Counter Braids: A novel counter architecture

Balaji Prabhakar
Stanford University

Joint work with:
Yi Lu, Andrea Montanari, Sarang Dharmapurikar and Abdul Kabbani

Overview

- Counter Braids
  - Background: current approaches
    - Exact, per-flow accounting
    - Approximate, large-flow accounting
  - Our approach
    - The Counter Braid architecture
    - A simple, efficient message passing algorithm
  - Performance, comparisons and further work

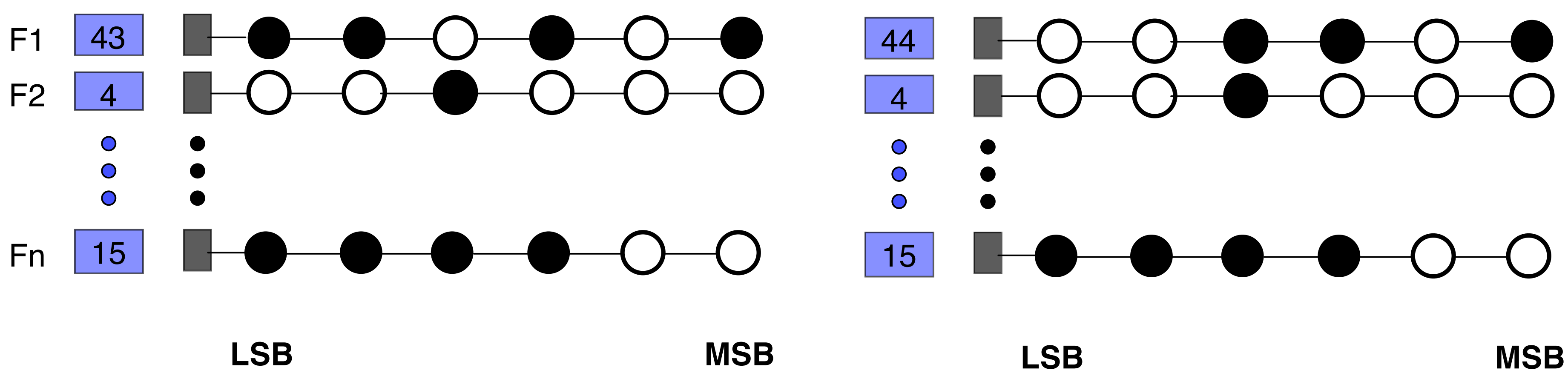
- Congestion notification in Ethernet
  - Overview of IEEE standards effort

Traffic Statistics: Background

- Routers collect traffic statistics; useful for
  - Accounting/billing, traffic engineering, security/forensics
  - Several products in this area; notably, Cisco's NetFlow, Juniper's cflowd, Huawei's NetStream
- Other areas
  - In databases: number and count of distinct items in streams
  - Web server logs
- Key problem: At high line rates, memory technology is a limiting factor
  - 500,000+ active flows, packets arrive once every 10 ns on 40 Gbps line
  - We need *fast* and *large* memories for implementing counters: v.expensive
- This has spawned two approaches
  - Exact, per-flow accounting: Use hybrid SRAM-DRAM architecture
  - Approximate, large-flow accounting: Use heavy-tailed nature of flow size distribution

Per-flow Accounting

- Naïve approach: one counter per flow

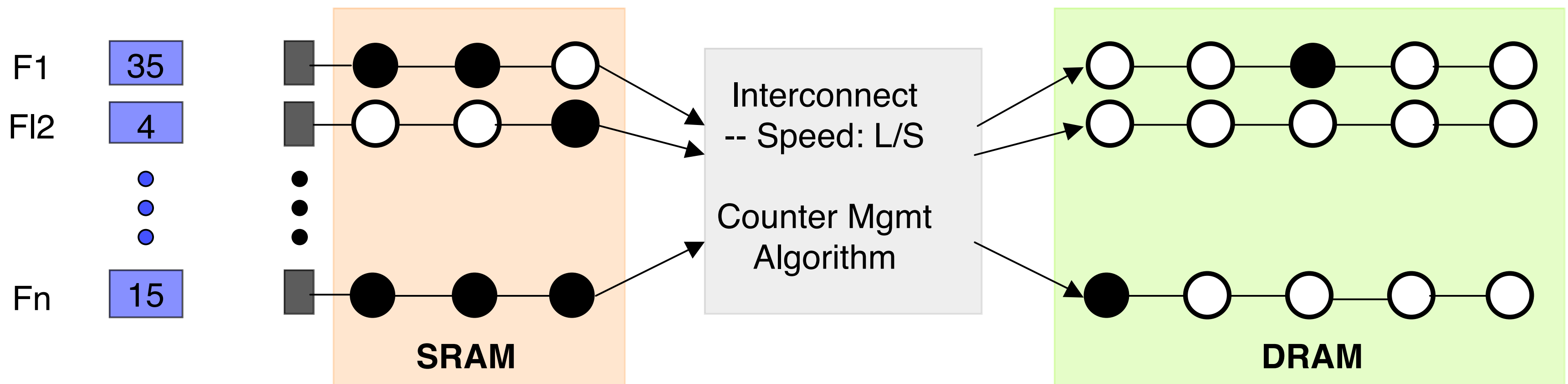


- Problem: Need fast and large memories; infeasible

An initial approach

Shah, Iyer, Prabhakar, McKeown (2001)

- Hybrid SRAM-DRAM architecture
  - LSBs in SRAM: high-speed updates, on-chip
  - MSBs in DRAM: less frequent updates; can use slower speed, off-chip DRAMs

