The Edge --- Randomized Algorithms for Network Monitoring

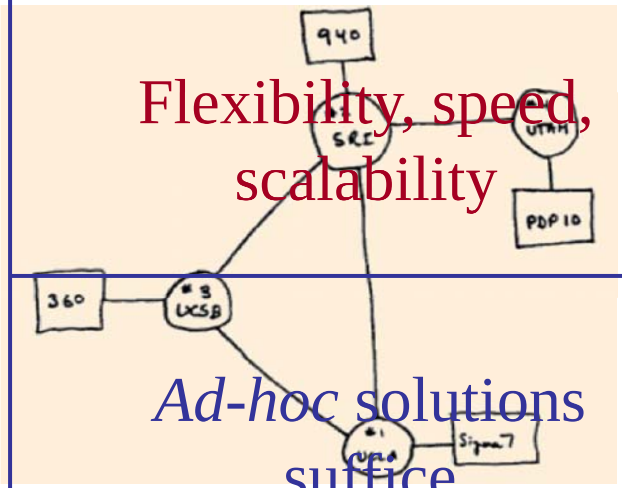
George Varghese

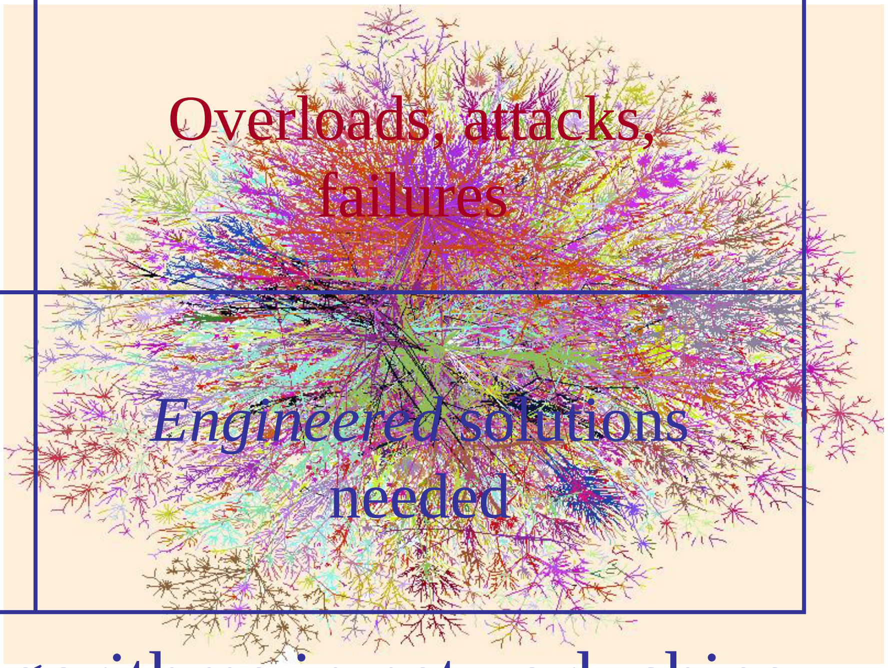
November 13 2008

UCSDCSE
Computer Science and Engineering

cns
center for networked systems

Research motivation





	The Internet in 1969	The Internet today
Problems	Flexibility, speed, scalability	Overloads, attacks, failures
Measurement & control	*Ad-hoc* solutions suffice	*Engineered* solutions needed

This talk: using randomized algorithms in network chips for monitoring performance and security in routers

Focus on 3 Monitoring Problems

- Problem 1: Finding heavy-bandwidth flows
- Problem 2: Measuring usec network latencies
- Problem 3: Logging all infected nodes during an attack with limited memory

In each case, a simple “sampling” scheme works

But in each case, if the router can add some memory and processing, we can get an edge . . .

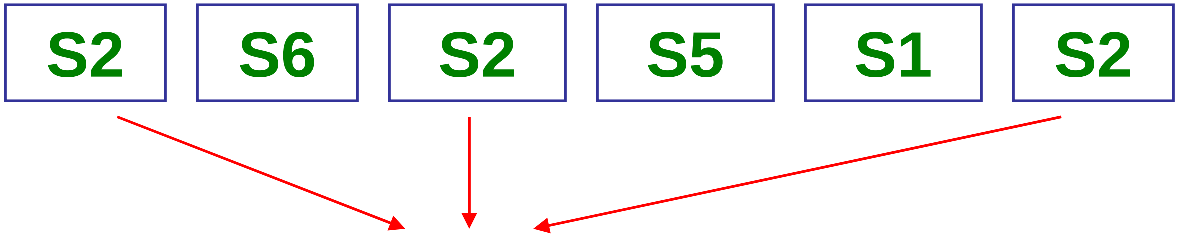
Get edge subject to constraints

- Low memory: On-chip SRAM limited to around 32 Mbits. Not constant but is not scaling with number of concurrent conversations/packets

- Small processing: For wire-speed at 40 Gbps, using 40 byte packets, have 8 nsec. Using 1 nsec SRAM, 8 memory accesses. Factor of 30 in parallelism buys 240 accesses.

