ABC: Adaptive Binary Cuttings for Multidimensional Packet Classification

Authors: Haoyu Song, Member, IEEE,
and Jonathan S. Turner, Fellow, IEEE, ACM
Presenter: Qing Lyu
2016/11/23

ToN’13, 21(1)

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Packet Classification

- Net-work security, network virtualization, and network quality of service (QoS)
  - large-scale packet classification involving thousands to tens of thousands of filters in a single router.
- Increasing network traffic poses greater challenges for fast packet classification
  - 10-Gb/s,40Gb/s, even 100-Gb/s connections in edge and core networks
  - 10-GbE line card needs to classify 15 million packets per second

Packet Classification

- Exhaustive search
  - examine all entries in the filter set
- Decomposition
  - decompose the multiplefield search into instances of single field searches, perform independent searches on each packet field, then combine the results
- Decision tree
  - construct a decision tree from the filters in the filter set and use the pack et fields to traverse the decision tree
- Tuple space
  - partition the filter set according to the number of specified bits in the filters, probe the partitions or a subset of the partitions using simple exact match searches

TCAM and algorithmic solutions

- TCAM:
  - engineering considerations.
  - cost, power dissipation, and board footprint considerations.
- Algorithmic solutions
  - flexibility, embed the memory on-chip (ASIC), system-on-chip integration
  - smaller storage allows the use of faster (but more expensive) memory devices to boost the throughput.
  - satisfy the worst-case throughput with a small amount of memory (DRAM, SRAM, or embedded memory)
  - Intrinsic complexity of the problem increases memory consumption

TCAM and algorithmic solutions

- TCAM cell :14–16 transistors to store a bit, SRAM cell uses 6 transistors, SDRAM cell uses 1 transistor and 1 capacitor .
- A bit in a TCAM component costs about 10 bits in an SRAM component.
- TCAM component usually consumes 18 B (144 bits) to store a 5-tuple filter
- SRAM-based algorithm should consume no more than 180 B per filter to compete with TCAM.

Decision tree algorithms

- Easy to implement and allow the tradeoff between storage and throughput.
- Splitting the filter set recursively until each subset contains fewer filters than a predefined bucket size.
- The filters in each of these subsets are organized in a priority list.
- NP-complete:
  - building optimal decision trees in the sense of minimizing the search steps (Hyafil and Rivest)
  - construction of storage-optimal decision tree (Murphy and McCraw)

Decision tree algorithms

- A globally optimal decision?
- Heuristic-based ! local optimality only
- Testing the decision tree and linearly searching the stored filters when a leaf node is reached.
- Easy to implement in both software and hardware but sensitive to the filter set structure

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Decision tree construction

- Crucial to Limit the storage when
  - implement the data structure
  - traverse the tree in order to find the best matching filter.
- Split the current filter set S into a number of subsets $s_1, s_2, \ldots, s_n$
- Keep splitting, until the size of nodes is no bigger than a bucket size, and the node become a leaf.

Advantages of decision tree method

- Achieve very high throughput when using a deep pipeline and the parallel bucket matching scheme
- Simple and efficient implementations by using binary encoding techniques.
- Incremental updates are also easily supported by decision trees
- Flexibly tradeoff between throughput and storage

Disadvantage of decision tree method

- Filter duplication
  - many filters are weakly specified on some dimensions with wildcards or large ranges.
  - more cuts at each tree node so that each child node contains fewer filters, thus aggravates the duplication problem.
  - e.g. firewall filter contain many heavy wildcard filters in source or the destination IP address fields

Disadvantage of decision tree method

- Skewed filter distribution
  - a small number: distribute across a wide range
  - most: concentrate in a small region
  - equal size cuts lead to imbalanced decision tree.
  - e.g. (ACL) filter sets, fairly specific on the IP address fields, but the distribution is highly skewed.
- The filter distribution directly affects the DT efficiency.

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Desired properties of DT

- The tree consists of as few nodes as possible
- The path from the root to any leaf node is short and balanced.
- Previous algorithms: set a space expansion factor to bound the number of duplicated filters, achieve as many cuts as possible per step without exceeding the threshold

A simple cutting procedure

- First find the set of optimal cutting points that can balance the distribution of filters and minimize the filter duplication effect.
- Then sort and register the cutting points
- Simply perform a binary search when a DT node is retrieved during the lookups.

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Adaptive Binary Cuttings (ABC)

- Split the filter set based on the evenness of the filter distribution, rather than the evenness of the cut volumes (three filter-set-splitting strategies)
  - adapt to the filter distribution geometrically or virtually: leads to a balanced decision tree and also reduces the filter duplication effect
- Binary encoding scheme, encodes the space-cutting sequence and can directly map to the bit string of the packet header fields.

