# Compressing IP Forwarding Tables: Towards Entropy Bounds and Beyond

**Revised on Feb 10, 2014**

Gábor Rétvári, János Tapolcai, Attila Kőrösi, András Majdán, Zalán Heszberger

Budapest Univ. of Technology and Economics
Dept. of Telecomm. and Media Informatics
{retvari,tapolcai,korosi,majdan,heszi}@tmit.bme.hu

*SIGCOMM'13, August 12–16, 2013, Hong Kong, China*

MŰEGYETEM 1782

# Motto

IP forwarding table compression is boring. . .

but compressed data structures are beautiful!

# Encoding Strings

- Suppose we want to encode the string `"labanana"`
- Just $4$ symbols, so we can use $2$ bits per symbol

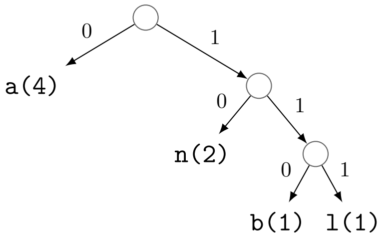
| symbol | code |
|---|---|
| a | 00 |
| b | 01 |
| l | 10 |
| n | 11 |

| l | a | b | a | n | a | n | a |
|---|---|---|---|---|---|---|---|
| 10 | 00 | 01 | 00 | 11 | 00 | 11 | 00 |

- Size is information-theoretic limit: 16 bits
- Fast access to symbol at any position, fast search, etc.
- But this format is not particularly memory efficient

# Huffman Coding

- Compression by encoding popular symbols on fewer bits
- Huffman tree sorted by symbol frequencies



# Huffman Coding

- Compression by encoding popular symbols on fewer bits
- Huffman tree sorted by symbol frequencies
- Use tree-prefix as symbol code

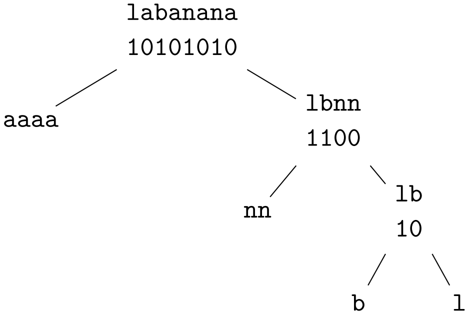
| symbol | code |
|---|---|
| a | 0 |
| b | 110 |
| l | 111 |
| n | 10 |

| l | a | b | a | n | a | n | a |
|---|---|---|---|---|---|---|---|
| 111 | 0 | 110 | 0 | 10 | 0 | 10 | 0 |

- Size is $nH_0$ bits, where $n$ is length and $H_0$ is entropy
- Only 14 bits, minimal for a zero order source
- But **no fast access to symbols, no search!**

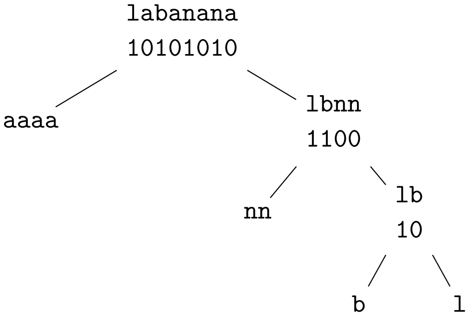
# Wavelet Trees

- Indexing and Huffman coding simultaneously
- A bitmap at each node of the Huffman tree
- Tells whether symbol belongs to the left/right branch



# Wavelet Trees: Access

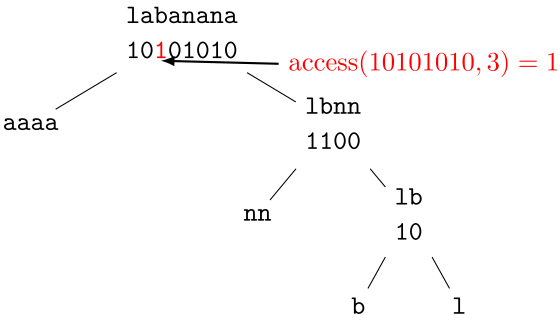
- Store bitmaps in **succinct bitstring indexes** (e.g., RRR)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$
- E.g., accessing the $3$rd position



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., `RRR`)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

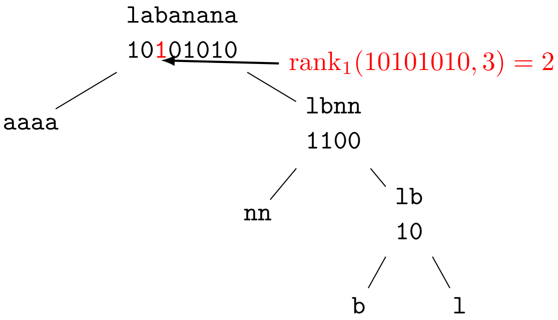
1. “Which branch the $3$rd symbol belongs to?”



