End-to-end Window-Constrained Scheduling for Real-Time Communication

Yuting Zhang
Richard West

10th International Conference on Real-Time and Embedded Computing Systems and Applications

RTCSA 2004
Motivation

Movie Server Application
- Media streams travel multiple hops
- Have end-to-end QoS
  - Deadline requirement
  - Jitter requirement
- Can tolerate some lost or late packets
Motivation

Movie Server Application

- Media streams travel multiple hops
- Have end-to-end QoS
  - Deadline requirement
  - Jitter requirement
  - Can tolerate some lost or late packets
- End-to-end Window-constrained Scheduling Problem
Talk Outline

- Introduction
- Problem statement
- MVDS algorithm
- Evaluation
- Concluding remarks
Window-constrained Model (1/3)

Suitable for e.g., multimedia & weakly-hard real-time systems:

- Not every deadline needs to be met
  - Impossible to meet every deadline in overload case
  - Can tolerate some lost or late packets without degrading service too much
- Constraints on loss patterns
Window-constrained Model (2/3)

- Guarantee a fraction of service over a fixed window of packets in real-time streams
  - \((m, k)\) window-constraint: \(m\) out of every \(k\) packets meet their deadlines

- Example:
  - \((m,k) = (2,5)\)
    

    ![Example Diagram](image)

  - \((m,k) = (2,5)\)
    

    ![Example Diagram](image)
Window-constrained Model (3/3)

- Characteristics:
  - Independent service guarantees
    - Each stream gets at least a fixed share of service without being affected by others
  - Suitable for overload cases
    - Strategically skip some packets
    - **Min utilization** may still be 100% for feasible schedule
  - Bounded delay and jitter
    - Within a given window
Prior Research

- **DWCS [West, Zhang et al: IEEE TOC’04]**
  - Window-constrained service guarantees with unit processing time and same packet inter-arrival time

- **VDS [Zhang, West, Qi: RTSS’04]**
  - Outperforms DWCS especially when packet inter-arrival times are different

🤔 Previously assumed single server

💡 **Problem**: How to extend original window-constrained scheduling problem across multiple hops (or servers)?
E2E Window-constrained Problem

- Each stream $S_i$ is characterized by:
  - Packet size (transmission time = pkt size/bandwidth)
  - Inter-arrival time at 1st hop (request period)
  - Path length: (# of hops to travel)
  - End-to-end delay bound $D_i$
    - Mainly determined by queue delay due to scheduling
  - End-to-end window-constraint $(m_i,k_i)$

- Goal:
  - Minimize end-to-end window-constraint violations
  - Maximize link utilization
Challenges

- Assumption:
  - No global control mechanism or feedback signal from downstream to upstream servers
  - All actions are taken locally

- Challenge: Given end-to-end QoS requirement, what is:
  - Local (per hop) scheduling policy?
  - Local QoS requirement?
  - Local drop scheme?

- Approach:
  - Use MVDS – an extension of VDS for a single server
Virtual Deadline Scheduling (VDS)

- Serve the head packet of eligible stream with the lowest virtual deadline
- **Virtual Deadline**
  - Combines *request deadline* and *window-constraint* together
  - if current constraint is \((m', k')\), next packet should be served within \((k'\times T)/m'\) time units

\[
Vd_i(t) = \frac{k_i'T_i}{m_i'} + ts_i(t) \quad (m_i' > 0)
\]

(\(ts_i(t)\) is start of current request period at time unit)

---

Current time, \(t=14\), \(Vd(14) = (k' \times T /m') + ts = (3 \times 4/2) + 12 = 18\)
VDS Algorithm

- Service constraint updates for $S_i$:
  - if (packet from $S_i$ serviced before deadline)
    - $m_i' = m_i' - 1$
  - if (new packet arrives from $S_i$)
    - $k_i' = k_i' - 1$
    - if ($k_i' == 0$) {
      - if ($m_i' > 0$) tag stream with a violation;
      - $k_i' = k_i'$; $m_i' = m_i$
    }

Current time, $t=17$, $Vd(17) = (k' * T / m') + ts$

$C=1$, $T=4$, $m=2$, $k=3$

Blue = served

Current time, $t=17$, $Vd(17) = (2*4/1) + 16 = 24$
MVDS Algorithm

- At each hop:
  - Local Virtual deadline $\leftrightarrow$ Local Real-time deadline
    + Local window-constraint

