Edge-RT: OS Support for Controlled Latency in the Multi-Tenant, Real-Time Edge

Wenyuan Shao, Bite Ye, Huachuan Wang, Gabriel Parmer and Yuxin Ren
{shaow, bitye, hcwang, gparmer, ryx}@gwu.edu
Interacting with the Cloud

Main purpose of interacting with cloud:

1. Offload computations to the cloud.
2. Aggregate data from multiple clients.
Interacting with the Cloud

Main purpose of interacting with cloud:
1. **Offload computations to the cloud.**
2. Aggregate data from multiple clients.
Interacting with the Cloud

Main purpose of interacting with cloud:

1. Offload computations to the cloud.
2. Aggregate data from multiple clients.
Interacting with the Cloud

Main purpose of interacting with cloud:
1. Offload computations to the cloud.
2. Aggregate data from multiple clients.

Problem:
For both examples, multiple clients require real-time responses from the cloud.
Interacting with the Cloud

Main purpose of interacting with cloud
1. Offload computations to the cloud.
2. Aggregate data from multiple clients.

Problem:
For both examples, multiple clients require real-time responses from the cloud.

Core Challenges:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.
Interacting with the Edge

What is Edge cloud?

Why Edge cloud?
Interacting with Edge

What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?
Interacting with Edge

What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Large-scale Cloud

Edge Cloud

Autonomous Vehicles (AVs)
Cameras
Sensors

Edge RTT: 1ms

Internet
What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.
Interacting with the Edge

What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.

Edge-Cloud Solutions:
1. 1ms RTT with 5G.
2. Using local bandwidth.

Autonomous Vehicles (AVs)  Cameras  Sensors
What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.

Edge-Cloud Solutions:
1. 1ms RTT with 5G.
2. Using local bandwidth.

Interacting with the Edge

Large-scale Cloud

Edge Cloud

Edge RTT: 1ms

Autonomous Vehicles (AVs) Cameras Sensors

Internet
Interacting with the Edge

What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Core Challenges of the Edge:
1. Multi-tenancy.
2. High density dynamic workload.
3. Deadline-aware.

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.

Edge-Cloud Solutions:
1. 1ms RTT with 5G.
2. Using local bandwidth.

Edge RTT: 1ms

Autonomous Vehicles (AVs)  Cameras  Sensors
Multi-tenancy on the Edge

- **Network slicing → tenants sharing server capacity.**
- Client isolation.
- Tenant isolation.

Cloud RTT: 20-50ms

Edge RTT: 1ms

Large-scale Cloud

Edge Cloud

Internet

Robot A
Robot B
AV A
AV B
Multi-tenancy on the Edge

- Network slicing → tenants sharing server capacity.
- **Client isolation.**
- Tenant isolation.
Multi-tenancy on the Edge

- Network slicing → tenants sharing server capacity.
- Client isolation.
- **Tenant isolation.**
Multi-tenancy on the Edge

- Network slicing → tenants sharing server capacity.
- Client isolation.
- **Tenant isolation.**
What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.

Edge-Cloud Solutions:
1. 1ms RTT with 5G.
2. Using local bandwidth.

Core Challenges of the Edge:
1. Multi-tenancy.
2. High density dynamic workload.
3. Deadline-aware.

Edge RTT: 1ms

Autonomous Vehicles (AVs)  Cameras  Sensors
High Density Workload

Large-scale Cloud

Edge Cloud

Edge RTT: 1ms

Robot A
Robot B
AV A
AV B
High Density Workload

Challenging for the Edge to achieve line-rate response time and support high density workloads with **significantly less** hardware resources.
What is Edge cloud?
1. Compute assets co-located with client devices.
2. Scale: few servers up-to a rack of servers.

Why Edge cloud?

Challenges of the Cloud:
1. Huge and unpredictable latency.
2. WAN bandwidth is running out.

Edge-Cloud Solutions:
1. 1ms RTT with 5G.
2. Using local bandwidth.

Core Challenges of the Edge:
1. Multi-tenancy.
2. High density dynamic workload.
3. Deadline-aware.

