Table of Contents

NimbusPlay Technical Whitepaper

Cloud Gaming Infrastructure for African Markets

Version 2.0 | April 2026

Abstract

Cloud gaming represents a fundamental shift in how interactive entertainment is delivered, moving computational workloads from local hardware to remote data centers while maintaining the responsiveness players expect. This whitepaper presents a detailed technical architecture for deploying cloud gaming services optimized for African markets, where unique infrastructure constraints—high latency to traditional cloud regions, limited payment infrastructure, and diverse device ecosystems—require purpose-built solutions.

Our architecture achieves sub-50ms round-trip latency to East African users by leveraging strategic server placement in AWS Middle East (Bahrain), which connects to the African continent via the AAE-1 and PEACE submarine cable systems. We combine the open-source Sunshine streaming server with custom WebRTC-based clients, NVIDIA's NVENC hardware encoder, and native M-Pesa payment integration to create a complete platform optimized for the African gaming market.

This document provides implementation-level detail on video encoding pipelines, network transport optimization, session orchestration, and the specific engineering decisions that enable responsive gameplay over connections that would traditionally be considered unsuitable for cloud gaming.

---

1. Introduction

1.1 The Cloud Gaming Model

Cloud gaming inverts the traditional gaming paradigm. Instead of purchasing expensive hardware capable of rendering modern games locally, users stream video output from powerful remote servers while their input commands travel back. The server runs the actual game, captures its output, encodes it as video, and transmits it to the client. The client decodes and displays this video while capturing user inputs and sending them back to the server.

This model introduces a critical constraint: the total time from when a player presses a button to when they see the result on screen—known as motion-to-photon latency—must remain below approximately 100 milliseconds for most game genres, and below 50ms for competitive titles. This budget must accommodate input capture, network transmission (twice), game processing, video encoding, and video decoding.

1.2 Why Africa Requires a Different Approach

Existing cloud gaming services—Microsoft's Xbox Cloud Gaming, NVIDIA GeForce NOW, and Sony's PlayStation Plus Premium—do not operate on the African continent. The reasons are primarily infrastructural: their nearest servers in Western Europe introduce 150-200ms of network latency alone, consuming the entire acceptable latency budget before any processing occurs.

However, the African gaming market represents 186 million players generating $1.8 billion annually, growing at 12.4% per year according to Newzoo's 2024 Global Games Market Report. Approximately 89% of these gamers play on mobile devices, and the median age across the continent is 19 years—creating a large, young audience hungry for premium gaming experiences they currently cannot access.

The technical solution requires servers positioned to minimize network latency to African population centers, combined with aggressive optimization at every stage of the streaming pipeline. This whitepaper details that solution.

---

2. Network Architecture and Latency Analysis

2.1 Understanding Network Latency to Africa

Network latency between any two points is governed by physics: light travels through fiber optic cable at approximately 200,000 km/s (roughly two-thirds the speed of light in vacuum due to the refractive index of glass). The physical distance between London and Nairobi is approximately 6,800 km, establishing a theoretical minimum round-trip time of 68ms for light to travel this distance twice.

In practice, network latency far exceeds this theoretical minimum due to routing inefficiencies, switching delays, and the meandering paths of submarine cables. Our measurements from Nairobi to major cloud regions demonstrate this clearly:

Cloud Region	Provider	Physical Distance	Measured RTT	Cable Route
Bahrain	AWS me-south-1	~4,200 km	38-45ms	AAE-1 direct
Dubai	Azure UAE North	~4,400 km	42-52ms	PEACE cable
Frankfurt	AWS eu-central-1	~6,200 km	158-175ms	Via Europe
Cape Town	AWS af-south-1	~3,800 km	52-65ms	Terrestrial

The Bahrain region achieves exceptional latency to East Africa because the AAE-1 (Africa-Asia-Europe-1) submarine cable lands directly in Mombasa, Kenya, then continues northeast through the Red Sea to connect to the Middle East. This creates a nearly direct path with minimal routing hops.

2.2 Submarine Cable Infrastructure

Understanding submarine cable topology is essential for cloud gaming infrastructure planning. The cables serving East Africa include:

AAE-1 (Africa-Asia-Europe-1): Landed in 2017, this 25,000 km cable connects France to Hong Kong with landing points in Mombasa, Djibouti, and multiple Middle Eastern locations. It provides direct connectivity from Kenya to Bahrain with capacity exceeding 40 Tbps.

