Probabilistic Traffic Monitoring

Basic Concepts, Large-Flow Detection, Duplicate Detection

Network Security AS 2020

15 December 2020

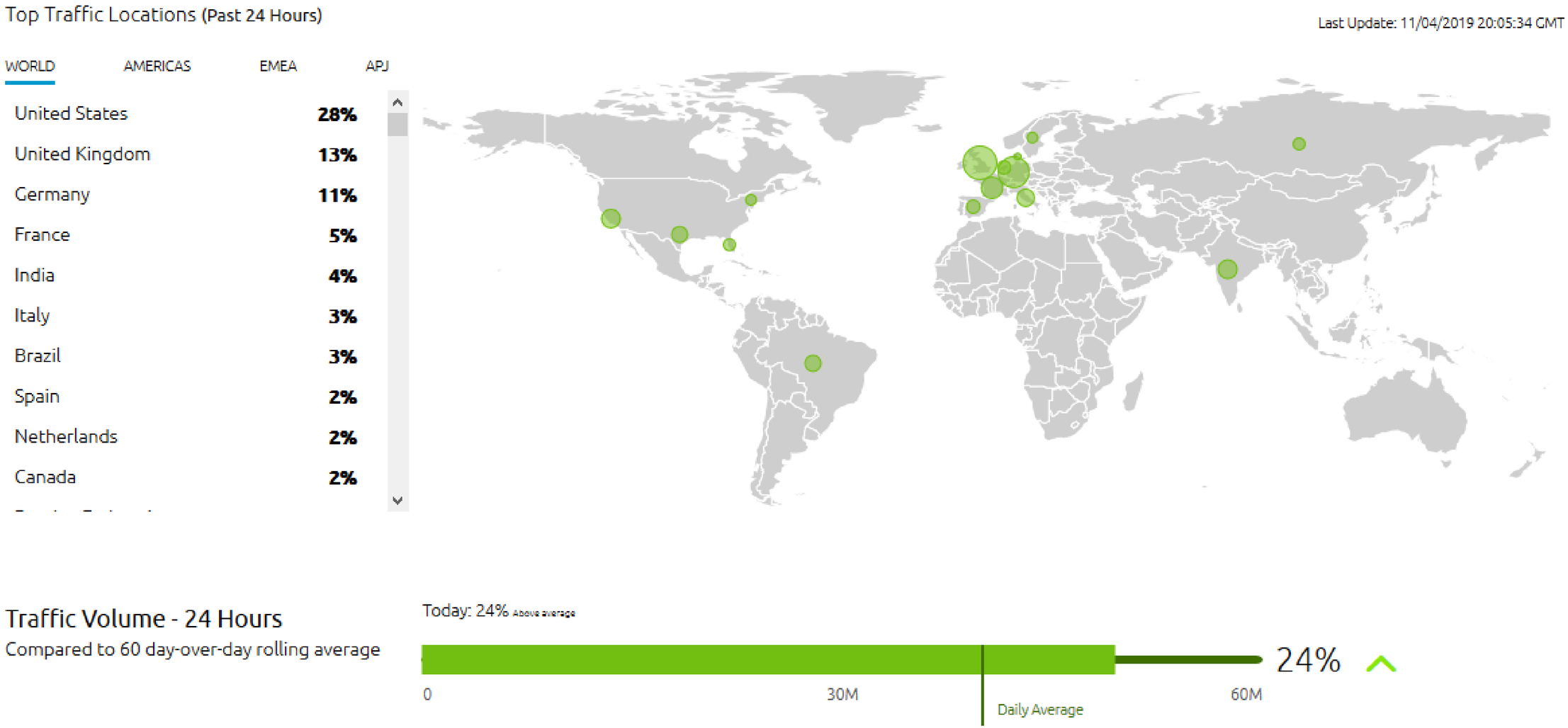
Adrian Perrig
(some slides by M. Legner, H.-C. Hsiao)