Edge RTT: 1ms
Deadline-aware Scheduling

Why challenging?
1. Chains of computations span cores.
2. Must meet end-to-end deadline of packets.
Deadline-aware Scheduling

Why challenging?
1. Chains of computations span cores.
2. Must meet end-to-end deadline of packets.
Deadline-aware Scheduling

Why challenging?
1. Chains of computations span cores.
2. Needs to meet end-to-end deadline of packets.
Existing Technologies

1. Optimize throughput by reducing system overhead.
   • Kernel bypass networking.
   • In-kernel Sandbox.

2. Thread-based deadline-aware scheduling.
Existing Technologies

1. Optimize throughput by reducing system overhead.
   • Kernel bypass networking.
   • In-kernel Sandbox.

2. Thread-based deadline-aware scheduling.
Kernel bypass Networking

Data Plane Development Kit (DPDK)

Without DPDK

With DPDK

User Space

Application

Kernel Space

Linux Kernel

Network Hardware

Server hardware

NIC
Kernel bypass Networking

Data Plane Development Kit (DPDK)
- Achieving isolation between tenants.
  1. DPDK + Single Root I/O Virtualization (SR-IOV)
  2. DPDK + Open vSwitch (OVS)
Kernel bypass Networking

Data Plane Development Kit (DPDK)

- Achieving isolation between tenants.
  1. DPDK + Single Root I/O Virtualization (SR-IOV)
     - Multi-tenancy: ✓
     - Scalability: ❌
     - Deadline-aware: ❌
  2. DPDK + Open vSwitch (OVS)
Kernel bypass Networking

Data Plane Development Kit (DPDK)

- Achieving isolation between tenants.
  1. DPDK + Single Root I/O Virtualization (SR-IOV)
     - Multi-tenancy:
     - Scalability:
     - Deadline-aware:
  2. DPDK + Open vSwitch (OVS)
Kernel bypass Networking

Data Plane Development Kit (DPDK)

- Achieving isolation between tenants.
  1. DPDK + Single Root I/O Virtualization (SR-IOV)
     - Multi-tenancy: ✓
     - Scalability: ×
     - Deadline-aware: ×
  2. DPDK + Open vSwitch (OVS)
     - Multi-tenancy: ✓
     - Scalability: ×
     - Deadline-aware: ×
In-kernel Sandbox

extended Berkeley Packet Filter (eBPF)
In-kernel Sandbox

extended Berkeley Packet Filter (eBPF)

• Evaluation:
  
  Multi-tenancy: ✗
  Deadline-aware: ✗
Existing Technologies

1. Optimize throughput by reducing system overhead.
   - Kernel bypass networking.
   - In-kernel Sandbox.

2. Thread-based deadline-aware scheduling.
Thread-based Scheduling.

**SCHED_DEADLINE**
- Multi-tenancy: ✔️
- Scalability: ❌
- Dynamic workload: ❌

Periodic tasks with known execution time

- Arrive time
- Deadline
- Period

Time

---

**THE GEORGE WASHINGTON UNIVERSITY**

WASHINGTON, DC
Thread-based Scheduling.

**SCHED DEADLINE**

- Multi-tenancy: ✅
- Scalability: ✗
- Dynamic workload: ✗

**Periodic tasks with known execution time**

- Arrive time
- Deadline
- Period
- Time

\[ T_1 \]
Thread-based Scheduling.

**SCHED_DEADLINE**
- Multi-tenancy: ✓
- Scalability: ×
- Dynamic workload: ×

Aperiodic tasks with unknown execution time

\[ T_1 \]

Arrive time | Deadline | Period | Deadline

Time
Thread-based Scheduling.

**SCHED_DEADLINE**
- Multi-tenancy: ✓
- Scalability: ✗
- Dynamic workload: ✗

Core Problem:
Schedule *tasks* instead of *packets*.

Aperiodic tasks with unknown execution time

- Arrive time
- Deadline
- Period
- Deadline
- Time
Edge-RT

1. Background:
   • Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP),
     2. DPDK-based fast networking,
     3. Fast memory movement between FWPs.

2. End-to-end deadline packet scheduling.
Edge-RT

1. Background:
   - Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP).
     2. DPDK-based fast networking.
     3. Fast memory movement between FWPs.

2. End-to-end deadline packet scheduling.
Edge-RT

1. Background:
   • Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP).
     2. DPDK-based fast networking.
     3. Fast memory movement between FWPs.

2. End-to-end deadline packet scheduling.
1. Background:
   • Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP).
     2. DPDK-based fast networking.
     3. Fast memory movement between FWPs.

2. End-to-end deadline packet scheduling.
Edge-RT

1. Background:
   • Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP).
     2. DPDK-based fast networking.
     3. Fast memory movement between FWPs.

2. End-to-end deadline packet scheduling.
Edge-RT

1. Background:
   - Built upon EdgeOS:
     1. Light-weighted isolation abstraction: feather weight process (FWP).
     2. DPDK-based fast networking.
     3. Fast memory movement between FWPs.