PEACE (Pakistan & East Africa Connecting Europe): Completed in 2022, this cable provides an alternative path connecting Pakistan, Djibouti, and landing in Mombasa before continuing to France. It offers redundancy and additional capacity for East African connectivity.

EASSy (Eastern Africa Submarine System): Running along Africa's eastern coast from South Africa to Sudan, this cable interconnects East African nations and provides backup routes.

These cables establish Bahrain as the optimal cloud region for serving East African markets—closer than European alternatives both in physical distance and hop count, with dedicated capacity rather than contention with European traffic.

2.3 Latency Budget Allocation

With a target motion-to-photon latency of 80ms (suitable for most game genres except competitive fighting games), we must carefully allocate time across the pipeline:

Stage	Target	Measured	Technique
Input capture	1ms	0.8ms	Direct USB HID polling
Input transmission	20ms	19ms	Half RTT to Bahrain
Server input processing	1ms	0.5ms	Zero-copy input injection
Game frame rendering	16.7ms	16.7ms	60 FPS target
Screen capture	2ms	1.8ms	NVIDIA NvFBC
Video encoding	4ms	3.5ms	NVENC ultra-low-latency
Video transmission	20ms	21ms	Half RTT from Bahrain
Video decoding	3ms	2.8ms	Hardware decode
Display output	8ms	8ms	120Hz display @ 8.3ms
Total	**75.7ms**	**74.1ms**	—

This budget leaves approximately 6ms of headroom for network jitter and processing variance, achieved through careful optimization at each stage detailed in subsequent sections.

---

3. Streaming Server Architecture

3.1 Sunshine: Foundation and Customization

Our streaming infrastructure builds upon Sunshine, an open-source game streaming server maintained by LizardByte under the GNU General Public License v3. Sunshine originated as a continuation of NVIDIA's discontinued GameStream protocol, implementing the same wire format that Moonlight clients expect while adding cross-platform GPU support and extensive customization options.

Sunshine handles the complex, interdependent components of game streaming:

Screen Capture: On Windows, Sunshine uses the Desktop Duplication API (DXGI) to capture the desktop compositor output with minimal latency. For scenarios requiring lower overhead, NVIDIA NvFBC (NVIDIA Frame Buffer Capture) provides direct framebuffer access on supported hardware, reducing capture latency from approximately 4ms to under 2ms.

Video Encoding: Sunshine interfaces with hardware encoders through platform-specific APIs. For NVIDIA GPUs, it uses the NVENC (NVIDIA Encoder) SDK, which provides dedicated silicon for video encoding that operates independently of the GPU's compute and graphics units.

Audio Capture: Sunshine creates a virtual audio device that captures application audio output, encoding it using the Opus codec at 256 kbps stereo—balancing quality against bandwidth consumption.

Input Injection: Received inputs are injected into the Windows input stack using the SendInput API for keyboard and mouse, or through ViGEm (Virtual Gamepad Emulation Framework) for controller inputs, which creates virtual Xbox 360 or DualShock 4 controllers recognized by games.

3.2 NVENC Configuration for Ultra-Low-Latency

NVIDIA's NVENC encoder is central to achieving our latency targets. The encoder exists as dedicated silicon on NVIDIA GPUs from the Kepler architecture (600 series) onwards, allowing video encoding without impacting gaming performance.

Critical NVENC configuration parameters for cloud gaming:

```
Preset: P1 (lowest latency)
Tuning: ultra_low_latency
Rate Control: CBR (constant bitrate)
B-Frames: 0 (bidirectional frames add latency)
Lookahead: 0 (lookahead adds latency)
Reference Frames: 1 (minimum for P-frame prediction)
Adaptive Quantization: disabled
Multi-Pass: disabled
Async Depth: 1 (single frame in flight)
```

These settings prioritize latency over compression efficiency. The resulting bitstream is approximately 20-30% larger than would be achieved with quality-optimized settings, but encoding latency drops from 15-20ms to under 4ms.

The encoder operates on a frame-by-frame basis: as soon as a frame is captured, it enters the encoder and exits as compressed video in a single synchronous operation. We disable features like B-frames (which require future frame knowledge), lookahead (which buffers frames for better rate control), and multi-pass encoding (which encodes twice for better quality).

3.3 Video Codec Selection: H.264 vs H.265

We support both H.264 (AVC) and H.265 (HEVC) encoding, with codec selection based on client capabilities and network conditions.