Problem 1: Heavy-bandwidth users

Heavy-hitters: In a measurement interval, (e.g., 1 minute) measure the flows (e.g., sources) on a link that send more than a threshold T (say 1% of the traffic) on a link using memory M < < F, the number of flows



Source S2 is 30 percent of traffic sequence

Estan,Varghese, ACM TOCS 2003

Getting an Edge for heavy-hitters

- Sample: Keep a M size sample of packets. Estimate heavy-hitter traffic from sample
- Sample and Hold: Sampled sources held in a CAM of size M. All later packets counted
- Edge: Standard error of bandwidth estimate is O(1/M) for S&H instead of O(1/sqrt(M))
- Improvement: (Prabhakar et al): Periodically remove “mice” from “elephant trap”

Problem 2: Fine-Grain Loss and Latency Measurement

(with Kompella, Levchenko, Snoeren)

SIGCOMM 2009, to appear

Fine-grained measurement critical

- Delay and loss requirements have intensified:
    - VoIP, IPTV, Gaming
        - » < 200 msec latency, small loss
    - Automated financial programs
        - » < 100 usec latency, very small (1 in 100,000) loss?
    - High-performance computing
        - » < 10 usec, very small loss
- New end-to-end metrics of interest
    - Average delay (accurate to < msec, possibly microsecs)
    - Jitter (delay variance helps)
    - Loss distribution (random vs microbursts, TCP timeouts)

Existing router infrastructure

- SNMP (simple aggregate packet counters)
  - Coarse throughput estimates not latency
- NetFlow (packet samples)
  - Need to coordinate samples for latency. Coarse

Applying existing techniques

- Standard approach is active probes and tomography
  - Join results from many paths to infer per-link properties
  - Can be applied to measuring all the metrics of interest

- Limitations
  - Overheads for sending probes limits granularity
    - Cannot be used to measure latencies in 100’s of μsecs)
  - Tomography inaccurate due to under-constrained formulation

No guarantee that metrics measured by *probes* are representative of those experienced by any particular traffic flow

Our approach

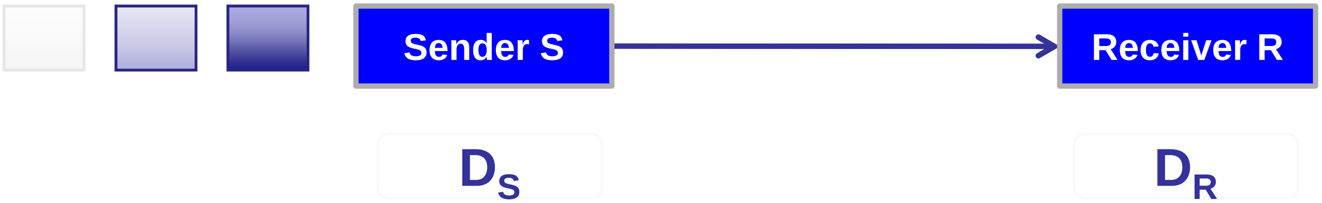
- Add hardware to monitor each segment in path
  - Use a low-cost primitive for monitoring individual segments
  - Compute path properties through segment composition
  - Ideally, segment monitoring uses few resources
    - Maybe even cheap enough for ubiquitous deployment!
- This talk shows our first steps
  - Introduce a data structure called an LDA as key primitive
  - We’ll use a only small set of registers and hashing
  - Compute loss, delay average and variance, loss distribution

We measure *real traffic* as opposed to injected probes

Outline

- Model
- Why simple data structures do not work
- LDA for average delay and variance

Abstract segment model



Sender S

Receiver R

D_S

D_R

- Packets always travel from S to R
  - R to S is considered separately
- Divide time into equal bins (measurement intervals)
  - Interval depends on granularity required (typically sub-second)
- Both S and R maintain some state D about packets
  - State is updated upon packet departure
- S transmits D_S to R
  - R computes the required metric as $f(D_S , D_R)$

Assumptions

- Assumption 1: FIFO link between sender and receiver
- Assumption 2: Fine-grained *per-segment* time synchronization
    - Using IEEE 1588 protocol, for example
- Assumption 3: Link susceptible to loss as well as variable delay
- Assumption 4: A little bit of hardware can be put in the routers

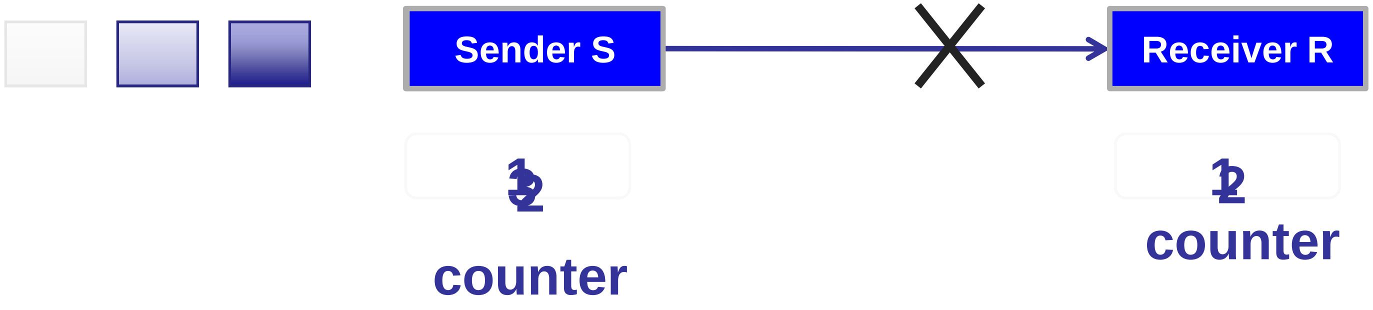
- You may have objections, we will address common ones later

Constraints

- Constraint 1: Very little high-speed memory
- Constraint 2: Limited measurement communication budget
- Constraint 3: Constrained processing capacity

- Consider a run-of-the-mill OC-192 (10-Gbps) link
  - 250-byte packets implies 5 million packets per second
  - At most 1 control packet every msec, more likely once per sec

Computing loss



Loss =
3 – 2 = 1

- Store a packet counter at S and R.
- S sends the counter value to R periodically
- R computes loss by subtracting its counter value from S's counter

Computing delay (naïve)