- Challenge:
  - How to derive the local values from the global service requirements

⚠️ Problem to solve:
1. Mapping end-to-end deadlines to per-hop local deadlines
2. Updating local current window-constraints and local scheduling states
3. When to drop late packets
Local Deadline Assignment

- Downstream server can compensate for upstream service
- Local deadline = previous deadline + local delay bound
  - Local delay bound = \( \frac{\text{end-to-end delay bound}}{\text{# of hops}} \)
  - e.g. Local delay bound = \( \frac{\text{end-to-end delay bound}}{\text{# of hops}} \)

Diagram:

- \( \downarrow \): Pkt arrival
- \( \uparrow \): deadline
- \( \downarrow \): Pkt departure

Timeline:
- At first server:
  - Times: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
  - Events: arrivals and deadlines
- At second server:
  - Times: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
  - Events: arrivals and deadlines
Local Current Window-constraints

Key: keep the original window at each hop

- e.g. using packet sequence number
Local Drop Scheme

- Meet the local deadline at *every* hop $\Rightarrow$ meet e2e deadline
- Miss the local deadline at *some* hop $\Rightarrow$ miss e2e deadline?
  - Delay can be made up at the following hop
  - May be possible to still meet e2e deadline

◊ **Problem**: Whether packet should be serviced or dropped if it missed its local deadline, given e2e deadline can still be met
Different Drop Schemes

- “Drop-local”: drop if the local deadline is missed
  - May be too early 😞

- “Drop-end”: drop if the end-to-end deadline is missed
  - May be too late 😞

- “Drop-prob”: drop according to some probability
  - Adaptive and fair 😊
Probabilistic Drop Scheme

- How to decide drop probability?
  - Minimum Utilization (at hop h): minimum required service
    \[ U_{min}^h = \sum_{i \in S(h)} \frac{m_i C_i}{k_i T_i}, \quad |S(h) = \{i \mid S_i \text{ passes through hop } h\} \]
    - 1-\(U_{min}\): Surplus capacity to compensate for wasted service (due to missed deadlines)
    - As tolerable wasted service \(\uparrow\) drop probability \(\downarrow\)
      No tolerable wasted service, always drop
    - Drop probability \(\propto 1/(1-U_{min})\), \(U_{min} < 1.0\)
      
      \[ \text{Prob} = 1, \quad U_{min} = 1.0 \]
Latency

- How late is packet relative to local and e2e deadlines?
- Intuition: As latency ↑ chance to meet e2e deadline ↓ & drop probability ↑
- Distinguish the packets based on delay

Drop probability

- F(1/(1-Umin), Latency)
MVDS Algorithm

- **MVDS**
  - Local virtual deadline:
    \[ V_d^{h_i,j}(t) = \frac{k_i^{h_i'} \delta_{i,j}^{h_i}}{m_i^{h_i'}} + ts_{i,j}^{h_i} \quad (m_i^{h_i'} > 0) \]
    (jth packet is the head packet of stream \( S_i \))

The packet with earliest local virtual deadline has highest service priority
Evaluation

- Experimental setup: (NS-simulation)

- Performance metrics:
  - Violation rate
  - Miss/drop rate
Different Drop Schemes - Violation Rate

- Drop-prob performs well in both under-load and over-load case
- Drop-local (favors cross traffic), drop-end (main-stream), drop-prob (fairer)
Different Drop Schemes – Violation Rate

All streams

Main streams

Under-load case
Different Drop schemes – Miss/Drop Rate

- Drop-prob drops less than drop-local in under-load
- Drop-local (favors cross traffic), drop-end (main-stream), drop-prob (fairer)
MVDS performs well in both under-load and over-load case
MDWCS, CEDF (favors cross traffic), MVDS (fairer)
Different Priority Schemes – Violation Rate

All streams

Main streams

Under-load case
Conclusions

- We propose a multi-hop VDS algorithm (MVDS) for the end-to-end window-constrained scheduling problem

- Have shown:
  - how to transform global service constraints of real-time streams into localized values for use at each hop
    - to exploit cooperation between servers
  - how to combine window-constraints and deadlines to decide the scheduling priority – *virtual deadline*
  - how to drop packets to minimize service violation rates while maximizing link utilization – *probabilistic drop*
The End

Thank You!

Questions?