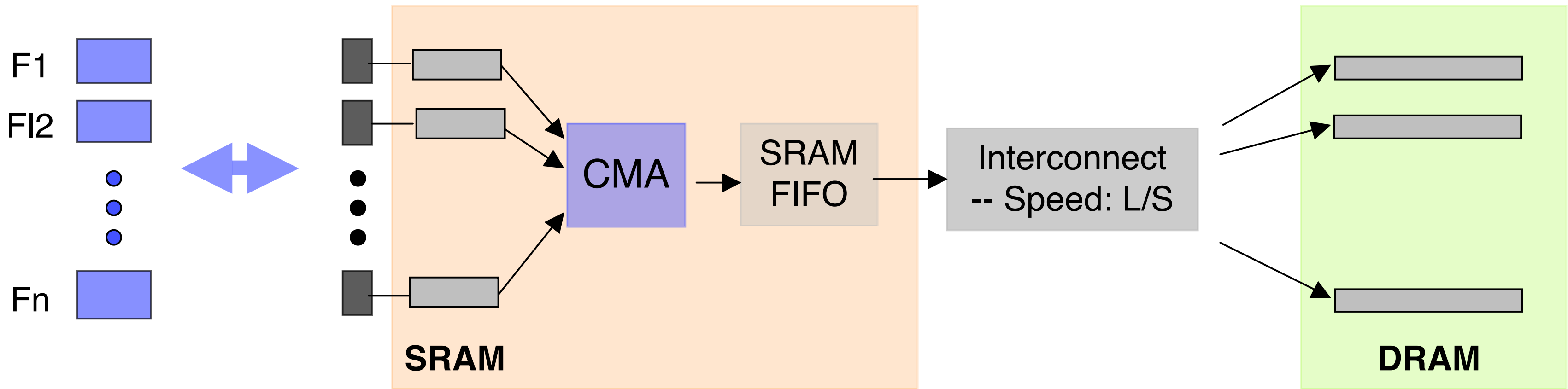


- The setup
  - Line speed = SRAM speed = L; Interconnect speed = DRAM speed = L/S
  - Adversarial packet arrival process

- Results
  1. The counter management algorithm Longest Counter First is optimal
  2. Min. num. of bits for each SRAM counter: $\log\left(\frac{\log(SN)}{\log(S/S-1)}\right) \approx \log\log N$

Related work

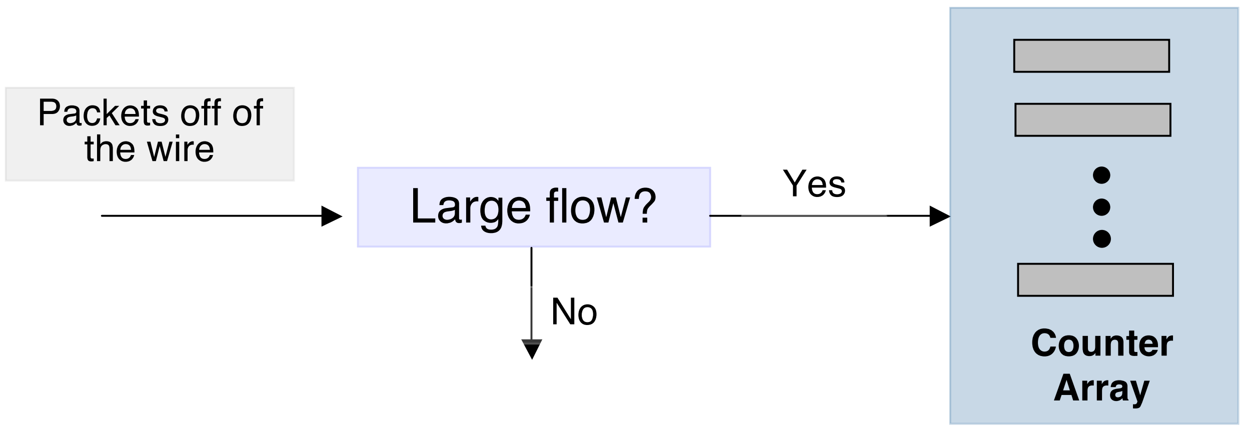
- Ramabhadran and Varghese (2003) obtained a simpler version of the LCF algorithm
- Zhao et al (2006) randomized the initial values in the SRAM counters to prevent the adversary from causing several counters to overflow closely



- Main problem of exact methods
  - Can't fit counters into single SRAM
  - Need to know the flow-counter association
    - Need perfect hash function; or, fully associative memory (e.g. CAM)

Approximate counting

- Statistical in nature
  - Use heavy-tailed (often Pareto) distribution of network flow sizes
  - Roughly, 80% of data brought by the biggest 20% of the flows
  - So, it makes sense to quickly identify these big flows and count their packets
- Sample and hold: Estan et al (2004) propose sampling packets to catch the large “elephant” flows and then counting just their packets
  - Significantly simpler, but approximate



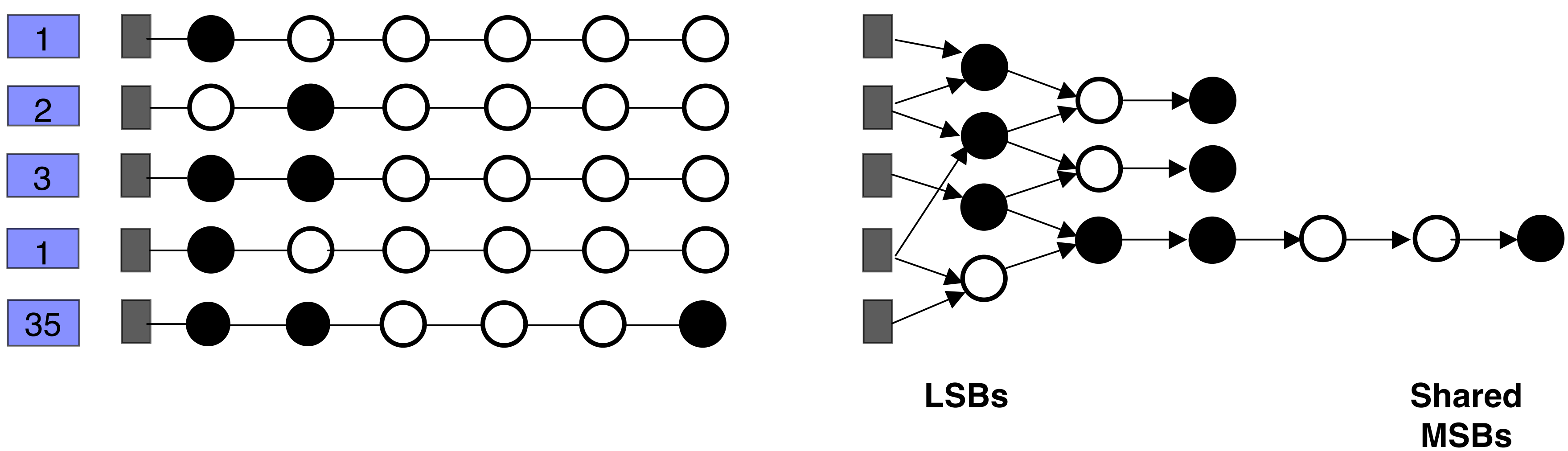
- This approach spawned a lot of follow-on work
  - Given the cost of memory, it strikes an excellent trade-off
  - Moreover, the flow-to-counter association problem is manageable

