NuStream: A Lightweight, Predictable Deterministic Real-Time Media Pipeline with GStreamer-like API for **NuttX** RTOS

Namdak Tonpa, Skynet

July 14, 2026

Abstract

We present a deterministic, hard real-time media processing framework designed specifically for the NuttX RTOS. The framework provides a GStreamer-inspired pipeline architecture while eliminating non-deterministic elements such as dynamic allocation, GObject, and complex caps negotiation. It prioritizes strict timing guarantees, zero-copy processing, and compile-time predictability suitable for safety-critical and high-performance embedded applications.

1 Introduction

Modern embedded systems increasingly require sophisticated multimedia capabilities — real-time video encoding, audio processing with echo cancellation, and low-latency streaming — while operating under strict deterministic constraints.

Traditional multimedia frameworks like GStreamer excel in flexibility on Linux systems but introduce significant sources of non-determinism: dynamic plugin loading, runtime memory allocation, complex object systems (GObject), and unpredictable negotiation phases. These characteristics make them unsuitable for hard real-time environments such as automotive, industrial control, medical devices, or mission-critical communication systems running on microcontrollers.

The primary motivation for this work is to create a mathematically clean, auditable, and highly predictable media pipeline for NuttX — an RTOS known for its POSIX compatibility and small footprint. We aim for critical levels of performance and determinism: every operation must have bounded execution time, fixed memory usage, and verifiable worst-case latency.

This framework, tentatively named **NuStream**, follows the philosophy of TRON relative to POSIX: it preserves the essential concepts of a media pipeline while providing a much stricter, more lightweight, and real-time-oriented implementation.

2 NuttX Deterministic Real-Time Operation

The framework must satisfy the following requirements:

1. **Hard Real-Time Guarantees**: All critical paths must have statically analyzable worst-case execution time (WCET).
2. **Zero Runtime Allocation**: No dynamic memory allocation (`malloc`/`free`) in the data plane.
3. **Fixed Memory Footprint**: All buffers and structures allocated at compile time or initialization.
4. **Static Pipeline Topology**: Pipeline graph defined at compile time (no dynamic pad linking).
5. **Deterministic Scheduling**: Integration with NuttX priority-based or deadline scheduling.
6. **Minimal Dependencies**: No GLib, no GObject. Pure C99 with NuttX APIs.
7. **Zero-Copy Where Possible**: Data flows through buffer references rather than copies.
8. **Auditable & Verifiable**: Code structure must support formal methods and timing analysis.

3 Real-Time Facilities as Object Attributes

Every core object in **NuStream** exposes its real-time characteristics as explicit attributes. This design allows static analysis tools and the scheduler to reason about timing properties.

```
typedef struct {
    const char* name;
    uint32_t wcet_us;           // Worst-Case Execution Time in microseconds
    uint32_t period_us;         // Required invocation period
    uint8_t  priority;          // NuttX task priority
    uint32_t deadline_us;       // Relative deadline
    bool     is_preemptible;
    bool     uses_dma;          // Zero-copy DMA capable
} rt_attributes_t;
```

All elements inherit or declare these attributes at initialization.

4 Architecture

The **NuStream** framework follows a static pipeline model with explicit real-time contracts on every object.

4.1 Core Objects (C99 Structs)

```
/* Real-time attributes (attached to every object) */
typedef struct {
    const char* name;
    uint32_t    wcet_us;          /* Worst-Case Execution Time */
    uint32_t    period_us;        /* Recommended invocation period */
    uint32_t    deadline_us;      /* Relative deadline from release */
    uint8_t     priority;         /* NuttX task priority */
    bool        zero_copy;        /* DMA / reference passing capable */
    bool        is_critical;      /* Cannot be preempted in data path */
} rt_attrs_t;

/* Buffer - fixed size, reference-counted where needed */
typedef struct rt_buffer {
    uint8_t*    data;
    size_t      size;
    size_t      capacity;
    uint64_t    pts_us;           /* Presentation timestamp */
    uint32_t    flags;
    uint32_t    refcount;
    void*       priv;             /* Element-specific data */
} rt_buffer_t;

/* Pad - connection point (simplified, static) */
typedef struct rt_pad {
    const char*       name;
    bool              is_source;
    rt_element_t*     element;
    struct rt_pad*    peer;
    rt_buffer_t*      buffer_pool; /* Fixed pool for this pad */
} rt_pad_t;

/* Element - core processing unit (GstElement equivalent) */
typedef struct rt_element {
    const char*      name;
    rt_attrs_t       rt_attrs;

    rt_pad_t**       src_pads;
    rt_pad_t**       sink_pads;
    size_t           num_src_pads;
```

```
    size_t          num_sink_pads;

    /* Lifecycle */
    int (*init)(struct rt_element* self, void* config);
    int (*start)(struct rt_element* self);
    int (*stop)(struct rt_element* self);
    int (*deinit)(struct rt_element* self);

    /* Data flow - deterministic core */
    int (*chain)(struct rt_element* self, rt_pad_t* pad, rt_buffer_t* buf);

    void*           priv;       /* Private data */
} rt_element_t;

/* Pipeline - static container */
typedef struct rt_pipeline {
    rt_element_t**  elements;
    size_t          num_elements;
    rt_buffer_t*    shared_pool;  /* Global fixed buffer pool */
    bool            running;
} rt_pipeline_t;
```

