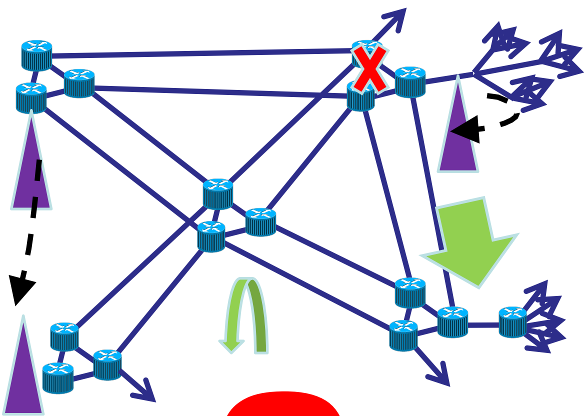
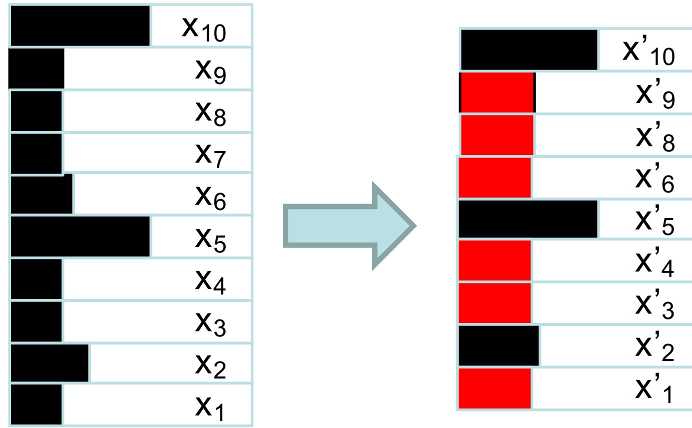
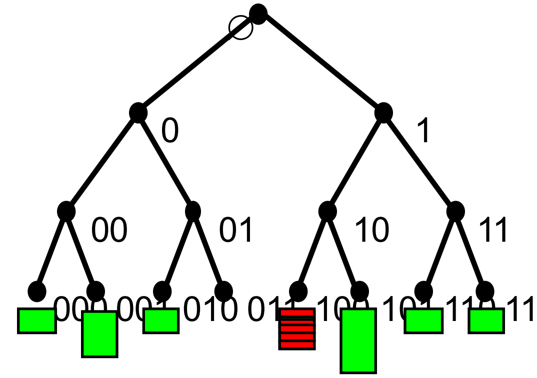
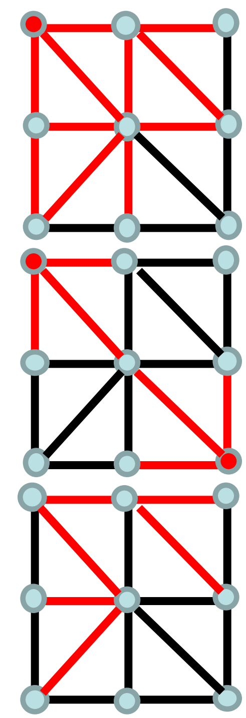
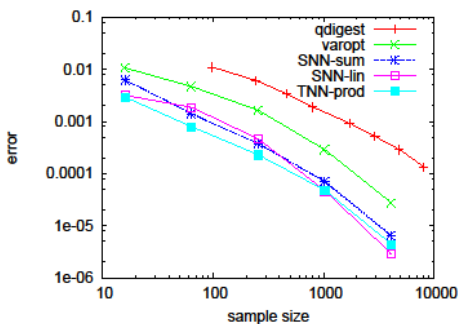






Sampling for Big Data



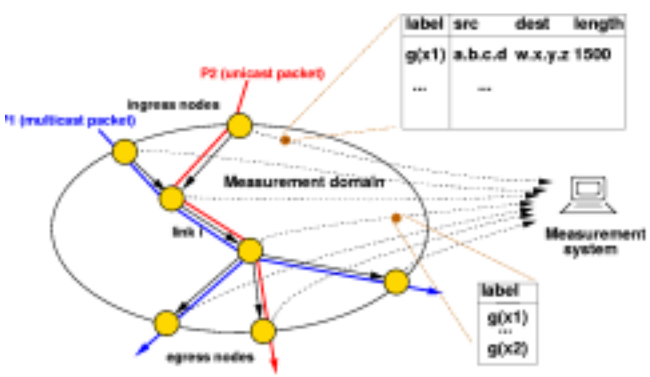


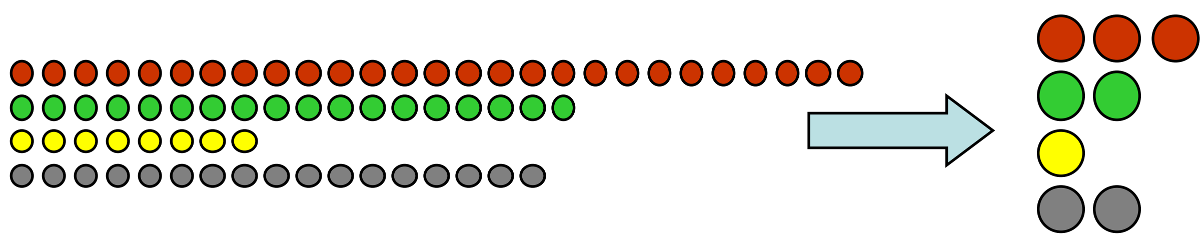
Graham Cormode, University of Warwick

G.Cormode@warwick.ac.uk

Nick Duffield, Texas A&M University

duffieldng@tamu.edu



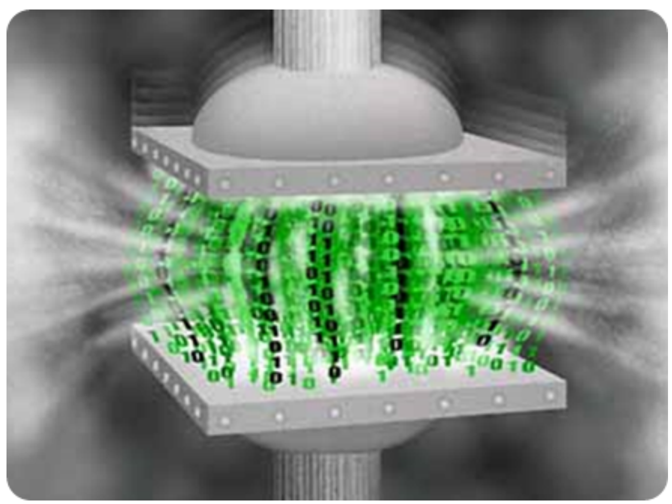


Big Data

- ◊ "Big" data arises in many forms:
  - Physical Measurements: from science (physics, astronomy)
  - Medical data: genetic sequences, detailed time series
  - Activity data: GPS location, social network activity
  - Business data: customer behavior tracking at fine detail
- ◊ Common themes:
  - Data is large, and growing
  - There are important patterns and trends in the data
  - We don't fully know where to look or how to find them

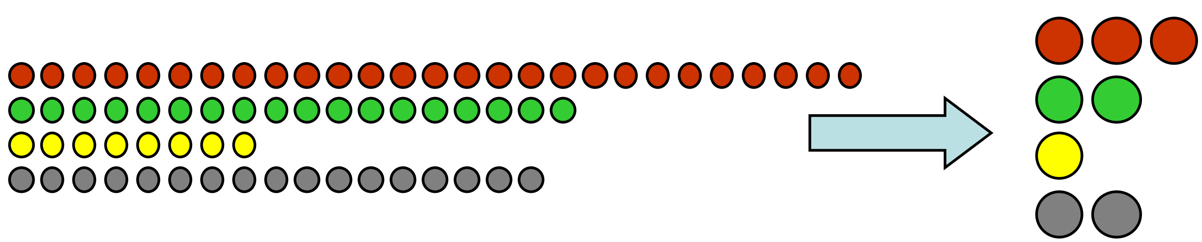
Why Reduce?

- ◊ Although "big" data is about more than just the volume…
  …most big data is big!
- ◊ It is not always possible to store the data in full
  - – Many applications (telecoms, ISPs, search engines) can't keep everything
- ◊ It is inconvenient to work with data in full
  - – Just because we can, doesn't mean we should
- ◊ It is faster to work with a compact summary
  - – Better to explore data on a laptop than a cluster



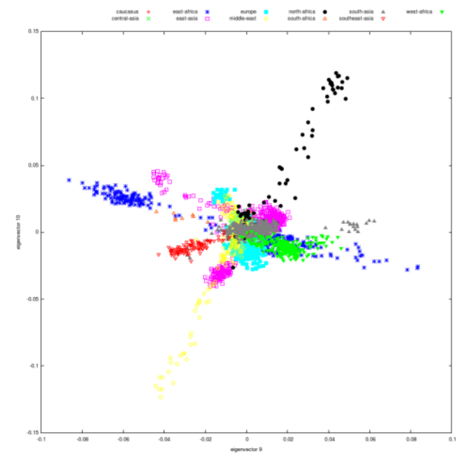
Why Sample?

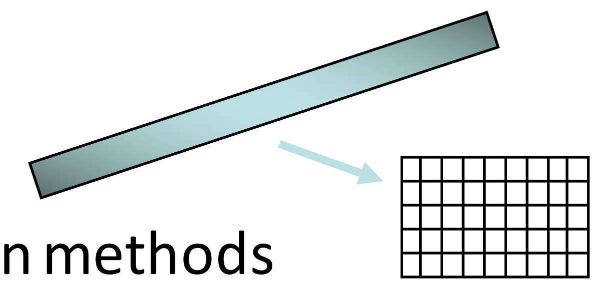
- Sampling has an intuitive semantics
  - We obtain a smaller data set with the same structure
- Estimating on a sample is often straightforward
  - Run the analysis on the sample that you would on the full data
  - Some rescaling/reweighting may be necessary
- Sampling is general and agnostic to the analysis to be done
  - Other summary methods only work for certain computations
  - Though sampling can be tuned to optimize some criteria
- Sampling is (usually) easy to understand
  - So prevalent that we have an intuition about sampling



Alternatives to Sampling

- Sampling is not the only game in town
  - Many other data reduction techniques by many names
- Dimensionality reduction methods
  - PCA, SVD, eigenvalue/eigenvector decompositions
  - Costly and slow to perform on big data
- “Sketching” techniques for streams of data
  - Hash based summaries via random projections
  - Complex to understand and limited in function
- Other transform/dictionary based summarization methods
  - Wavelets, Fourier Transform, DCT, Histograms
  - Not incrementally updatable, high overhead
- All worthy of study – in other tutorials





Health Warning: contains probabilities

- Will avoid detailed probability calculations, aim to give high level descriptions and intuition
- But some probability basics are assumed
  - Concepts of probability, expectation, variance of random variables
  - Allude to concentration of measure (Exponential/Chernoff bounds)
- Feel free to ask questions about technical details along the way

$$\begin{aligned}
\operatorname{var}\left(\frac{k}{n}\right) &= \mathrm{E}\left[\operatorname{var}\left(\frac{k}{n}\middle|\theta\right)\right] + \operatorname{var}\left[\mathrm{E}\left(\frac{k}{n}\middle|\theta\right)\right] \\
&= \mathrm{E}\left[\left(\frac{1}{n}\right)\theta(1-\theta)\middle|\mu, M\right] + \operatorname{var}\left(\theta|\mu, M\right) \\
&= \frac{1}{n}\left(\mu(1-\mu)\right) + \frac{n-1}{n}\frac{(\mu(1-\mu))}{M+1} \\
&= \frac{\mu(1-\mu)}{n}\left(1+\frac{n-1}{M+1}\right).
\end{aligned}$$