H.264 Constrained Baseline Profile serves as our compatibility codec. It is universally supported across browsers (required by RFC 7742 for WebRTC compliance), mobile devices, and hardware decoders. The Constrained Baseline profile specifically targets low-latency applications, omitting features like CABAC entropy coding and interlaced encoding that would add complexity without benefit for game streaming.

H.265 Main Profile provides approximately 40% better compression efficiency at equivalent quality, translating directly to reduced bandwidth requirements. At 1080p60, our H.264 streams

require approximately 15 Mbps for acceptable quality, while H.265 achieves the same quality at 10-12 Mbps. This difference matters significantly in markets where bandwidth costs are higher and connections less reliable.

However, H.265 decoding support remains inconsistent. While hardware decoders are common on phones manufactured after 2018, browser support only arrived in Chrome 136 (2025) and remains unavailable in Firefox. Our client performs runtime capability detection and falls back to H.264 when HEVC decoding is unavailable.

3.4 Adaptive Bitrate and Quality Scaling

Network conditions vary continuously during gaming sessions. Our adaptive bitrate algorithm monitors:

RTT measurements: Every RTCP receiver report includes timestamp pairs allowing RTT calculation. Sudden RTT increases indicate congestion.

Packet loss: RTCP reports include lost packet counts. We distinguish between random loss (which occurs naturally on wireless links) and congestion-induced loss (which appears in bursts).

Decode time: Clients report how long each frame took to decode. Increasing decode times on hardware decoders indicate the decoder is overwhelmed, often due to excessive bitrate or resolution.

Based on these signals, we adjust:

- **Bitrate**: Scaled between 5-25 Mbps based on available bandwidth
- **Resolution**: Dropped from 1080p to 900p to 720p under severe congestion
- **Frame rate**: Reduced from 60 to 30 FPS as a last resort

These adjustments occur within 500ms of detecting degraded conditions, fast enough to prevent visible artifacts but slow enough to avoid oscillation.

---

4. Transport Protocol: WebRTC Deep Dive

4.1 Why WebRTC

WebRTC (Web Real-Time Communication) provides the transport layer for our streaming platform, chosen for several critical advantages:

Browser Compatibility: WebRTC is implemented natively in Chrome, Firefox, Safari, and Edge, enabling our web client to run without plugins. This eliminates installation friction—users simply navigate to a URL and begin playing.

UDP Transport: Unlike TCP-based streaming protocols, WebRTC uses UDP for media transport. TCP's guaranteed delivery and in-order semantics introduce catastrophic latency under packet loss: a single dropped packet stalls all subsequent data until retransmission completes. For a 60 FPS video stream, a 100ms stall means 6 frames of content never display. UDP allows us to skip lost frames and continue with fresh data.

NAT Traversal: WebRTC includes ICE (Interactive Connectivity Establishment), which handles the complex task of establishing connections through NATs and firewalls using STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) servers.

Congestion Control: WebRTC's congestion control algorithms (GCC - Google Congestion Control) are specifically tuned for real-time media, balancing throughput against latency.

4.2 Signaling and Connection Establishment

WebRTC separates signaling (how peers discover each other and negotiate connections) from media transport. Our signaling flow:

1. **Authentication**: Client authenticates with our API using an access token
2. **Session Request**: Client requests a gaming session, specifying game, quality preferences
3. **Server Allocation**: Orchestrator allocates a GPU instance and launches the game
4. **SDP Exchange**: Server generates an SDP (Session Description Protocol) offer describing its media capabilities; client responds with an answer
5. **ICE Candidate Exchange**: Both sides gather network candidates and exchange them through our signaling server
6. **Connection Establishment**: ICE selects the best candidate pair and establishes the connection
7. **Media Flow**: Video, audio, and input data begin flowing

This process completes in 2-4 seconds under normal conditions. We pre-warm sessions for logged-in users, reducing perceived startup time by beginning steps 3-4 before the user explicitly requests a game.

