Analysis on Caching Large Content for Information Centric Networking

Yoji Yamamoto, Junji Takemasa, Yuki Koizumi, Toru Hasegawa
Osaka University
Packet-level Caching

- **In-network caching in ICN**
  - Each packet is named
  - ICN naturally supports packet-level caching

- **Packet-level caching may be more efficient than object-level caching**
  - An object is divided into packets and cached
  - This allows to utilize the entire cache space

---

**Object-level caching**

**Packet-level caching**

- No empty space

Some space not used
Inefficiency Due to Looped Replacement

- **Looped replacement** [1]
  - Packets of an object are evicted by packets of the same object

- **Degradation in performance of packet-level caching**
  - The remainder of the packets waste some space of the cache
    - They never contributes to cache hits
  - Large objects tend to waste more space of the cache [2]

- **Research issues are twofold:**
  - To develop an analytical model of looped replacement in order to simulate packet-level caching storing large objects
  - To develop a method to suppress looped replacement

---


Why Looped Replacement Occurs?

- Packets are inserted/evicted individually
  - There are \textit{time gaps between insertion/eviction of the first and the last packets of an object}

- Three transient states
  1. Insertion state
     - \textit{A part of packets of an object} is in the cache
     - The remainder is arriving at the cache
  2. Steady state
     - \textit{All the packets of an object} are in the cache
  3. Eviction state
     - \textit{A part of packets of an object} is in the cache
     - The remainder is being evicted from the cache

- Occurrence of looped replacement
  - It occurs when \textit{the first packet arrives at the cache during eviction state}
Analytical Packet-level Caching Model

**Overview**

Calculate the probability that the first packet arrives at the cache when the object is in eviction state

**Steps**

1. Calculate **the duration of the three states**
2. Calculate **the time when eviction state starts/finishes according to the duration** derived in step 1
   - Eviction state starts at $\tau_1^i + \tau_2^i$
   - Eviction states finishes at $\tau_1^i + \tau_2^i + \tau_3^i$

3. Calculate **the probability that the first packet arrives during eviction state** according to the arrival process (e.g., Poisson process)
Challenges of Deriving Three States

- **Difference between packet-level and object-level caching**
  - Packets of an object are inserted and evicted individually
  - Several objects arrive simultaneously at the cache

- **Two challenges of modeling the duration**
  1. **Factors to determine the duration** of insertion, steady and eviction states are different
  2. Packets of **objects simultaneously arriving** at the cache also affect the duration
Simultaneously Arriving Objects

- **Packets of other objects also arrive during insertion state of object** \( i \)
  
  - Some packets of such objects are inserted between the first and the last packets of object \( i \)
    - Such packets are referred to as **interrupting packets**
  
  - They increase the number of packets between the first packet and the last packet of object \( i \)
  
  - They makes the duration of eviction state longer

The case without incorporating interrupting packets

The case with incorporating interrupting packets

Interrupting packets (red packets) make the duration of eviction state longer
Duration of Three States

- **Main factors to determine the three states**
  - Insertion state
    - Arrival rate of *packets of each object*
  - Steady state
    - Arrival rate of *all objects except itself*
    - Number of interrupting packets
  - Eviction state
    - Arrival rate of *all objects*
    - Number of interrupting packets
  - Common factors
    - Object size

- **Duration of the three states**
  - Extend Che’s model [3] so that the above factors are incorporated

Analytical Evaluation Condition

- **Aim of analyses**
  - To validate that looped replacement occurs
  - To evaluate effects of object size on looped replacement
  - To investigate causes of looped replacement

- **Environments: VoD scenario**
  - **Traffic:** Netflix [4]
    - Number of objects: 18,000
    - Packet size: 1 Kbyte
    - Number of packets of an object: 2,250,000 (60 minutes, 5Mbps)
    - Total request rate of objects: 0.05 requests/sec

  - **Cache size**
    - 1% of the number of entire packets

Occurrence of Looped Replacement

- Looped replacement often occurs for highly (but not the most highly) popular objects(*)

- The looped replacement probabilities of the 4th to the 126th popular objects are higher than 0.1

Fig. The hit and looped replacement probability of top 500 popular objects (X-axis: the object IDs sorted in descending order of their arrival rates)

*) A **popular objects** is defined as an object of which arrival rate is high
Effects of Object Size

- **Condition**
  - Pickup one object of which looped replacement probability is highest in the previous evaluation
  - Change its size from 1% to 100% of the other objects

- **The looped replacement probability increases as the object size grows (Fig. a)**
  - This is because the number of interrupting packets gets large (Fig. b)
  - This results in increase in the duration of eviction state becomes longer (Fig. c)
    - Note again that looped replacement occurs when the first packet arrives during eviction state

![Graphs](#)

- a) Looped replacement probability
- b) The number of interrupting packets
- c) The length of duration of eviction state
Causes of Looping Replacement

- Unpopular objects dominate interrupting packets

![Graph showing proportion of unpopular objects](image)

- Solution of looped replacement
  - Not to insert packets of unpopular objects into the cache
    - **Cache admission**
Suppressing Looped Replacement

■ Cache admission
  - Decide whether a packet arriving at a cache should be inserted or not
  - Cache admission reduces the number of interrupting packets caused by unpopular objects

■ Filter [3]
  - A frequency-based cache admission algorithm
  - It does not insert a packet of which number of recent arrivals is smaller than a threshold

threshold = 2

3

head cache tail

Resolving Looped Replacement

- Cache admission suppresses looped replacement
  - The highest looped replacement probability
    - LRU: 0.173
    - Filter 0.016

- Cache admission reduces the number of interrupting packets
  - Reduce 70% of the number of interrupting packets

Graphs showing looped replacement probability and proportion of cumulative interrupting packets.
Conclusion

- **Packet-level caching causes looped replacement**
  - There are time gaps between insertion/eviction of the first and the last packets of an object
  - This causes looped replacement

- **Develop an analytical model of packet-level caching**
  - Model the duration between insertion and eviction of packets constituting an object
  - Model the number of packets of simultaneously arriving objects

- **Suppress looped replacement with cache admission**
  - Cache admission reduces the number of interrupting packets
    - Many of interrupting packets are packets of unpopular objects
APPENDIX
Resolving Looped Replacement

- The solution of looped replacement is different between cache admission and OPC

- Cache admission: make $\tau^i$ short as much as possible by reducing interrupting packets

- OPC: avoid being evicted state by evicting packets of an objects from the last packet
The hit probability

- OPC improves hit probability by the half of looped replacement probability
Reducing Interrupting Packets