Adaptive Binary Cuttings (ABC)

- Discards the notion of the expansion factor:
  - adapt to the filter distribution
  - make fully usage of the capacity by making as many cuts as possible.
  - tree node the same size
- Discards the notion of the bucket size:
  - improve the throughput until the given storage is used up
  - evaluate all the current DT branches and the number of filters remaining in each current leaf node
  - choose the branch that causes the current worst-case throughput to continue working on, as long as the storage budget allows

Algorithm design

- Choose bits from any dimension and any position to split the filter set
- Produce one or more full binary Cut-ting Shape Trees (CSTs). Encode with Cutting Shape Bitmap (CSB)

Build_DT

1. Initialize a single-node tree in which the root contains all the filters;
2. while (current storage $\leq$ the predefined storage budget AND
3. some current leaf nodes have >3 filters) {
4. let S_3 = the set of leaf nodes with >3 filters;
5. select $v \in S_3$ which requires the longest time to search a filter in the worst case;
6. split node v to produce the CSTs and the new child DT nodes;
7. }

Basic idea of ABC

- Guided by the memory consumption
  - seeks to use it to improve the lookup throughput when memory is expendable---L2
- A tree node is not worth splitting if it contains less than four filters---L3
  - cost of retrieving and decoding one more tree node_is greater than linear search

Basic idea of ABC

- Choose the worst-case searching path---L5
  - the maximum cost to access the last filter at any leaf node
  - leaf node depth, the tree node size, the number of filters in the list, and the filter size
  - a dynamic sorting data structure such as a heap
  - the current cost as the key
  - In each loop, remove the highest path from the heap
  - split the corresponding leaf node, and insert the newly generated paths into the heap.

Basic idea of ABC

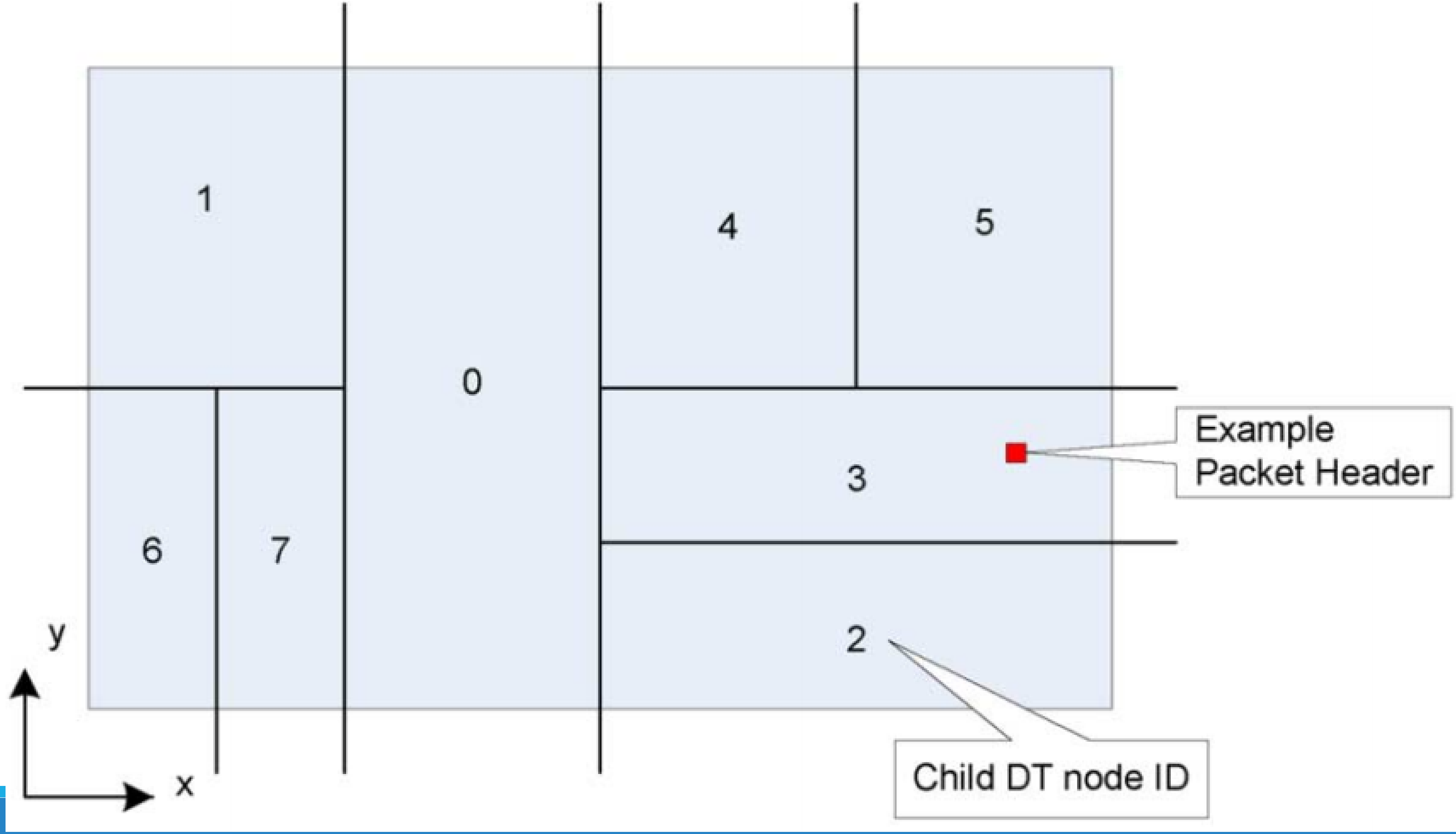
- split a leaf node and produce the CSTs ---L6
  - Three different approaches
  - A preference value is used as a metric for the quality of the DT node cuttings
  - The preference is smallest when the subsets are equal in size duplicated filters is minimized.
  - Best decision: minimizes the preference value