10	23	13
12	26	14
15	35	20

Avg delay **47/3 = 15.7**

- A naïve first cut: timestamps
    - Store timestamps for each packet at sender, receiver
    - After every cycle, sender sends the packet timestamps to the receiver
    - Receiver computes individual delays, and computes average
    - 5M packets require ~ 25,000 packets (200 labels per packet)

Extremely high communication and storage costs

Computing delay (sampled)

Sender S

Receiver R

10

15

23

35

13

20

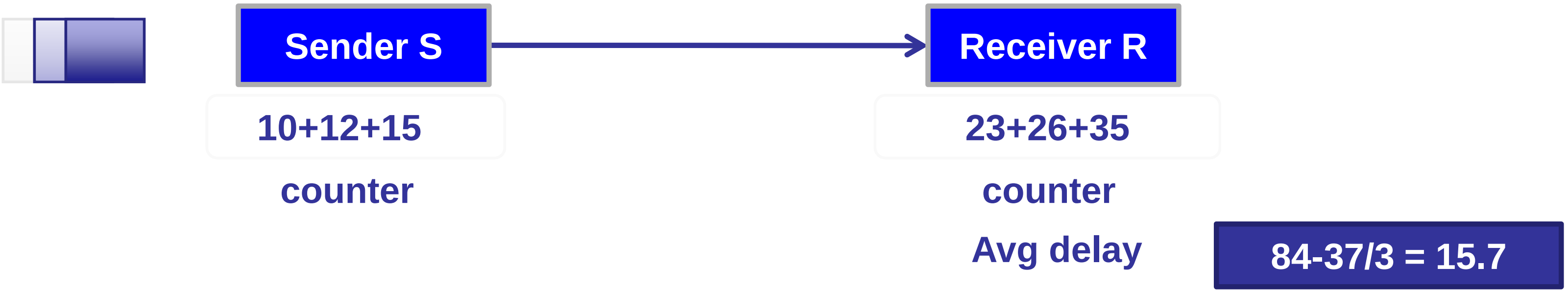
Avg delay

33/2 = 16.5

- (Slightly) better approach: sampling
  - Store timestamps for only sampled packets at sender, receiver
  - 1 in 100 sampling means ~ 250 packets

Less expensive, but we can get an edge . . .

Delay with no packet loss



Sender S

Receiver R

10+12+15

counter

23+26+35

counter

Avg delay

84-37/3 = 15.7

- Observation: Aggregation can reduce cost
  - Store sum of the timestamps at S & R
  - After every cycle, S sends its sum C_S to R
  - R computes average delay as $(C_S – C_R) / N$
  - Only one counter and one packet to send

Works great, if packets were never lost…

Delay in the presence of loss

- Consider two packets, first sent at T/2 and lost. Second sent at T, received at T. Receiver gets D = T/4
- Lost packets can cause Error = O(T) where T is the size of the measurement interval
- Failed quick fix: Bloom filter will not work
  - Always a finite false positive probability

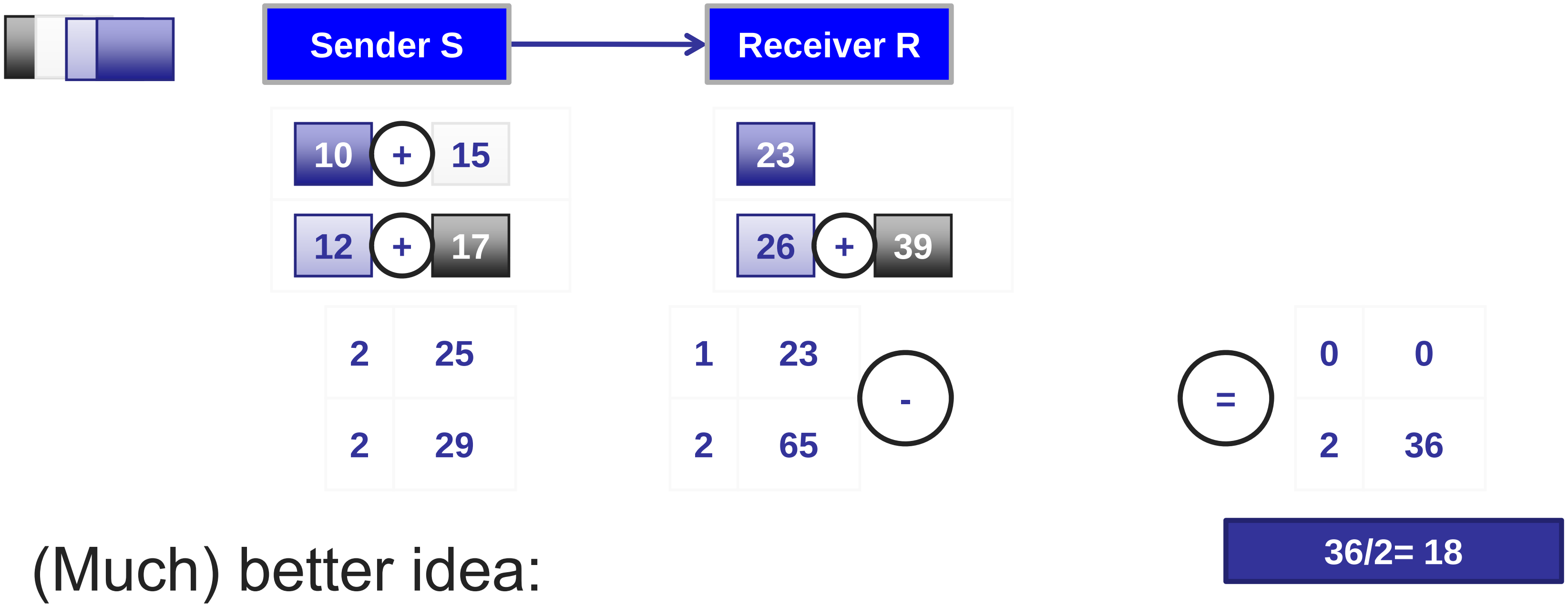
Theory Perspective: Streaming

- Streaming algorithms a massive field of study in theory, databases, and web analysis