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., RRR)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

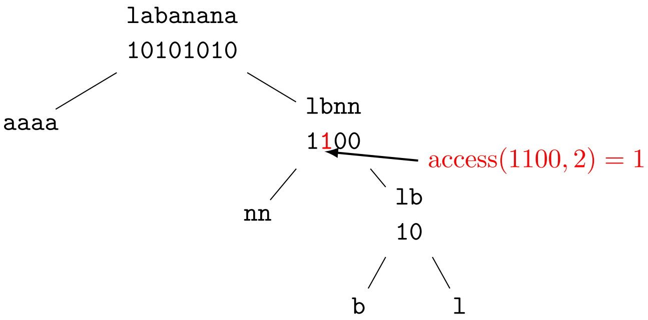
2. "How many symbols from this branch occurred this far?"



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., RRR)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

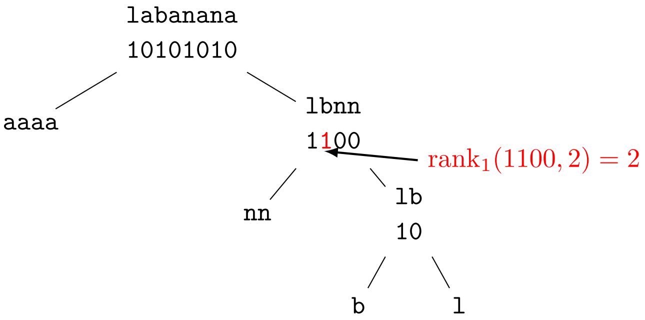
3. "Which branch this symbol belongs to?"



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., `RRR`)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

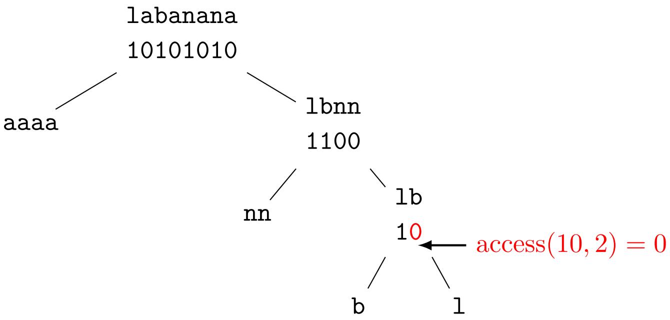
4. "How many symbols from this branch occurred this far?"



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., RRR)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

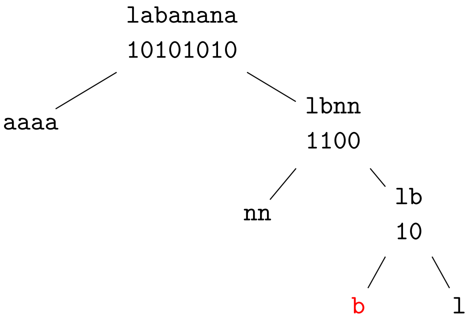
5. "Which of the remaining two symbols is the result?"



# Wavelet Trees: Access

- Store bitmaps in **succinct bitstring indexes** (e.g., `RRR`)
  - encode an $n$ bit long bitmap on roughly $n$ bits
  - support access/rank queries in $O(1)$

6. The $3$rd symbol is `b`



# Wavelet Trees: Size

- We only store the bitmaps at each level

labanana
10101010

lbnn
1100

lb
10

l a b a n a n a
111 0 110 0 10 0 10 0

- Every symbol appears with its Huffman code
- Size is $nH_0$ bits (plus negligible overhead)
- But we still have efficient access

# Compressed Data Structures

- **Compression not necessarily sacrifices fast access!**
- Store information in entropy-bounded space **and** provide fast in-place access to it
  - take advantage of regularity, if any, to compress
  - data drifts closer to the CPU in the cache hierarchy
  - operations are even faster than on the original uncompressed form
- No space-time trade-off!
- **This paper: advocate compressed data structures to the networking community**
- IP forwarding table compression as a use case

# IP Forwarding Information Base

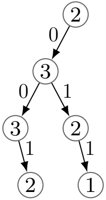
- The fundamental data structure used by IP routers to make forwarding decisions
- Stores more than 440K IP-prefix-to-nexthop mappings as of January, 2013
  - consulted on a packet-by-packet basis at line speed
  - queries are complex: longest prefix match
  - updated couple of hundred times per second
  - takes several MBytes of fast line card memory and counting
- May or may not become an Internet scalability barrier