$$pref = \sqrt{\sum_{i=1}^{k} r_i^2}.$$

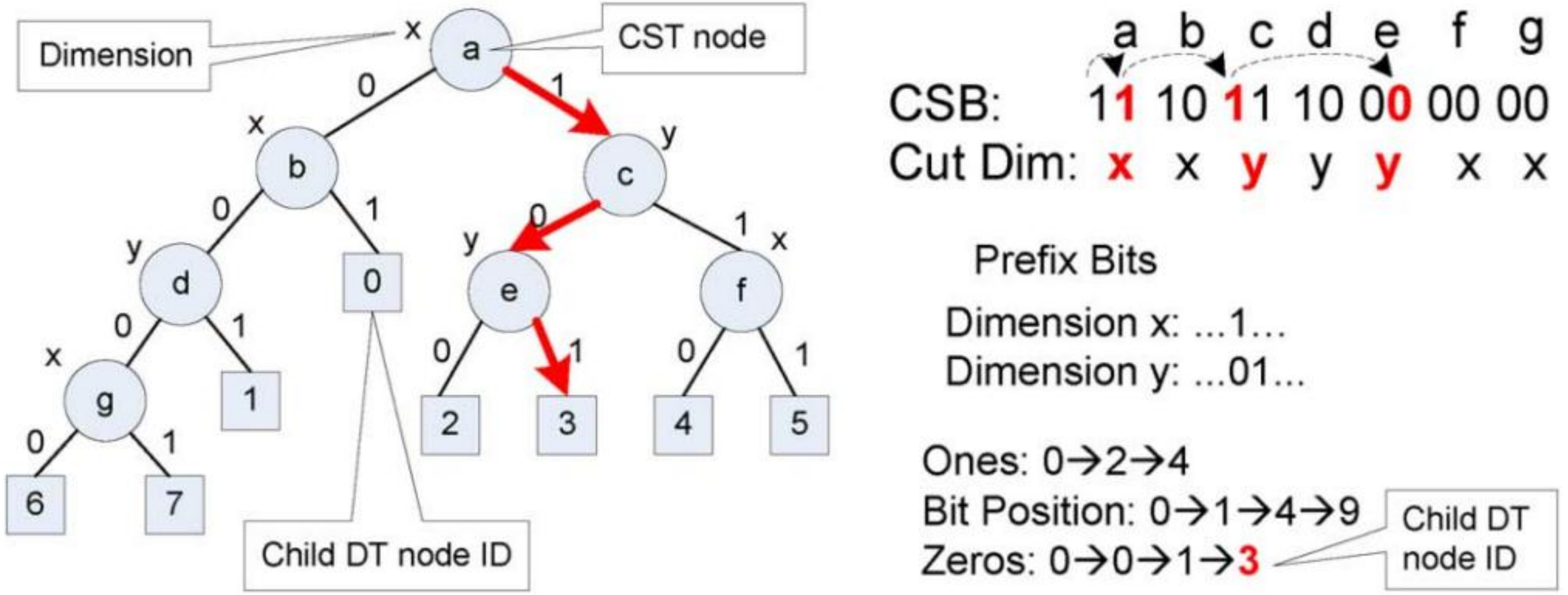
ABC Variation I

- Produces a single CST at each DT node
- Map each header field to a space dimension
- Perform multiple variable-sized cuttings per tree node
- Split into two equal-sized subregions along a certain dimension
- a full binary tree with *k* leaf nodes.

ABC Variation I



ABC Variation I



ABC Variation I

- If current number of leaf nodes $k' < k$
- Split the node i on the dimension d

$$pref_{i,d} = \sqrt{r_1^2 + ... + r_{i,d,l}^2 + r_{i,d,r}^2 + ... + r_{k'}^2}$$

$$= \sqrt{pref^2 + r_{i,d,l}^2 + r_{i,d,r}^2 - r_i^2}.$$

- Minimize the preference value

ABC Variation II

- DT node is split on multiple dimensions, but generate up to *D* separate CSTs, each for one dimension.
- Choose the leaf node on one of the CSTs to split if the cutting leads to the minimum preference value.