2. Goal: End-to-end deadline scheduling of packets.
End-to-end Packet Scheduling

1. Deadline inheritance.
   - Thread inherit deadline from the packet.

2. Schedule chains of computations.
End-to-end Packet Scheduling

1. **Deadline inheritance.**
   - Thread inherit deadline from the packet.

2. Schedule chains of computations.
End-to-end Packet Scheduling

1. Deadline inheritance.
   • Thread inherit deadline from the packet.

2. Schedule chains of computations.
End-to-end Packet Scheduling

1. Deadline inheritance.
   - Thread inherit deadline from the packet.

2. Schedule chains of computations.
End-to-end Packet Scheduling

1. Deadline inheritance.
   - Thread inherit deadline from the packet.

2. Schedule chains of computations.
End-to-end Packet Scheduling

1. Deadline inheritance.
   • Thread inherit deadline from the packet.

2. Schedule chains of computations.
   • Same deadline for all stages.
   • Makes no assumption based on WCET.
End-to-end Packet Scheduling

1. Deadline inheritance.
   • Thread inherit deadline from the packet.

2. Schedule chains of computations.
   • Same deadline for all stages.
   • Makes no assumption based on WCET.
Remaining Challenges

1. Frequent thread activation and deactivation.
2. Frequent (inter-core) event notification.
3. EDF policy overheads for frequent activation.
Optimization

1. Frequent thread activation and deactivation.
2. Frequent (inter-core) event notification.
3. EDF policy overheads for frequent activation.

1. Deadline-aware batching.
2. Periodic Event Notification.
3. Constant-Time EDF.
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.

FWP$_1$

FWP$_2$

$D_1 = 20$

$D_2 = 10$

$20$ $15$ $12$ $10$
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.

Scheduled earlier than they should be
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{\text{batch}}$).
3. Batching with controlled size.
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.

$D_1 = 50$

$D_2 = 8$

$(P_i, d_i), d \in [d_i - \Delta_{batch}, d_i]$
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{\text{batch}}$).
3. Batching with controlled size.

Caused by System overhead
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.

Batching reduces system overhead
Deadline-aware batching

1. Deadline inheritance of multiple packets.
2. Priority inversions ($\Delta_{batch}$).
3. Batching with controlled size.

Caused by priority inversion
Optimization

1. Frequent thread activation and deactivation.
2. Frequent (inter-core) event notification.
3. EDF policy overheads for frequent activation.

1. Deadline-aware batching.
2. Periodic Event Notification.
3. Constant-Time EDF.
Optimization

1. Frequent thread activation and deactivation.
2. Frequent (inter-core) event notification.
3. EDF policy overheads for frequent activation.

1. Deadline-aware batching.
2. Periodic Event Notification.
3. Constant-Time EDF.
Constant-time EDF Scheduling (CT-EDF)

1. Quantize time into fixed quanta.
2. Using array to track each quanta.
3. Priority inversion ($\Delta_{\text{window}}$).
Constant-time EDF Scheduling (CT-EDF)

1. Quantize time into fixed quanta.
2. Using array to track each quanta.
3. Priority inversion ($\Delta_{\text{window}}$).
Constant-time EDF Scheduling (CT-EDF)

1. Quantize time into fixed quanta.
2. Using array to track each quanta.
3. Priority inversion ($\Delta_{\text{window}}$).
Constant-time EDF Scheduling (CT-EDF)

1. Quantize time into fixed quanta.
2. Using array to track each quanta.
3. Priority inversion ($\Delta_{\text{window}}$).

$$(P, d), d \in [d - \Delta_{\text{window}}, d]$$
Priority Inversions

Edge-RT achieves line-rate throughput by creating *bounded* deadline inversions:
1. Batching ($\Delta_{batch}$).
2. Periodic event notification ($\Delta_{timer}$).
3. CT-EDF ($\Delta_{window}$).
Priority Inversions

Edge-RT achieves line-rate throughput by creating **bounded** deadline inversions:
1. Batching ($\Delta_{batch}$).
2. Periodic event notification ($\Delta_{timer}$).
3. CT-EDF ($\Delta_{window}$).

\[
d \in [d_i - \max(\Delta_{batch}, \Delta_{window}), d_i]
\]
System Architecture

User-level

Kernel

Physical

Hardware

Composite κ-kernel

CT-EDF Scheduler

FWPs

Flow table

DPDK

NIC

NIC rx

NIC tx

DPDK rx

DPDK tx

MMA
System Architecture

- **Kernel**
  - Composite \( \mu \)-kernel
  - Notification queue
  - CT-EDF Scheduler