ETH zürich

Lecture Objectives

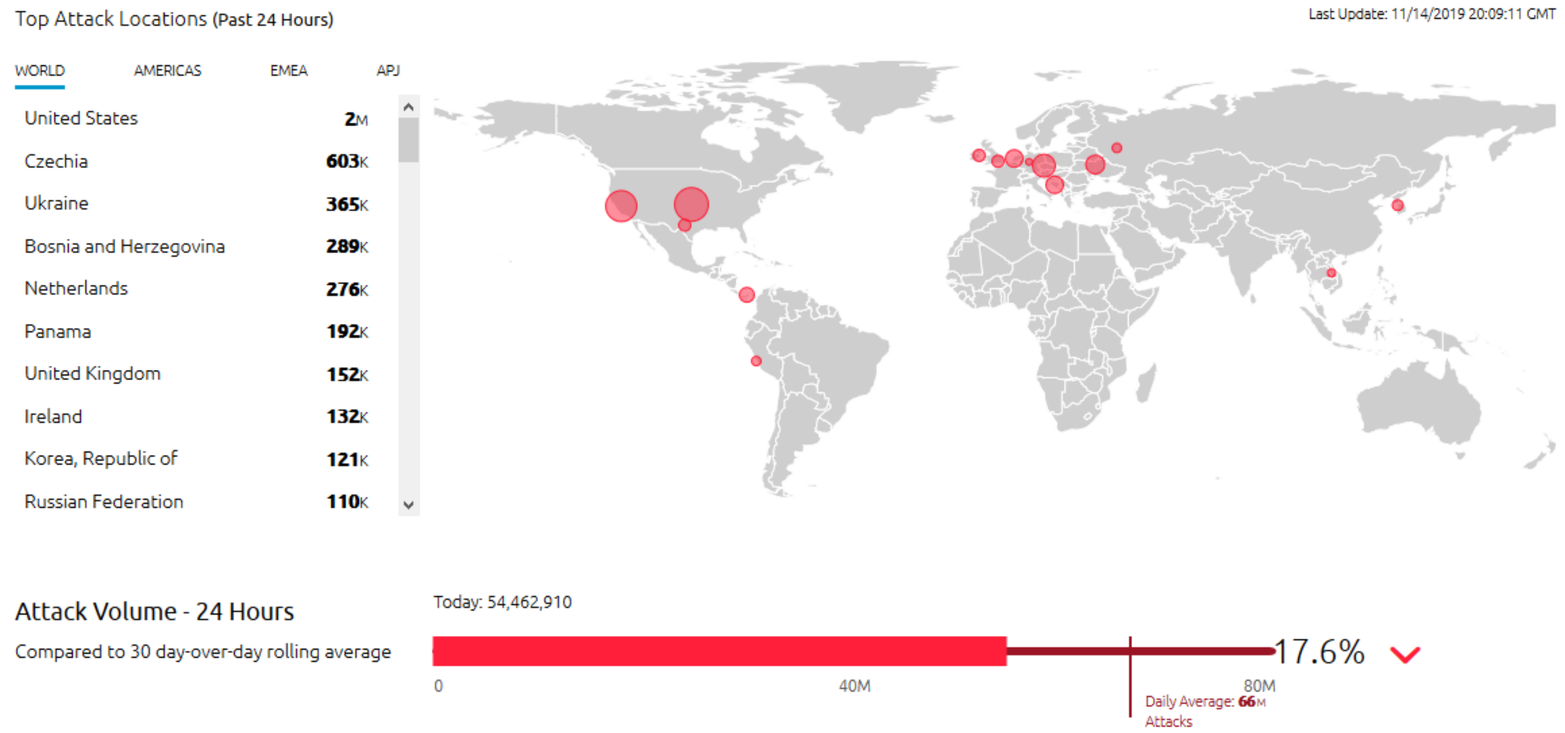
1. Ability to recognize challenges of designing traffic-monitoring schemes
2. Ability to analyze monitoring schemes and identify their pros and cons
3. Ability to apply tricks and primitives learned today

Network Traffic Monitoring



Akamai Real-Time Web Monitor.
https://www.akamai.com/us/en/resources/visualizing-akamai/real-time-web-monitor.jsp

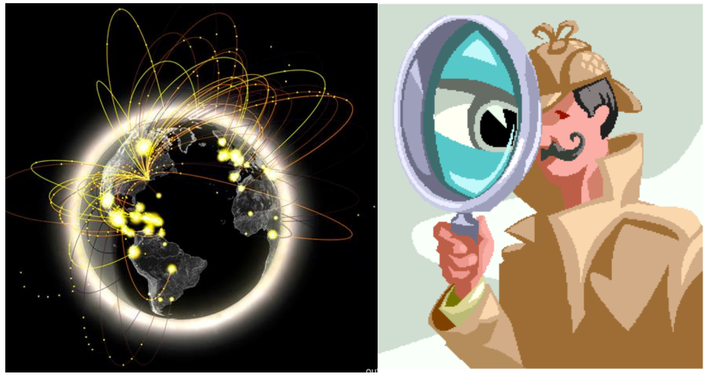
Network Traffic Monitoring



Akamai Real-Time Web Monitor.

Applications of Traffic Monitoring

- Anomaly detection
  - Volume-based (denial of service – DoS) or port scans
  - ISP wants to enforce quality of service or maximal sending rate
- Network management
  - Accounting, such as usage-based pricing
    - Often done with deterministic methods
  - Traffic engineering, such as balancing traffic loads



http://senseable.mit.edu/nyte/visuals.html

On what granularity do we perform monitoring? Traffic Flows

- What's a flow?
  - A set of packets with common values in one or more header fields (or a flow identifier) observed in a given time interval
  - A flow identifier (FID) is typically a 5-tuple:
    {Src IP, Dst IP, Src Port, Dst Port, Protocol}
  - IPv6 has an explicit 20bit "Flow Label"
- Some applications are interested in a subset of header fields:
  - DDoS detection: {*, **Dst IP**, *, *, *}
  - Source bandwidth monitoring for usage-based pricing: {**Src IP**, *, *, *, *}

Why is traffic monitoring difficult?

- Core routers in the Internet forward **multiple terabit of traffic per second**
  - Links of 40/100 Gbps are widespread; 400 Gbps are being deployed
  - Routers can have dozens of these links
- Empirical evidence: 20 million different flows on a 1 Tbps router

CISCO ASR 9922
https://twitter.com/olesovhcom/status/215452504614379521

Why is traffic monitoring difficult?