- However, our problem has two big differences:
  - Coordination: Instead of calculating F(s_i) on **one** stream s_i. we compute F(s_i, r_i) on **two** streams s_i and r_i
  - Loss: Some of the r_i can be undefined because of loss

- Example: Max is trivial in streaming setting but provably requires linear memory in coordinated setting

Delay in the presence of loss



Sender S → **Receiver R**

10 + 15

12 + 17

23

26 + 39

2	25
2	29

1	23
2	65

\-

=

0	0
2	36

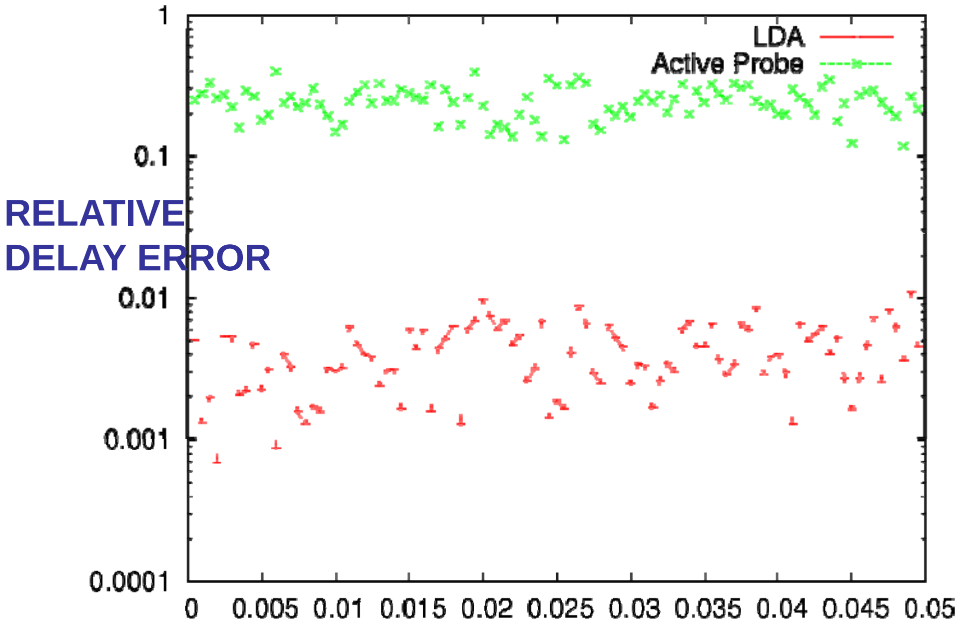
36/2= 18

- (Much) better idea:
  - Spread loss across several buckets
  - Discard buckets with lost packets
- Lossy Difference Aggregator (LDA)
  - Hash table with packet count and timestamp sum

Analysis and Refinements

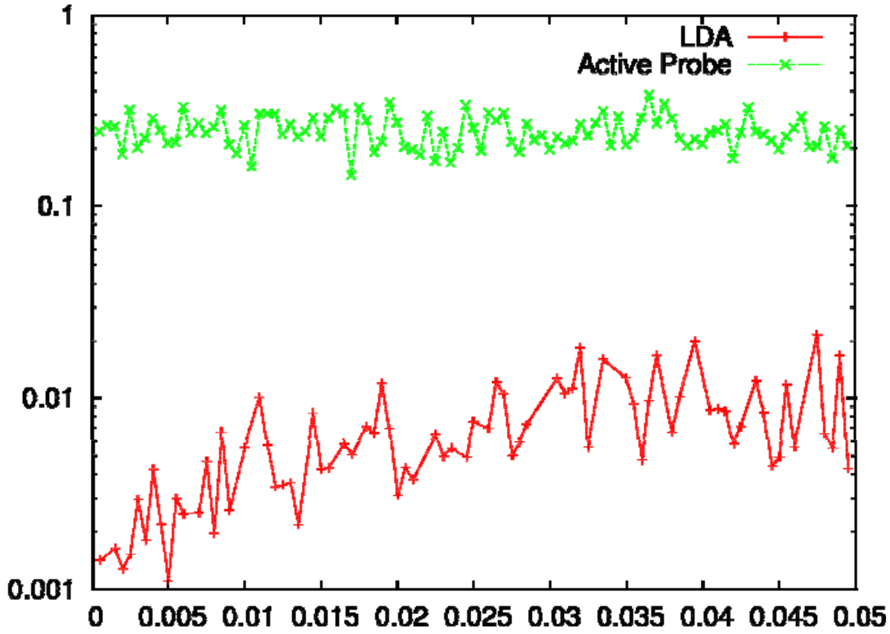
- Packet loss
  - k packet losses can corrupt up to k buckets
  - If k << B, then only a small subset of buckets corrupted
- Problem: High loss implies many bad buckets
- Solution: Sampling
  - Control sampling rate such that no more than B/2 buckets corrupted (based on loss rate)
- Problem: Loss rate is unpredictable
- Solution: Run parallel copies for several loss rates
  - Logarithmic copies suffice in theory, smaller in practice