- **User-level**
  - DPDK
  - FWP
  - FWP

- **Physical**
  - NIC
  - NIC
  - NIC rx
  - NIC tx

- **Hardware**

- **Composites**
  - kernel
  - Hardware
  - NIC

- **Notations**
  - NIC rx
  - NIC tx
  - DPDK rx
  - DPDK tx
  - FWP
  - flow table
  - notification queue
Evaluation

Experiment setup:

- Power Edge R740 servers.
- Two socket Intel(R) Xeon(R) Platinum 8160 CPUs @2.10GHz each with 24 cores.
- Intel X710 for10GbE NIC.
- Compare Linux, EdgeOS and Edge-RT
Evaluation

Workload description:
1. Bimodal workloads.
2. Light computation $WCET = 40\mu s, \text{deadline} = 10\text{ms}$, (Kalman filtering)
3. Heavy computation $WCET = 5\text{ms}, \text{deadline} = 500\text{ms}$, (ML inference)
4. EdgeRT $\Delta_{batch} = 8\text{ms}, \Delta_{timer} = 250\mu s$.
5. 480 clients/chains, chain length 4, 1920 FWPs in total.
Utilization Sensitivity

Fig. 1. The behavior of light tasks with increasing utilization

Fig. 2. The behavior of heavy tasks with increasing utilization
Utilization Sensitivity

Fig. 1. The behavior of light tasks with increasing utilization

Fig. 2. The behavior of heavy tasks with increasing utilization
Utilization Sensitivity

Fig. 1. The behavior of light tasks with increasing utilization

Fig. 2. The behavior of heavy tasks with increasing utilization
Utilization Sensitivity

Fig. 1. The behavior of **light** tasks with increasing utilization

Fig. 2. The behavior of **heavy** tasks with increasing utilization
Utilization Sensitivity

Fig. 1. The behavior of **light** tasks with increasing utilization

Fig. 2. The behavior of **heavy** tasks with increasing utilization
Conclusion

Edge-RT provides a solution for the **multi-tenant, dense, latency-sensitive** edge cloud.

- **Multi-tenant**: Strong FWP-based isolation.
- **Density**: throughput-centric implementation.
- **Deadlines**:
  - FWP inheritance of packet deadlines,
  - Bounded deadline inversions,
  - End-to-end packet deadline scheduling.
Conclusion

Edge-RT provides a solution for the **multi-tenant, dense, latency-sensitive** edge cloud.

- **Multi-tenant**: Strong FWP-based isolation.
- **Density**: throughput-centric implementation.
- **Deadlines**:
  - FWP inheritance of packet deadlines,
  - Bounded deadline inversions,
  - End-to-end packet deadline scheduling.

**Edge-RT: strong foundation for the real-time edge**
Questions and Comments?

? || /* */
## Existing Technologies

- A summary of edge-cloud configurations

<table>
<thead>
<tr>
<th>Edge Configurations</th>
<th>Deadline-aware</th>
<th>Preemptivity</th>
<th>Client Isolation</th>
<th>Computation Chain</th>
<th>Dynamic Workloads</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFS</td>
<td>not deadline-aware</td>
<td>preemptive</td>
<td>process-based</td>
<td>per-client chain</td>
<td>supported</td>
<td>&gt; 2000</td>
</tr>
<tr>
<td>DPDK + OVS/SR-IOV</td>
<td>not deadline-aware</td>
<td>non-preemptive</td>
<td>process-based</td>
<td>no chain</td>
<td>supported</td>
<td>~ 256</td>
</tr>
<tr>
<td>SCHED_DEADLINE</td>
<td>per-thread</td>
<td>preemptive</td>
<td>process-based</td>
<td>no chain</td>
<td>not supported</td>
<td>&lt; 1000</td>
</tr>
<tr>
<td>eBPF + XDP</td>
<td>not deadline-aware</td>
<td>non-preemptive</td>
<td>no isolation</td>
<td>no chain</td>
<td>not supported</td>
<td>-</td>
</tr>
<tr>
<td>EdgeOS</td>
<td>not deadline-aware</td>
<td>preemptive</td>
<td>FWP-based</td>
<td>per-client chain</td>
<td>supported</td>
<td>&gt; 2000</td>
</tr>
<tr>
<td>Edge-RT</td>
<td>per-packet</td>
<td>preemptive</td>
<td>FWP-based</td>
<td>per-client chain</td>
<td>supported</td>
<td>&gt; 2000</td>
</tr>
</tbody>
</table>