- Consider a **1 Tbps** router and **1KB** packets
- 1 Tbps ≈ 2^{40} bit/s, 1KB = 2^{13} bit

  → 2^{40} / 2^{13} = 2^{40-13} = 2^{27} ≈ 10^{8} = 100 million packets per second

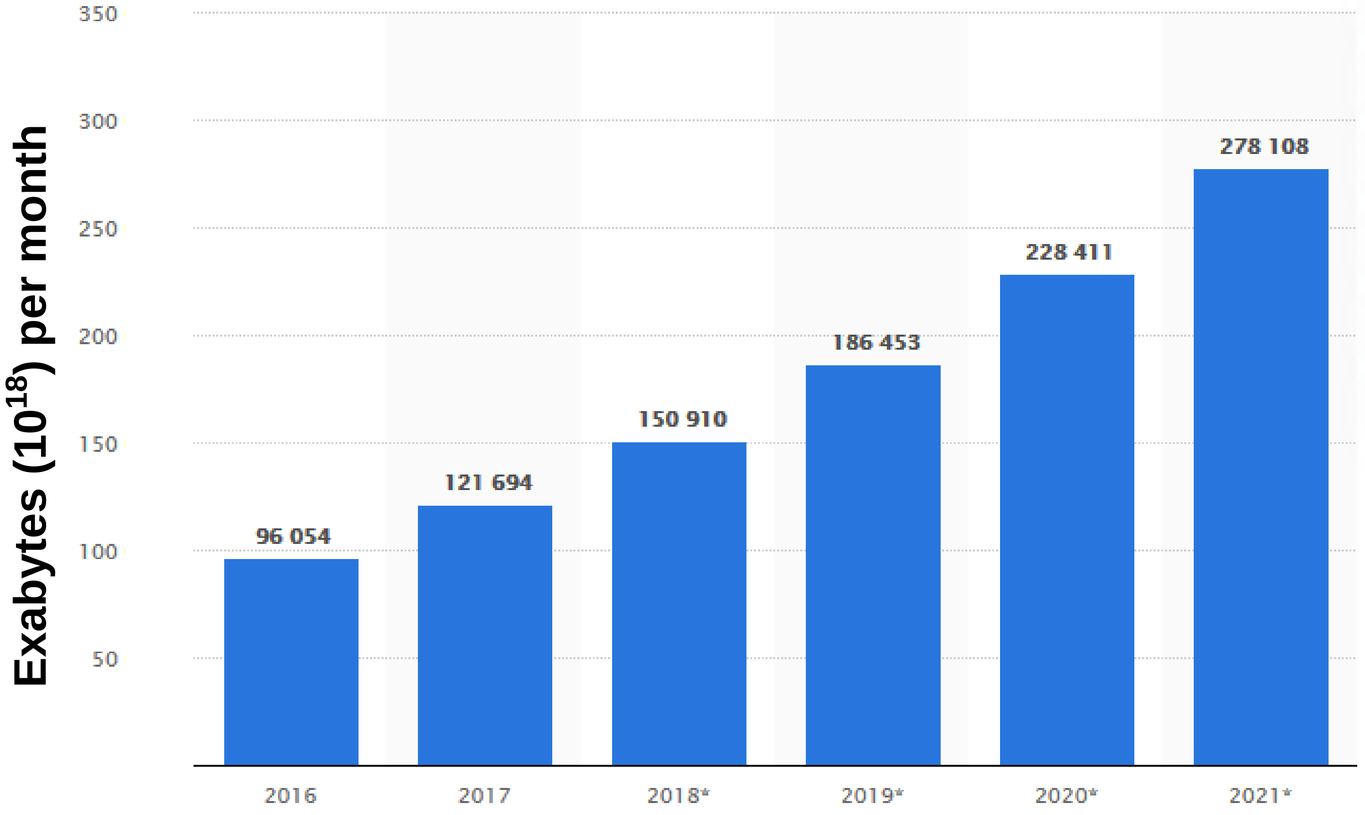
→ One packet every 10 ns

- 10 ns: one AES operation, light travels 3 m
- DRAM lookup takes >100 ns

→ Packet processing must be extremely efficient

Why is traffic monitoring difficult?

Year	Global Internet Traffic
1992	100 GB per day
1997	100 GB per hour
2002	100 GB per second
2007	2 TB per second
2017	47 TB per second
2022	150 TB per second



https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html
https://www.statista.com/statistics/499431/global-ip-data-traffic-forecast/

Why is traffic monitoring difficult?