Comparison to active probes



LOSS

Sampling rate chosen *statically* for 5% loss to lose B/2 packets



LOSS

Sampling rate chosen *dynamically* for each loss rate to lose B/2 packets

Computing jitter

- Propose measuring jitter as variance in delay
- Can we adapt LDA to measure variance ?
- Solution idea: inspired by sketching [AMS96]
  - Consider random variable X_i that takes values +1 and -1 with probability ½
  - At S and R, packet p_i has timestamps a_i and b_i
  - S transmits $\sum a_i * X_i$ to R
  - R computes $(\sum b_i * X_i - \sum a_i * X_i)^2 / n - \mu^2$ to obtain variance

Why this works (AMS 96)

$$E[(\sum b_i \times X_i - \sum a_i \times X_i)^2]$$

$$= E[(\sum \delta_i \times X_i)^2]$$

$$= E[\sum \delta_i^2 \times X_i^{\ 2} + 2\sum \delta_i \delta_j \times X_i X_j]$$

$$= \sum \delta_i^2 \times \underbrace{E[X_i^{\ 2}]}_{=1} + 2\sum \delta_i \delta_j \times \underbrace{E[X_i X_j]}_{=0}$$

$$= \sum \delta_i^2$$

Other issues

- Implementation: counters plus increment/decrement. 200 SRAM counters < 1% of 95 nm ASIC
- FIFO model: load balancing breaks model, need to enforce by doing on each link in hunt group
- Deployment: deploy within single router first using flow through logic: majority of loss, delay within routers
- Time synchronization: being done within routers, also across links with IEEE 1588 and GPS (Corvil)

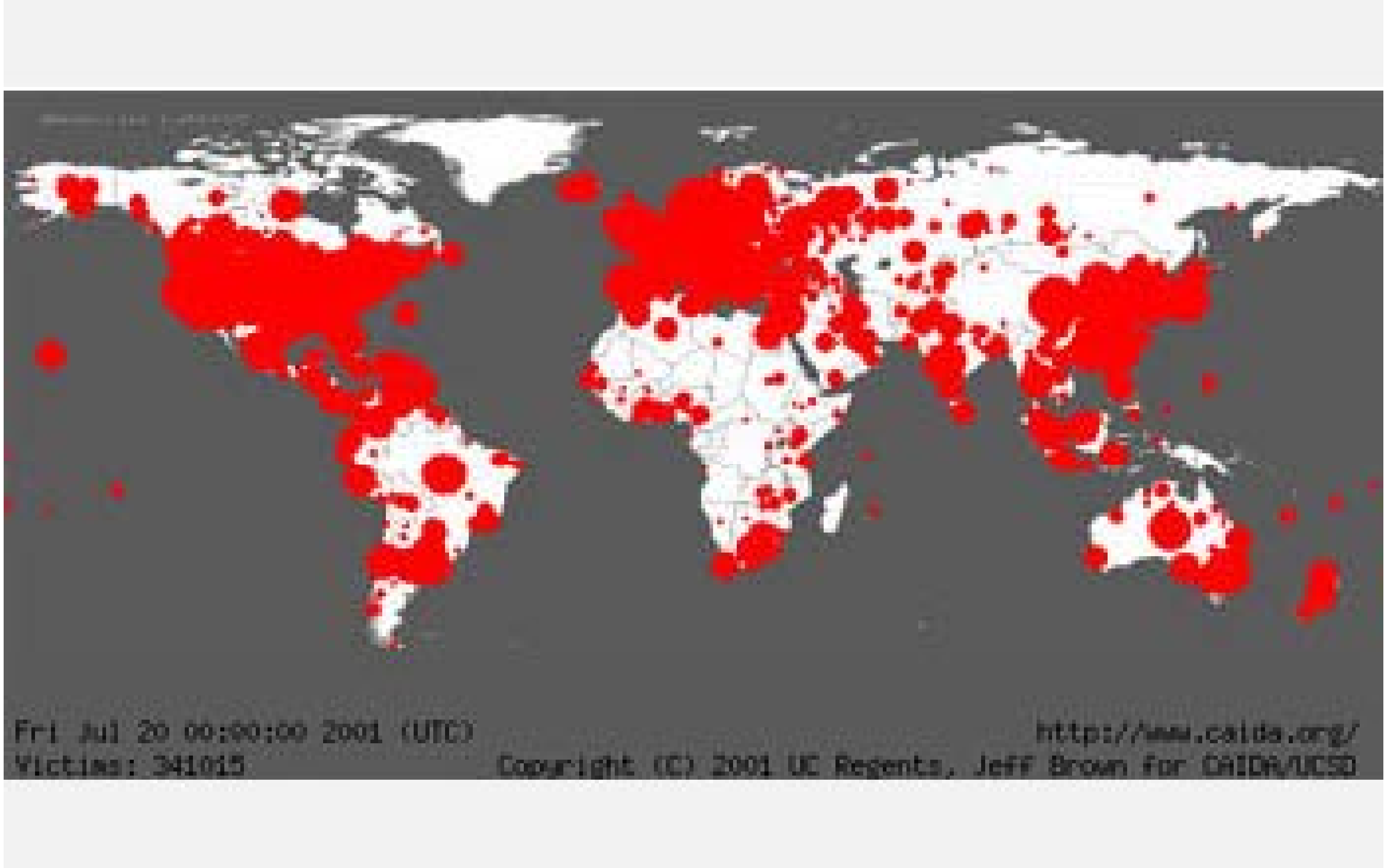
Summary of Problem 2

- With rise in modern trading and video applications, fine grained latency is important. Active probes cannot provide latencies down to microseconds
- Proposed LDAs for performance monitoring as a new synopsis data structure
  - Simple to implement and deploy ubiquitously
  - Capable of measuring average delay, variance, loss and possibly detecting microbursts
  - Edge is N samples (1 million) versus M samples (1000) for no-error case. Reduces error by 300.