Summary

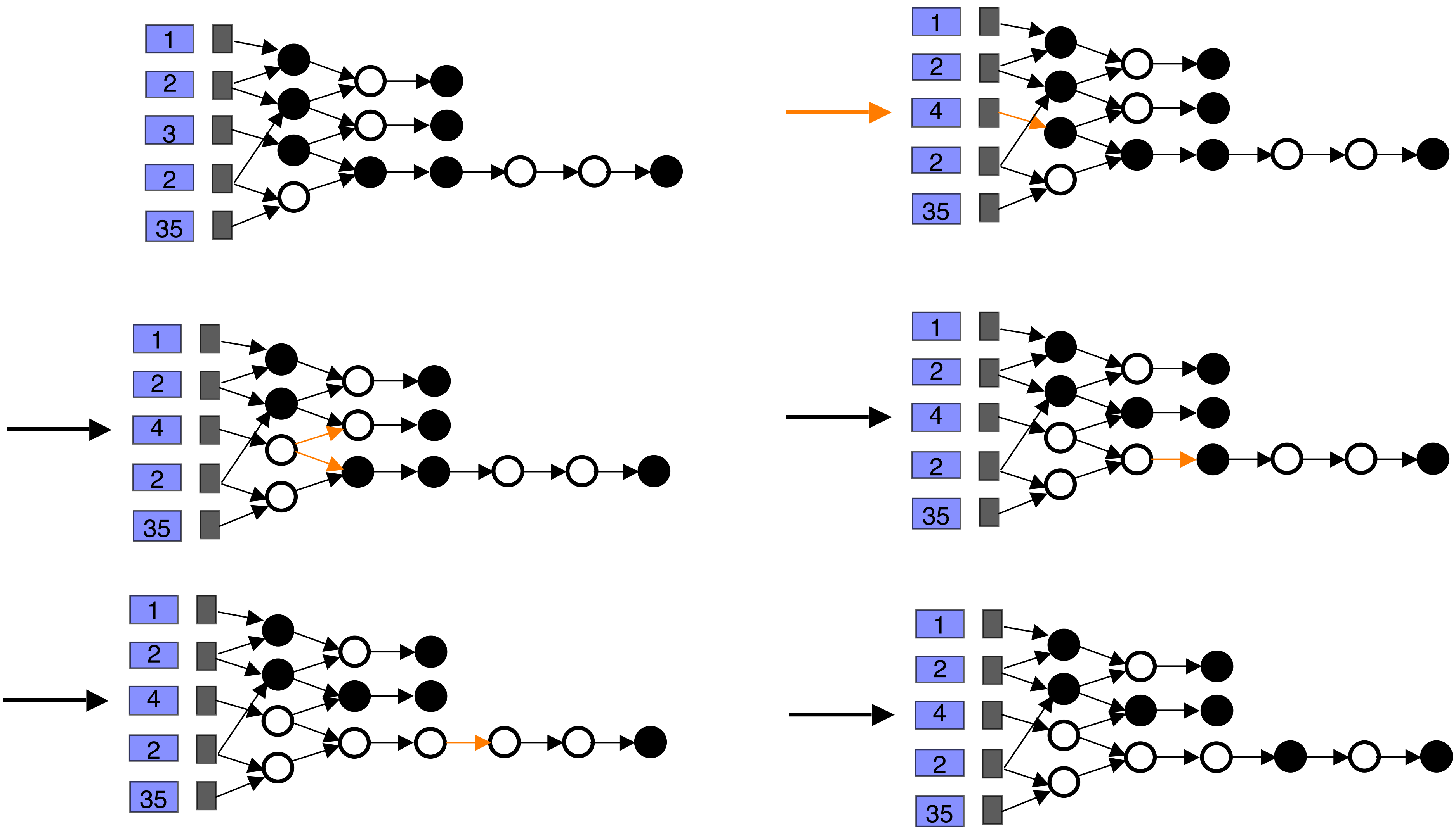
- Exact counting methods
  - Space intensive
  - Complex
- Approximate methods
  - Focus on large flows
  - Not as accurate