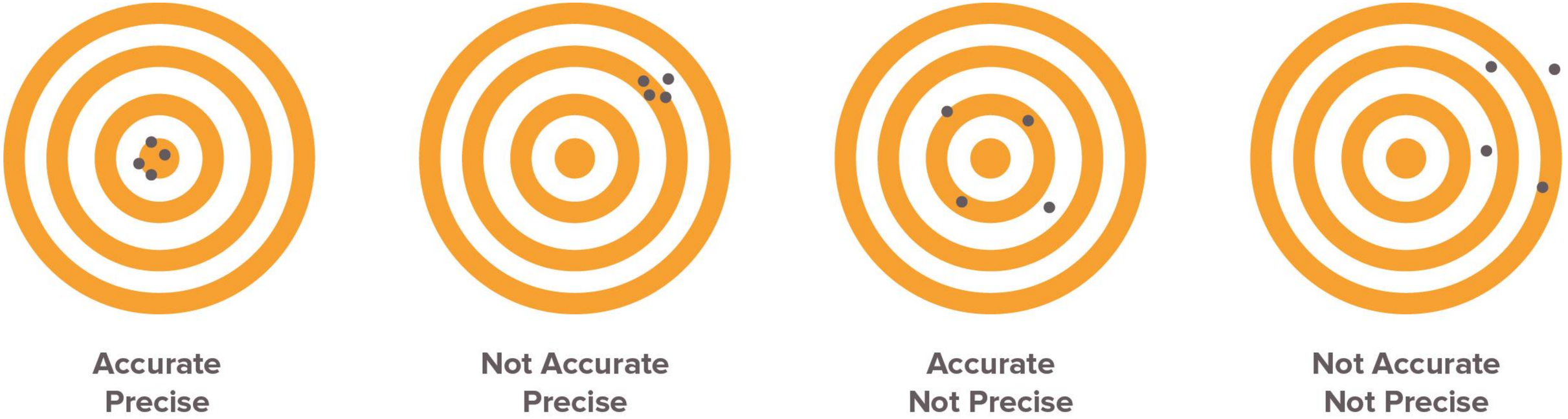
- Traffic grows fast → expensive to add additional memory
- Additional problem: traffic and number of flows increase during attacks → monitoring fails when it is needed the most
- Attackers can craft traffic to target monitoring system

Basic Concepts of Probabilistic Traffic Monitoring

Probabilistic Traffic Monitoring

- Intuition: trade accuracy/precision for efficiency
  - Probabilistic traffic monitoring can reduce overhead
  - Input: network traffic,
    Output: estimate of traffic statistics (e.g., #of packets in a flow)
- Challenges
  - Measure with limited memory/processing
  - Output an “accurate” estimate with high probability
    - Accurate: close to the real value
      (the output of the naïve per-flow-counter algorithm)

Precision vs. Accuracy



- Accurate: Average of measurements is close to the real value (**low bias**)
- Precise: Measurements are close to each other (**low variance**)

https://archive-media.formlabs.com/upload/accuracy-precision-3d-printing.jpg

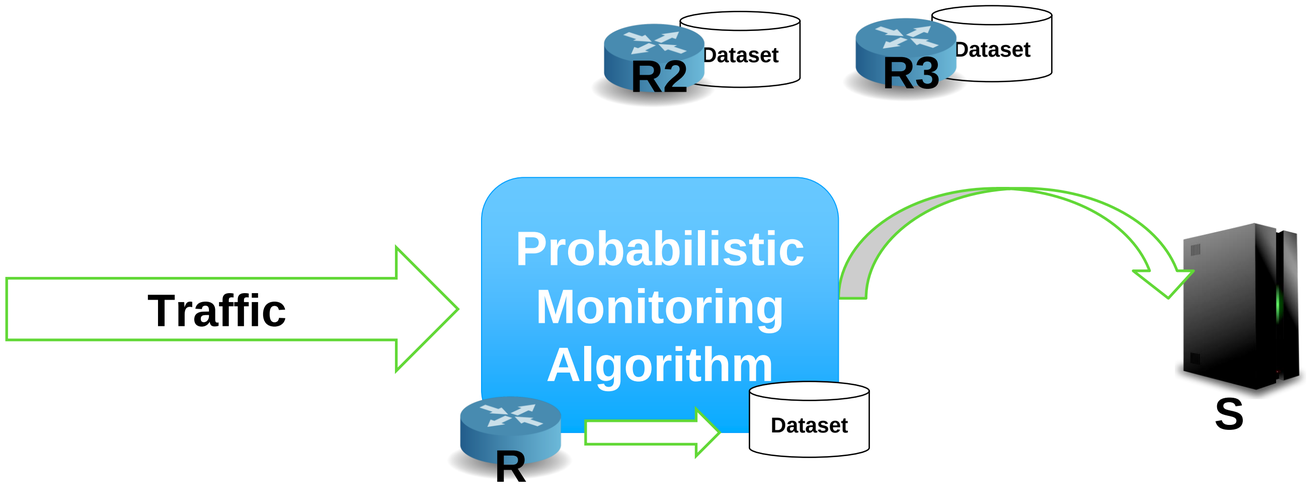
Different Types of Errors

	Actually Positive	**Actually Negative**
Positive Prediction		
Negative Prediction		

Different Types of Errors

	Actually Positive	**Actually Negative**
Positive Prediction	True Positive (TP)	False Positive (FP)
Negative Prediction	False Negative (FN)	True Negative (TN)

Overview of Probabilistic Monitoring



- Router (R) summarizes traffic into a compact dataset
- (Optional) R periodically reports the dataset to a server (S)
- (Optional) S estimates certain statistics based on multiple datasets

Outline

- General-purpose measurement: Cisco's (original) NetFlow
- Identifying **large flows**: elephant detectors
- Finding **duplicates**: Bloom filters
- Estimating the **number of flows**: probabilistic counting
- Discussion and summary

Warmup Exercises

- **Situation 1**: population of known size *n*
- **Task**: algorithm to select *one individual* with *uniform probability* 1/*n*
- **Answer**: take one (pseudo-)random value *v* in [0,1)