4.2 Full API

```
/* Pipeline Management */
rt_pipeline_t* rt_pipeline_create(void);
int rt_pipeline_destroy(rt_pipeline_t* pipe);
int rt_pipeline_add(rt_pipeline_t* pipe, rt_element_t* elem);
int rt_pipeline_link_pads(rt_element_t* src, const char* srcpad,
                          rt_element_t* sink, const char* sinkpad);
int rt_pipeline_start(rt_pipeline_t* pipe);
int rt_pipeline_stop(rt_pipeline_t* pipe);

/* Element Management */
rt_element_t* rt_element_new(const char* name, size_t srcpads, size_t sinkpads);
int rt_element_set_attrs(rt_element_t* elem, const rt_attrs_t* attrs);

/* Standard Elements (Factory) */
rt_element_t* rt_element_camera_create(void);      /* Source */
rt_element_t* rt_element_h264_encoder_create(void);
rt_element_t* rt_element_opus_encoder_create(void);
rt_element_t* rt_element_rtp_pay_create(void);
rt_element_t* rt_element_network_sink_create(void); /* UDP/RTP */
rt_element_t* rt_element_display_sink_create(void);
```

```
/* Buffer Management */
rt_buffer_t*  rt_buffer_new(size_t capacity);
void          rt_buffer_ref(rt_buffer_t* buf);
void          rt_buffer_unref(rt_buffer_t* buf);
```

5 Potential Market Applications for NuStream on NuttX

1. **Automotive & ADAS (Advanced Driver Assistance Systems)**
   - Real-time camera processing, object detection preprocessing, surround-view stitching, and low-latency video encoding.
   - Driver monitoring systems (DMS) with infrared cameras and audio alerts.
   - *Why it fits:* Strict safety standards (ISO 26262), deterministic timing for vision pipelines, and integration with NuttX's real-time scheduler.
2. **Industrial Automation & Machine Vision**
   - High-speed inspection systems on production lines (defect detection via cameras).
   - Robotic vision with real-time video feedback.
   - Predictive maintenance using multi-sensor (visual + audio) fusion.
   - *Why it fits:* Harsh environments, need for predictable latency, and long-term reliability without Linux overhead.
3. **Medical Devices & Healthcare**
   - Ultrasound, endoscopy, or surgical robotics video pipelines.
   - Patient monitoring with real-time audio/video streaming (telemedicine).
   - Wearable or portable diagnostic devices requiring low power and guaranteed timing.
   - *Why it fits:* Regulatory requirements (IEC 62304) favor auditable, deterministic code.
4. **Drones / UAVs & Aerospace**
   - Onboard video encoding and low-latency streaming for FPV or surveillance.
   - Payload processing (multispectral cameras, thermal imaging).
   - Real-time sensor fusion for autonomous navigation.
   - *Why it fits:* Weight/power constraints + hard real-time needs for flight safety.
5. **Mission-Critical Communications**
   - Secure real-time video/audio over tactical radios or satellite links.
   - Body-worn or vehicle-mounted cameras for law enforcement/defense.
   - *Why it fits:* Zero-copy + deterministic behavior reduces jitter and improves reliability under bandwidth constraints.

6. **IoT Edge Devices & Smart Infrastructure**
    - Smart city cameras with onboard analytics.
    - Industrial IoT gateways handling multiple video/audio streams.
    - Environmental monitoring stations with multimedia logging.
7. **Consumer / Prosumer Embedded Products**
    - High-end security cameras or action cameras running a lightweight RTOS.
    - Audio processing devices (e.g., conference systems with echo cancellation).
    - Robotics hobbyist/pro projects needing better real-time media than Linux.

6 Conclusion

NuStream delivers a deterministic, lightweight, and auditable media pipeline framework specifically engineered for the constraints of hard real-time embedded systems running on NuttX. By eliminating dynamic memory allocation, complex object systems, and runtime negotiation—hallmarks of traditional frameworks like GStreamer—NuStream achieves strict timing predictability, fixed memory footprints, and zero-copy data paths essential for safety-critical applications.

The architecture rests on three foundational principles:

- **Static everything**: Pipeline topology, buffer pools, and real-time contracts are defined at compile time or initialization.
- **Explicit real-time attributes**: Every element exposes WCET, deadlines, priorities, and DMA capabilities, enabling static analysis and integration with NuttX schedulers.
- **Minimalism with power**: A clean C99 API that preserves the expressive pipeline model while remaining suitable for microcontrollers and formal verification.

This design opens the door to sophisticated multimedia processing—real-time video encoding, multi-channel audio with echo cancellation, low-latency sensor fusion, and secure streaming—on resource-constrained devices without compromising determinism. Potential application domains include automotive driver assistance systems, industrial vision, medical imaging, UAV payload processing, and mission-critical communications.

Future work will focus on formal WCET analysis integration, additional high-performance elements (e.g., GPU/DSP offloading where available), comprehensive benchmarking on NuttX targets, and tools for automated pipeline verification and code generation.

By providing a media framework for NuttX, **NuStream** demonstrates that flexibility and predictability are not mutually exclusive. It brings modern media pipeline concepts to the embedded real-time domain while maintaining the auditable simplicity required for safety-critical certification.