大阪大学
OSAKA UNIVERSITY

Identifying Highly Popular Content Improves Forwarding Speed of NDN Software Router

○**Junji Takemasa**, Kosuke Taniguchi,

Yuki Koizumi and Toru Hasegawa

Osaka University, Japan

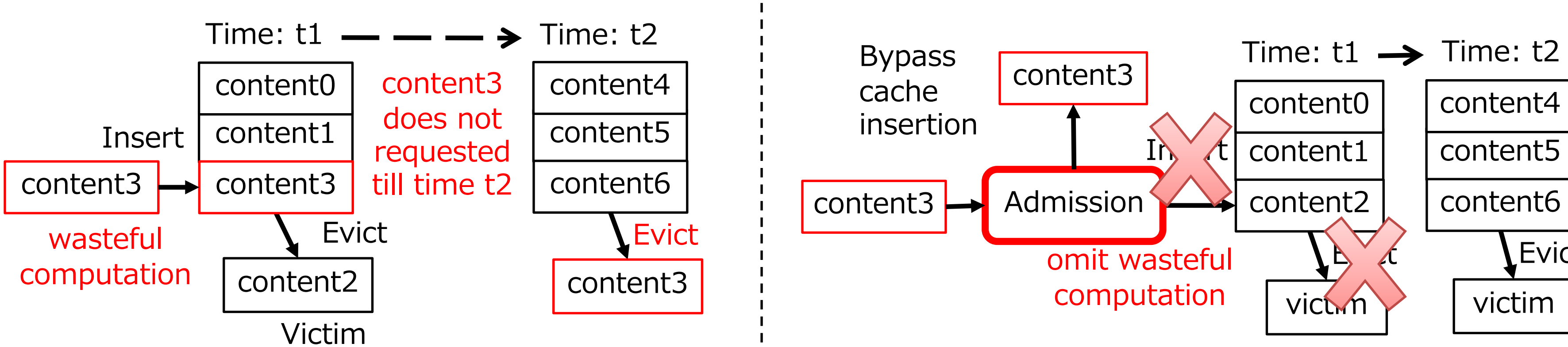
Toward High-Speed NDN Routers

- **Two heaviest functions of NDN routers [1]**
  1. FIB lookup
     - Existing studies try to resolve this issue
       - Fast LPM algorithms
     - FIB lookup cannot be bypassed
       - One of the essential functions of name-based forwarding
  2. **Cache insertion**
     - Algorithm for cache insertion is so simple that reducing its computation is difficult
     - However, cache insertion for some content can be omitted
- **Cache insertion for unpopular content**
  - Inserting content that will not be requested in the future can be omitted
  - Computation time for cache insertion can be reduced

[1] K. Taniguchi et al., "Poster : A Method for Designing High-speed Software NDN Routers," in Proc. of ACM ICN, 2016.

Wasteful Cache Computation

- **Unnecessary cache insertion due to cache eviction**
  - Cache eviction (like LRU and LFU) decides which content should be evicted from the cache
    - Incoming content is always inserted into the cache even if it will not be requested in the future
- **Bypassing the unnecessary cache insertion with cache admission**
  - Cache admission decides whether content should be inserted to the cache
    - Content is not inserted into a cache if it may not be requested in the future
    - Wasteful computations can be reduced