  → Select individual *i* if (*i*-1) / *n* ≤ *v* < *i* / *n*

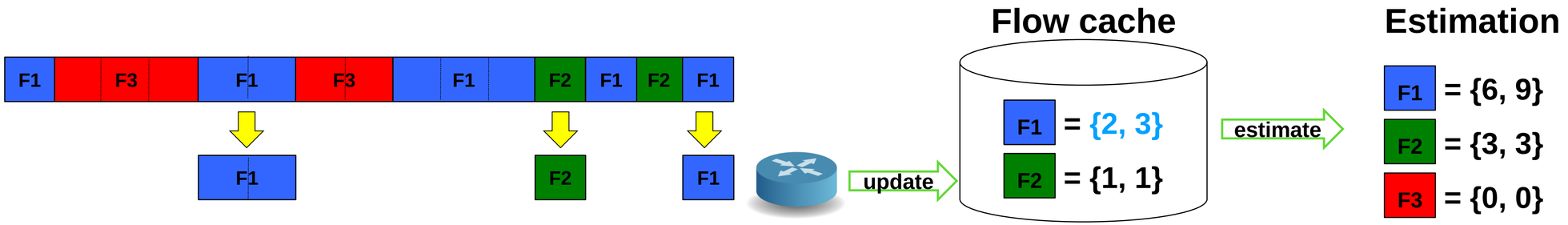
Warmup Exercises

- **Situation 2**:
  - Population of unknown size *n*
  - You see each individual exactly once
  - There is a “cache” where you can temporarily keep exactly one individual
- **Task**: algorithm to select *one individual* with *uniform probability* 1/*n*
- **Answer**: keep the *i*-th individual in the “cache” with probability 1/*i*
  - Select last “cached” individual
  - Fact: after *n* individuals, probability of selection is exactly 1/*n* for each one

General-Purpose Measurements (NetFlow)

Sampled NetFlow

- Standard NetFlow vs. Sampled NetFlow
  - Standard NetFlow samples *every* packet
  - Sampled NetFlow samples every *k*-th packet
- Keeps flow entry {C_{pkt}, C_{byte}} for each flow and updates it with the sampled packet
  - C_{pkt} = # of sampled packets
  - C_{byte} = # of sampled bytes
- Estimates number of packets & bytes of a flow by multiplying recorded values by *k*
  - $\tilde{n}_{pkt} = k * C_{pkt}$; $\tilde{n}_{byte} = k * C_{byte}$
  - Example below when $k = 3$: the estimate of F1 is {6, 9}, while the real value is {5, 8}



Analysis of Sampled NetFlow

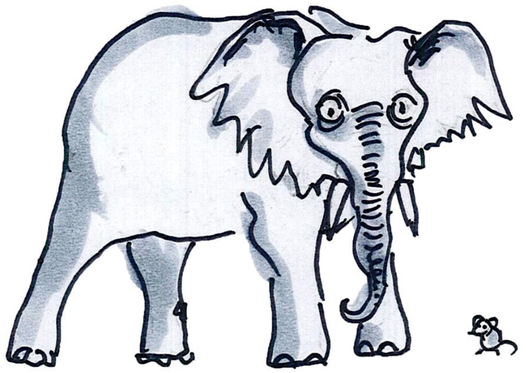
- Advantages
  - Simple to implement
  - Reduced processing time (process only a subset of packets)
- Disadvantages
  - Memory overhead: one entry per flow in worst case
  - Imprecise estimate, especially for short-lived flows
    - Most precise for flows sending many small packets
  - Security issues?

Issues with General-Purpose Measurement

- Observation: general-purpose flow measurement is either imprecise or infeasible
- Solution: focus on specific traffic information
  - **Large flows**
  - Subpopulations (small-volume flows)
  - **Number of flows**
  - Address access patterns
  - Per-source behavior of small flows
  - Flow distribution

Large-Flow (Elephant) Detectors

Focusing on the Elephants, Ignoring the Mice



- What are large flows?
  - Flows that consume more than a given threshold of link capacity during a given measurement interval
  - E.g., flows that take more than 1% of link capacity
- Rise of the Hyper Giants [Arbor Networks 2009]:[1]
  Already in 2009, 30 large providers (“hyper giants”) generated 30% of all Internet traffic
- Less than 1% of flows account for more than 90% of traffic volume (measurements in a university network in 2018)[2]

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.478.6620&rep=rep1&type=pdf
[2] https://arxiv.org/pdf/1809.03486.pdf
http://noticing.co/wp-content/uploads/2015/08/elephant-and-mouse.jpg

Identifying Large Flows

- Possible to efficiently identify large flows because
  #of large flows $\ll$ # of flows in total
  - Without keeping per-flow state on routers
- Accuracy metrics for large flow detection:
  - False positive: wrongly include a small flow in the result
  - False negative: fail to identify a large flow
  - Measurement error: error in traffic volume of the identified large flows
- We will look at three types of large-flow detectors:
  - Sampling-based: NetFlow, sample-and-hold
  - Sketch-based: Multistage filters
  - Eviction-based: EARDet