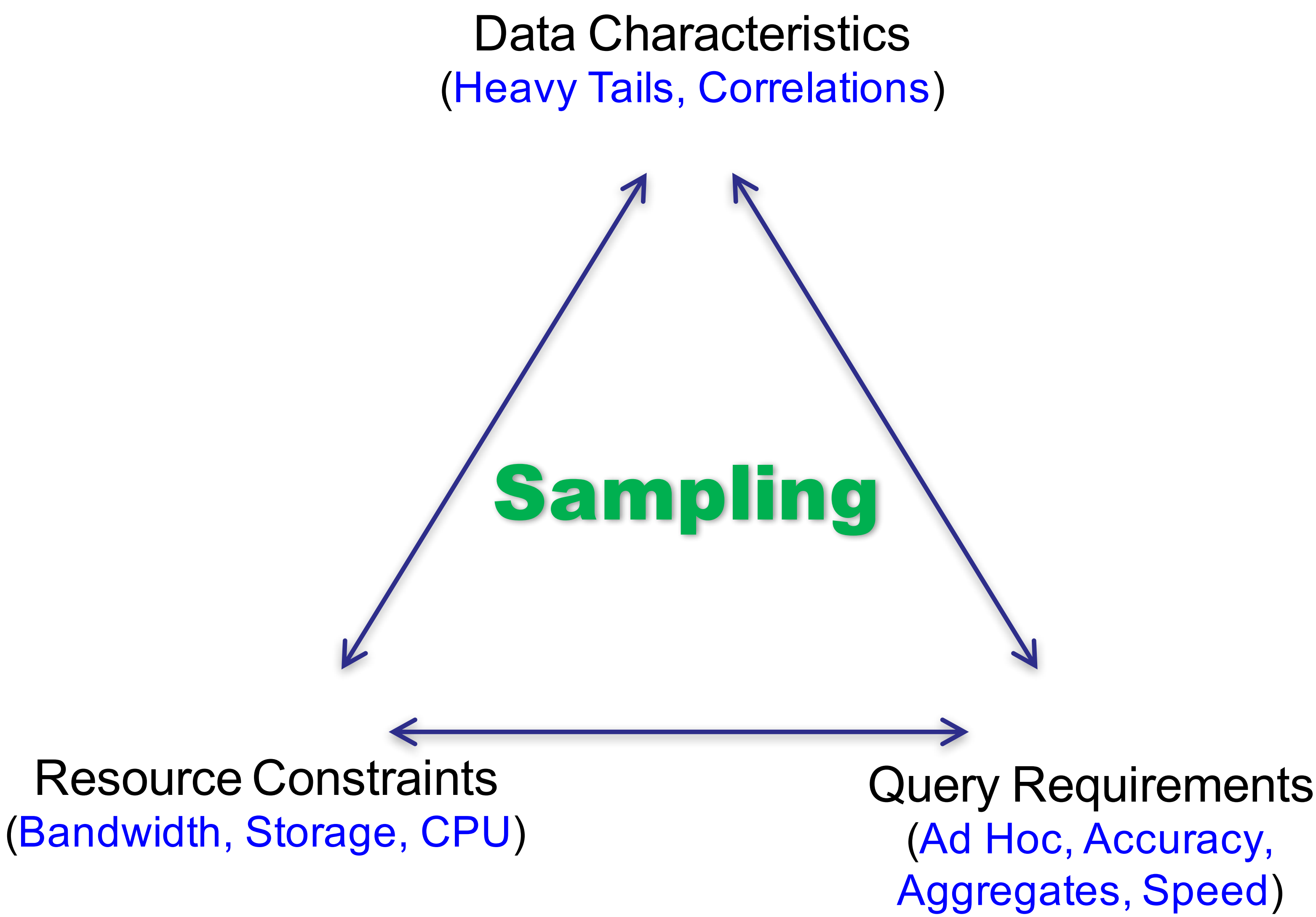
Outline

- Motivating application: sampling in large ISP networks
- Basics of sampling: concepts and estimation
- Stream sampling: uniform and weighted case
  - Variations: Concise sampling, sample and hold, sketch guided

BREAK

- Advanced stream sampling: sampling as cost optimization
  - VarOpt, priority, structure aware, and stable sampling
- Hashing and coordination
  - Bottom-k, consistent sampling and sketch-based sampling
- Graph sampling
  - Node, edge and subgraph sampling
- Conclusion and future directions

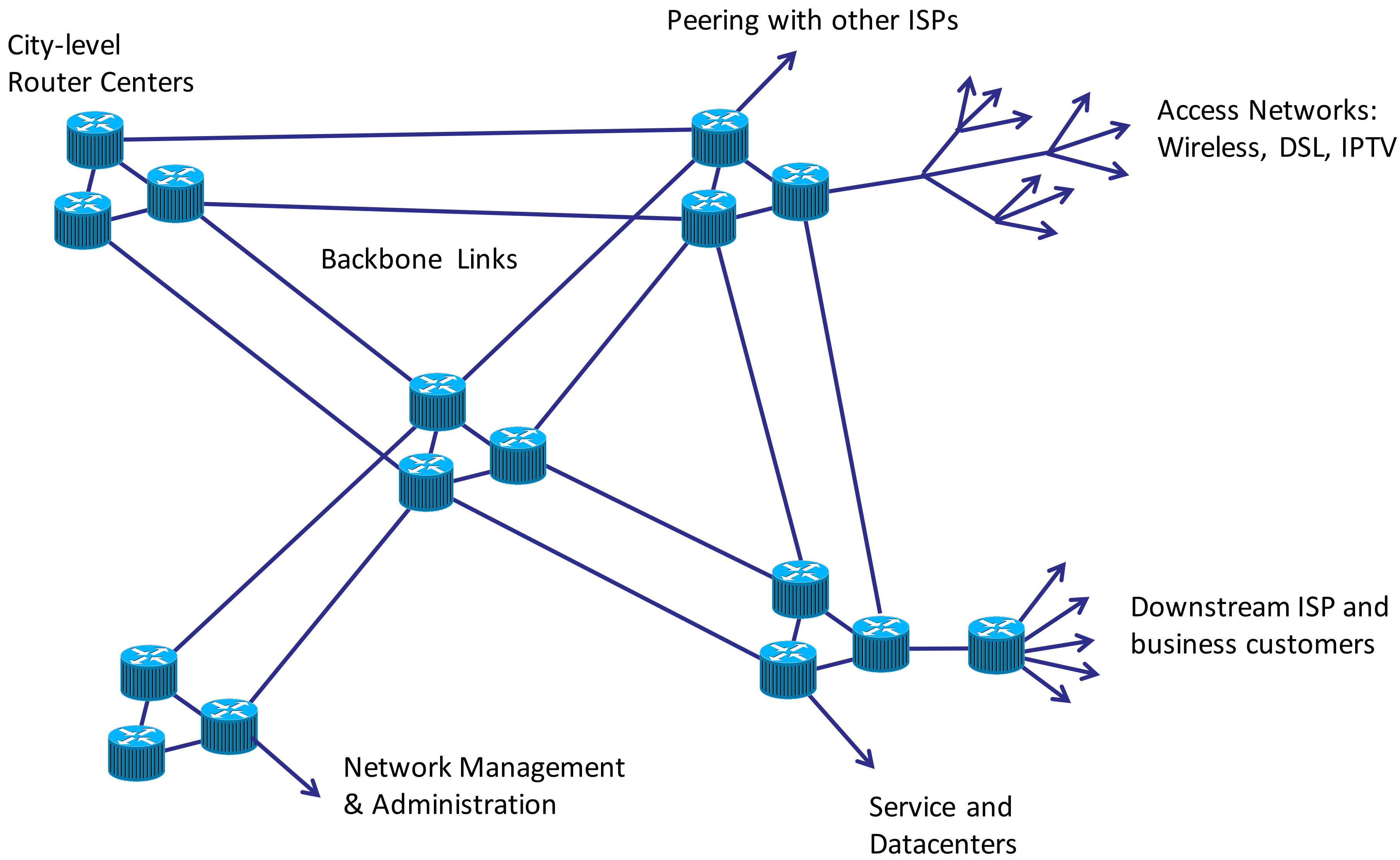
Sampling as a Mediator of Constraints



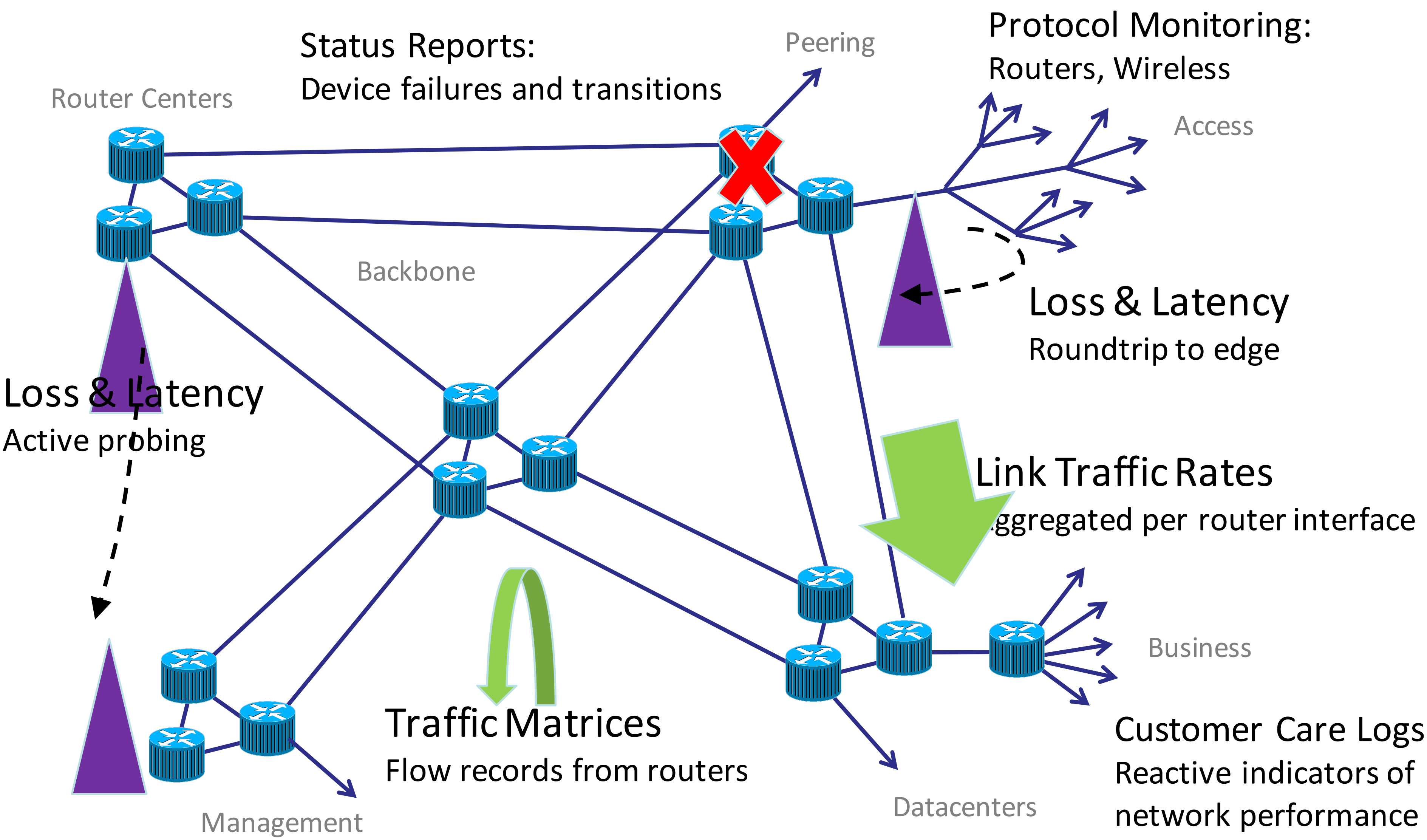
Motivating Application: ISP Data

- Will motivate many results with application to ISPs
- Many reasons to use such examples:
  - Expertise: tutors from telecoms world
  - Demand: many sampling methods developed in response to ISP needs
  - Practice: sampling widely used in ISP monitoring, built into routers
  - Prescience: ISPs were first to hit many "big data" problems
  - Variety: many different places where sampling is needed
- First, a crash-course on ISP networks…

Structure of Large ISP Networks



Measuring the ISP Network: Data Sources



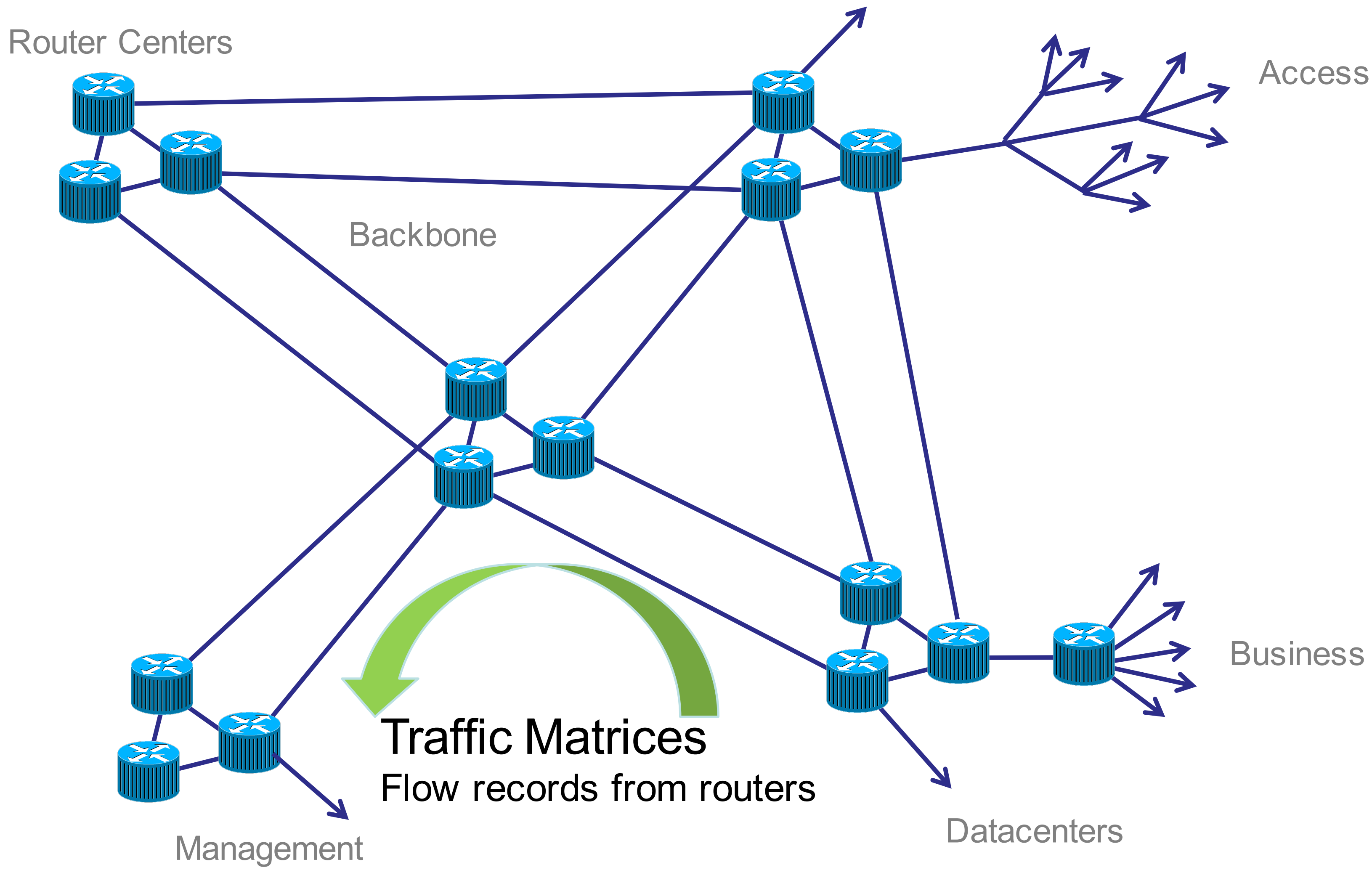
Why Summarize (ISP) Big Data?