Problem 3: Scalable Logging

(with Terry Lam)

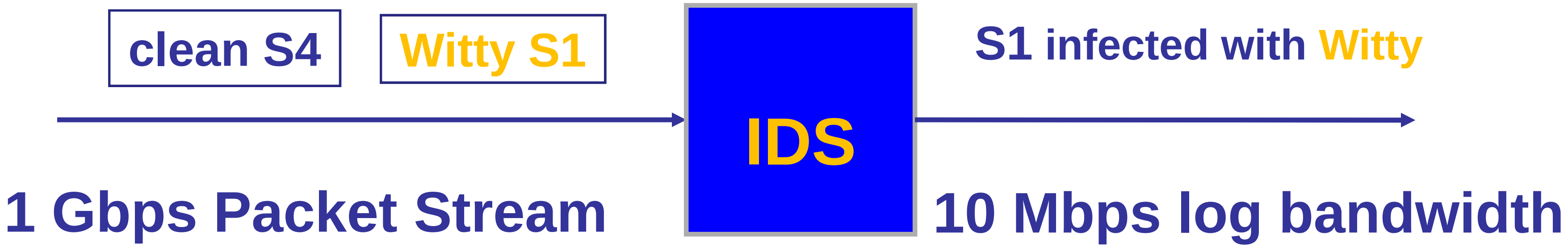
Spread of Code Red



Source: CAIDA Visualization

"Worms!? She must have picked them up on the Internet."

Scalable logging problem



clean S4

Witty S1

IDS

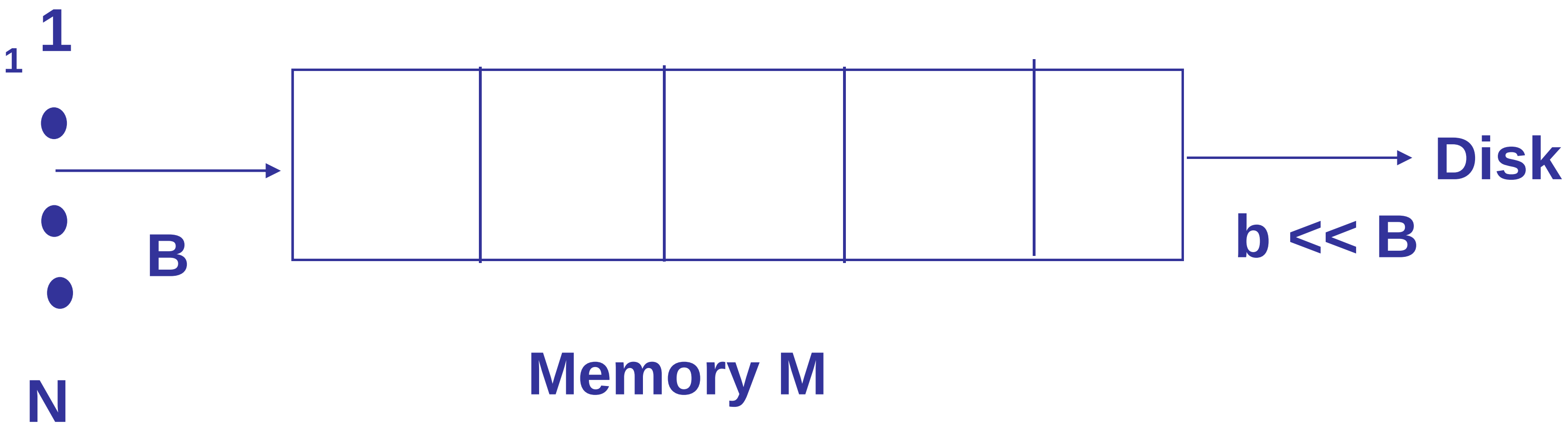
S1 infected with Witty

1 Gbps Packet Stream

10 Mbps log bandwidth

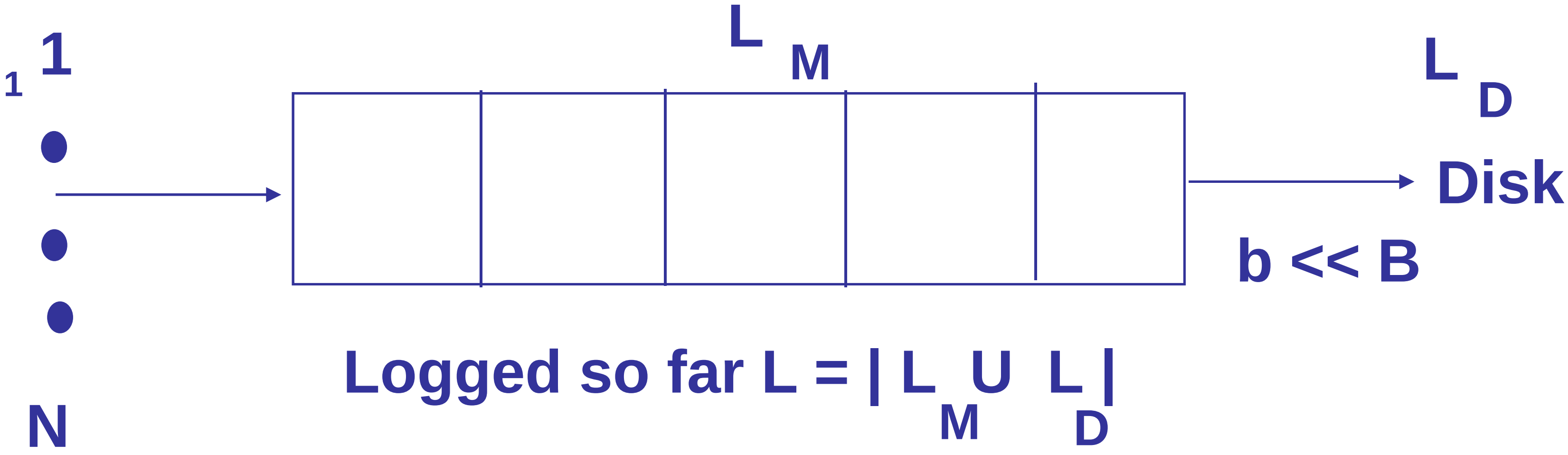
- **Setting:** IDS has a list of signatures, manual (Snort) or automatically learned.
- **Function:** Each time a packet matches a signature, IDS should log the packet source to disk
- **Difficulty:** Millions of infected sources, small memory at IDS, small logging bandwidth