System 1 (Sampling-Based): Sample and Hold

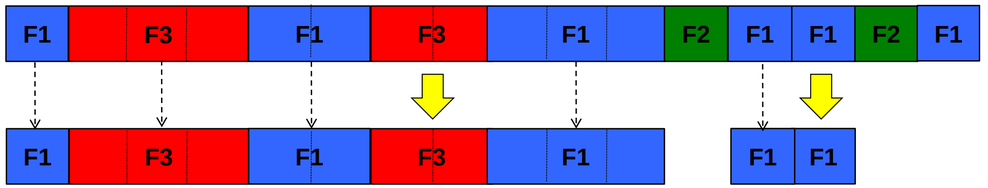
[Estan & Varghese 2002]

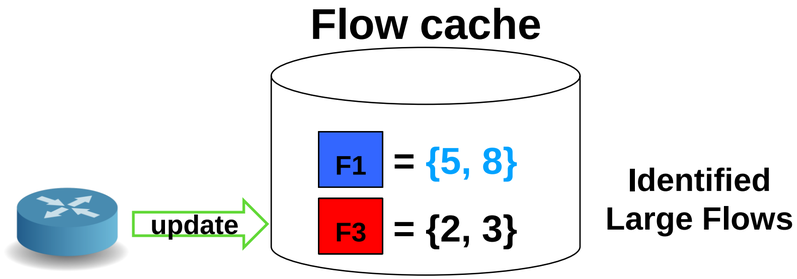
- **Sample**: byte-sampling technique (in contrast to packet sampling in NetFlow) taking packet size into account
  - Samples each byte with probability p
  - Practically, samples a packet of size s with probability p_s,
  - $p_s = 1-(1-p)^s \approx p \times s$ (when p is small)
  - Reduces memory overhead due to the non-uniform sampling biased toward “elephant” flows
- **Hold**: updates a flow entry for all subsequent packets once it is created
  - Requires flow-table lookup for every incoming packet

System 1 (Sampling-Based): Sample and Hold

[Estan & Varghese 2002]

1. For every packet, check if its flow record exists
   - If yes, “hold” ( ) the packet
   - If no, “sample” ( ) with probability p_s
2. Update flow entries $\{C_{pkt}, C_{byte}\}$ associated with “sampled & held” packets
3. Flows in the cache = identified large flows
   - Estimated number of packets & bytes: $\tilde{n}_{pkt} = C_{pkt}$; $\tilde{n}_{byte} = C_{byte}$





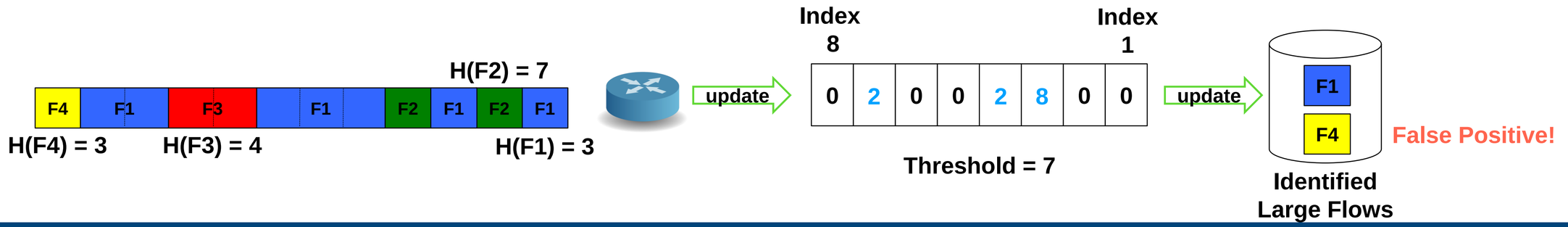
Sample-and-Hold vs. NetFlow

	Sample-and-Hold	**NetFlow**
Sampling technique	Byte sampling	Packet sampling
Estimate of number of packets/bytes in a flow	No over-counting	May over-count or underestimate
Error rate (Given memory overhead M)	$O(M^{-1})$	$O(M^{-1/2})$
Inspect headers of all packets?	Yes	No

System 2 (Sketch-Based): Multistage Filter

[Estan & Varghese 2002]

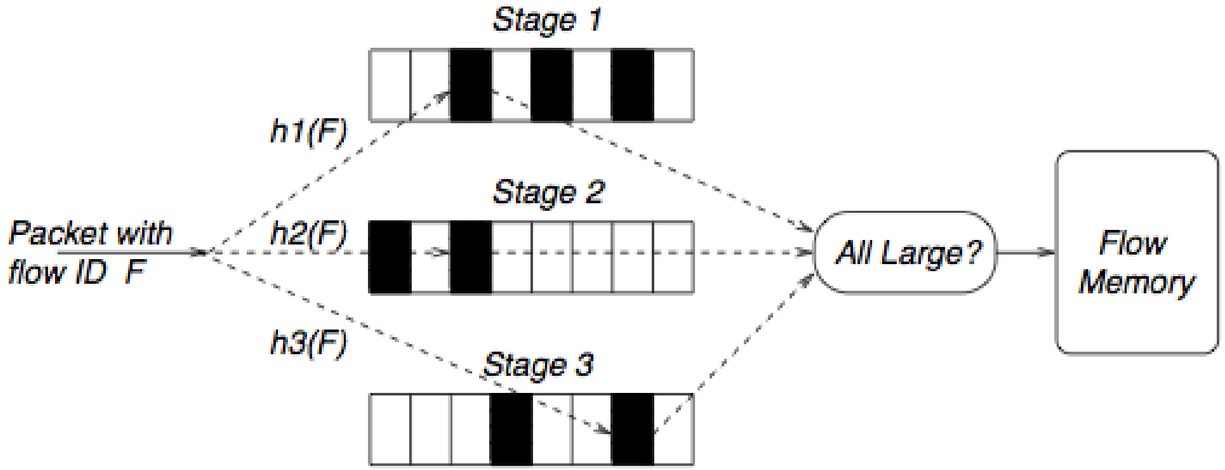
- How a single-stage filter works:
  1. Keep array of *n* counters
  2. Router hashes the flow ID of the incoming packet
     - Output of hash function (H) is used as pseudorandom integer in {1, …, *n*}
  3. Increase *i*-th counter if the hash output is *i*
  4. A flow is considered a large flow if the counter value in the associated counter ≥ threshold