# Prefix Trees

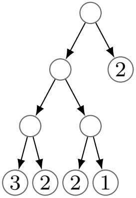
- Tries are the most convenient way to store IP FIBs

| prefix | label |
|---|---|
| -/0 | 2 |
| 0/1 | 3 |
| 00/2 | 3 |
| 001/3 | 2 |
| 01/2 | 2 |
| 011/3 | 1 |

FIB




Prefix tree




Prefix-free trie

# FIB Space Bounds

- A FIB can be uniquely represented by a binary prefix-free trie $T$
- Let $T$ have $n$ leaves labeled from an alphabet of size $\delta$ with Shannon-entropy $H_0$
- The **information-theoretic lower bound** to encode $T$ is

$$2n + n \log_2 \delta \text{ bits}$$

- The **zero-order entropy** of $T$ is

$$2n + nH_0 \text{ bits}$$

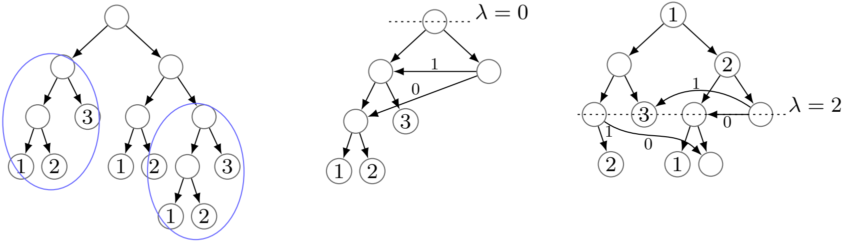
- The tree structure imposes an additive term $2n$ to the string size $nH_0$

# Static Compressed FIBs: `XBW-l`

- Apply the state-of-the-art in compressed data structures
  - convert FIB to prefix-free form
  - serialize the prefix tree into a set of strings
  - compress using wavelet trees and `RRR`
- We call the resultant data structure `XBW-l`
  + realizes the zero-order entropy bound
  + in fact, also attains higher-order entropy
  + lookup goes in $O(\log n)$ time
  – but update is linear
  – lookup is too slow for practical applications
- Problem turns out that `XBW-l` is **pointerless**

# Dynamic FIBs: Trie-folding

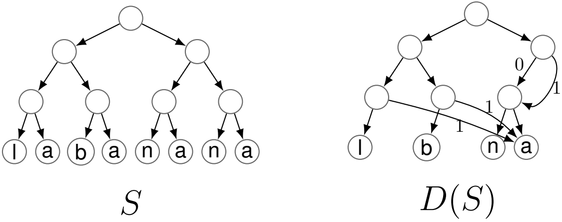
- Practical FIB compression, a good old **pointer machine**
- Fold the trie into a **prefix DAG** (DAFSA, DAWG, BDD)




- For good compression, we need the tree to be in a prefix-free form
- But prefix-free forms are expensive to update
- Balance by a parameter $\lambda$, called the leaf-push barrier

# Prefix DAG Size

- View the problem as string compression: encode a string $S$ into a prefix DAG $D(S)$




- **Theorem 1:** $D(S)$ needs $5n \log_2 \delta$ bits at most
- **Theorem 2:** $D(S)$ can be squeezed into $\sim 7nH_0$ bits in expectation
- **Theorem 3:** update goes in $O((1 + {}^1\!/_{H_0}) \log n)$ steps

# Evaluation

| FIB | $N$ | $\delta$ | $H_0$ | $I$ | $E$ | XBW-l | pDAG | $\mu$ |
|---|---|---|---|---|---|---|---|---|
| taz | 410K | 4 | 1.00 | 94KB | 56KB | 63KB | 178KB | 3.17 |
| access(d) | 444K | 28 | 1.06 | 206KB | 90KB | 100KB | 369KB | 4.1 |

- Entropy bound ($E$) is way smaller than information-theoretic limit ($I$): IP FIBs contain high regularity!
- XBW-l attains entropy bounds very closely, with prefix DAGs (pDAG) off by only a factor $\mu$ of $2$–$4$
- FIBs can be encoded on roughly $1$–$2$ bits per prefix(!)
  - that's roughly $100$–$400$ KBytes of memory
- Several million lookups per sec both in HW and SW
  - faster than the uncompressed form
- pDAG tolerates more than $100,000$ updates per sec

# Conclusions

- Compressed data structures are essential in information retrieval, computational biology, geometry, etc.
  - allow to sidestep notorious space-time trade-offs
  - as such, compressing comes essentially for free
- FIB compression is a poster child of why the networking field is in a sore need of good compression methods
  - permits to reason about size, lookup, and update performance (analyzability)
  - allows to state theoretical storage size bounds (predictability)
  - faster operations than on the uncompressed form (efficiency)