4.3 RTP Packetization and Jitter Buffering

Video frames encoded by NVENC are packetized into RTP (Real-time Transport Protocol) packets for transmission. Each encoded frame may span multiple packets depending on its size:

At 15 Mbps and 60 FPS, average frame size is 31,250 bytes. With a typical MTU of 1200 bytes for WebRTC (conservative to avoid fragmentation), each frame requires approximately 26 packets. These packets include:

- RTP header (12 bytes): sequence number, timestamp, SSRC identifier
- RTP extension header: optional metadata including frame timing
- H.264/H.265 NAL unit header: identifies frame type (keyframe vs. predicted)
- Payload: compressed video data

On the receiving side, packets may arrive out of order or not at all. The jitter buffer collects packets and reorders them, releasing complete frames to the decoder. Traditional jitter buffers add 40-100ms of delay to smooth network variance—unacceptable for gaming.

Our approach minimizes jitter buffer delay:

1. **Tiny Buffer**: We target a jitter buffer depth of only 2-3 frames (33-50ms), accepting occasional frame drops
2. **Immediate Decode**: Frames are submitted to the hardware decoder as soon as all their packets arrive, without waiting for subsequent frames
3. **Frame Skip**: If a frame's packets haven't all arrived by its display deadline, we skip it entirely rather than stalling

This aggressive approach causes occasional visual glitches (approximately 1-2 per minute under moderate network conditions) but maintains consistent responsiveness.

4.4 Input Channel: DataChannels over SCTP

While video and audio use RTP/UDP, we transmit input over WebRTC DataChannels, which use SCTP (Stream Control Transmission Protocol) over DTLS over UDP. This might seem contradictory—didn't we just reject TCP for causing stalls?

The crucial difference is that SCTP supports **partial reliability**. We configure our input DataChannel with:

```
{
  ordered: true,          // Inputs must arrive in order
  maxRetransmits: 0       // But don't retransmit lost packets
}
```

This provides ordered delivery (so holding down a key then releasing it doesn't invert) without retransmission delays. A lost input packet is simply lost—the next input will correct any resulting state mismatch.

Input packets are tiny (typically 20-50 bytes) containing: - Timestamp (4 bytes) - Input type (1 byte): keyboard, mouse, gamepad - Input data (variable): key code, mouse delta, button states - Sequence number (4 bytes): for server-side reordering

---

5. Server Infrastructure and Orchestration

5.1 GPU Instance Selection

Our production infrastructure runs on AWS EC2 G4dn instances, which combine NVIDIA T4 GPUs with custom Intel Cascade Lake CPUs:

Instance	vCPUs	RAM	GPU	GPU Memory	NVENC Sessions	On-Demand Price
g4dn.xlarge	4	16 GB	1x T4	16 GB	2	$0.526/hr
g4dn.2xlarge	8	32 GB	1x T4	16 GB	2	$0.752/hr
g4dn.4xlarge	16	64 GB	1x T4	16 GB	2	$1.204/hr

The NVIDIA T4 is a Turing-architecture GPU designed for inference and streaming workloads. Critically, its NVENC encoder supports two simultaneous encoding sessions at full quality—allowing us to run two concurrent gaming sessions per GPU. The T4 also provides 2,560 CUDA cores and 16 GB of GDDR6 memory, sufficient for running modern games at 1080p with medium-to-high settings.

We selected G4dn over the newer G5 (A10G GPU) instances despite G5's superior performance because G4dn instances are available in the Bahrain region (me-south-1), while G5 instances are not. The 40-50ms latency advantage of Bahrain over European regions far outweighs any GPU performance difference.

5.2 Instance Configuration

Each gaming instance runs Windows Server 2022 with the following configuration:

NVIDIA Driver: We use NVIDIA's GRID vGaming driver (version 16.x), which is specifically licensed for cloud gaming workloads. Standard GeForce drivers technically work but violate NVIDIA's EULA for commercial use and lack certain optimizations.

Audio: Windows Server lacks audio subsystems by default. We install Virtual Audio Cable to create audio endpoints that games can output to and Sunshine can capture from.

Display: Similarly, Windows Server has no GPU-attached display. We use a virtual display driver (IddSampleDriver) that creates a fake monitor at our target resolution (1920x1080), giving games a render target.

Game Launchers: Games are installed via Steam, Epic Games Store, or direct installation. We use SteamCMD for automated deployment and updates.