System 2 (Sketch-Based): Multistage Filter

[Estan & Varghese 2002]

- Multiple filters operate in parallel to reduce FPs
  - Each stage uses an *independent* hash function
  - A flow is considered a large flow if the associated counter values at **all** stages ≥ threshold
- Properties: fixed memory resources, no FNs, low FPs
  - FP rate decreases exponentially in the number of stages



System 3 (Eviction-Based): Finding Frequent Items

- Equivalent to the “finding frequent items” problem in database literature
  - Given a stream of items
  - Find those items: frequency > a threshold θ
- Item → packet
- A frequent item → a large flow
- E.g., identify flows that take more than 1% of link capacity

Majority Algorithm

[Fischer and Salzberg 1982, Demaine et al. 2002, Karp et al. 2003]

- An exact, linear-time algorithm to find frequent items in two passes with $1/\theta$ space
  - The first pass identifies all frequent items (no FN!)
  - The second pass eliminates all FPs
- Example question: in an election, how to identify the candidate that won more than a half of the votes (if there is one)?
  - Keep only *one* counter and make two passes
  - Generalized question: how to identify candidates (if exist) winning more than θn votes using only $1/\theta$ - 1 counters?

Majority Algorithm

[Fischer and Salzberg 1982, Demaine et al. 2002, Karp et al. 2003]

- Example question: in an election, how to identify the candidate that won more than a half of the votes (if there is one)?
- Intuition: repeatedly remove pairs of distinct items

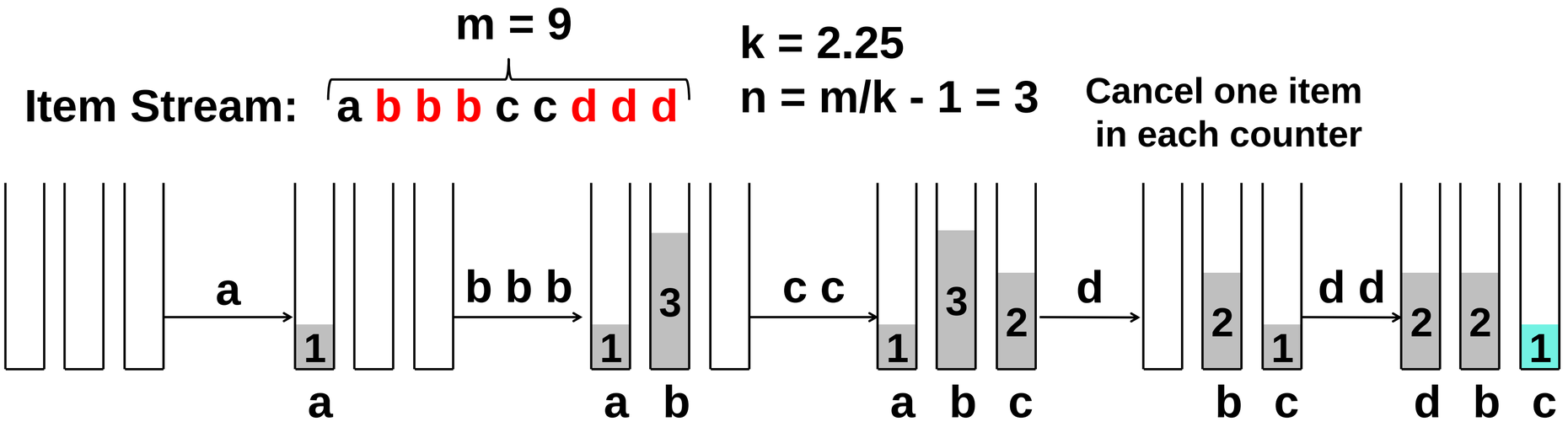
  → If one type occurs more than 50% of the time, some items must remain.