Our approach

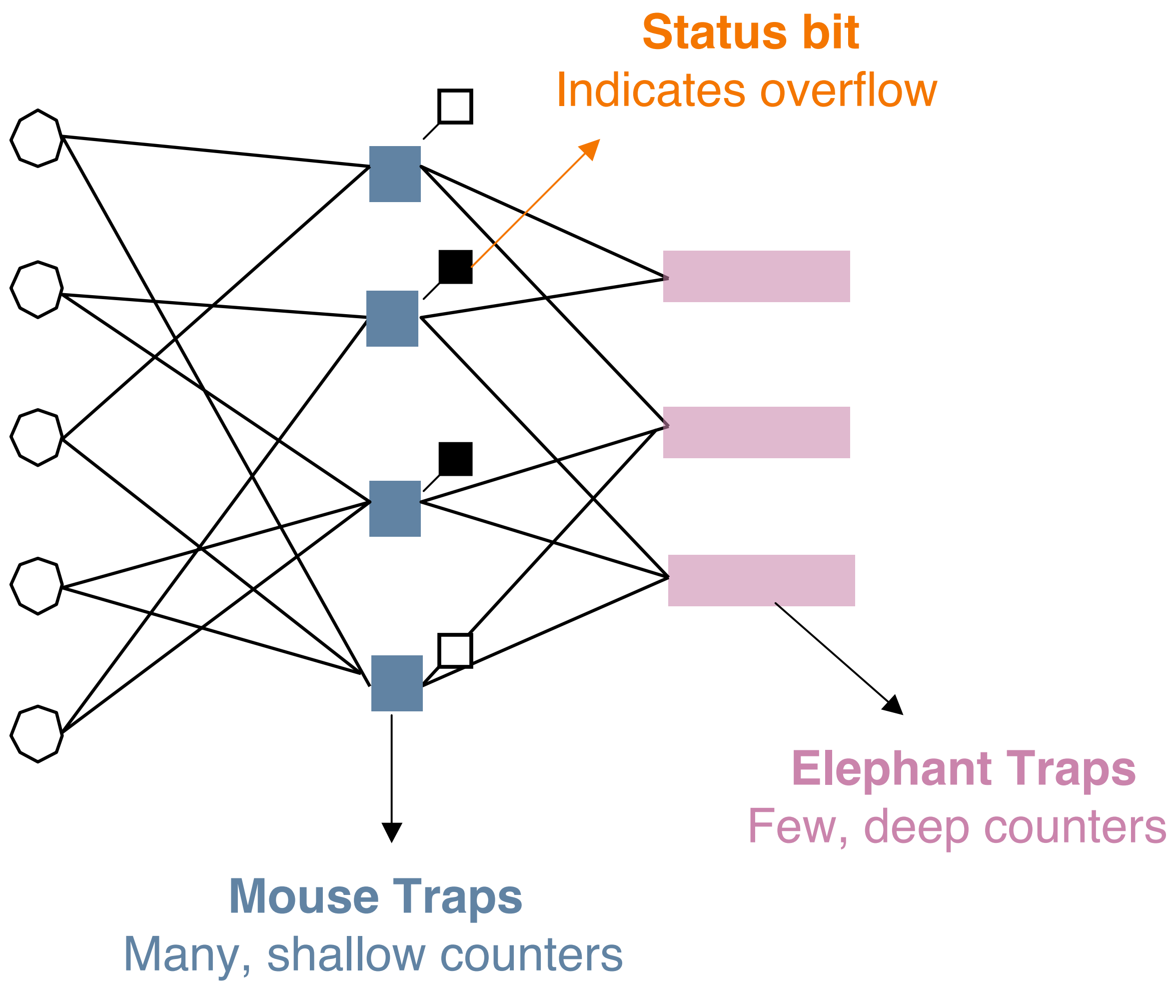
- The two problems of exact counting methods solved as follows
  1. Large counter space
     – By "braiding" the counters
  2. Flow-to-counter association problem
     – By using multiple hash functions and a "decoder"
- Braiding



Incrementing

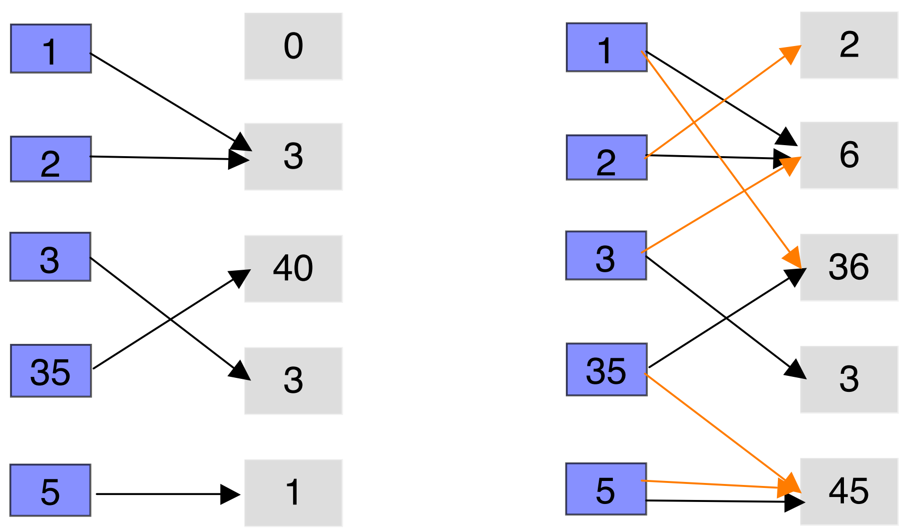


Counter Braids for Measurement
(in anticipation)



Flow-to-counter association

- Multiple hash functions
  - Single hash function leads to collisions
  - However, one can use *two* hash functions and use the redundancy to recover the flow size



- Find flow sizes from counter values; i.e. solve C = MF
  - Need a decoding algorithm
  - It's performance: how much space? what decoding accuracy?

Optimality

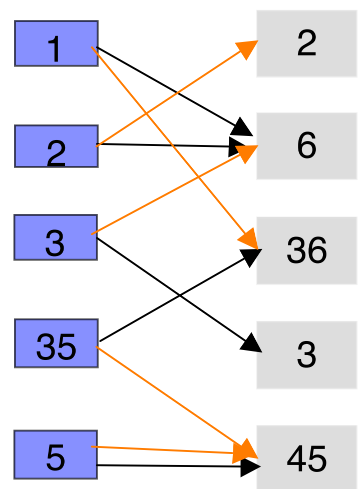
- Counter Braids are optimal, i.e.
  - When using the maximum likelihood (ML) decoder, the space needed for the counters reaches the *entropy lower bound*

- The ML decoder
  - Let $F^1, \ldots, F^k$ be the list of all solutions to $C = MF$
  - F^{ML} is that solution which is most likely

- This is interesting because C is a *linear, incremental* function of the data, F
  - By contrast, the Lempel-Ziv compressor, which is also optimal, is a non-linear function of data
  - However, the ML decoder is NP-hard in general; need something simpler

The Count-Min Algorithm

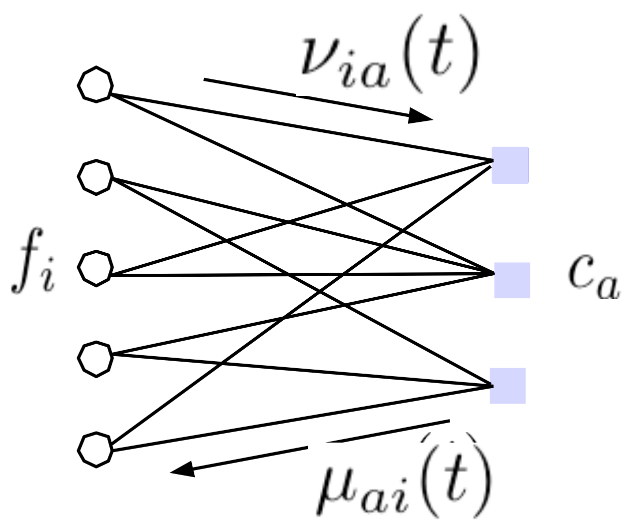
- Let us first look at this algorithm is due to Cormode and Muthukrishnan
  - Algorithm:
    - Hash flow j to multiple counters, increment all of them
    - Estimate flow j's size as the minimum counter it hits
  - The flow sizes for the example below would be estimated as: 6, 2, 3, 36, 45