- When transmission bandwidth for measurements is limited
  - Not such a big issue in ISPs with in-band collection
- Typically raw accumulation is not feasible (even for nation states)
  - High rate streaming data
  - Maintain historical summaries for baselining, time series analysis
- To facilitate fast queries
  - When infeasible to run exploratory queries over full data
- As part of hierarchical query infrastructure:
  - Maintain full data over limited duration window
  - Drill down into full data through one or more layers of summarization

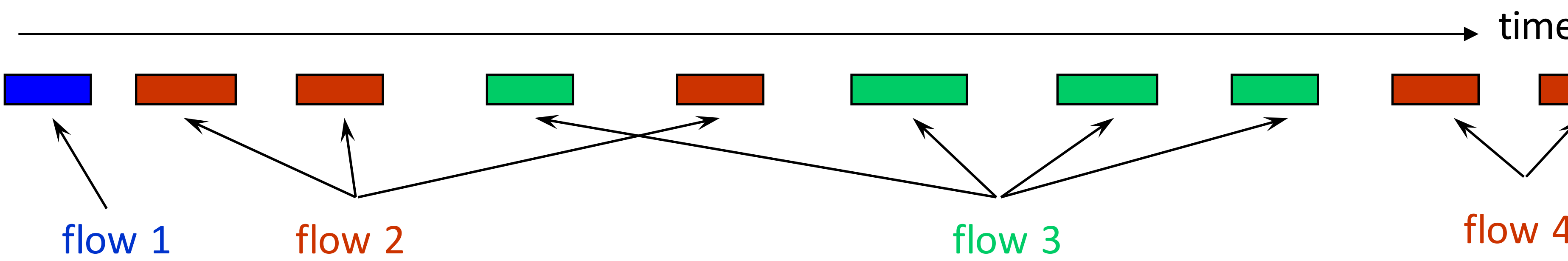
Sampling has been proved to be a flexible method to accomplish this

Data Scale:
Summarization and Sampling

Traffic Measurement in the ISP Network



Massive Dataset: Flow Records



- **IP Flow**: set of packets with common key observed close in time
- **Flow Key**: IP src/dst address, TCP/UDP ports, ToS,… [64 to 104+ bits]
- **Flow Records**:
  - Protocol level summaries of flows, compiled and exported by routers
  - Flow key, packet and byte counts, first/last packet time, some router state
  - Realizations: Cisco Netflow, IETF Standards
- **Scale**: 100's TeraBytes of flow records daily are generated in a large ISP
- Used to manage network over range of timescales:
  - Capacity planning (months),…., detecting network attacks (seconds)
- Analysis tasks
  - Easy: timeseries of predetermined aggregates (e.g. address prefixes)
  - Hard: fast queries over exploratory selectors, history, communications subgraphs

Flows, Flow Records and Sampling

- Two types of sampling used in practice for internet traffic:
  1. Sampling packet stream in router prior to forming flow records
     - Limits the rate of lookups of packet key in flow cache
     - Realized as Packet Sampled NetFlow (more later…)
  2. Downstream sampling of flow records in collection infrastructure
     - Limits transmission bandwidth, storage requirements
     - Realized in ISP measurement collection infrastructure (more later…)
- Two cases illustrative of general property
  - Different underlying distributions require different sample designs
  - Statistical optimality sometimes limited by implementation constraints
    - Availability of router storage, processing cycles

Abstraction: Keyed Data Streams

- Data Model: objects are keyed weights
  - Objects (x,k): Weight x; key k
    - Example 1: objects = packets, x = bytes, k = key (source/destination)
    - Example 2: objects = flows, x = packets or bytes, k = key
    - Example 3: objects = account updates, x = credit/debit, k = account ID
- Stream of keyed weights, $\{(x_i, k_i): i = 1,2,\ldots,n\}$
- Generic query: subset sums
  - $X(S) = \Sigma_{i\in S}\, x_i$ for $S \subset \{1,2,\ldots,n\}$ i.e. total weight of index subset S
  - Typically $S = S(K) = \{i: k_i \in K\}$: objects with keys in K
    - Example 1, 2: $X(S(K))$ = total bytes to given IP dest address / UDP port
    - Example 3: $X(S(K))$ = total balance change over set of accounts
- Aim: Compute fixed size summary of stream that can be used to estimate arbitrary subset sums with known error bounds

Inclusion Sampling and Estimation

- Horvitz-Thompson Estimation:
  - Object of size x_i sampled with probability p_i
  - Unbiased estimate $x'_i = x_i / p_i$ (if sampled), 0 if not sampled: $E[x'_i] = x_i$
- Linearity:
  - Estimate of subset sum = sum of matching estimates
  - Subset sum $X(S)= \Sigma_{i\in S} x_i$ is estimated by $X'(S) = \Sigma_{i\in S} x'_i$
- Accuracy:
  - Exponential Bounds: $\Pr[\ |X'(S) - X(S)| > \delta X(S)] \leq \exp[-g(\delta)X(S)]$
  - Confidence intervals: $X(S) \in [X^-(\varepsilon) , X^+(\varepsilon)]$ with probability $1 - \varepsilon$
- Futureproof:
  - Don't need to know queries at time of sampling
    - "Where/where did that suspicious UDP port first become so active?"
    - "Which is the most active IP address within than anomalous subnet?"
  - Retrospective estimate: subset sum over relevant keyset

Independent Stream Sampling

- Bernoulli Sampling
  - IID sampling of objects with some probability p
  - Sampled weight x has HT estimate x/p
- Poisson Sampling
  - Weight x_i sampled with probability p_i ; HT estimate x_i / p_i
- When to use Poisson vs. Bernoulli sampling?
  - Elephants and mice: Poisson allows probability to depend on weight…
- What is best choice of probabilities for given stream $\{x_i\}$?

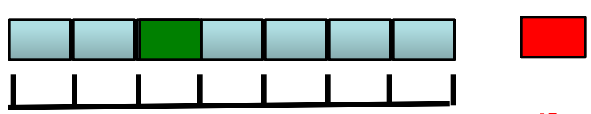
Bernoulli Sampling

- ◊ The easiest possible case of sampling: all weights are 1
  - – N objects, and want to sample k from them uniformly
  - – Each possible subset of k should be equally likely
- ◊ Uniformly sample an index from N (without replacement) k times
  - – Some subtleties: truly random numbers from [1…N] on a computer?
  - – Assume that random number generators are good enough
- ◊ Common trick in DB: assign a random number to each item and sort
  - – Costly if N is very big, but so is random access
- ◊ Interesting problem: take a single linear scan of data to draw sample
  - – Streaming model of computation: see each element once
  - – Application: IP flow sampling, too many (for us) to store
  - – (For a while) common tech interview question

Reservoir Sampling

"Reservoir sampling" described by [Knuth 69, 81]; enhancements [Vitter 85]

- ◊ Fixed size k uniform sample from arbitrary size N stream in one pass
  - – No need to know stream size in advance
  - – Include first k items w.p. 1
  - – Include item $n > k$ with probability $p_n = k/n$, $n > k$
    - □ Pick j uniformly from $\{1,2,\ldots,n\}$
    - □ If $j \le k$, swap item n into location j in reservoir, discard replaced item
- ◊ Neat proof shows the uniformity of the sampling method:
  - – Let S_n = sample set after n arrivals



New item: selection probability

$\text{Prob}[n \in S_n] = p_n := k/n$

Previously sampled item: induction

$m \in S_{n-1}$ w.p. $p_{n-1} \Rightarrow m \in S_n$ w.p. $p_{n-1} * (1 - p_n / k) = p_n$

Reservoir Sampling: Skip Counting

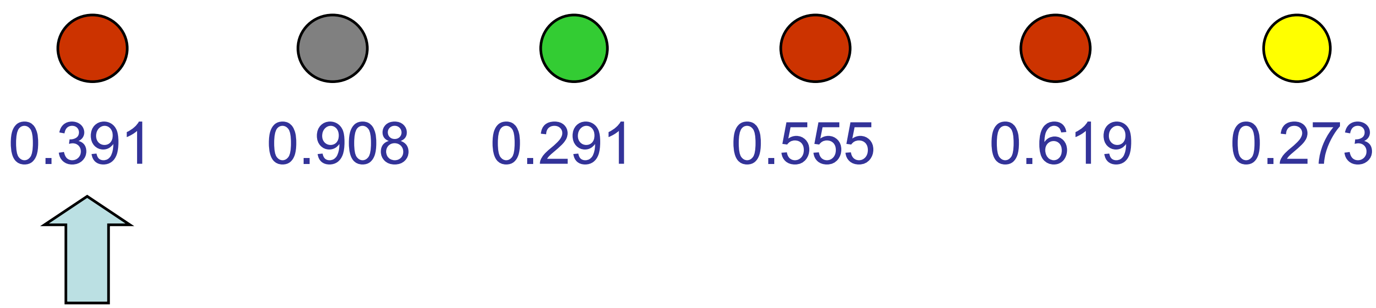
- Simple approach: check each item in turn
  - $O(1)$ per item:
  - Fine if computation time < interarrival time
  - Otherwise build up computation backlog $O(N)$
- Better: "skip counting"
  - Find random index $m(n)$ of next selection $> n$
  - Distribution: $\text{Prob}[m(n) \leq m] = 1 - (1-p_{n+1})*(1-p_{n+2})*\ldots*(1-p_m)$
- Expected number of selections from stream is

$$k + \Sigma_{k<m\leq N}\ p_m = k + \Sigma_{k<m\leq N}\ k/m = O(k\,(\,1 + \ln\,(N/k)\,))$$

- Vitter'85 provided algorithm with this average running time

Reservoir Sampling via Order Sampling

- Order sampling a.k.a. bottom-k sample, min-hashing
- Uniform sampling of stream into reservoir of size k
- Each arrival n: generate one-time random value $r_n \in U[0,1]$
  - r_n also known as hash, rank, tag…
- Store k items with the smallest random tags



- Each item has same chance of least tag, so uniform
- Fast to implement via priority queue
- Can run on multiple input streams separately, then merge

Handling Weights

- So far: uniform sampling from a stream using a reservoir
- Extend to non-uniform sampling from weighted streams
  - Easy case: k=1
  - Sampling probability $p(n) = x_n/W_n$ where $W_n = \Sigma_{i=1}^{n} x_i$
- k>1 is harder
  - Can have elements with large weight: would be sampled with prob 1?
- Number of different weighted order-sampling schemes proposed to realize desired distributional objectives
  - Rank $r_n = f(u_n, x_n)$ for some function f and $u_n \in U[0,1]$
  - k-mins sketches [Cohen 1997], Bottom-k sketches [Cohen Kaplan 2007]
  - [Rosen 1972], Weighted random sampling [Efraimidis Spirakis 2006]
  - Order PPS Sampling [Ohlsson 1990, Rosen 1997]
  - Priority Sampling [Duffield Lund Thorup 2004], [Alon+DLT 2005]

Weighted random sampling

- Weighted random sampling [Efraimidis Spirakis 06] generalizes min-wise
  - For each item draw r_n uniformly at random in range [0,1]
  - Compute the 'tag' of an item as $r_n^{(1/x_n)}$
  - Keep the items with the k smallest tags
  - Can prove the correctness of the exponential sampling distribution
- Can also make efficient via skip counting ideas

Priority Sampling

- Each item x_i given priority $z_i = x_i / r_i$ with r_n uniform random in $(0,1]$
- Maintain reservoir of $k+1$ items (x_i , z_i) of highest priority
- Estimation
  - Let $z^* = (k+1)^{st}$ highest priority
  - Top-k priority items: weight estimate $x'_I = \max\{ x_i , z^* \}$
  - All other items: weight estimate zero
- Statistics and bounds
  - x'_I unbiased; zero covariance: $Cov[x'_i , x'_j] = 0$ for $i \neq j$
  - Relative variance for any subset sum $\leq 1/(k-1)$ [Szegedy, 2006]