ABC Variation II

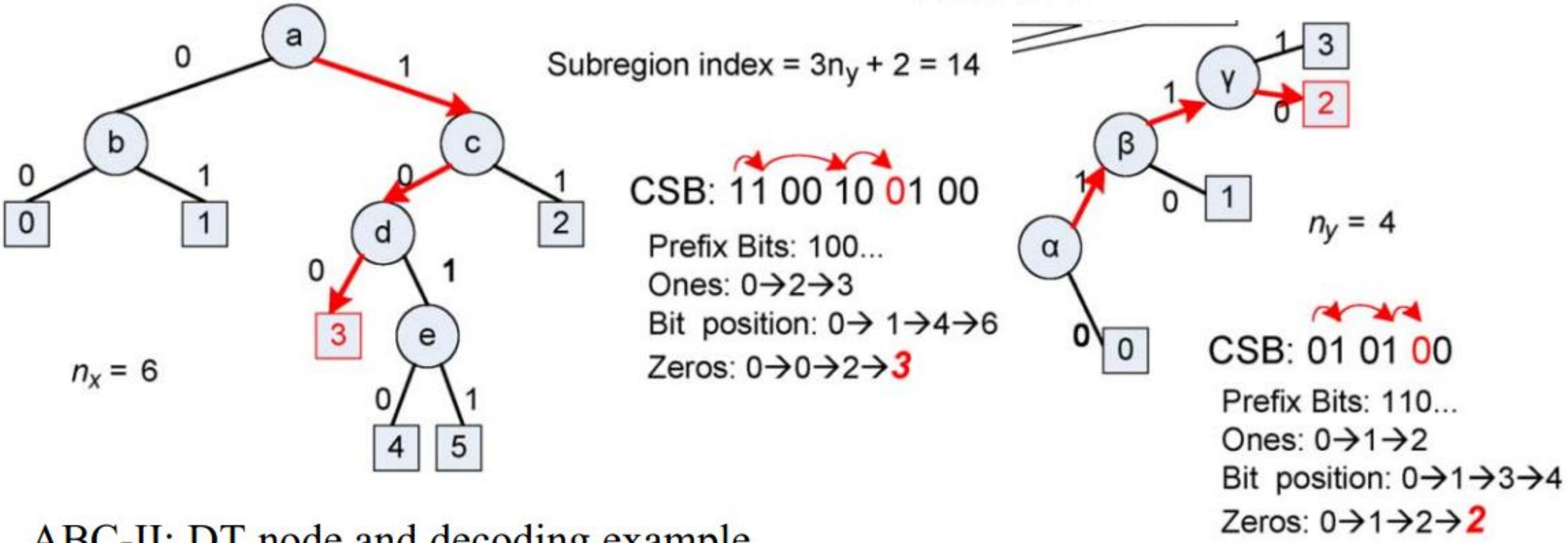


Fig. 3. ABC-II: DT node and decoding example.

ABC Variation III

- produces only a single CST at each DT node
- treats each filter as an integral ternary bit string
- start from a single-node CST and keep splitting some leaf node using a bit from the filter string until we run out of the storage space
- evaluate the new preference values for all the leaf nodes if they are split on any filter bit
- Choose the leaf node and the filter bit that can minimize the preference value to grow the CST

Comparison

- The decision tree performance is generally better if more cuttings can be done at each DT node.
  - ABC-I supports at most 22 cuts per DT node and ABC-III supports at most 13 cuts per DT node.
  - ABC-II can produce more cuts per DT node than the other two variations due to the product effect

Comparison

- Implementation Complexity
  - ABC-I and ABC-III generate a single CST per DT node and the CST can be very tall, the DT node processing latency is typically larger than that for ABC-II
  - ABC-II the CSTs can be decoded in parallel， pipelines or multiple parallel lookup engines are used to fill the memory bandwidth
  - However, preprocessing time of ABC-II is the largest because it requires more computations to produce the CSTs at each DT node.

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Optimization

- Reduce Filters Using a Hash Table
  - A filter is hashed into $H(i,j)$ if its source IP prefix specifies more than i bits and its destination IP prefix specifies more than j bits.

Optimization

- Filter Partition on the Protocol Field
  - The cutting does not work well on the protocol field.
  - Each cutting unavoidably duplicates all the filters with the wildcard protocol specification.
  - Build a decision tree for each specified protocol value.

Optimization

- Partitioning filters based on duplication factor
  - some filters suffer more duplications than the others.
  - the higher-priority filters tend to receive fewer duplications than the lower-priority filters.
  - remove the filters that results in excessive duplications (identified as spoilers) from the filter set
  - then build the decision tree on the remaining filters.
  - spoilers can be handled by a small on-chip TCAM.

Optimization

- Holding filters internally and reversing search order
  - at a DT node, if a filter would otherwise be duplicated into all the child DT nodes, keep it in the current DT node, near to root node.
  - search the filter lists using the bottom-up order to avoid unnecessary memory accesses
  - when find a stored filter list along the searching path, do not retrieve the filter list right away, Instead, push its pointer into a stack. Then begin to pop the pointers in the stack and search the filter lists only when reach a leaf node.

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Evaluation

- Synthetic filter sets generated by ClassBench.
- A packet header trace in which the number of packets is ten times of the number of filters.
- Collect the average number of bytes retrieved per packet as the average-case performance measure.

Evaluation

- Scalability on Filter Set Size.