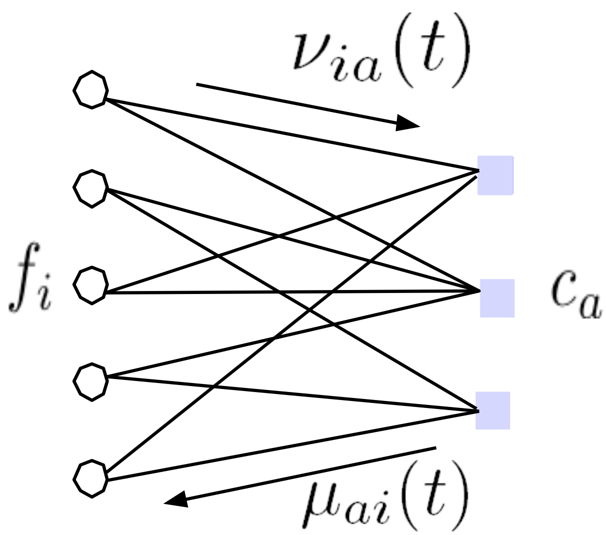
- Major drawbacks
  - Need lots of counters for accurate estimation
  - Don't know how much the error is; in fact, don't know ***if*** there is an error
- We shall see that applying the "Turbo-principle" to this algorithm gives terrific results

Decoder 2: The MP estimator

- An Iterative Message Passing Decoder
  - For solving the system of (underdetermined) linear equations: C = MF
  - Messages in the t^{th} iteration
    - $\mu_{ai}(t)$ from counter *a* to flow *i*: estimate of flow *i*'s size by counter *a* based on messages from flow's ***other than*** *i*
    - $\nu_{ia}(t)$ from flow *i* to counter *a*: flow *i*'s estimate of its own size based on messages from counters ***other than*** *a*



The MP Estimator



Initialize

min = minimum possible flow size;

$\nu_{ia}(0) = min\ \forall i \in I$ and $a \in R$;

$c_a = a^{th}$ counter value

Iterations

for iteration number $t = 1$ to $niter$

$$\mu_{ai}(t) = \max\left\{\left(c_a - \sum_{j \neq i} \nu_{ja}(t-1)\right), min\right\};$$

$$\nu_{ia}(t) = \begin{cases} \min_{b \neq a} \mu_{bi}(t) & \text{if } t \text{ is odd,} \\ \max_{b \neq a} \mu_{bi}(t) & \text{if } t \text{ is even.} \end{cases}$$

Final Estimate

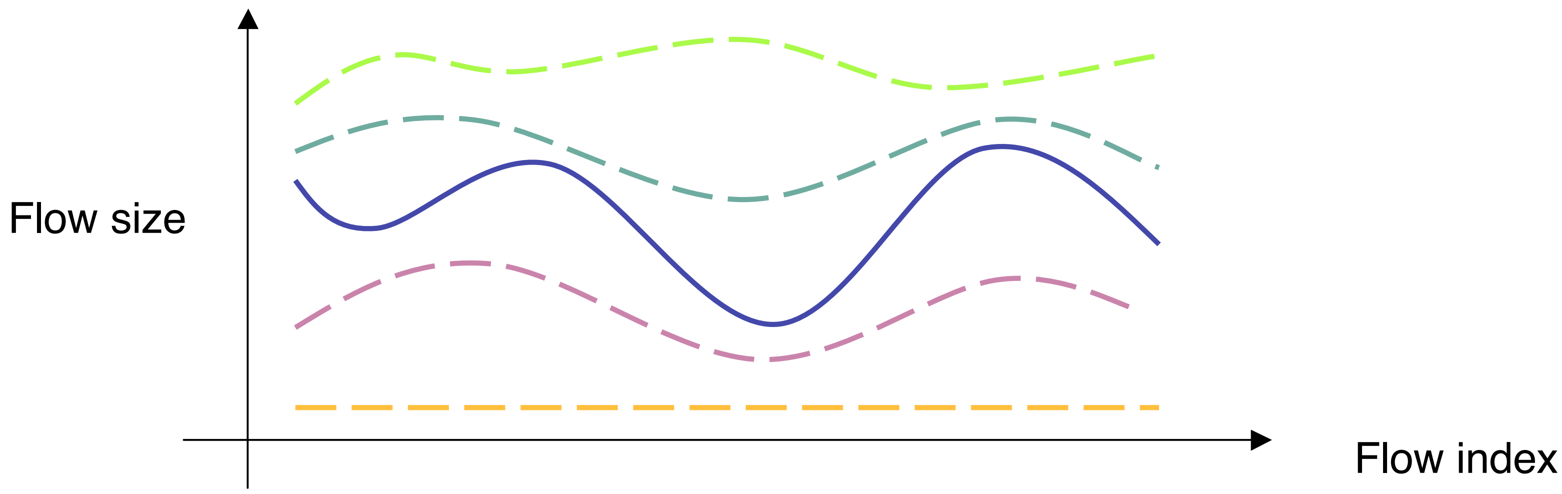
$$\widehat{f}_i(t) = \begin{cases} \min_a\{\mu_{ai}(niter)\} & \text{if } t \text{ is odd,} \\ \max_a\{\mu_{ai}(niter)\} & \text{if } t \text{ is even.} \end{cases}$$

- Note: Count-min is just the first iteration of the algorithm if initial flow estimates are 0

Properties of the MP Algorithm

- Anti-monotonicity: With initial estimates of 1 for the flow sizes,

$$\hat{f}_i(2t) \leq \hat{f}_i(2t+2) \leq ... \leq f_i \leq ... \leq \hat{f}_i(2t+3) \leq \hat{f}_i(2t+1)$$



- Note: Because of this property, estimation errors are both detectable and have a bound!

When does the sandwich close?