Priority Sampling in Databases

- One Time Sample Preparation
  - Compute priorities of all items, sort in decreasing priority order
    - No discard
- Sample and Estimate
  - Estimate any subset sum $X(S) = \Sigma_{i \in S} x_i$ by $X'(S) = \Sigma_{i \in S} x'_I$ for some $S' \subset S$
  - Method: select items in decreasing priority order
- Two variants: bounded variance or complexity
  1. S' = first k items from S: relative variance bounded $\leq 1/(k-1)$
     - $x'_I = \max\{x_i, z^*\}$ where $z^* = (k+1)^{st}$ highest priority in S
  2. S' = items from S in first k: execution time $O(k)$
     - $x'_I = \max\{x_i, z^*\}$ where $z^* = (k+1)^{st}$ highest priority

[Alon et. al., 2005]

Making Stream Samples Smarter

- Observation: we **see** the whole stream, even if we can't store it
  - Can keep more information about sampled items if repeated
  - Simple information: if item sampled, count all repeats
- Counting Samples [Gibbons & Mattias 98]
  - Sample new items with fixed probability p, count repeats as c_i
  - Unbiased estimate of total count: $1/p + (c_i – 1)$
- Sample and Hold [Estan & Varghese 02]: generalize to weighted keys
  - New key with weight b sampled with probability $1 - (1-p)^b$
- Lower variance compared with independent sampling
  - But sample size will grow as pn
- Adaptive sample and hold: reduce p when needed
  - "Sticky sampling": geometric decreases in p [Manku, Motwani 02]
  - Much subsequent work tuning decrease in p to maintain sample size

Sketch Guided Sampling

- Go further: avoid sampling the heavy keys as much
  - Uniform sampling will pick from the heavy keys again and again
- Idea: use an oracle to tell when a key is heavy [Kumar Xu 06]
  - Adjust sampling probability accordingly
- Can use a "sketch" data structure to play the role of oracle
  - Like a hash table with collisions, tracks approximate frequencies
  - E.g. (Counting) Bloom Filters, Count-Min Sketch
- Track probability with which key is sampled, use HT estimators
  - Set probability of sampling key with (estimated) weight w as $1/(1 + \varepsilon w)$ for parameter ε : decreases as w increases
  - Decreasing ε improves accuracy, increases sample size

Challenges for Smart Stream Sampling

- Current router constraints
  - Flow tables maintained in fast expensive SRAM
    - To support per packet key lookup at line rate
- Implementation requirements
  - Sample and Hold: still need per packet lookup
  - Sampled NetFlow: (uniform) sampling reduces lookup rate
    - Easier to implement despite inferior statistical properties
- Long development times to realize new sampling algorithms
- Similar concerns affect sampling in other applications
  - Processing large amounts of data needs awareness of hardware
  - Uniform sampling means no coordination needed in distributed setting

Future for Smarter Stream Sampling

- Software Defined Networking
  - Current: proprietary software running on special vendor equipment
  - Future: open software and protocols on commodity hardware
- Potentially offers flexibility in traffic measurement
  - Allocate system resources to measurement tasks as needed
  - Dynamic reconfiguration, fine grained tuning of sampling
  - Stateful packet inspection and sampling for network security
- Technical challenges:
  - High rate packet processing in software
  - Transparent support from commodity hardware
  - OpenSketch: [Yu, Jose, Miao, 2013]
- Same issues in other applications: use of commodity programmable HW

Stream Sampling:
Sampling as Cost Optimization

Matching Data to Sampling Analysis

◊ Generic problem 1: Counting objects: weight $x_i = 1$

Bernoulli (uniform) sampling with probability p works fine

- Estimated subset count $X'(S) = \#\{\text{samples in } S\} / p$
- Relative Variance $(X'(S)) = (1/p - 1)/X(S)$
  - □ given p, get any desired accuracy for large enough S

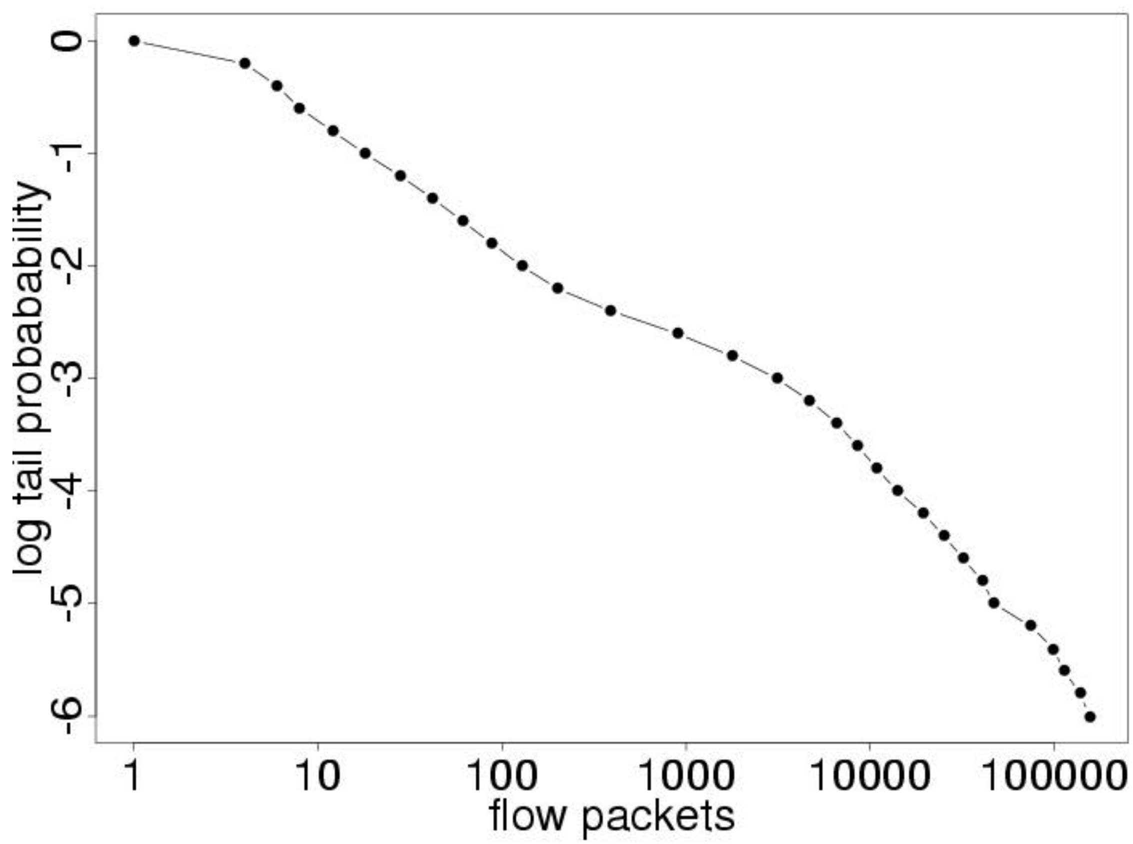
◊ Generic problem 2: x_i in Pareto distribution, a.k.a. 80-20 law

- Small proportion of objects possess a large proportion of total weight
  - □ How to best to sample objects to accurately estimate weight?
- Uniform sampling?
  - □ likely to omit heavy objects $\Rightarrow$ big hit on accuracy
  - □ making selection set S large doesn't help

- Select m largest objects ?
  - □ biased & smaller objects systematically ignored

Heavy Tails in the Internet and Beyond

- Files sizes in storage
- Bytes and packets per network flow
- Degree distributions in web graph, social networks



Non-Uniform Sampling

- Extensive literature: see book by [Tille, “Sampling Algorithms”, 2006]
- Predates “Big Data”
  - Focus on statistical properties, not so much computational
- IPPS: Inclusion Probability Proportional to Size
  - Variance Optimal for HT Estimation
  - Sampling probabilities for multivariate version: [Chao 1982, Tille 1996]
  - Efficient stream sampling algorithm: [Cohen et. al. 2009]

Costs of Non-Uniform Sampling

- Independent sampling from n objects with weights $\{x_1,\ldots,x_n\}$
- Goal: find the "best" sampling probabilities $\{p_1, \ldots, p_n\}$
- Horvitz-Thompson: unbiased estimation of each x_i by

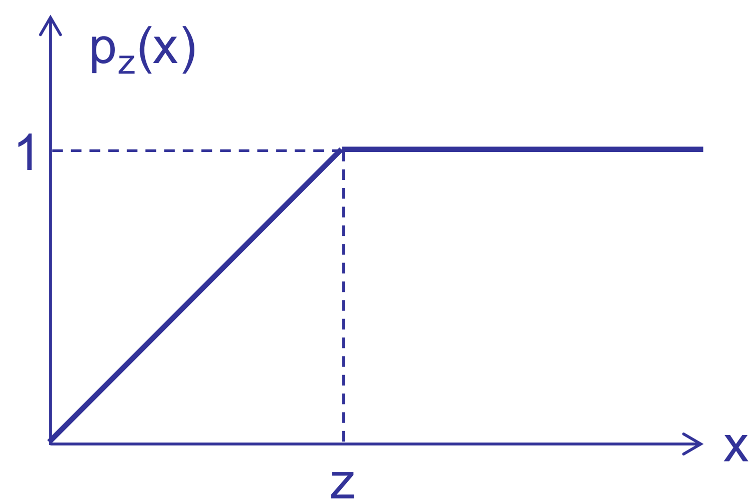
$$x'_i = \begin{cases} x_i / p_i & \text{if weight } i \text{ selected} \\ 0 & \text{otherwise} \end{cases}$$

- Two costs to balance:
  1. Estimation Variance: $Var(x'_i) = x^2_i\,(1/p_i - 1)$
  2. Expected Sample Size: $\sum_i p_i$
- Minimize Linear Combination Cost: $\sum_i (x_i^2(1/p_i - 1) + z^2\, p_i)$
  - z expresses relative importance of small sample vs. small variance

Minimal Cost Sampling: IPPS

IPPS: Inclusion Probability Proportional to Size

- Minimize Cost $\Sigma_i (x_i^2 (1/p_i - 1) + z^2 p_i)$ subject to $1 \geq p_i \geq 0$
- Solution: $p_i = p_z(x_i) = \min\{1, x_i / z\}$
  - small objects $(x_i < z)$ selected with probability proportional to size
  - large objects $(x_i \geq z)$ selected with probability 1
  - Call z the "sampling threshold"
  - Unbiased estimator $x_i/p_i = \max\{x_i , z\}$
- Perhaps reminiscent of importance sampling, but not the same:
  - make no assumptions concerning distribution of the x



Error Estimates and Bounds

- Variance Based:
  - HT sampling variance for single object of weight x_i
    - $Var(x'_i) = x^2_i (1/p_i - 1) = x^2_i (1/\min\{1, x_i/z\} - 1) \leq z\, x_i$
  - Subset sum $X(S) = \Sigma_{i \in S} x_i$ is estimated by $X'(S) = \Sigma_{i \in S} x'_i$
    - $Var(X'(S)) \leq z\, X(S)$
- Exponential Bounds
  - E.g. $Prob[X'(S) = 0] \leq \exp(- X(S) / z)$
- Bounds are simple and powerful
  - depend only on subset sum $X(S)$, not individual constituents

Sampled IP Traffic Measurements

- Packet Sampled NetFlow
  - Sample packet stream in router to limit rate of key lookup: uniform $1/N$
  - Aggregate sampled packets into flow records by key
- Model: packet stream of (key, bytesize) pairs $\{ (b_i, k_i) \}$
- Packet sampled flow record (b,k) where $b = \Sigma \{b_i : i \text{ sampled} \land k_i = k\}$
  - HT estimate $b*N$ of total bytes in flow
- Downstream sampling of flow records in measurement infrastructure
  - IPPS sampling, probability $\min\{1, b*N/z\}$
- Chained variance bound for any subset sum X of flows
  - $Var(X') \leq (z + Nb_{max}) X$ where b_{max} = maximum packet byte size
  - Regardless of how packets are distributed amongst flows

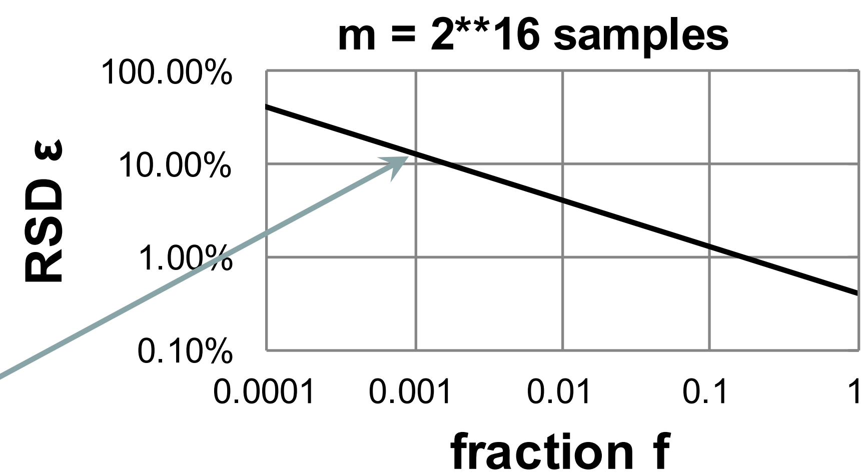
[Duffield, Lund, Thorup, IEEE ToIT, 2004]