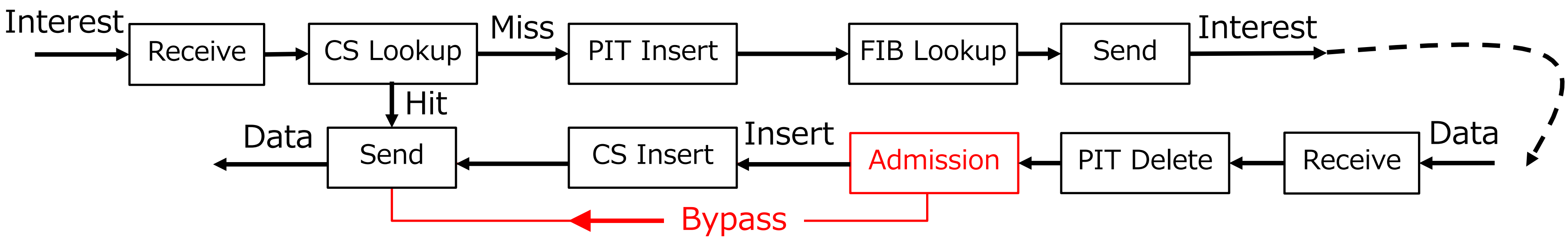


(a) Cache Eviction

(b) Cache Admission

Forwarding with Cache Admission

- **Add cache admission before cache insertion**

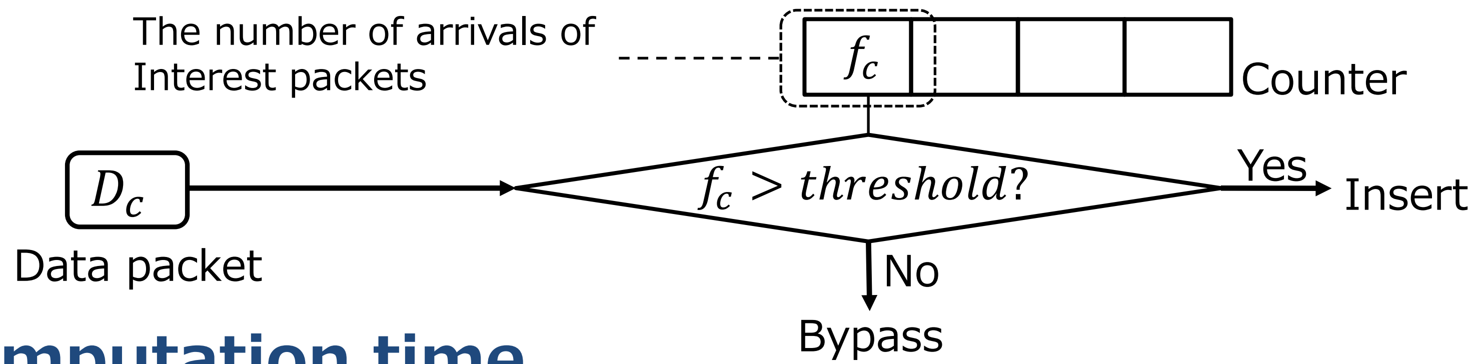


- **Requirements of cache admission:**
  - Fast computation
    - Computation for the cache admission should not degrade forwarding speed
  - Small memory consumption
    - Data structures for the cache admission should be implemented on DRAM
- **Approach: Filter (Fast frequency-based cache admission)**
  - It requires only a few instructions without any loops
  - Counting bloom filter and small queue for fast computation and small memory consumption

Algorithm for Fast Computation

■ Algorithm: compare frequency with threshold

- Decides insertion of Data packets based on its frequency
- Records the number of arrivals of Interest packets as frequency

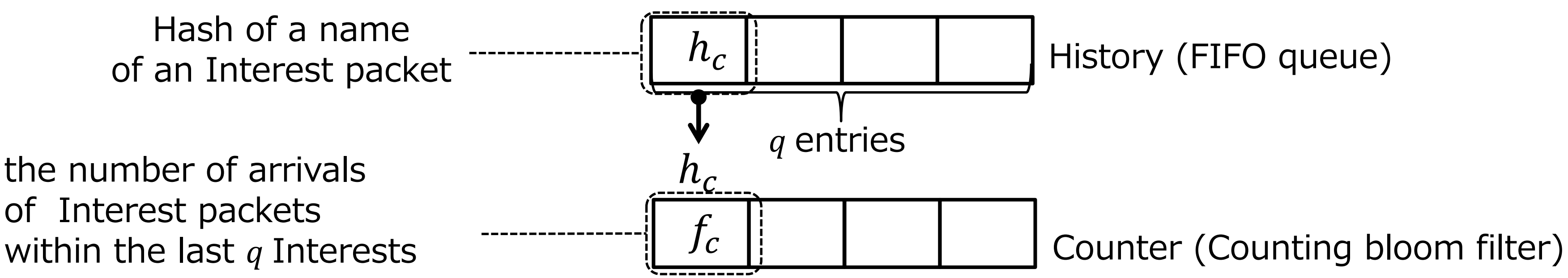


■ Computation time

- Sufficiently low (less than 100 CPU cycles)
  - The computation time does not have much impact on forwarding speed (we will show the results later)
- Time complexity: O(1)
  - No loops

Data Structure for Small Memory Consumption

- **Data structures:**
  - Counter: records a frequency of a Data packet to decide its cache insertion
    - Entry: the number of arrivals of Interest packets within history
  - History: limits the number of recorded arrivals of Interest packets in counter
    - Entry: a hash of a name of an Interest packet