Scalable logging Model



- Challenges:
  - Small logging bandwidth: b < < arrival rate B
  - Small memory: M < < number of sources N
  - Memory can fill with sources logged to disk

Naïve scheme performance



Logged so far $L = |L_M \cup L_D|$

- In steady state, every 1/b time, head leaves to disk
- Probability replacement is new is (N – L) / N
- Expected time L → L + 1 is N /(N – L) b
- Time to log all sources is (ln N – ln M) N /b

In worst case model, time can be infinite!

Simple techniques...

- Naïve scheme is ln N/M worse than optimal even in optimistic random model
- Keeping a hash table or Bloom filter does not help significantly because we have only M memory and so cannot keep track of sources logged to disk.
- Clearing hash table or Bloom Filter periodically does not help as same sources may reappear

Congestion Control to the rescue?

- Many packets must complete to obtain value. Random dropping leads to congestion collapse
- Closed loop congestion control (TCP, Ethernet): needs cooperative sources
- Classical solution: admission control. Again requires cooperation
- What can a poor resource do to protect itself unilaterally without cooperation from senders?

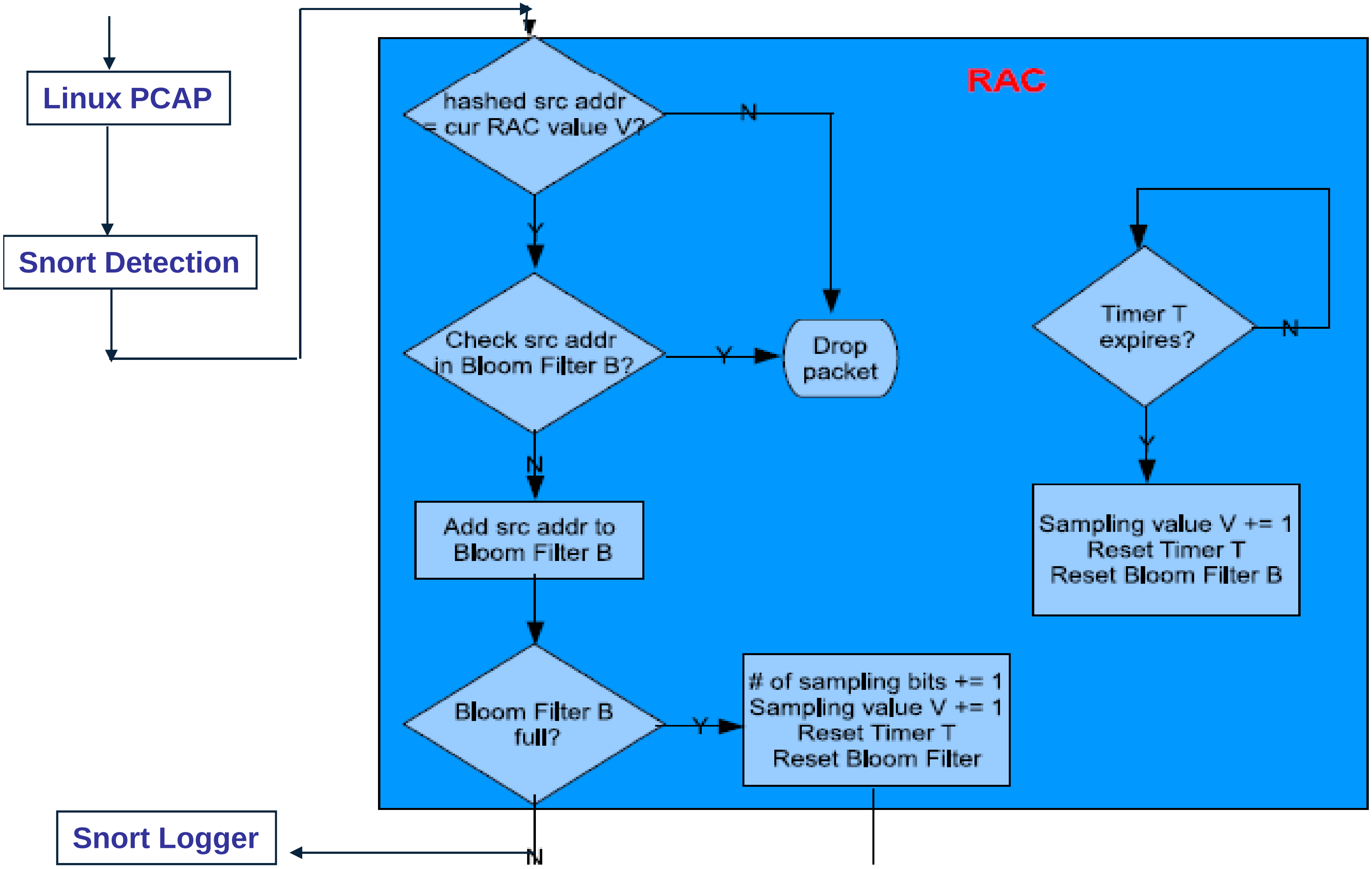
Randomized admission control

Randomly select as many sources as possible

- **Select:** Only if low order k bits of Hash(S) = V. Add S to Bloom Filter
- **Adjust:** Halve sources (k→ k+1) if Bloom Filter is full
- **Iterate:** V → V + 1 after time T to capture all sources