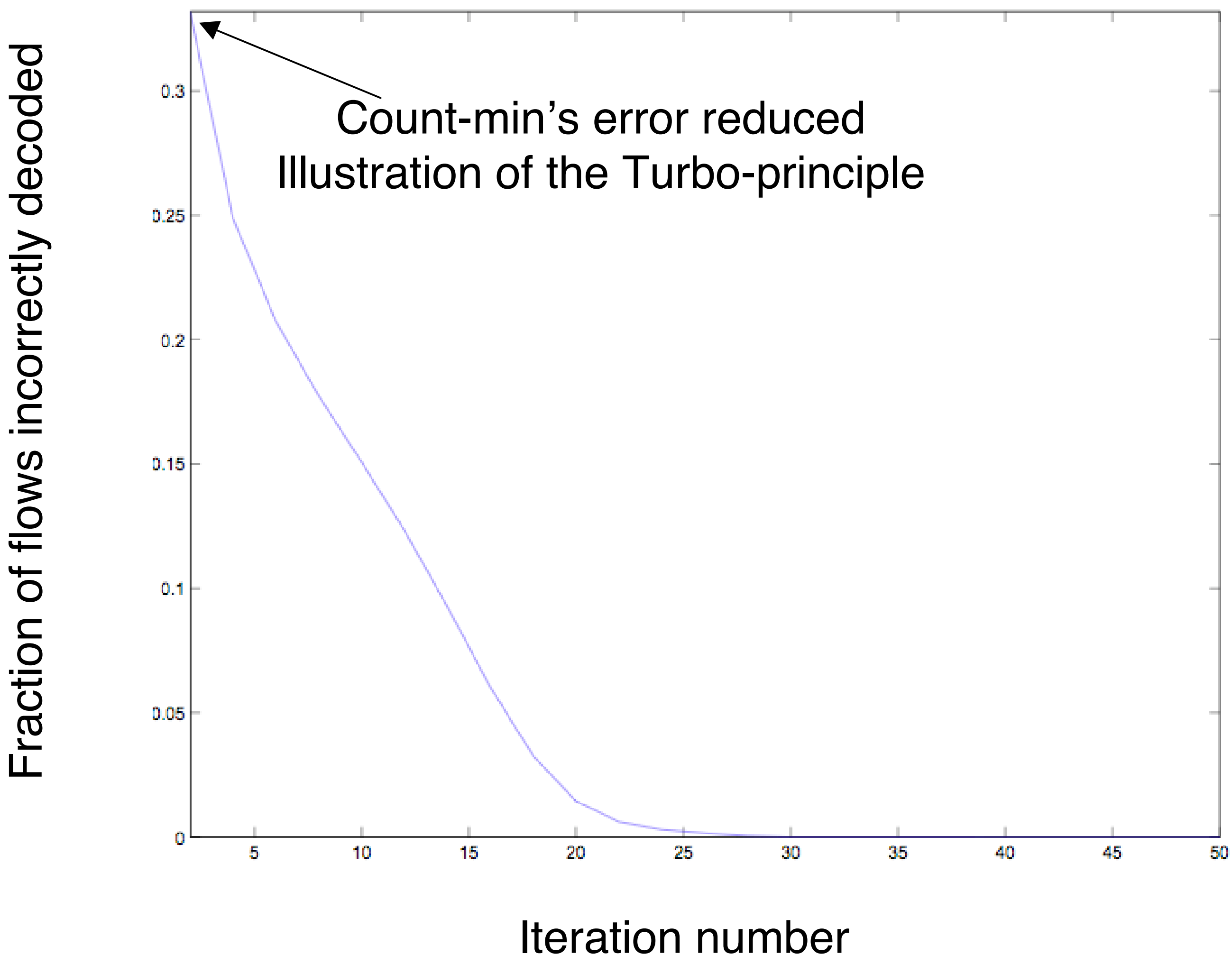
- Using the "density evolution" technique of Coding Theory, one can show that it suffices for m > c*n, where

$$c^* = \sqrt{P(f > min)}$$

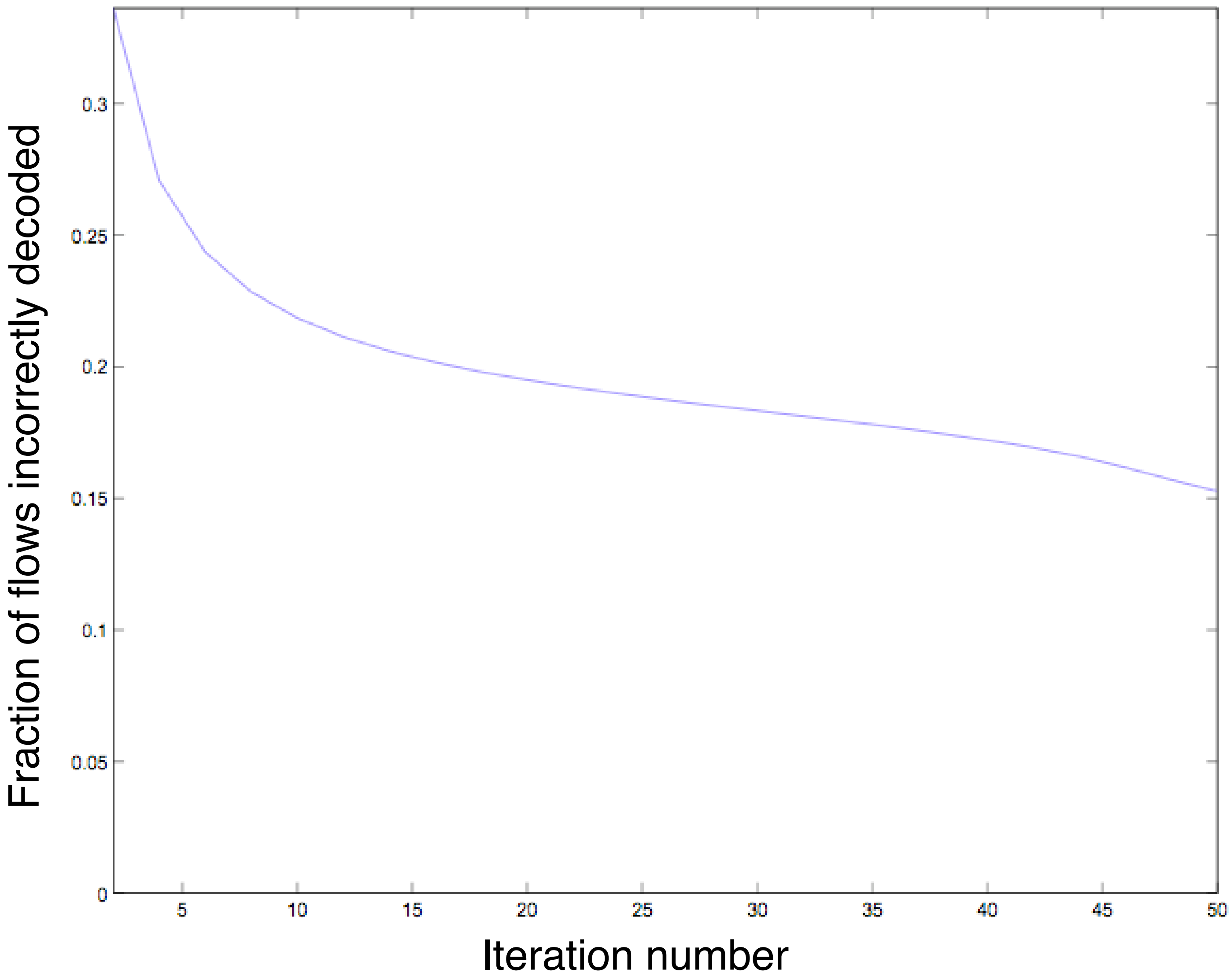
  - This means for heavy-tailed flow sizes, where there are approximately 35% 1-packet flows, c* is roughly 0.8