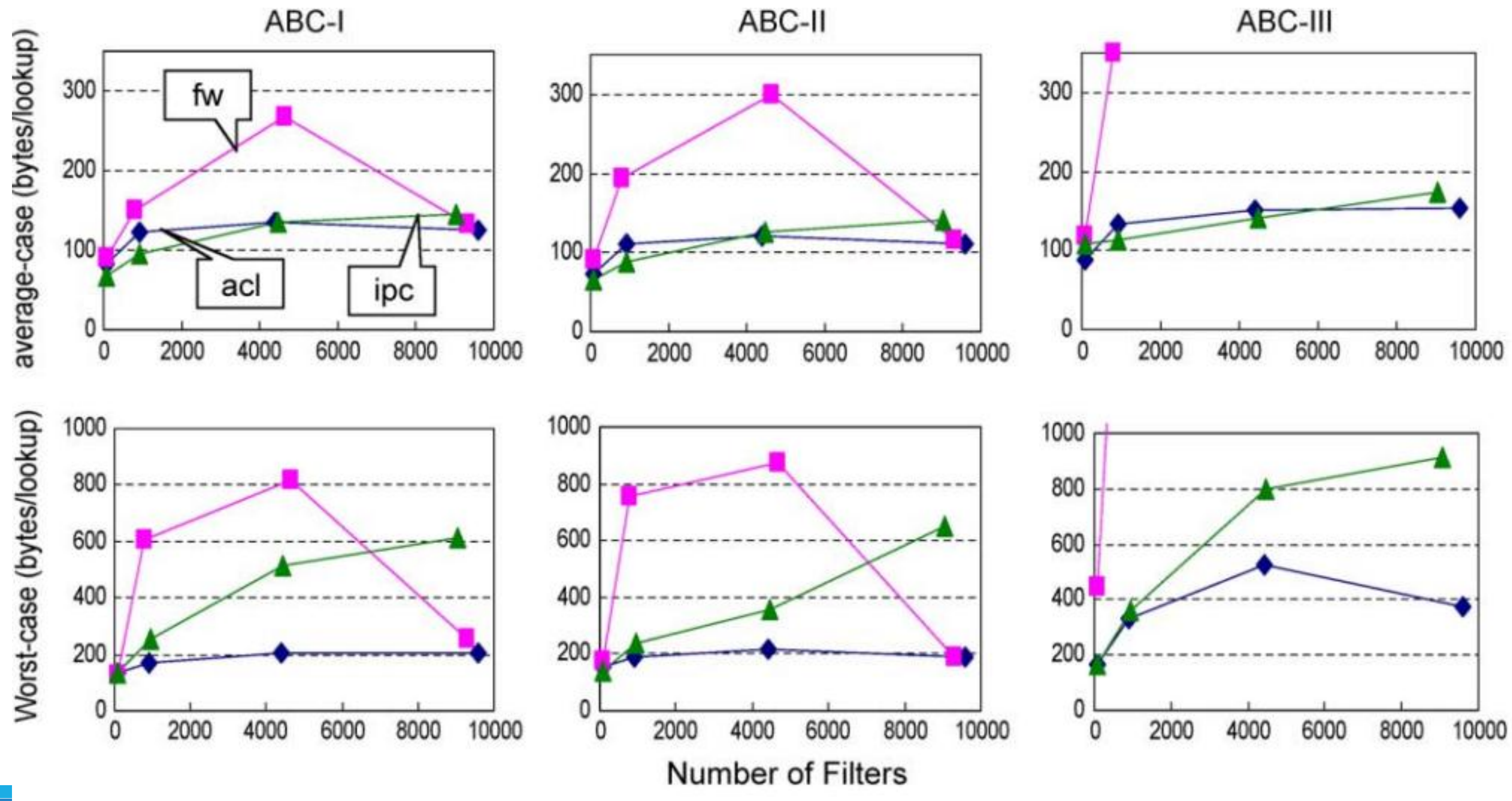


Fig. 8. Algorithm scalability on filter set size.

Evaluation

- Tradeoff of storage and throughput.

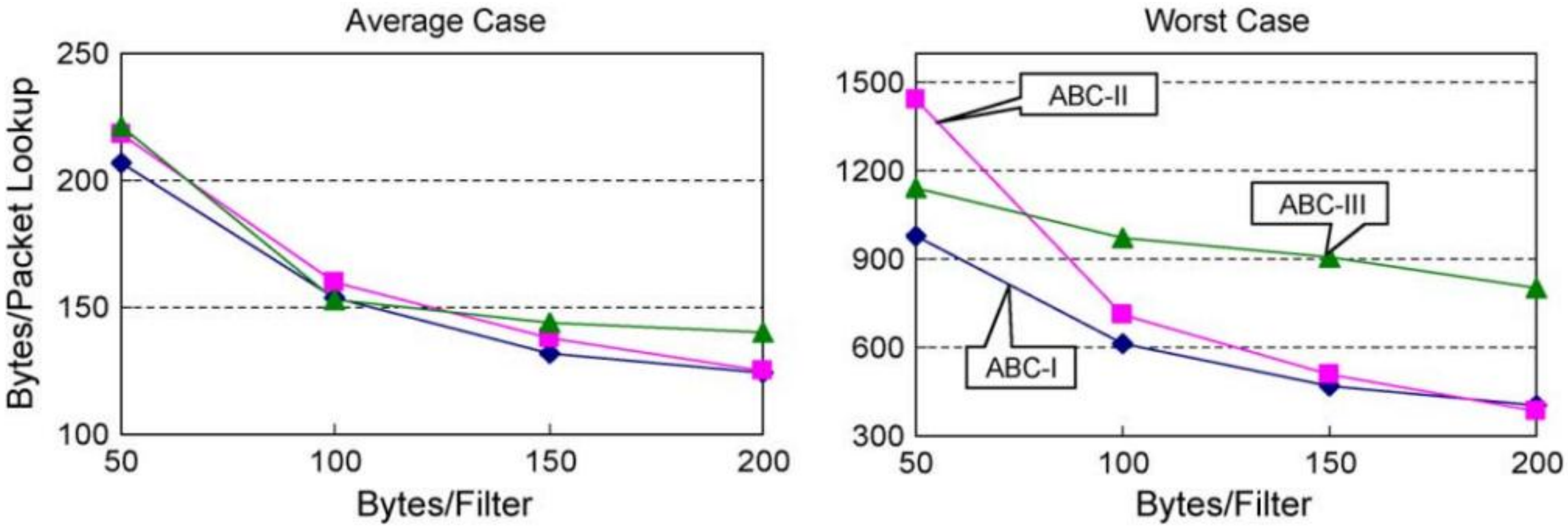


Fig. 9. Tradeoff of storage and throughput.

Evaluation

- Sensitivity to Optimizations-Effect of filter reduction using a Hash table.