Long term fairness and *small* memory and processing

Adding RAC to Snort

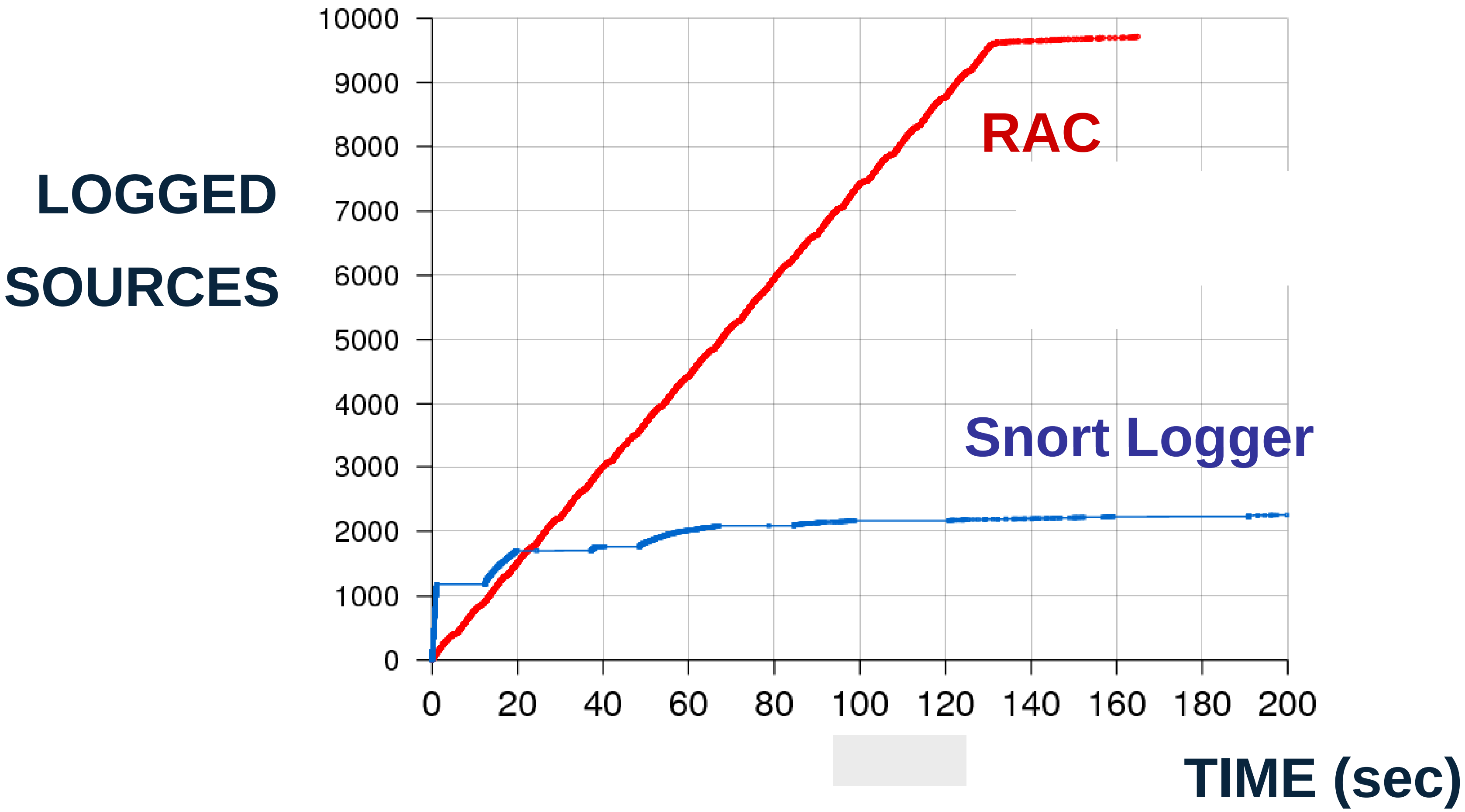


Linux PCAP

Snort Detection

Snort Logger

RAC Logging vs Snort Logging



- Ratio of inbound traffic/logging rate = 10.
- RAC logs 96% in 120 second, Snort saturates at 20%

Edge can be an order of magnitude

- RAC is factor of 2 off from optimal to log all sources versus ln N/M off for naïve.
  - For N = 1 million and M small, edge is close to 14 for random arrivals; infinite for worst-case

- LDA offers N samples versus M samples for naïve ‘
  - For N = 1 million, M = 10,000, edge is close to 10

- Sample and Hold offers O(1/M) standard error versus O(1/sqrt(M)) for naïve
  - For M = 10,000, edge in standard error is 100

Related Work

- LDA:
  - Streaming Algorithms: less work on 2-party streaming algorithms between a sender and receiver
  - Network tomography: joins the result of black box measurements to infer link delays and losses
- RAC:
  - Random partitions a common idea. We apply to admission control and add cycling through partitions
  - Alto Scavenger "discards information for half the files" if disk full

Summary

- Monitoring networks for performance problems and security at high speeds is important but hard
- Randomized streaming algorithms can offer cheaper (in gates) solutions at high speeds.
- Described two simple randomized algorithms
    - LDA: Aggregate by summing, hash to withstand loss
    - RAC: Randomly partition input into small enough sets. Cycle through sets for fairness.

In conclusion . . .