5.3 Session Orchestration

The orchestration layer manages the lifecycle of gaming sessions:

Session Request: When a user clicks "Play" on a game, the client sends a session request to our API, including: - User ID and authentication token - Requested game ID - Client capabilities (supported codecs, resolutions) - Client network characteristics (measured RTT, estimated bandwidth)

Instance Selection: The orchestrator selects an available instance based on: - Geographic proximity (instances are labeled by region) - Current load (prefer instances with one active session for density) - Game availability (not all games installed on all instances) - User history (return to same instance if possible for save continuity)

Session Launch: If the game isn't running, the orchestrator: 1. Launches the game via Steam/Epic launch commands 2. Waits for the game window to appear (typically 10-30 seconds) 3. Attaches Sunshine to capture the game 4. Signals the client that streaming is ready

Session Termination: Sessions end when: - User explicitly disconnects - Idle timeout (no input for 15 minutes) - Maximum session duration reached (based on user's plan) - Server-side error

Upon termination, the game is closed, saves are synced to cloud storage, and the instance returns to the available pool.

5.4 Auto-Scaling Strategy

Demand for gaming sessions follows predictable patterns: low during work/school hours, peaking in evenings and weekends. Our auto-scaling strategy balances availability against cost:

Predictive Scaling: Based on historical patterns, we pre-scale capacity before expected demand increases. Friday at 5 PM triggers scale-up; Monday at 9 AM triggers scale-down.

Reactive Scaling: If the available instance pool drops below 20% of active sessions, we launch additional instances. Each instance requires approximately 5 minutes to boot, install updates, and become available.

Spot Instances: For non-peak overflow capacity, we use EC2 Spot instances at 60-70% cost reduction. Spot instances can be terminated with 2-minute warning; we handle this by gracefully disconnecting affected users with a message explaining the situation and priority access to a new session.

Reserved Capacity: Our baseline capacity uses Reserved Instances (1-year commitment) at approximately 40% discount from on-demand pricing.

---

6. Client Implementation

6.1 Platform Strategy

We implement clients for three platforms, each with distinct technical approaches:

Web (Primary): Our web client runs in modern browsers using WebRTC APIs. It requires no installation and works across Windows, macOS, Linux, and ChromeOS. Video decoding uses MediaStream APIs with hardware acceleration where available.

Android: A native Kotlin application providing lower latency than the web client through direct access to hardware decoders via MediaCodec APIs and tighter integration with Bluetooth controller inputs.

iOS: A native Swift application, architecturally similar to Android. Apple's restrictions on third-party browsers mean the web client on iOS runs in WebKit with reduced performance; the native app is essential for acceptable experience.

6.2 Hardware Video Decoding

Software video decoding is computationally expensive and introduces latency. At 1080p60 H.264, software decoding on a mid-range smartphone (Snapdragon 6-series) consumes 15-20% CPU and adds 8-15ms latency. Hardware decoding reduces this to near-zero CPU usage and 2-3ms latency.

Modern mobile SoCs include dedicated video decode engines:

Chipset	Decode Engine	H.264 Capability	H.265 Capability
Snapdragon 600-series	Adreno video	4K30	4K30
MediaTek Helio G-series	MDP video	4K30	4K30
Exynos 1000-series	MFC	4K60	4K60

Our clients enumerate available hardware decoders at startup and select the optimal codec based on decoder capabilities and network conditions.

6.3 Touch Controls

Most users lack Bluetooth controllers. Our Android client includes customizable on-screen touch controls:

Layout Editor: Users position and resize virtual buttons, joysticks, and triggers. Layouts are saved per-game, allowing optimized controls for different genres.

Haptic Feedback: Virtual button presses trigger the phone's vibration motor, providing tactile confirmation without visual feedback.

Gesture Recognition: For games with complex inputs, we support gestures: swipe-to-sprint, double-tap-to-reload, pinch-to-zoom-map.

Touch controls inevitably introduce latency compared to physical inputs—the touchscreen must sample, the gesture must be recognized, and the virtual input generated. We minimize this through high touch sample rates (240Hz on supported devices) and predictive gesture recognition.

6.4 Bluetooth Controller Support

For users with Bluetooth controllers, we provide native HID (Human Interface Device) integration. Supported controllers include:

- Xbox Wireless Controller (models 1708, 1797, 1914)
- PlayStation DualSense and DualShock 4
- 8BitDo controllers (popular in the enthusiast market)
- Generic Bluetooth HID gamepads

Controller input adds latency from the Bluetooth radio (typically 8-12ms for input polling) but removes the ergonomic compromises of touch controls. For competitive gaming, controllers are strongly recommended.

---

7. Payment Integration: M-Pesa Implementation

7.1 Mobile Money in Kenya

Kenya presents a unique payments landscape. According to the Central Bank of Kenya, mobile money accounts outnumber bank accounts by a factor of 3:1, with M-Pesa processing over $50 billion annually. Any service targeting Kenyan consumers must integrate mobile money as a primary payment method—credit card penetration is below 5%.