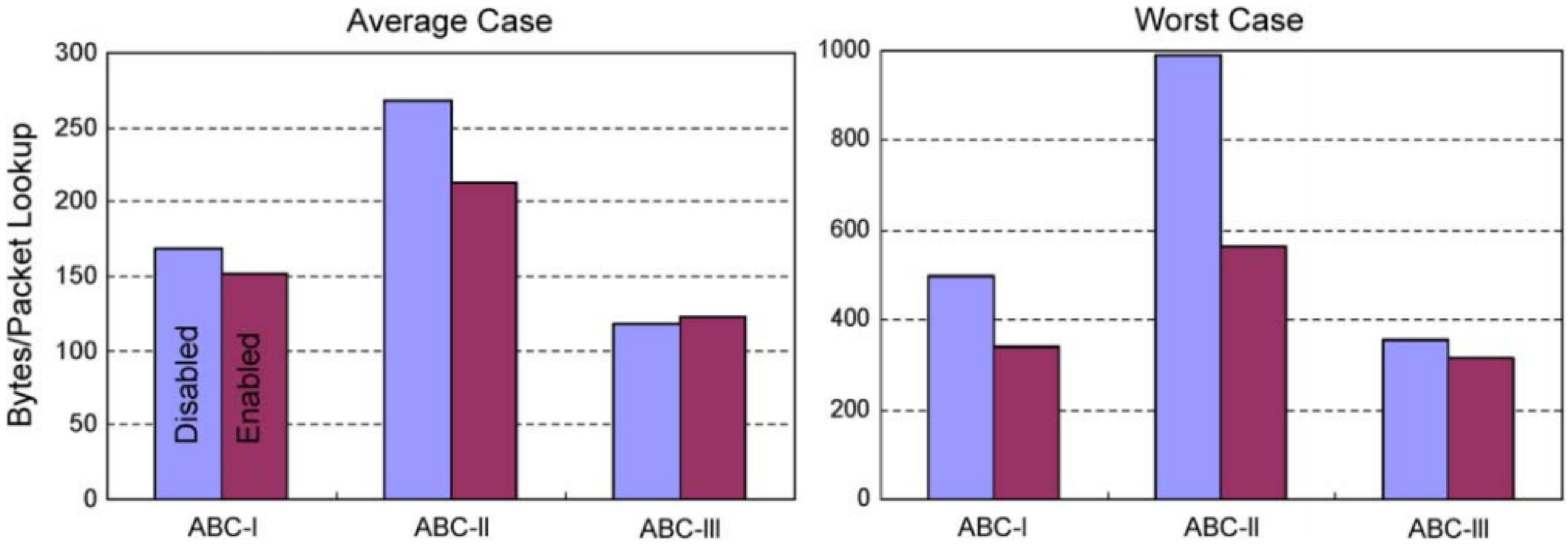


Fig. 10. Effect of filter reduction using a Hash table.

Evaluation

- Sensitivity to Optimizations-Effect of looking upon protocol field first.

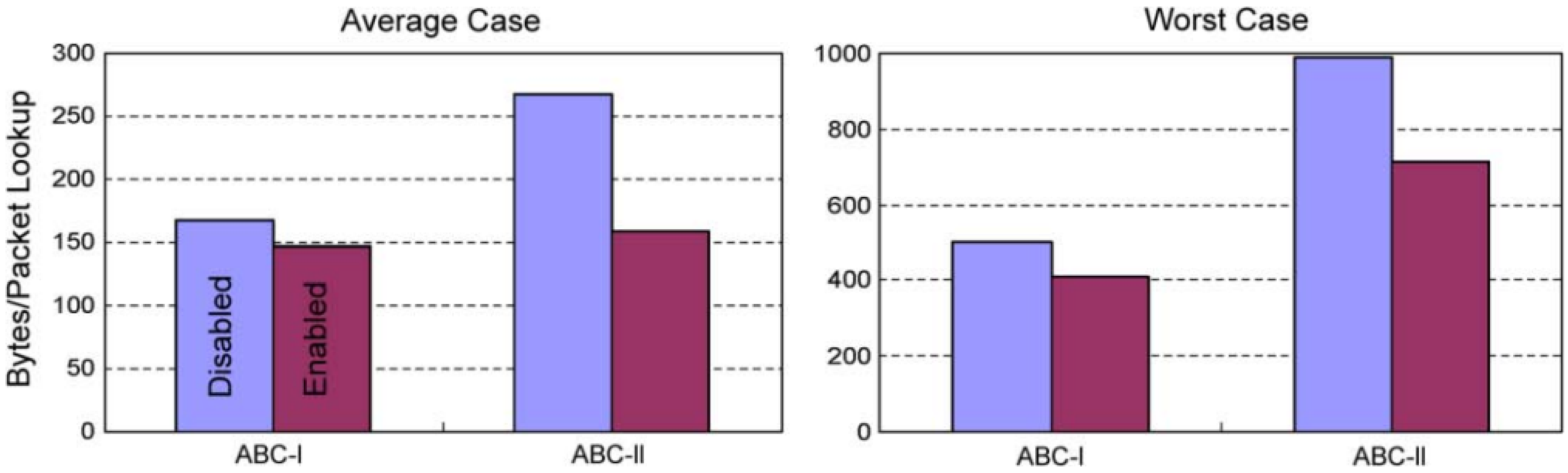


Fig. 11. Effect of looking up on protocol field first.

Evaluation

- Sensitivity to Optimizations-Effect of holding filters internally and reversing search order.

