BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations

Minlan Yu
Princeton University
minlanyu@cs.princeton.edu
Joint work with Alex Fabrikant, Jennifer Rexford
Large-scale SPAF Networks

• Large network using flat addresses
  – Simple for configuration and management
  – Useful for enterprise and data center networks

• SPAF: Shortest Path on Addresses that are Flat
  – Flat addresses (e.g., MAC addresses)
  – Shortest path routing (link state, distance vector)
Scalability Challenges

• Recent advances in control plane
  – E.g., TRILL, SEATTLE
  – Topology and host information dissemination
  – Route computation

• Data plane remains a challenge
  – Forwarding table growth (in # of hosts and switches)
  – Increasing link speed
State of the Art

• Hash table in SRAM to store forwarding table
  – Map MAC addresses to next hop
  – Hash collisions: extra delay

• Overprovision to avoid running out of memory
  – Perform poorly when out of memory
  – Difficult and expensive to upgrade memory
Bloom Filters

• Bloom filters in fast memory (SRAM)
  – A compact data structure for a set of elements
  – Calculate $s$ hash functions to store element $x$
  – Easy to check membership
  – Reduce memory at the expense of false positives
BUFFALO: Bloom Filter Forwarding

• One Bloom filter (BF) per next hop
  – Store all addresses forwarded to that next hop

Packet destination → query

Nexthop 1
Nexthop 2
……
Nexthop T

Bloom Filters

hit
BUFFALO Challenges

How to optimize memory usage?
• Minimize the false-positive rate

How to handle false positives quickly?
• No memory/payload overhead

How to handle routing dynamics?
• Make it easy and fast to adapt Bloom filters
BUFFALO Challenges

How to optimize memory usage?
• Minimize the false-positive rate

How to handle false positives quickly?
• No memory/payload overhead

How to handle routing dynamics?
• Make it easy and fast to adapt Bloom filters
Optimize Memory Usage

• Consider fixed forwarding table
• **Goal:** Minimize overall false-positive rate
  – Probability one or more BFs have a false positive
• **Input:**
  – Fast memory size \( M \)
  – Number of destinations per next hop
  – The maximum number of hash functions
• **Output:** the size of each Bloom filter
  – Larger BF for next-hops with more destinations
Constraints and Solution

• Constraints
  – Memory constraint
    • Sum of all BF sizes fast memory size $M$
  – Bound on number of hash functions
    • To bound CPU calculation time
    • Bloom filters share the same hash functions

• Proved to be a convex optimization problem
  – An optimal solution exists
  – Solved by IPOPT (Interior Point OPTimizer)
Minimize False Positives

- Forwarding table with 200K entries, 10 next hop
- 8 hash functions
- Optimization runs in about 50 msec
Comparing with Hash Table

• Save 65% memory with 0.1% false positives

• More benefits over hash table
  – Performance degrades gracefully as tables grow
  – Handle worst-case workloads well
BUFFALO Challenges

How to optimize memory usage?
• Minimize the false-positive rate

How to handle false positives quickly?
• No memory/payload overhead

How to handle routing dynamics?
• Make it easy and fast to adapt Bloom filters
False Positive Detection

- Multiple matches in the Bloom filters
  - One of the matches is correct
  - The others are caused by false positives
Handle False Positives

• Design goals
  – Should not modify the packet
  – Never go to slow memory
  – Ensure timely packet delivery

• BUFFALO solutions
  – Exclude incoming interface
    • Avoid loops in “one false positive” case
  – Random selection from matching next hops
    • Guarantee reachability with multiple false positives
One False Positive

• Most common case: one false positive
  – When there are multiple matching next hops
  – Avoid sending to incoming interface

• Provably at most a two-hop loop
  – Stretch $\leq Latency(AB) + Latency(BA)$
Multiple False Positives

• Handle multiple false positives
  – Random selection from matching next hops
  – Random walk on shortest path tree plus a few false positive links
  – To eventually find out a way to the destination
Stretch Bound

• **Provable expected stretch bound**
  – With $k$ false positives, proved to be at most $O(3^{k/3})$
  – Proved by random walk theories

• **However, stretch bound is actually not bad**
  – False positives are independent
  – Probability of $k$ false positives drops exponentially

• **Tighter bounds in special topologies**
  – For tree, expected stretch is polynomial in $k$
Stretch in Campus Network

When $fp=0.001\%$
99.9\% of the packets have no stretch.
0.0003\% packets have a stretch of shortest path length.

When $fp=0.5\%$, 0.0002\% packets have a stretch 6 times of shortest path length.

CCDF

Stretch normalized by shortest path length
BUFFALO Challenges

How to optimize memory usage?
• Minimize the false-positive rate

How to handle false positives quickly?
• No memory/payload overhead

How to handle routing dynamics?
• Make it easy and fast to adapt Bloom filters
Problem of Bloom Filters

• **Routing changes**
  – Add/delete entries in BFs

• **Problem of Bloom Filters (BF)**
  – Do not allow deleting an element

• **Counting Bloom Filters (CBF)**
  – Use a counter instead of a bit in the array
  – CBFs can handle adding/deleting elements
  – But, require more memory than BFs
Update on Routing Change

- Use CBF in slow memory
  - Assist BF to handle forwarding-table updates
  - Easy to add/delete a forwarding-table entry
Occasionally Resize BF

- Under significant routing changes
  - # of addresses in BFs changes significantly
  - Re-optimize BF sizes
- Use CBF to assist resizing BF
  - Large CBF and small BF
  - Easy to expand BF size by contracting CBF
BUFFALO Switch Architecture

– Prototype implemented in kernel-level Click
Prototype Evaluation

• Environment
  – 3.0 GHz 64-bit Intel Xeon
  – 2 MB L2 data cache, used as fast memory size $M$

• Forwarding table
  – 10 next hops
  – 200K entries
Evaluation Results

• Peak forwarding rate
  – 365 Kpps, 1.9 μs per packet
  – 10% faster than hash-based EtherSwitch

• Performance with FIB updates
  – 10.7 μs to update a route
  – 0.47 s to reconstruct BFs based on CBFs
    • On another core without disrupting packet lookup
    • Swapping in new BFs is fast
Conclusion

• Three properties of BUFFALO
  – Small, bounded memory requirement
  – Gracefully increase stretch with the growth of forwarding table
  – Fast reaction to routing updates

• Key design decisions
  – One Bloom filter per next hop
  – Optimizing of Bloom filter sizes
  – Preventing forwarding loops
  – Dynamic updates using counting Bloom filters
• Thanks!

• Questions?