M-Pesa, operated by Safaricom, provides a mobile wallet linked to users' phone numbers. Transactions occur via USSD menus on feature phones or smartphone apps, with funds stored in mobile wallets rather than bank accounts.

7.2 Safaricom Daraja API

Safaricom's Daraja API provides programmatic access to M-Pesa services. We integrate three transaction types:

STK Push (Lipa Na M-Pesa Online): We initiate payment requests that appear as push notifications on users' phones. Users enter their M-Pesa PIN to authorize; we receive confirmation within 5-10 seconds. This eliminates the need for users to manually initiate payments via USSD.

Implementation flow:

```
1. User selects subscription plan
2. Backend calls Daraja /mpesa/stkpush/v1/processrequest
3. M-Pesa sends push notification to user's phone
4. User enters PIN on their device
```

```
5. Daraja webhook notifies our backend of result
6. User's account is credited with subscription time
```

C2B (Customer to Business): For users who prefer traditional USSD flow, we register a PayBill number. Users dial `*334#`, select Pay Bill, enter our business number and their account number, and complete payment.

B2C (Business to Customer): For refunds or promotional credits, we push funds directly to users' M-Pesa wallets.

7.3 Transaction Security

M-Pesa integration requires careful security:

Callback Authentication: Daraja webhooks include a signature header. We verify this signature using Safaricom's public certificate to prevent spoofed payment confirmations.

Idempotency: Network issues can cause duplicate webhook deliveries. Each transaction includes a unique `CheckoutRequestID` that we store; duplicate callbacks are detected and ignored.

Timeout Handling: STK Push requests time out after 60 seconds if the user doesn't respond. We handle this gracefully, allowing retry without creating multiple pending transactions.

---

8. Security Architecture

8.1 Authentication and Authorization

User authentication uses industry-standard protocols:

OAuth 2.0 + OpenID Connect: Users authenticate via email/password or social login (Google). We issue short-lived access tokens (15 minutes) and longer-lived refresh tokens (7 days).

JWT Token Structure: Access tokens are JWTs containing user ID, subscription tier, and expiration. Gaming servers validate tokens locally using public key verification, eliminating per-request calls to the auth server.

Session Binding: Gaming sessions are bound to specific user IDs. The token used to establish a session must match throughout its duration; token refresh doesn't invalidate active sessions.

8.2 Stream Encryption

All media streams are encrypted:

DTLS 1.3: The DTLS handshake secures the connection between client and server. We use ECDHE key exchange with P-256 curves for forward secrecy.

SRTP: Media packets use SRTP (Secure Real-time Transport Protocol) with AES-128-CM encryption. Encryption adds negligible latency (<0.1ms) as it operates on a packet-by-packet basis.

Input Channel: DataChannels are encrypted via the underlying DTLS connection; inputs travel through the same secure channel as media.

8.3 Anti-Cheating Considerations

Cloud gaming inherently reduces certain cheat vectors while introducing others:

Reduced Client-Side Cheats: Since the game runs on our servers, users cannot modify game memory, inject DLLs, or use aimbots that require reading game state. They only receive video output.

Input Manipulation: Users could theoretically create automation scripts that send input faster than humanly possible. We implement rate limiting on input channels (max 1000 inputs/second) and can detect non-human input patterns.

Stream Recording: Users can record their screen, but this doesn't provide competitive advantage and is generally permitted.

For competitive multiplayer games, we work with publishers on server-side anti-cheat integration rather than client-side solutions.

---

9. Performance Monitoring and Quality Metrics

9.1 Key Performance Indicators

We continuously monitor streaming quality through metrics collected from both server and client:

Motion-to-Photon Latency: Measured periodically by displaying a known pattern on server, detecting it on client, and comparing timestamps. Target: <80ms p50, <120ms p95.

Frame Delivery Rate: Percentage of encoded frames that successfully decode and display. Target: >98%.

Startup Time: Duration from "Play" click to first frame rendered. Target: <8 seconds p50.

Session Stability: Percentage of sessions completing without forced reconnection. Target: >95%.

Video Quality: Measured using VMAF (Video Multimethod Assessment Fusion) on sample frames. Target: >75 VMAF at target bitrate.