- In fact, there is a ***sharp threshold***
  - Less than that many counters means you cannot decode correctly, more is not required!

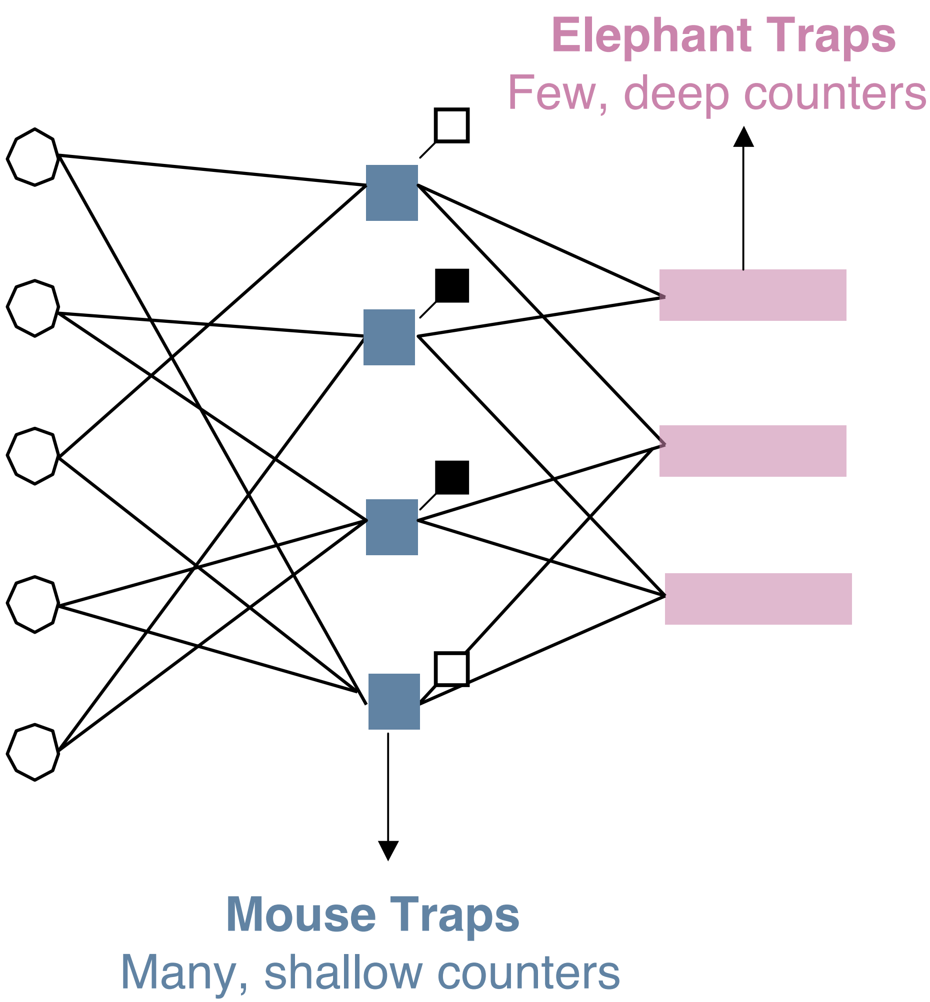
Above Threshold (= 72,000)
100,000 flows and 75,000 ctrs



Below Threshold
100,000 flows and 71,000 ctrs



The 2-stage Architecture: Counter Braids



-- First stage: Lots of shallow counters

-- Second stage: V.few deep counters

-- First stage counters hash into the second stage; an "overflow" status bit on first stage counters indicates if the counter has overflowed to the second stage

