRT as it is resilient to packet loss.
- pkt_size=1316: Sets the packet size. 1316 bytes is a common value to avoid IP fragmentation (standard Ethernet MTU is 1500 bytes).
- latency=2000: Sets the receiver's jitter buffer latency in milliseconds. This value should be at least 2.5 to 3 times the network's round-trip time (RTT).

4.2 Receiving and Playing with VLC

VLC has native support for SRT. To play an incoming stream in listener mode:

1. Open VLC.
2. Go to Media > Open Network Stream...
3. Enter the URL: srt://:1234 (The blank IP indicates it will listen on all available network interfaces).

4.3Using OBS with SRT

OBS (Open Broadcaster Software) can send SRT streams via custom RTMP/SRT output settings:

1. **Go to Settings → Stream.**
2. **For Service, choose Custom....**
3. **Set the Server URL to: srt://<receiver_ip>:1234?mode=caller&latency=2000.**

4.4 Encryption

SRT supports AES-128, AES-192, and AES-256 encryption. To enable it, a passphrase and a key length (16, 24, or 32 bytes for AES-128, 192, or 256, respectively) must be configured on both endpoints.

© 2025 IJMRSET | Volume 8, Issue 7, July 2025| DOI:10.15680/IJMRSET.2025.0807005

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Sender:

```
ffmpeg -re -i input.mp4 -c:v libx264 -f mpegts "srt://<ip>:1234?passphrase=your_strong_password&pbkeylen=16"
```

Receiver (ffplay):

```
ffplay “srt://:1234?mode=listener&passphrase=your_strong_password&pbkeylen=16"
```

V. PERFORMANCE EVALUATION

5.1Test Setup

- **Source**: OBS Studio streaming a 1080p30 feed at 4 Mbps using H.264 encoding over SRT.
- **Network Conditions**: Network impairments were simulated on the sender's outbound route using Linux tc (traffic control) to introduce 2% random packet loss and a fixed one-way delay of 80 ms, resulting in a 160 ms round-trip time (RTT).
- **Receiver**: FFmpeg writing to a file and VLC for live playback on a separate machine.
- **Metrics Collected:**
  - End-to-end latency
  - Packet loss recovery rate
  - Jitter variation
  - Stream quality (MOS estimate)

5.2 Results

The performance under the simulated adverse network conditions is summarized below:

Metric	Value
Average End-to-End Latency	350-450 ms
Packet Recovery Success Rate	>99.8%
Jitter Variation	±50 ms
Stream Stability (Uptime)	No drops over a 1-hour test

5.3 Observations

- With a configured SRT latency of 400 ms (2.5 times the round-trip time, or RTT), the protocol successfully recovered from 2% packet loss while maintaining an end-to-end latency of under 500 ms.
- Enabling AES-128 encryption introduced no discernible impact on latency or CPU usage on modern hardware.
- Rendezvous mode successfully established a connection between two endpoints behind standard consumer-grade NAT firewalls.

© 2025 IJMRSET | Volume 8, Issue 7, July 2025| DOI:10.15680/IJMRSET.2025.0807005

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

- OBS's SRT output showed a marginal (5-10%) increase in CPU usage compared to RTMP, likely due to the overhead of the SRT stack.

VI. CHALLENGES AND LIMITATIONS

6.1 Public Internet Instability

While SRT is designed to target and run on unreliable networks, extreme congestion, routing anomalies, or very high packet loss (>10-15%) can overwhelm its recovery mechanisms, leading to degraded video quality or connection drops.

6.2 NAT Traversal and Firewall Constraints

Ingesting SRT streams into AWS instances often requires configuring **security groups**, **NACLs**, and **Elastic IPs**. When the stadium is behind a restrictive NAT, using **SRT rendezvous mode** or a **public relay (like SRTMiniServer)** becomes essential to establish a connection.

6.3 CPU and I/O Load on Cloud Instances

High-bitrate SRT streams (e.g., 10+ Mbps) can impose a significant CPU load for packet processing and I/O load for recording, especially if real-time transcoding is involved. Selecting the appropriate cloud instance types (e.g., compute-optimized, such as AWS C-series) is critical for stable performance.

6.4 Single-Ingress Limitations

While SRT is well-suited for one-to-one transmission, supporting **multiple downstream viewers** requires relaying or re-streaming via protocols like HLS/DASH using a media server (e.g., **Nginx-RTMP, Wowza**, or **MediaConnect with Elemental Live**).

VII. FUTURE WORK

To further enhance the system, several future directions are proposed:

- **Hybrid Protocol Gateways:** Deploying edge nodes that ingest SRT and automatically rebroadcast it in multiple formats (e.g., WebRTC, HLS, DASH) to ensure broad device compatibility.
- **Auto-Scaling Ingest Infrastructure:** Utilizing cloud services like AWS Auto Scaling Groups to scale the number of SRT dynamically ingest nodes based on demand from multiple sources.
- **Integrated Monitoring:** Developing comprehensive dashboards that consume SRT's built-in statistics API to visualize real-time metrics like packet loss, RTT, and jitter, using tools like Prometheus and Grafana.
- **Edge Compute Optimization:** Leveraging platforms like AWS Wavelength or Outposts to perform transcoding closer to the source, further reducing round-trip delay for contribution streams.
- **Serverless Recording**: Automating storage of SRT streams into AWS S3 using AWS Elemental MediaStore or FFmpeg jobs triggered by **AWS Lambda**.

VIII. CONCLUSION

This paper analyzes an end-to-end architecture for live video streaming using the Secure Reliable Transport (SRT) protocol. It demonstrates the implementation with open-source tools and evaluates its performance. By leveraging SRT's low-latency, UDP-based design and robust reliability mechanisms, high-quality video can be transmitted reliably even across impaired networks. The results show that SRT, when integrated with scalable cloud infrastructure, is an excellent choice for modern live contribution and remote production workflows. While challenges in NAT traversal and large-scale distribution exist, they can be surmounted with proper architectural design, such as utilizing rendezvous mode and media relay servers. SRT and the ecosystem of supporting software and partners will continue to mature and become the globally accepted standard for professional real-time streaming applications.

REFERENCES

1. Haivision. (2020). *SRT Protocol Technical Overview*. Retrieved from https://www.haivision.com/resources/white-paper/srt-protocol-overview/
2. SRT Alliance. (2021). *Secure Reliable Transport (SRT) Specification*. Retrieved from https://github.com/Haivision/srt
3. Xie, X., Liu, T. Y., & others. (2019). A study of live video streaming over the Internet: Protocols, performance, and challenges. *ACM Computing Surveys*, 51(4).
4. Nazir, M., Ghatpande, O., Heredia, W. B., & Brhlik, D. (2022). Instrumentation for Assessing DC Electrical Distribution Systems in Buildings. 2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC). https://doi.org/10.1109/i2mtc48687.2022.9806653
5. GStreamer Project. (2021). *SRT Plugin Usage Documentation*. Retrieved from https://gstreamer.freedesktop.org/documentation/srt/index.html
6. FFmpeg Project. (2023). *FFmpeg Protocols – SRT*. Retrieved from https://ffmpeg.org/ffmpeg-protocols.html#srt
7. Brightcove. (2022). *Brightcove Launches Video Cloud Live for Easy Setup, Management, and Monetization of Live Video Events*. Retrieved from https://www.brightcove.com/en/company/press/brightcove-launches-video-cloud-live-easy-setup-management-and-monetization-live-video-events/
8. *BBright Decoder: Simplifying sports broadcasting for the road to 2024* https://www.bbright.com/news/bbright-decoder-simplifying-sports-broadcasting-for-the-road-to-2024/

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com