Estimation Accuracy in Practice

- Estimate any subset sum comprising at least some fraction f of weight
- Suppose: sample size m
- Analysis: typical estimation error ε (relative standard deviation) obeys

$$\varepsilon \quad {}^{1}\!/\!\sqrt{f\,m}$$

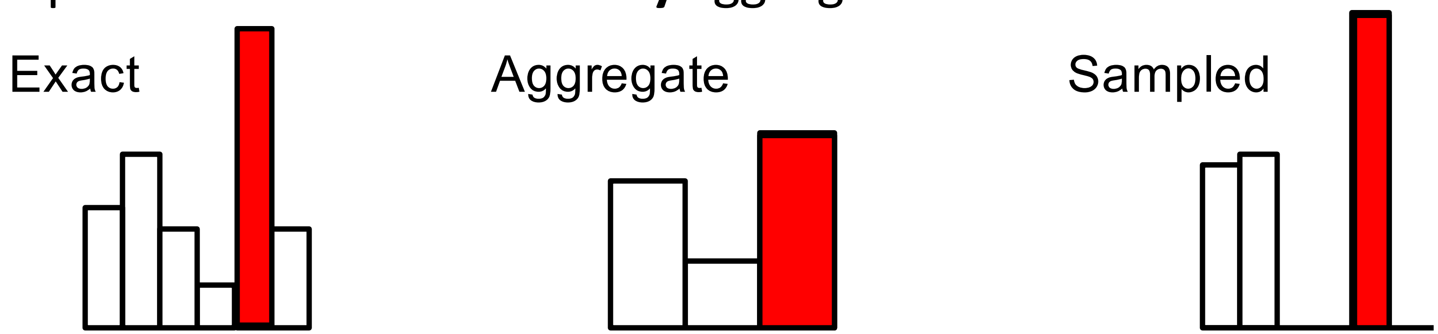
Estimate fraction f = 0.1% with typical relative error 12%:



- 2**16 = storage needed for aggregates over 16 bit address prefixes
  - But sampling gives more flexibility to estimate traffic within aggregates

Heavy Hitters: Exact vs. Aggregate vs. Sampled

- Sampling does not tell you where the interesting features are
  - But does speed up the ability to find them with existing tools
- Example: Heavy Hitter Detection
  - Setting: Flow records reporting 10GB/s traffic stream
  - Aim: find Heavy Hitters = IP prefixes comprising ≥ 0.1% of traffic
  - Response time needed: 5 minute
- Compare:
  - Exact: 10GB/s x 5 minutes yields upwards of 300M flow records
  - 64k aggregates over 16 bit prefixes: no deeper drill-down possible
  - Sampled: 64k flow records: **any** aggregate ≥ 0.1% accurate to 10%



Cost Optimization for Sampling

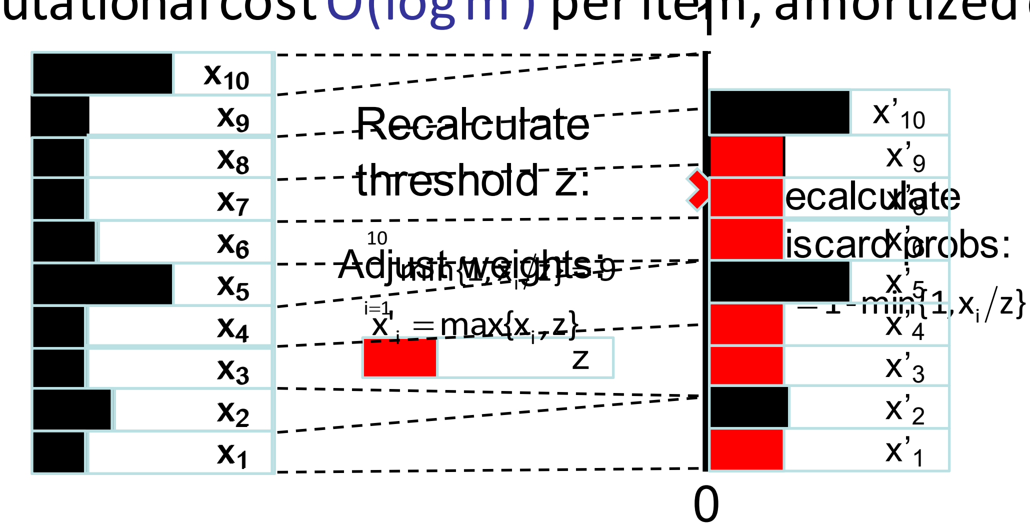
Several different approaches optimize for different objectives:

1. Fixed Sample Size IPPS Sample
   – Variance Optimal sampling: minimal variance unbiased estimation
2. Structure Aware Sampling
   – Improve estimation accuracy for subnet queries using topological cost
3. Fair Sampling
   – Adaptively balance sampling budget over subpopulations of flows
   – Uniform estimation accuracy regardless of subpopulation size
4. Stable Sampling
   – Increase stability of sample set by imposing cost on changes

IPPS Stream Reservoir Sampling

- Each arriving item:
  - Provisionally include item in reservoir
  - If m+1 items, discard 1 item randomly
    - Calculate threshold z to sample m items on average: z solves $\Sigma_i\, p_z(x_i) = m$
    - Discard item i with probability $q_i = 1 - p_z(x_i)$
    - Adjust m surviving x_i with Horvitz-Thompson $x'_i = x_i / p_i = \max\{x_i, z\}$
- Efficient Implementation:
  - Computational cost $O(\log m)$ per item, amortized cost $O(\log \log m)$

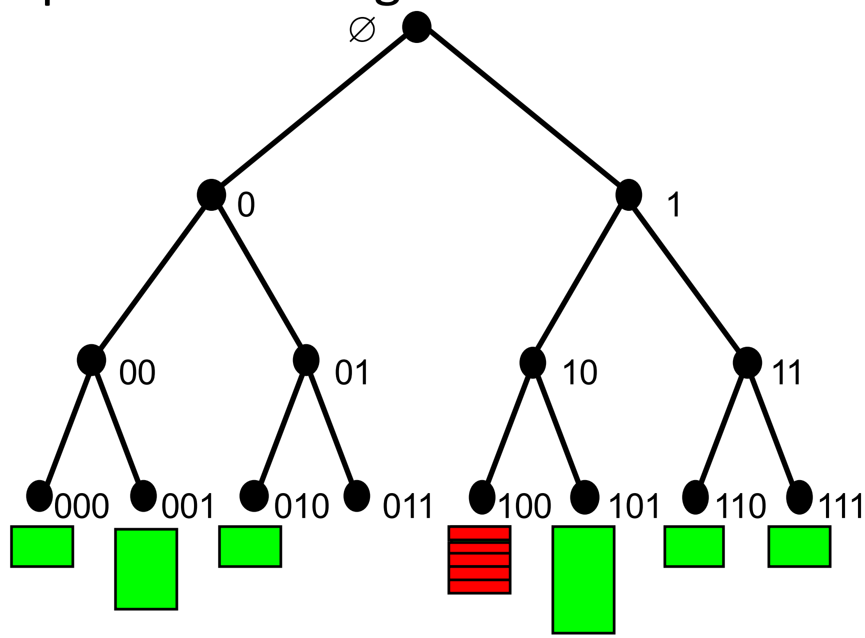
Example:
m=9



[Cohen, Duffield, Lund, Kaplan, Thorup; SODA 2009, SIAM J. Comput. 2011]

Structure (Un)Aware Sampling

- Sampling is oblivious to structure in keys (IP address hierarchy)
  - Estimation disperses the weight of discarded items to surviving samples

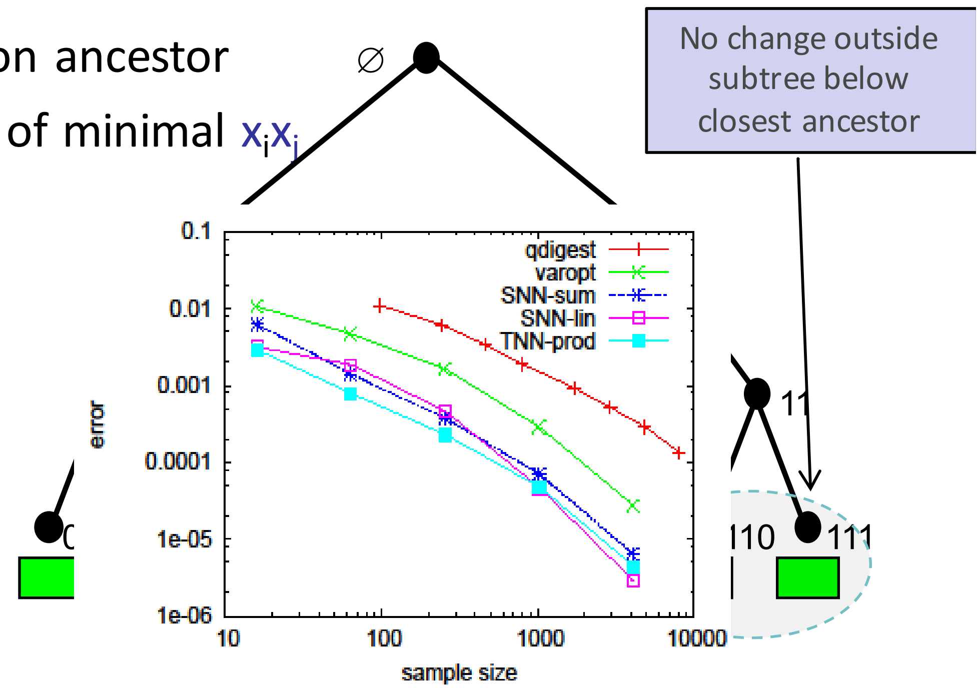


- Queries structure aware: subset sums over related keys (IP subnets)
  - Accuracy on LHS is decreased by discarding weight on RHS

Localizing Weight Redistribution

- Initial weight set $\{x_i : i \in S\}$ for some $S \subset \Omega$
  - E.g. Ω = possible IP addresses, S =observed IP addresses
- Attribute "range cost" $C(\{x_i : i \in R\})$ for each weight subset $R \subseteq S$
  - Possible factors for Range Cost:
    - □ Sampling variance
    - □ Topology e.g. height of lowest common ancestor
  - Heuristics: R^* = Nearest Neighbor $\{x_i, x_j\}$ of minimal $x_i x_j$
- Sample k items from S:
  - Progressively remove one item from subset with minimal range cost:
  - While($|S| > k$)
    - □ Find $R^* \subseteq S$ of minimal range cost.
    - □ Remove a weight from R^* w/ VarOpt

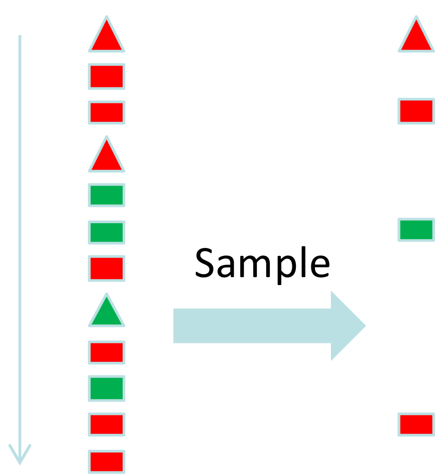
[Cohen, Cormode, Duffield; PVLDB 2011]



Order of magnitude reduction in average subnet error vs. VarOpt

Fair Sampling Across Subpopulations

- Analysis queries often focus on specific subpopulations
  - E.g. networking: different customers, user applications, network paths
- Wide variation in subpopulation size
  - 5 orders of magnitude variation in traffic on interfaces of access router
- If uniform sampling across subpopulations:
  - Poor estimation accuracy on subset sums within small subpopulations



Color = subpopulation

▲ , ▲ = interesting items

– occurrence proportional to subpopulation size

Uniform Sampling across subpopulations:

– Difficult to track proportion of interesting items within small subpopulations: ▲

Fair Sampling Across Subpopulations

- Minimize **relative** variance by sharing budget m over subpopulations
  - Total n objects in subpopulations $n_1,\ldots,n_d$ with $\Sigma_i n_i=n$
  - Allocate budget m_i to each subpopulation n_i with $\Sigma_i m_i=m$
- Minimize average population relative variance R = const. $\Sigma_i 1/m_i$
- Theorem:
  - R minimized when $\{m_i\}$ are Max-Min Fair share of m under demands $\{n_i\}$
- Streaming
  - Problem: don't know subpopulation sizes $\{n_i\}$ in advance
- Solution: progressive fair sharing as reservoir sample
  - Provisionally include each arrival
  - Discard 1 item as VarOpt sample from any maximal subpopulation
- Theorem [Duffield; Sigmetrics 2012]:
  - Max-Min Fair at all times; equality in distribution with VarOpt samples $\{m_i$ from $n_i\}$