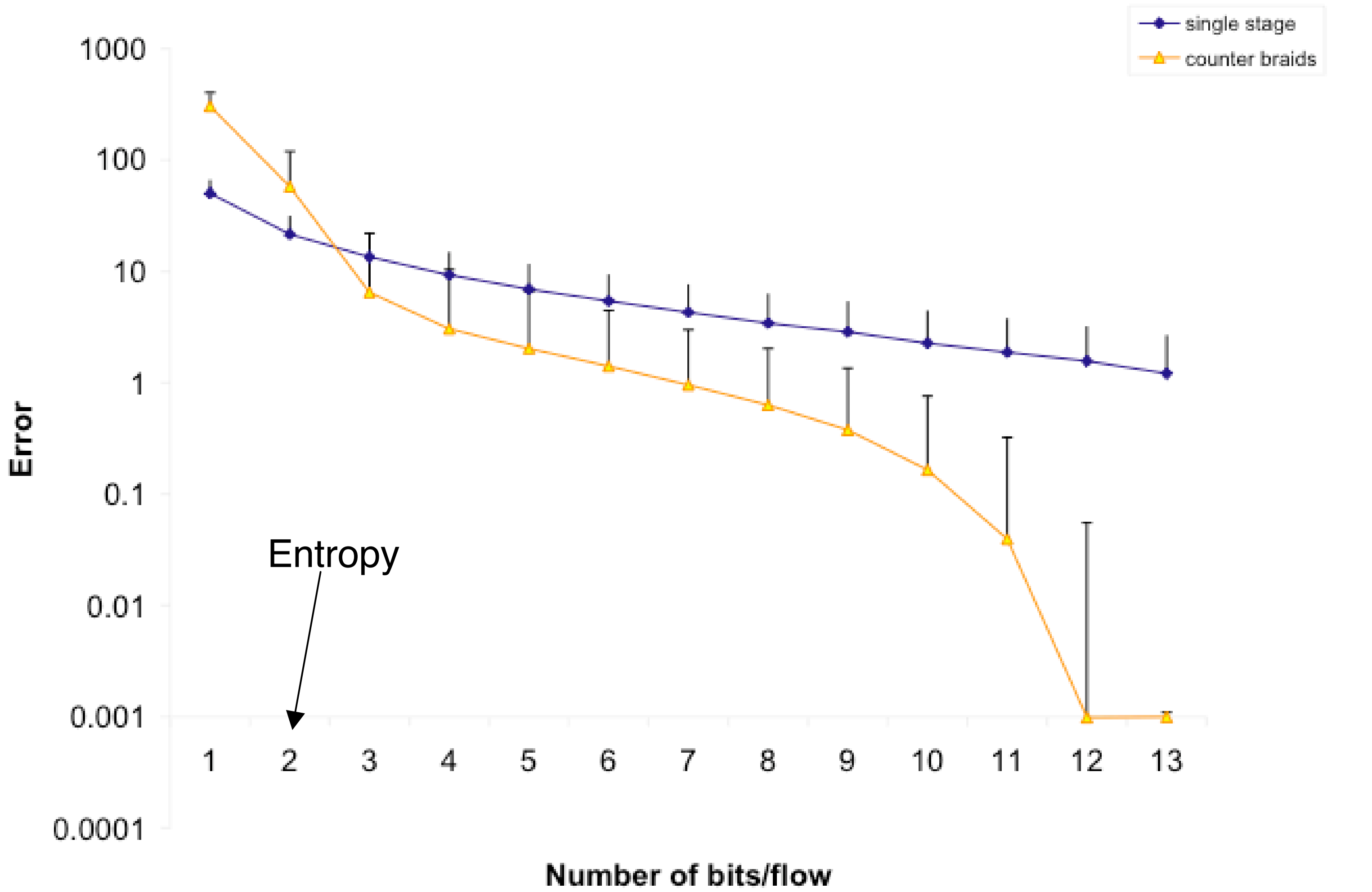
-- If a first stage counter overflows, it resets and counts again; second stage counters track most significant bits

-- Apply MP algorithm recursively

Performance of the MP Algorithm

- Interested in absolute error as a function of flow size
  - Pareto flow sizes
  - Entropy = 1.96 bits
  - Max flow size = 7364
  - Number of flows = 100,000

Counter Braids vs. the Single-stage Architecture



Internet trace simulations

- Used two OC-48 (2.5 Gbps) one-hour contiguous traces collected by CAIDA at a San Jose router.
- Divided traces into 12 5-minute segments. Each segment has 0.9 million flows and 20 million packets in trace 1, and 0.7 million flows and 9 million packets in trace 2.
- We used total counter space of 1.28 MB.
- We ran 50 experiments, each with different hash functions. There were a total of 1200 runs. ***No error*** was observed.

Comparison

	Hybrid	Sample-and-Hold	Counter Braids
Purpose	All flow sizes	Elephant Flows	All flow sizes
Number of flows	900,000	98,000	900,000
Memory Size (SRAM) for counters	4.5 Mbit (31.5 Mbit in DRAM + counter-management algorithm)	1 Mbit	10 Mbit
Memory Size (SRAM) flow-to-counter association	>25 Mbit (infeasible)	1.6 Mbit	Not needed
Error	Exact	Fractional Large: 0.03745% Medium: 1.090% Small: 43.87%	Lossless recovery. Pe ~ 10^(-7)

Conclusions for Counter Braids

- Cheap and accurate solution to the network traffic measurement problem
  - Message Passing Decoder
  - Counter Braids
- Initial results showed that the performance was quite good
- Further work
  - Multi-stage generalization of Counter Braids
  - Analyze MP algorithm
  - Multi-router solution: same flow passes through many routers

Congestion Notification in Ethernet: Part of the IEEE 802.1 Data Center Bridging standardization effort

Berk Atikoglu, Abdul Kabbani, Balaji Prabhakar
Stanford University
Rong Pan
Cisco Systems
Mick Seaman

Backgrond

- Switches and routers send congestion signals to end-systems to regulate the amount of network traffic.
  - We distinguish two types of congestion.
    - Transient: Caused by random fluctuations in the arrival rate of packets, and effectively dealt with using buffers and link-level pausing (or dropping packets in the case of the Internet).
    - Oversubscription: Caused by an increase in the applied load either because existing flows send more traffic, or (more likely) because new flows have arrived.
  - A congestion notification mechanism is concerned with dealing with the second type of congestion.
  - We've been involved in developing QCN (for Quantized Congestion Notification), an algorithm which is being studied as a part of the IEEE 802.1 Data Center Bridging group for deployment in Ethernet

Congestion control in the Internet

- In the Internet
  - Various queue management schemes, notably RED, drop or mark packets using ECN at the links
  - TCP at end-systems uses these congestion signals to vary the sending rate
  - There exists a rich history of algorithm development, control-theoretic analysis and detailed simulation of queue management schemes and congestion control algorithms for the Internet
    - Jacobson, Floyd et al, Kelly et al, Low et al, Srikant et al, Misra et al, Katabi et al …
    - The simulator ns-2

Switched Ethernet vs Internet

- Some significant differences …
  1. There is no end-to-end signaling in the Ethernet *a la* per-packet acks in the Internet
     - So congestion must be signaled to the source by switches
     - Not possible to know round trip time!
     - Algorithm not automatically self-clocked (like TCP)
  2. Links can be paused; i.e. packets may not be dropped
  3. No sequence numbering of L2 packets
  4. Sources do not start transmission gently (like TCP slow-start); they can potentially come on at the full line rate of 10Gbps
  5. Ethernet switch buffers are much smaller than router buffers (100s of KBs vs 100s of MBs)
  6. Most importantly, algorithm should be simple enough to be implemented completely in hardware

- An interesting environment to develop a congestion control algorithm
  - QCN derived from the earlier BCN algorithm
  - Closest Internet relatives: BIC TCP at source, REM/PI controller at switch