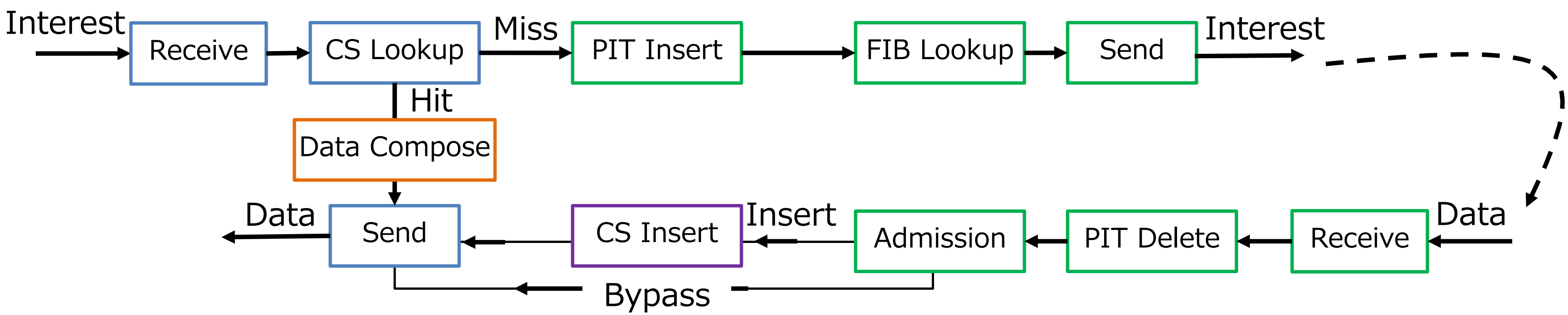


- **Small memory consumption**
  - Counting bloom filter is space-efficient data structure
  - The size of history is much smaller than that of counter

Method to Evaluate Forwarding Speed

1. Experimentation

- Implement an NDN software with Filter
- Measure computation time of each function in the NDN software



2. Model-based estimation

- Calculate cache hit and insertion probabilities
- Calculate the average computation time of NDN forwarding

$$\boldsymbol{Speed} = \boldsymbol{Cycles}_{NDN}^{FWD} = \Sigma_{f\in Always} 1 \cdot Cycles_f + \Sigma_{f\in Miss} p^{Miss} \cdot Cycles_f + \Sigma_{f\in Hit} p^{Hit} \cdot Cycles_f + \Sigma_{f\in Insert} p^{Miss} p^{Insertion} \cdot Cycles_f$$

Calculation of Insertion and Hit Probabilities

■ Develop Filter model

- Calculate insertion probabilities of each Data packet
  - Probability: "the num. of arrivals of Interest packets > threshold"
    - Filter: inserts a Data packet when its frequency > threshold
    - Frequency: the num. of arrivals of Interest packets within history
    - Derived from CDF of Poisson Distribution

■ Combine Filter model with FIFO* eviction model[2]

- Calculate insertion rates from Filter model and apply these
  - FIFO eviction model: calculate hit probability from insertion rates
  - Insertion rates: an Interest misses at cache and the Data is inserted

Filter model → Eviction model

$$\lambda_c^{Insertion} = \lambda_c (1 - p_c^{Hit}) p_c^{Insertion}$$

$$p_c^{Insertion} = \mathrm{P}[f_c > Threshold]$$

$$p_c^{Hit} = \mathrm{FIFO}(\lambda_c^{Insertion}, Csize)$$

* We choose FIFO because its implementation is simpler than LRU, LFU

[2] V. Martina etal., "A unified approach to the performance analysis of caching systems," in Proc. of IEEE INFOCOM 2014

Evaluation Conditions

- **NDN software: [3]**
  - Its computation is highly optimized by using hash tables
- **Cache eviction policies:**
  - With Filter: FIFO
  - Without Filter: FIFO, LRU, LFU
- **Settings of Data packets:**
  - The number of unique Data packets: 10^7 packets
  - Popularity of Data packets: Zipf distribution with α=0.8
  - Cache size: 1% of the number of unique Data packets
- **Filter's settings**
  - History Length: 10^6 entries
  - Threshold: 10

[3] W. So, et.al. "Named data networking on a router: Fast and DoS-resistant forwarding with hash tables." in Proc. of ACM/IEEE ANCS 2013

Evaluation Results

■ Improvement of Forwarding Speed

- Filter reduces computation time for NDN forwarding by 24%
  - Without Filter: 3201.0 cycles/packet → With Filter: 2431.2
- Filter improves forwarding speed of an NDN router

■ Reduction of the number of cache insertions

- Cache insertion probability of Filter : 5.6×10^{-6}
- Filter efficiently bypasses cache insertions

■ Deterioration of cache hit probability

- Filter doesn't degrade cache hit probability of cache evictions

FIFO with Filter	FIFO	LRU	LFU
0.36	0.22	0.26	0.37

Conclusion

- **Wasteful cache computation for unpopular content degrades forwarding speed**
  - Cache eviction always inserts content which may not be requested in the future
  - Cache admission allows the unnecessary cache insertion to be bypassed
- **Filter: a fast cache admission algorithm**
  - Filter identifies content that may not be requested in the future
  - Fast computation and small memory consumption
- **Filter improves forwarding speed**
  - Filter reduces the computation time by about 24%
  - Filter does not degrade the cache hit probability