Stable Sampling

- Setting: Sampling a population over successive periods
- Sample independently at each time period?
  - Cost associated with sample churn
  - Time series analysis of set of relatively stable keys
- Find sampling probabilities through cost minimization
  - Minimize Cost = Estimation Variance + z * E[#Churn]
- Size m sample with maximal expected churn D
  - weights $\{x_i\}$, previous sampling probabilities $\{p_i\}$
  - find new sampling probabilities $\{q_i\}$ to minimize cost of taking m samples
  - Minimize $\Sigma_i x^2_i / q_i$ subject to $1 \geq q_i \geq 0$, $\Sigma_I q_i = m$ and $\Sigma_I \mid p_i - q_i \mid \leq D$

[Cohen, Cormode, Duffield, Lund 13]

Summary of Part 1

- ◊ Sampling as a powerful, general summarization technique
- ◊ Unbiased estimation via Horvitz-Thompson estimators
- ◊ Sampling from streams of data
  - Uniform sampling: reservoir sampling
  - Weighted generalizations: sample and hold, counting samples
- ◊ Advances in stream sampling
  - The cost principle for sample design, and IPPS methods
  - Threshold, priority and VarOpt sampling
  - Extending the cost principle:
    - □ structure aware, fair sampling, stable sampling, sketch guided

Outline

- Motivating application: sampling in large ISP networks
- Basics of sampling: concepts and estimation
- Stream sampling: uniform and weighted case
  - Variations: Concise sampling, sample and hold, sketch guided

BREAK

- Advanced stream sampling: sampling as cost optimization
  - VarOpt, priority, structure aware, and stable sampling
- Hashing and coordination
  - Bottom-k, consistent sampling and sketch-based sampling
- Graph sampling
  - Node, edge and subgraph sampling
- Conclusion and future directions

Data Scale:
Hashing and Coordination

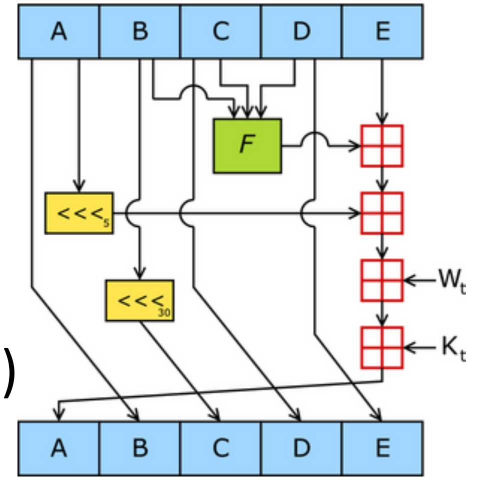
Sampling from the set of items

- ◊ Sometimes need to sample from the distinct set of objects
  - – Not influenced by the weight or number of occurrences
  - – E.g. sample from the distinct set of flows, regardless of weight
- ◊ Need sampling method that is invariant to duplicates
- ◊ Basic idea: build a function to determine what to sample
  - – A "random" function $f(k) \rightarrow R$
  - – Use $f(k)$ to make a sampling decision: consistent decision for same key

Permanent Random Numbers

- Often convenient to think of f as giving "permanent random numbers"
  - Permanent: assigned once and for all
  - Random: treat as if fully randomly chosen
- The permanent random number is used in multiple sampling steps
  - Same "random" number each time, so **consistent** (correlated) decisions
- Example: use PRNs to draw a sample of s from N via order sampling
  - If $s << N$, small chance of seeing same element in different samples
  - Via PRN, stronger chance of seeing same element
    - Can track properties over time, gives a form of stability
- Easiest way to generate PRNs: apply a **hash function** to the element id
  - Ensures PRN can be generated with minimal coordination
  - Explicitly storing a random number for all observed keys does not scale

Hash Functions



Many possible choices of hashing functions:

- Cryptographic hash functions: SHA-1, MD5, etc.
  - Results appear "random" for most tests (using seed/salt)
  - Can be slow for high speed/high volume applications
  - Full power of cryptographic security not needed for most statistical purposes
    - Although possible some trade-offs in robustness to subversion if not used
- Heuristic hash functions: srand(), mod
  - Usually pretty fast
  - May not be random enough: structure in keys may cause collisions
- Mathematical hash functions: universal hashing, k-wise hashing
  - Have precise mathematical properties on probabilities
  - Can be implemented to be very fast

Mathematical Hashing

- K-wise independence: $\Pr[h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge \ldots \wedge h(x_t) = y_t] = 1/R^t$
  - Simple function: $c_t x^t + c_{t-1} x^{t-1} + \ldots c_1 x + c_0 \bmod P$
  - For fixed prime P, randomly chosen $c_0 \ldots c_t$
  - Can be made very fast (choose P to be Mersenne prime to simplify mods)
- (Twisted) tabulation hashing [Thorup Patrascu 13]
  - Interpret each key as a sequence of short characters, e.g. 8 * 8bits
  - Use a "truly random" look-up table for each character (so 8 * 256 entries)
  - Take the exclusive-OR of the relevant table values
  - Fast, and fairly compact
  - Strong enough for many applications of hashing (hash tables etc.)

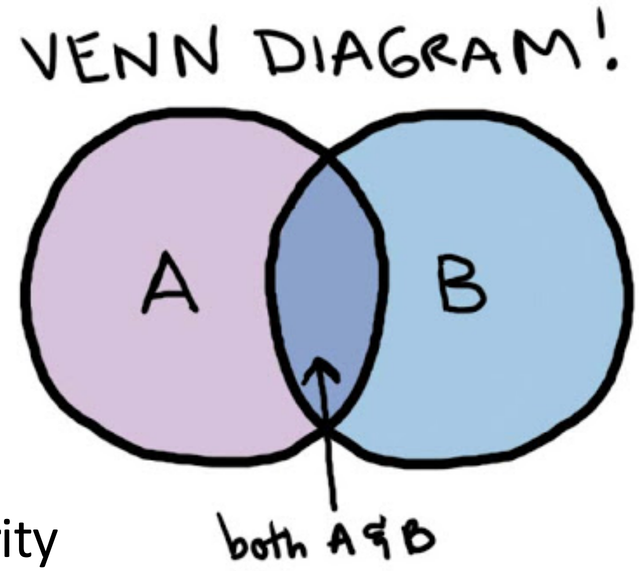
Bottom-k sampling

- Sample from the set of distinct keys
  - Hash each key using appropriate hash function
  - Keep information on the keys with the s smallest hash values
  - Think of as order sampling with PRNs…
- Useful for estimating properties of the support set of keys
  - Evaluate any predicate on the sampled set of keys
- Same concept, several different names:
  - Bottom-k sampling, Min-wise hashing, K-minimum values

Subset Size Estimation from Bottom-k

- Want to estimate the fraction $t = |A|/|D|$
  - D is the observed set of data
  - A is an arbitrary subset given later
  - E.g. fraction of customers who are sports fans from midwest aged 18-35
- Simple algorithm:
  - Run bottom-k to get sample set S, estimate $t' = |A \cap S|/s$
  - Error decreases as $1/\sqrt{s}$
  - Analysis due to [Thorup 13]: simple hash functions suffice for big enough s

Similarity Estimation



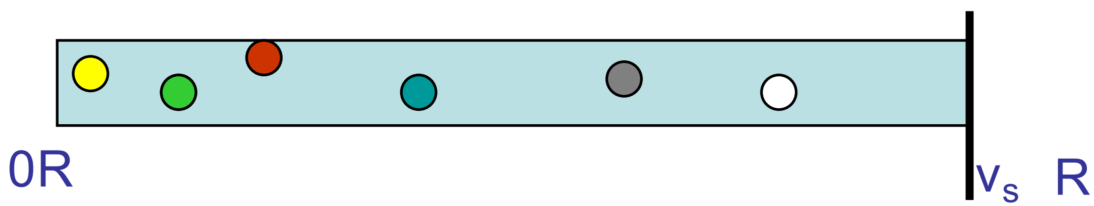
- How similar are two sets, A and B?
- Jaccard coefficient: $|A \cap B|/|A \cup B|$
  - 1 if A, B identical, 0 if they are disjoint
  - Widely used, e.g. to measure document similarity
- Simple approach: sample an item uniformly from A and B
  - Probability of seeing same item from both: $|A \cap B|/(|A| \times |B|)$
  - Chance of seeing same item too low to be informative
- Coordinated sampling: use same hash function to sample from A, B
  - Probability that same item sampled: $|A \cap B|/|A \cup B|$
  - Repeat: the average number of agreements gives Jaccard coefficient
  - Concentration: (additive) error scales as $1/\sqrt{s}$

Technical Issue: Min-wise hashing

- For analysis to work, the hash function must be fully random
  - All possibly permutations of the input are equally likely
  - Unrealistic in practice: description of such a function is huge
- "Simple" hash functions don't work well
  - Universal hash functions are too skewed
- Need hash functions that are "approximately min-wise"
  - Probability of sampling a subset is **almost** uniform
  - Tabulation hashing a simple way to achieve this

Bottom-k hashing for F_0 Estimation

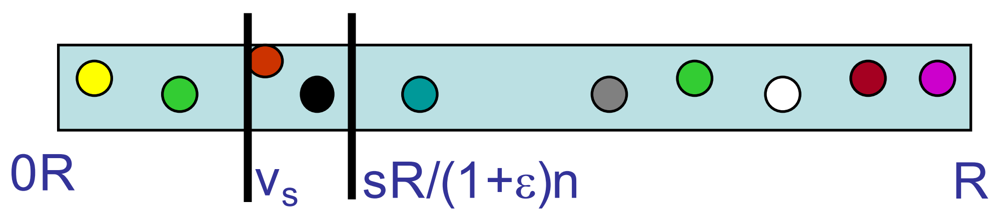
- F_0 is the number of distinct items in the stream
  - a fundamental quantity with many applications
  - E.g. number of distinct flows seen on a backbone link
- Let m be the domain of stream elements: each data item is $[1\ldots m]$
- Pick a random (pairwise) hash function $h: [m] \to [R]$



- Apply bottom-k sampling under hash function h
  - Let v_s = s'th smallest (distinct) value of $h(i)$ seen
- If $n = F_0 < s$, give exact answer, else estimate $F'_0 = sR/v_s$
  - $v_s/R \approx$ fraction of hash domain occupied by s smallest

Analysis of F_0 algorithm

- Can show that it is unlikely to have an overestimate
  - Too many items hashed below a fixed value
  - Can treat each event of an item hashing too low as independent
- Similar outline to show unlikely to have an overestimate
- (Relative) error scales as $1/\sqrt{s}$
- Space cost:
  - Store s hash values, so $O(s \log m)$ bits
  - Can improve to $O(s + \log m)$ with additional hashing tricks
  - See also "Streamed Approximate Counting of Distinct Elements", KDD'14



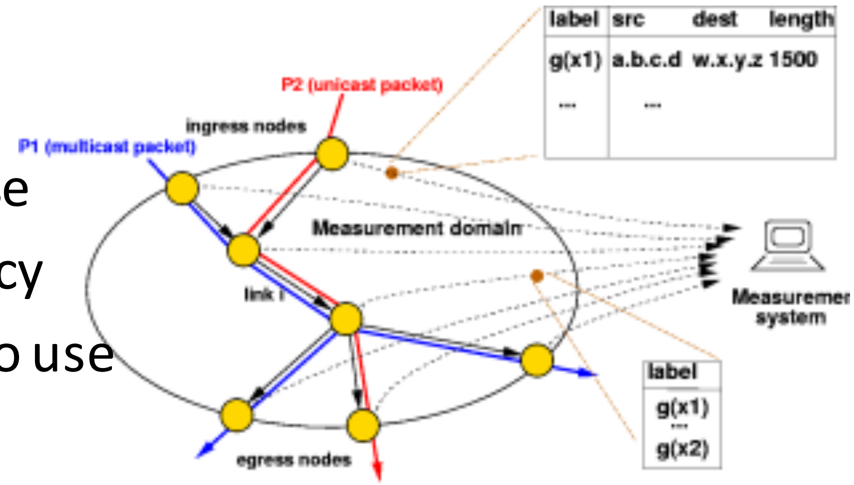
Consistent Weighted Sampling

- Want to extend bottom-k results when data has weights
- Specifically, two data sets A and B where each element has weight
  - Weights are aggregated: we see whole weight of element together
- Weighted Jaccard: want probability that same key is chosen by both to be $\sum_i \min(A(i), B(i))/\sum_i \max(A(i), B(i))$
- Sampling method should obey uniformity and consistency
  - Uniformity: element i picked from A with probability proportional to A(i)
  - Consistency: if i is picked from A, and B(i) > A(i), then i also picked for B
- Simple solution: assuming integer weights, treat weight A(i) as A(i) unique (different) copies of element i, apply bottom-k
  - Limitations: slow, unscalable when weights can be large
  - Need to rescale fractional weights to integral multiples

Consistent Weighted Sampling

- Efficient sampling distributions exist achieving uniformity and consistency
- Basic idea: consider a weight w as w/Δ different elements
  - Compute the probability that any of these achieves the minimum value
  - Study the limiting distribution as $\Delta \rightarrow 0$
- Consistent Weighted Sampling [Manasse, McSherry, Talway 07], [Ioffe 10]
  - Use hash of item to determine which points sampled via careful transform
  - Many details needed to contain bit-precision, allow fast computation
- Other combinations of key weights are possible [Cohen Kaplan Sen 09]
  - Min of weights, max of weights, sum of (absolute) differences