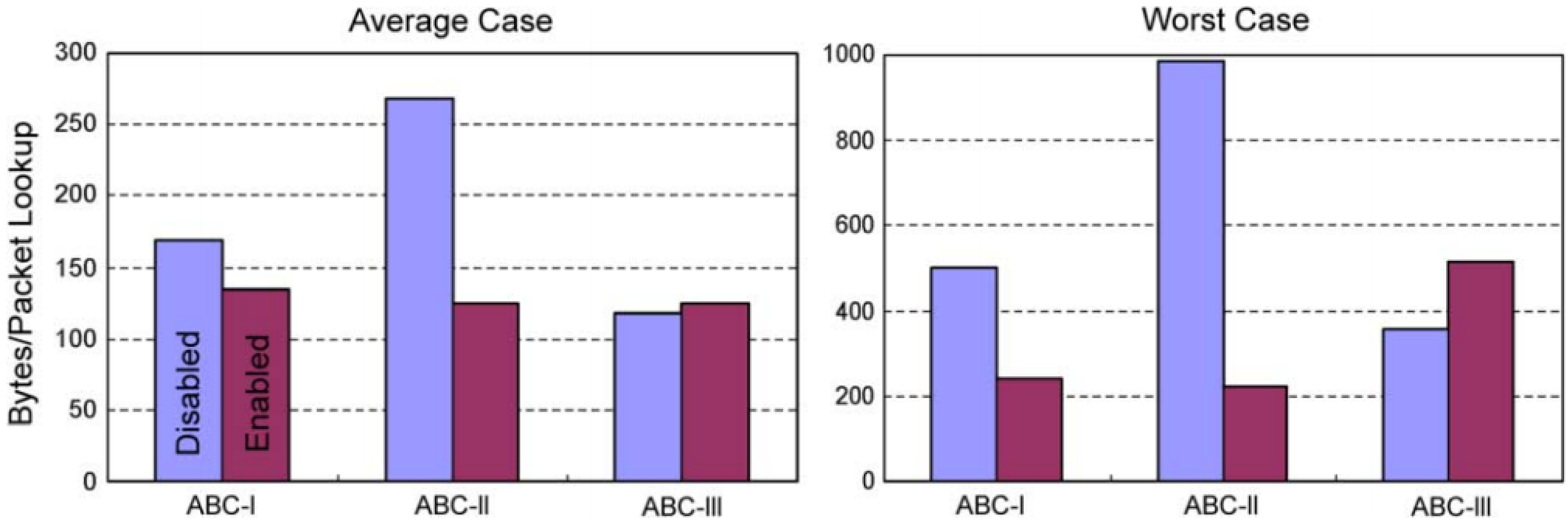


Fig. 12. Effect of holding filters internally and reversing search order.

Evaluation

- Sensitivity to Optimizations-Effect of removing highly duplicated filters.

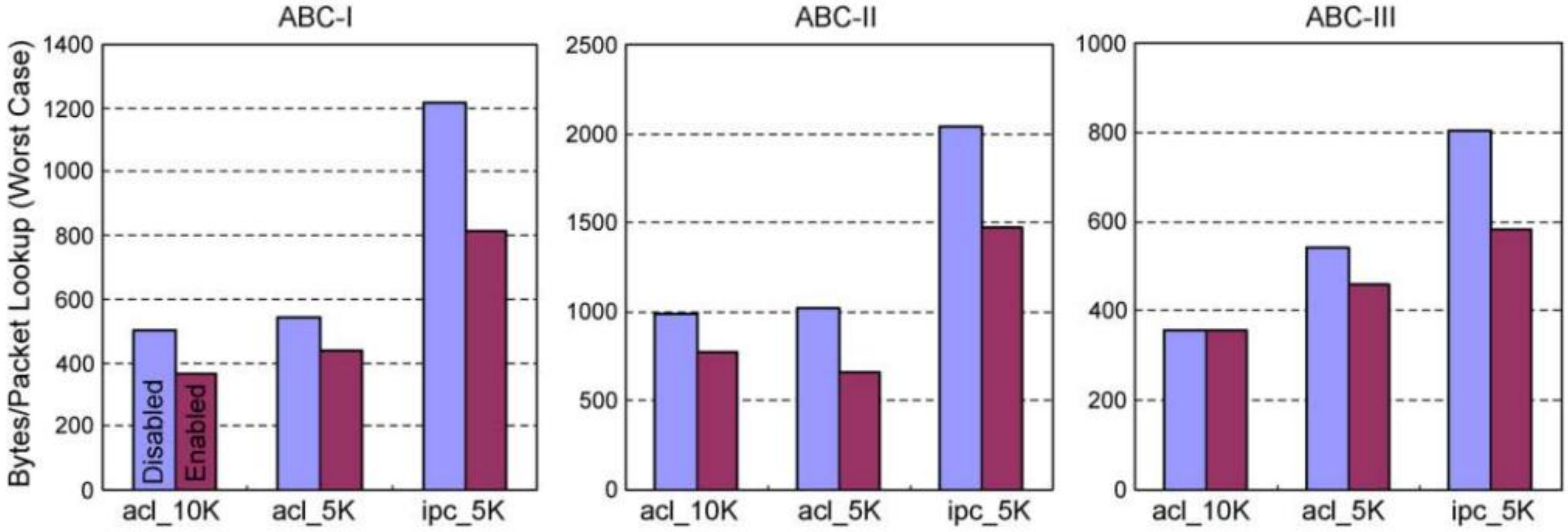


Fig. 13. Effect of removing highly duplicated filters.

Evaluation

- Sensitivity to Optimizations-Effect of changing DT node size.