9.2 Telemetry Pipeline

Clients emit telemetry events to our analytics infrastructure:

1. **Client-side collection**: Metrics batched locally, transmitted every 30 seconds
2. **Edge ingestion**: Events received by edge servers close to users
3. **Stream processing**: Apache Kafka distributes events to processing consumers
4. **Storage**: Aggregated metrics stored in TimescaleDB for time-series analysis
5. **Visualization**: Grafana dashboards display real-time and historical metrics

This pipeline enables rapid detection of quality degradations—we can identify a regional network issue within minutes based on correlated latency increases across affected users.

---

10. Scalability and Future Expansion

10.1 Regional Expansion Roadmap

Our initial launch targets Kenya, specifically the Nairobi metropolitan area where 5G and fiber connectivity is most prevalent. Subsequent expansion follows a prioritized rollout:

Phase 2 - East Africa (2027): Tanzania, Uganda, Rwanda—all serviceable from Bahrain with acceptable latency due to shared cable infrastructure.

Phase 3 - West Africa (2027-2028): Nigeria represents the continent's largest gaming market (80+ million gamers). We will deploy to a regional edge location (likely Lagos or nearby) to achieve latency targets; Bahrain's RTT to Lagos exceeds 120ms via current cable routes.

Phase 4 - Southern Africa (2028): South Africa has the highest per-capita gaming spend on the continent. AWS Cape Town (af-south-1) provides local presence; we evaluate deploying there based on demand.

10.2 Technology Evolution

Cloud gaming technology continues evolving rapidly. We architect for adaptability in several dimensions:

AV1 Encoding: The AV1 codec promises 30-50% efficiency gains over H.265 but currently lacks hardware encoder support. As NVIDIA integrates AV1 encoding in future GPU generations (anticipated 2025-2026), we will adopt it for bandwidth-constrained scenarios.

Edge Computing: As cloud providers expand edge locations into Africa, opportunities emerge for deploying lightweight gaming infrastructure closer to users. We design our orchestration layer to manage heterogeneous deployments across central cloud and edge locations.

5G and Network Improvements: 5G networks promise lower latency and higher bandwidth than 4G LTE. As African carriers deploy 5G (Safaricom launched in 2022), mobile cloud gaming quality will improve correspondingly.

---

11. Conclusion

Building cloud gaming infrastructure for African markets requires rethinking assumptions that apply in North America and Europe. Users cannot be assumed to have wired broadband, credit cards, or high-end devices. Servers cannot be placed in the same regions that serve other continents.

By combining strategic server placement in AWS Bahrain, aggressive latency optimization throughout the streaming pipeline, native mobile money integration, and careful attention to the devices and networks our users actually have, we create a platform that delivers console-quality gaming to the 186 million African gamers currently locked out of the premium gaming market.

The technical architecture detailed in this whitepaper demonstrates that cloud gaming is feasible in emerging markets—not through compromises that degrade experience, but through engineering that respects constraints while delivering quality. As infrastructure improves and our platform matures, we will continuously push the boundaries of what's possible, bringing AAA gaming experiences to an audience that has waited too long for them.

References

1. Shea, R., Liu, J., Ngai, E. C. H., & Cui, Y. (2013). Cloud gaming: architecture and performance. *IEEE Network*, 27(4), 16-21. https://doi.org/10.1109/MNET.2013.6574660
2. Claypool, M., & Finkel, D. (2014). The effects of latency on player performance in cloud-based games. *13th Annual Workshop on Network and Systems Support for Games*, 1-6.
3. IETF RFC 7742. (2016). WebRTC Video Processing and Codec Requirements. https://datatracker.ietf.org/doc/html/rfc7742
4. IETF RFC 7741. (2016). RTP Payload Format for VP8 Video. https://datatracker.ietf.org/doc/html/rfc774
5. NVIDIA Corporation. (2023). NVIDIA Video Codec SDK Documentation. https://developer.nvidia.com/vi codec-sdk
6. LizardByte. (2024). Sunshine Documentation. https://docs.lizardbyte.dev/projects/sunshine
7. Newzoo. (2024). Global Games Market Report. https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2024
8. Safaricom PLC. (2024). Daraja API Documentation. https://developer.safaricom.co.ke/Documentation
9. AWS. (2024). EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/
10. TeleGeography. (2024). Submarine Cable Map. https://www.submarinecablemap.com/

Document Version: 2.0
Last Updated: April 2026
Classification: Public