Trajectory Sampling



- ◊ Aims [Duffield Grossglauser 01]:
  - Probe packets at each router they traverse
  - Collate reports to infer link loss and latency
  - Need to sample; independent sampling no use
- ◊ Hash-based sampling:
  - All routers/packets: compute hash h of invariant packet fields
  - Sample if h ∈ some H and report to collector; tune sample rate with |H|
  - Use high entropy packet fields as hash input, e.g. IP addresses, ID field
  - Hash function choice trade-off between speed, uniformity & security
- ◊ Standardized in Internet Engineering Task Force (IETF)
  - Service providers need consistency across different vendors
  - Several hash functions standardized, extensible
  - Same issues arise in other big data ecosystems (apps and APIs)

Hash Sampling in Network Management

- Many different network subsystems used to provide service
  - Monitored through event logs, passive measurement of traffic & protocols
  - Need cross-system sample that captures full interaction between network and a representative set of users
- Ideal: hash-based selection based on common identifier
- Administrative challenges! Organizational diversity
- Timeliness challenge:
  - Selection identifier may not be present at a measurement location
  - Example: common identifier = anonymized customer id
    - Passive traffic measurement based on IP address
    - Mapping of IP address to customer ID not available remotely
    - Attribution of traffic IP address to a user difficult to compute at line speed

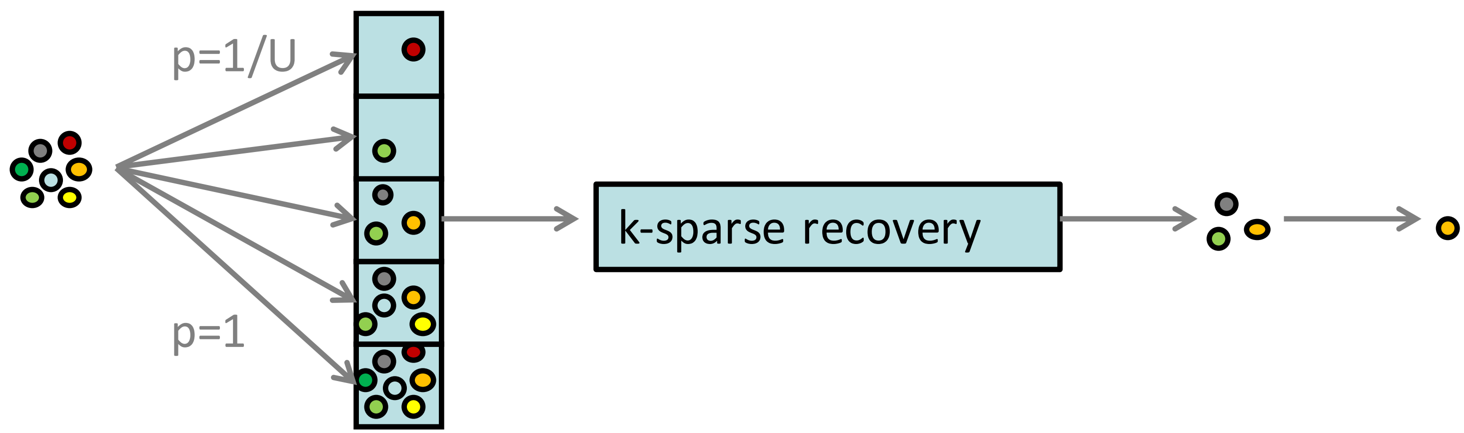
Advanced Sampling from Sketches

- Difficult case: inputs with positive and negative weights
- Want to sample based on the overall frequency distribution
  - Sample from support set of n possible items
  - Sample proportional to (absolute) total weights
  - Sample proportional to some function of weights
- How to do this sampling effectively?
  - Challenge: may be many elements with positive and negative weights
  - Aggregate weights may end up zero: how to find the non-zero weights?
- Recent approach: L_0 sampling
  - L_0 sampling enables novel "graph sketching" techniques
  - Sketches for connectivity, sparsifiers [Ahn, Guha, McGregor 12]

L_0 Sampling

- L_0 sampling: sample with prob $\approx f_i^0/F_0$
  - i.e., sample (near) uniformly from items with non-zero frequency
- General approach: [Frahling, Indyk, Sohler 05, C., Muthu, Rozenbaum 05]
  - Sub-sample all items (present or not) with probability p
  - Generate a sub-sampled vector of frequencies f_p
  - Feed f_p to a *k-sparse recovery* data structure
    - Allows reconstruction of f_p if $F_0 < k$
  - If f_p is k-sparse, sample from reconstructed vector
  - Repeat in parallel for exponentially shrinking values of p

Sampling Process



- Exponential set of probabilities, p=1, ½, ¼, 1/8, 1/16… 1/U
  - Let $N = F_0 = |\{ i : f_i \neq 0\}|$
  - Want there to be a level where k-sparse recovery will succeed
  - At level p, expected number of items selected S is Np
  - Pick level p so that $k/3 < Np \leq 2k/3$
- Chernoff bound: with probability exponential in k, $1 \leq S \leq k$
  - Pick $k = O(\log 1/\delta)$ to get $1-\delta$ probability

Hash-based sampling summary

- Use hash functions for sampling where some consistency is needed
  - Consistency over repeated keys
  - Consistency over distributed observations
- Hash functions have duality of random and fixed
  - Treat as random for statistical analysis
  - Treat as fixed for giving consistency properties
- Can become quite complex and subtle
  - Complex sampling distributions for consistent weighted sampling
  - Tricky combination of algorithms for L_0 sampling
- Plenty of scope for new hashing-based sampling methods

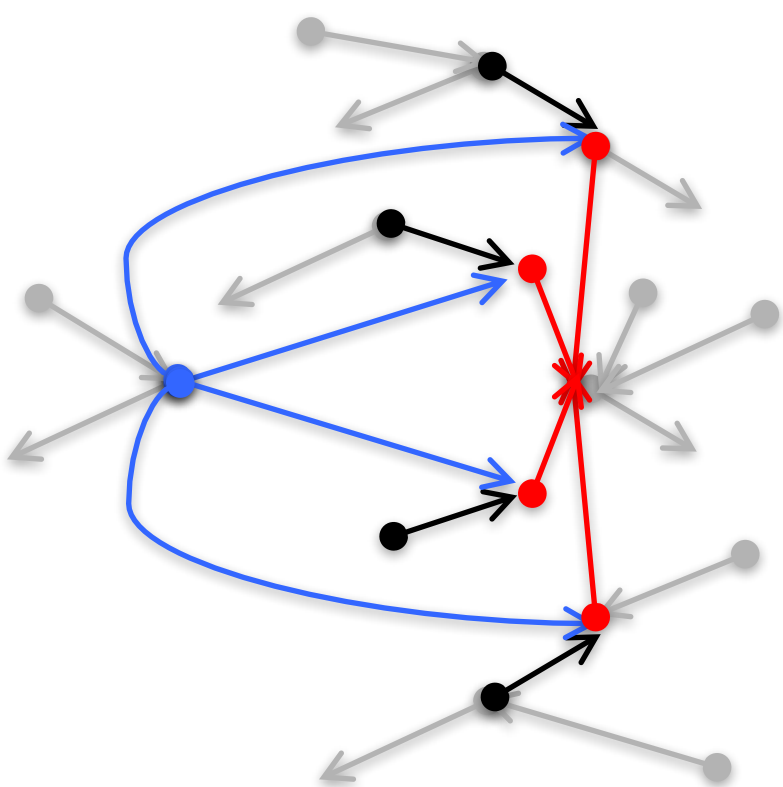
Data Scale:
Massive Graph Sampling

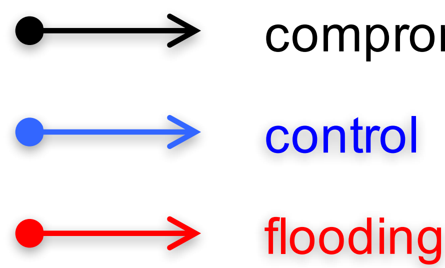
Massive Graph Sampling

- "Graph Service Providers"
  - Search providers: web graphs (billions of pages indexed)
  - Online social networks
    - Facebook: ~10^9 users (nodes), ~10^{12} links
  - ISPs: communications graphs
    - From flow records: node = src or dst IP, edge if traffic flows between them
- Graph service provider perspective
  - Already have all the data, but how to use it?
  - Want a general purpose sample that can:
    - Quickly provide answers to exploratory queries
    - Compactly archive snapshots for retrospective queries & baselining
- Graph consumer perspective
  - Want to obtain a realistic subgraph directly or via crawling/API

Retrospective analysis of ISP graphs

- Node = IP address
- Directed edge = flow from source node to destination node





BOTNET

- Hard to detect against background
- Known attacks can be detected:
  - Signature matching based on partial graphs, flow features, timing
- Unknown attacks are harder to spot:
  - exploratory & retrospective analysis
  - preserve accuracy if sampling?

Goals for Graph Sampling

Crudely divide into three classes of goal:

1. Study local (node or edge) properties
   - Average age of users (nodes), average length of conversation (edges)
2. Estimate global properties or parameters of the network
   - Average degree, shortest path distribution
3. Sample a “representative” subgraph
   - Test new algorithms and learning more quickly than on full graph

◊ Challenges: what properties should the sample preserve?

- The notion of “representative” is very subjective
- Can list properties that should be preserved
  (e.g. degree dbn, path length dbn), but there are always more…

Models for Graph Sampling

Many possible models, but reduce to two for simplicity

(see tutorial by Hasan, Ahmed, Neville, Kompella in KDD 13)

- Static model: full access to the graph to draw the sample
  - The (massive) graph is accessible in full to make the small sample
- Streaming model: edges arrive in some arbitrary order
  - Must make sampling decisions on the fly

- Other graph models capture different access scenarios
  - Crawling model: e.g. exploring the (deep) web, API gives node neighbours
  - Adjacency list streaming: see all neighbours of a node together

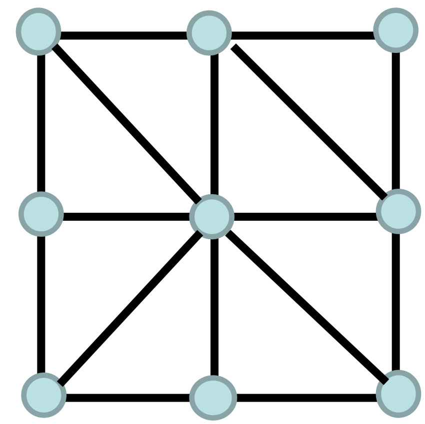
Node and Edge Properties

- Gross over-generalization:
  node and edge properties can be solved using previous techniques
  - Sample nodes/edge (in a stream)
  - Handle duplicates (same edge many times) via hash-based sampling
  - Track properties of sampled elements
    - E.g. count the degree of sampled nodes

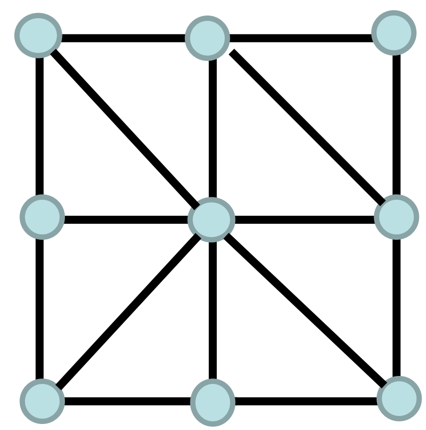
- Some challenges. E.g. how to sample a node proportional to its degree?
  - If degree is known (precomputed), then use these as weights
  - Else, sample edges uniformly, then sample each end with probability ½

Induced subgraph sampling

- Node-induced subgraph
  - Pass 1: Sample a set of nodes (e.g. uniformly)
  - Pass 2: collect all edges incident on sampled nodes
  - Can collapse into a single streaming pass
  - Can't know in advance how many edges will be sampled



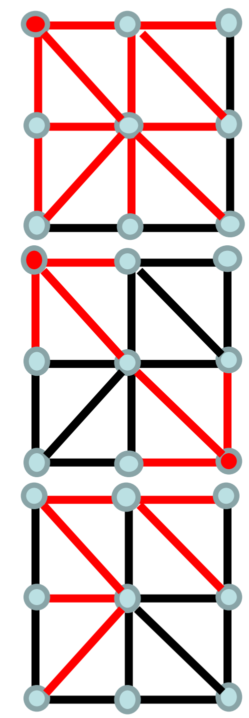
- Edge-induced subgraph
  - Sample a set of edges (e.g. uniformly in one pass)
  - Resulting graph tends to be sparse, disconnected
- Edge-induced variant [Ahmed Neville Kompella 13]:
  - Take second pass to fill in edges on sampled nodes
  - Hack: combine passes to fill in edges on current sample



HT Estimators for Graphs

- Can construct HT estimators from uniform vertex samples [Frank 78]
  - Evaluate the desired function on the sampled graph (e.g. average degree)
- For functions of edges (e.g. number of edges satisfying a property):
  - Scale up accordingly, by $N(N-1)/(k(k-1))$ for sample size k on graph size N
  - Variance of estimates can also be bounded in terms of N and k
- Similar for functions of three edges (triangles) and higher:
  - Scale up by $NC3/kC3 \approx 1/p^3$ to get unbiased estimator
  - High variance, so other sampling schemes have been developed