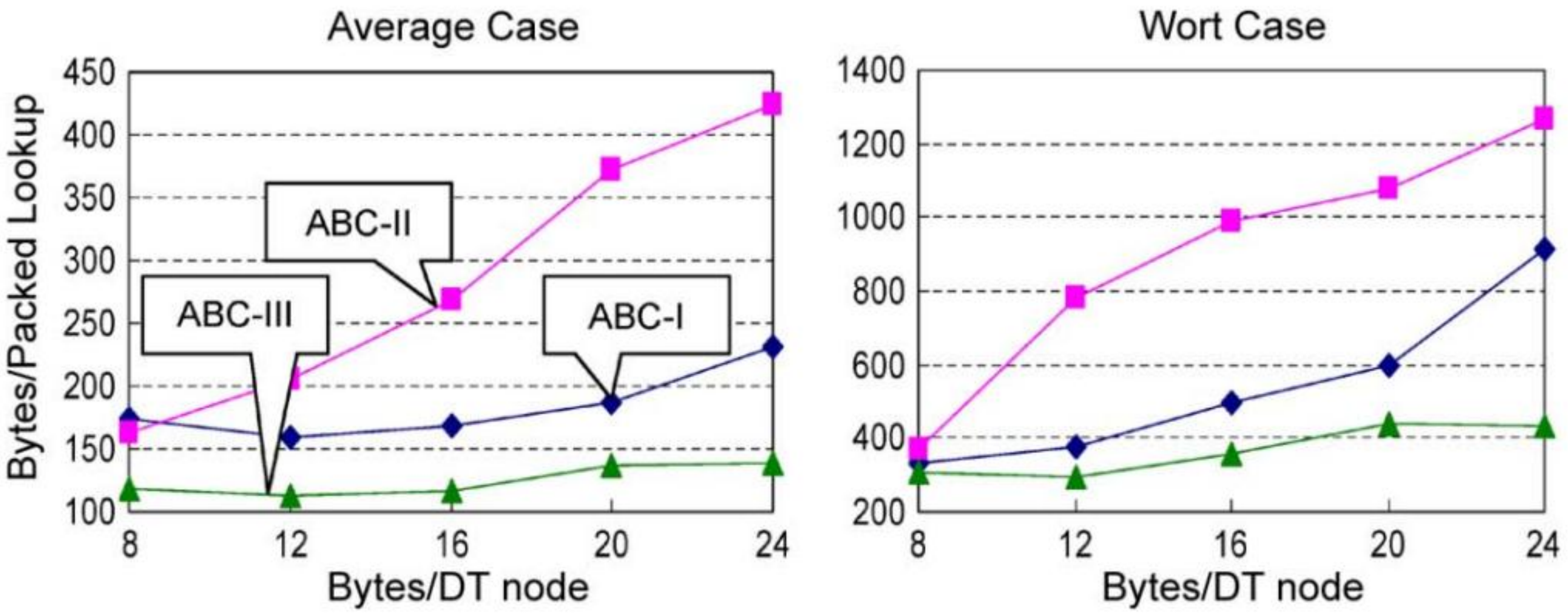


Fig. 14. Effect of changing DT node size.

Evaluation

- Comparison with HiCuts and HyperCuts.

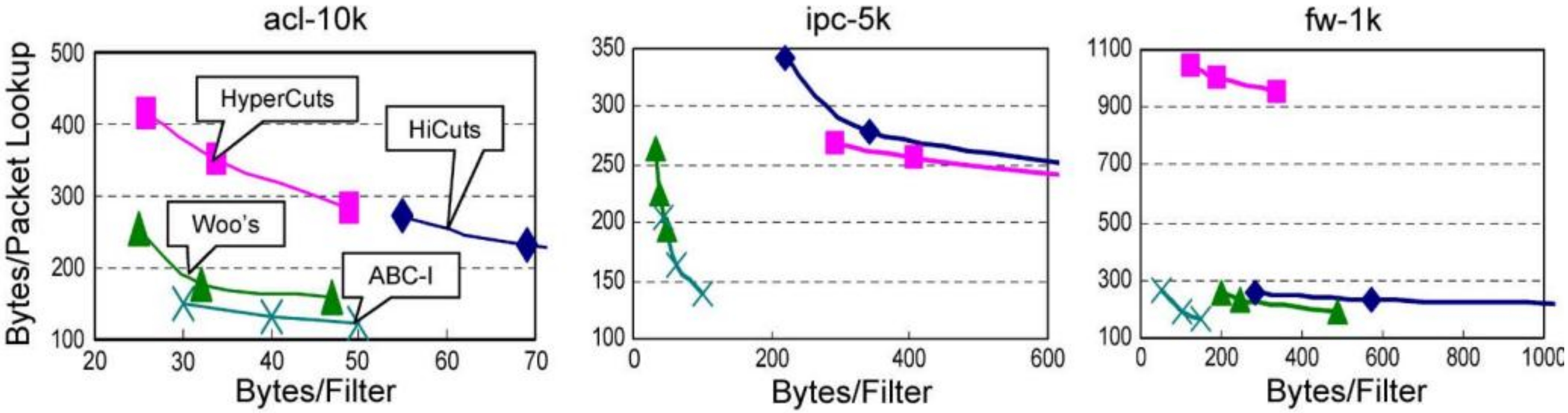


Fig. 15. Compare ABC to other DT-based algorithms.

Outline

- Background
- Related Work
- Observations
- Algorithm description
- Optimization
- Evaluation
- Conclusion

Conclusions

- ABC adopts variable sized cuts per decision step to even the filter distribution and reduce the filter duplication.
- ABC ensures all the DT nodes have the same size and are fully utilized.
- ABC is performance-guided: preset the storage budget and then look for best achievable throughput.
- ABC is scalable to large filter sets and is sufficient to sustain the real-time packet classification for 10-GbElines.

Thank you！