Graph Sampling Heuristics



"Heuristics", since few formal statistical properties are known

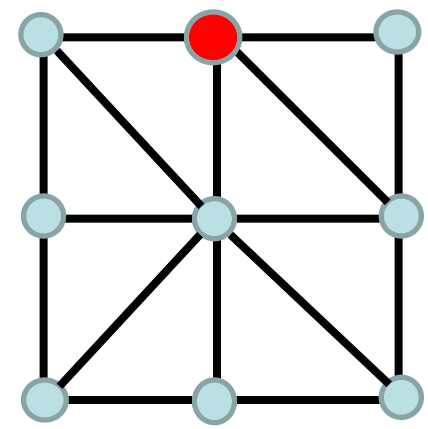
- ◊ Breadth first sampling: sample a node, then its neighbours…
  - – Biased towards high-degree nodes (more chances to reach them)
- ◊ Snowball sampling: generalize BF by picking many initial nodes
  - – Respondent-driven sampling: weight the snowball sample to give statistically sound estimates [Salganik Heckathorn 04]
- ◊ Forest-fire sampling: generalize BF by picking only a fraction of neighbours to explore [Leskovec Kleinberg Faloutsos 05]
  - – With probability p, move to a new node and "kill" current node

No "one true graph sampling method"

- – Experiments show different preferences, depending on graph and metric [Leskovec, Faloutsos 06; Hasan, Ahmed, Neville, Kompella 13]
- – None of these methods are "streaming friendly": require static graph
  - □ Hack: apply them to the stream of edges as-is

Random Walks Sampling

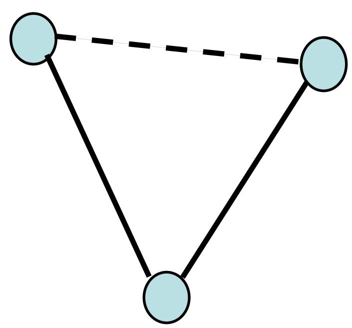
- ◊ Random walks have proven very effective for many graph computations
  - – PageRank for node importance, and many variations
- ◊ Random walk a natural model for sampling a node
  - – Perform "long enough" random walk to pick a node
  - – How long is "long enough" (for mixing of RW)?
  - – Can get "stuck" in a subgraph if graph not well-connected
  - – Costly to perform multiple random walks
  - – Highly non-streaming friendly, but suits graph crawling



- ◊ Multidimensional Random Walks [Ribeiro, Towsley 10]
  - – Pick k random nodes to initialize the sample
  - – Pick a random edge from the union of edges incident on the sample
  - – Can be viewed as a walk on a high-dimensional extension of the graph
  - – Outperforms running k independent random walks

Subgraph estimation: counting triangles

- Hot topic: sample-based triangle counting
  - Triangles: simplest non-trivial representation of node clustering
    - Regard as prototype for more complex subgraphs of interest
  - Measure of “clustering coefficient” in graph, parameter in graph models…
- Uniform sampling performs poorly:
  - Chance that randomly sampled edges happen to form subgraph is ≈ 0
- Bias the sampling so that desired subgraph is preferentially sampled



Subgraph Sampling in Streams

Want to sample one of the T triangles in a graph

- ◊ [Buriol et al 06]: sample an edge uniformly, then pick a node
  - – Scan for the edges that complete the triangle
  - – Probability of sampling a triangle is $T/(|E|\,(|V|-2))$
- ◊ [Pavan et al 13]: sample an edge, then sample an incident edge
  - – Scan for the edge that completes the triangle
  - – (After bias correction) probability of sampling a triangle is $T/(|E|\,\Delta)$
    - □ Δ = max degree, considerably smaller than $|V|$ in most graphs
- ◊ [Jha et.al. KDD 2013]: sample edges, the sample pairs of incident edges
  - – Scan for edges that complete "wedges" (edge pairs incident on a vertex)
- ◊ Advert: Graph Sample and Hold [Ahmed, Duffield, Neville, Kompella, KDD 2014]
  - – General framework for subgraph counting; e.g. triangle counting
  - – Similar accuracy to previous state of art, but using smaller storage

Graph Sampling Summary

- Sampling a representative graph from a massive graph is difficult!
- Current state of the art:
  - Sample nodes/edges uniformly from a stream
  - Heuristic sampling from static/streaming graph
- Sampling enables subgraph sampling/counting
  - Much effort devoted to triangles (smallest non-trivial subgraph)
- “Real” graphs are richer
  - Different node and edge types, attributes on both
  - Just scratching surface of sampling realistic graphs

Current Directions in Sampling

Outline

- ◊ Motivating application: sampling in large ISP networks
- ◊ Basics of sampling: concepts and estimation
- ◊ Stream sampling: uniform and weighted case
  - Variations: Concise sampling, sample and hold, sketch guided

BREAK

- ◊ Advanced stream sampling: sampling as cost optimization
  - VarOpt, priority, structure aware, and stable sampling
- ◊ Hashing and coordination
  - Bottom-k, consistent sampling and sketch-based sampling
- ◊ Graph sampling
  - Node, edge and subgraph sampling
- ◊ Conclusion and future directions

Role and Challenges for Sampling

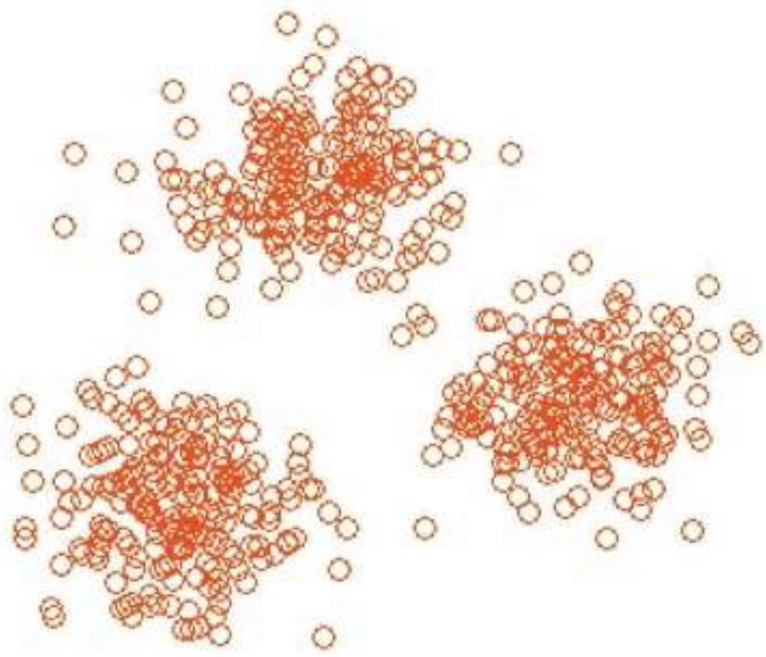
- Matching
  - Sampling mediates between data characteristics and analysis needs
  - Example: sample from power-law distribution of bytes per flow…
    - but also make accurate estimates from samples
    - simple uniform sampling misses the large flows
- Balance
  - Weighted sampling across key-functions: e.g. customers, network paths, geolocations
    - cover small customers, not just large
    - cover all network elements, not just highly utilized
- Consistency
  - Sample all views of same event, flow, customer, network element
    - across different datasets, at different times
    - independent sampling ⇒ small intersection of views

Sampling and Big Data Systems

- ◊ Sampling is still a useful tool in cluster computing
  - – Reduce the latency of experimental analysis and algorithm design
- ◊ Sampling as an operator is easy to implement in MapReduce
  - – For uniform or weighted sampling of tuples
- ◊ Graph computations are a core motivator of big data
  - – PageRank as a canonical big computation
  - – Graph-specific systems emerging (Pregel, LFgraph, Graphlab, Giraph…)
  - – But… sampling primitives not yet prevalent in evolving graph systems
- ◊ When to do the sampling?
  - – Option 1: Sample as an initial step in the computation
    - □ Fold sample into the initial “Map” step
  - – Option 2: Sample to create a stored sample graph before computation
    - □ Allows more complex sampling, e.g. random walk sampling

Sampling + KDD

- The interplay between sampling and data mining is not well understood
  - Need an understanding of how ML/DM algorithms are affected by sampling
  - E.g. how big a sample is needed to build an accurate classifier?
  - E.g. what sampling strategy optimizes cluster quality
- Expect results to be method specific
  - I.e. “IPPS + k-means” rather than “sample + cluster”



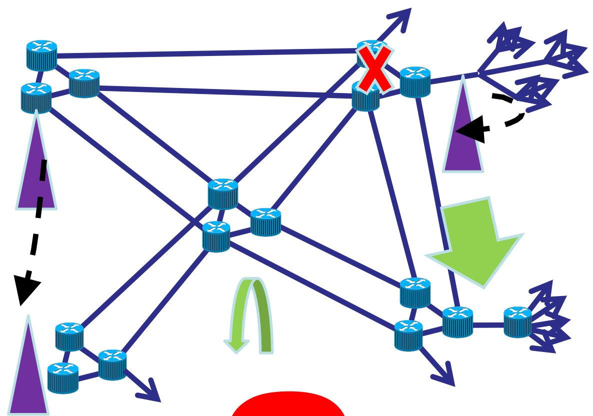
Sampling and Privacy

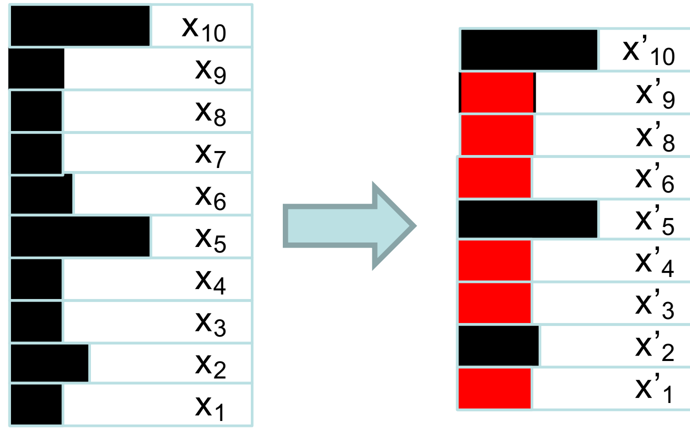
- Current focus on privacy-preserving data mining
  - Deliver promise of big data without sacrificing privacy?
  - Opportunity for sampling to be part of the solution
- Naïve sampling provides "privacy in expectation"
  - Your data remains private if you aren't included in the sample…
- Intuition: uncertainty introduced by sampling *contributes* to privacy
  - This intuition can be formalized with different privacy models
- Sampling can be analyzed in the context of differential privacy
  - Sampling alone does **not** provide differential privacy
  - But applying a DP method to sampled data does guarantee privacy
  - A tradeoff between sampling rate and privacy parameters
    - Sometimes, lower sampling rate improves overall accuracy

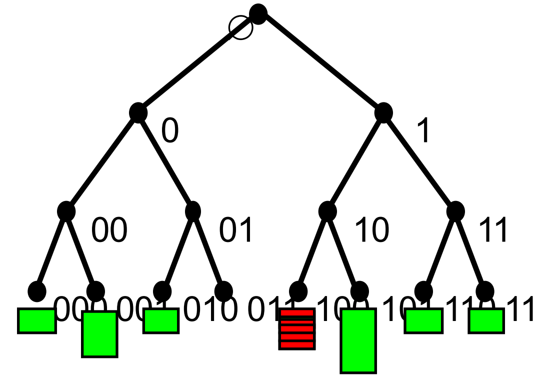
Advert: Now Hiring...

- Nick Duffield, Texas A&M
  - Phds in big data, graph sampling

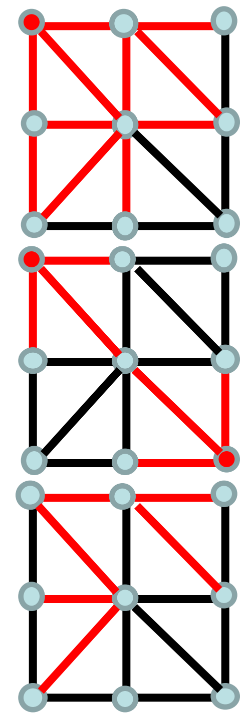
- Graham Cormode, University of Warwick UK
  - Phds in big data summarization (graphs and matrices, funded by MSR)
  - Postdocs in privacy and data modeling (funded by EC, AT&T)

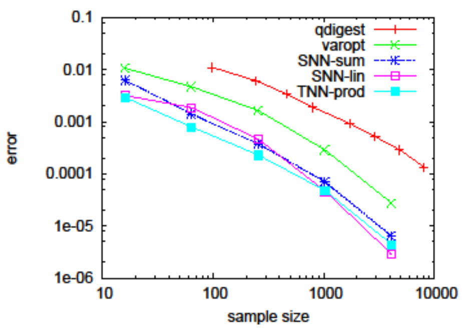






Sampling for Big Data





Graham Cormode, University of Warwick

G.Cormode@warwick.ac.uk

Nick Duffield, Texas A&M University

duffieldng@tamu.edu