- Algorithm:
  - Keep one item (*kept_item*) and one *counter*
  - First pass: for each *new_item*:
    - If *counter* = 0, then *kept_item* ← *new_item*, *counter*++
    - Else if *new_item* == *kept_item*, then *counter*++
    - Else *counter*--
  - Second pass: test the last *kept_item*

Frequent item finding (MG-algorithm)

[Misra and Gries, 1982]

- **Goal**: Find all items that appear in a stream of *m* items more than *k* times with **no false negatives**
- **Limited space**: *n* = *m*/*k* - 1 counters
- Challenge: Need second pass to eliminate false positives

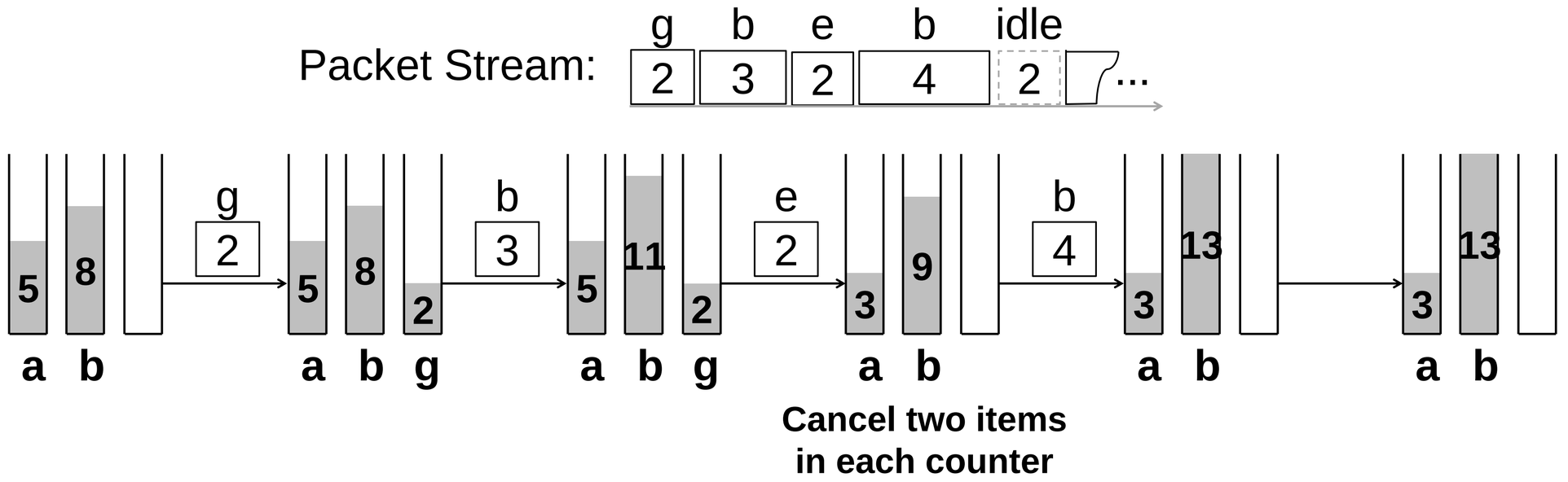


J. Misra and D. Gries. Finding Repeated Elements. Science of Computer Programming, 2(2):143–152, 1982.

EARDet Algorithm

[Wu, Hsiao, Hu, 2014]

- Modify MG-Algorithm:
  - Number of items → packet size:
    Increase counter by size of packets instead of number of items

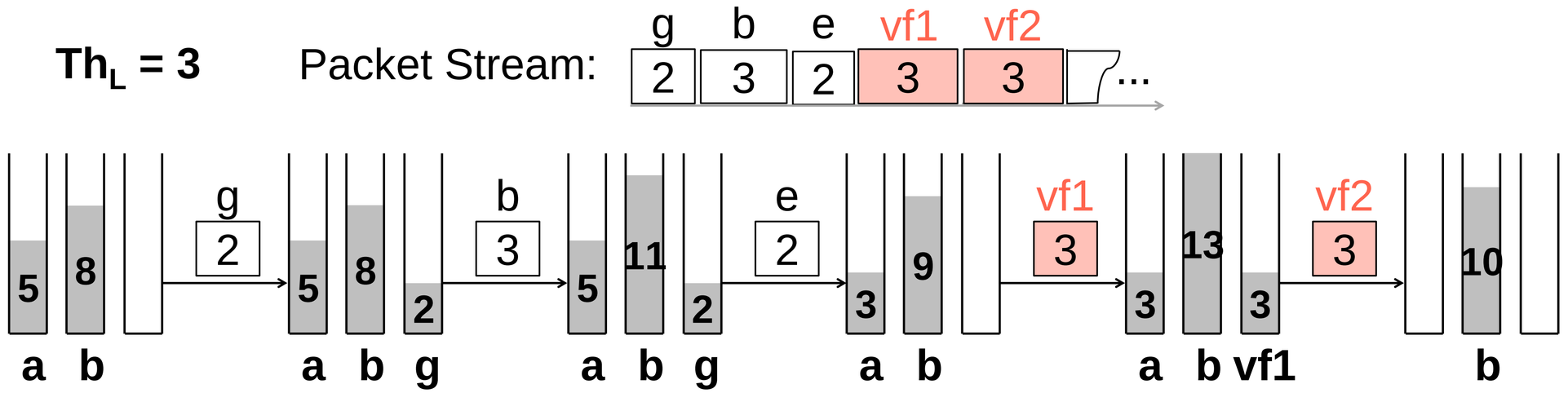


https://www.csie.ntu.edu.tw/~hchsiao/pub/2014_ACM_IMC.pdf

EARDet Algorithm

[Wu, Hsiao, Hu, 2014]

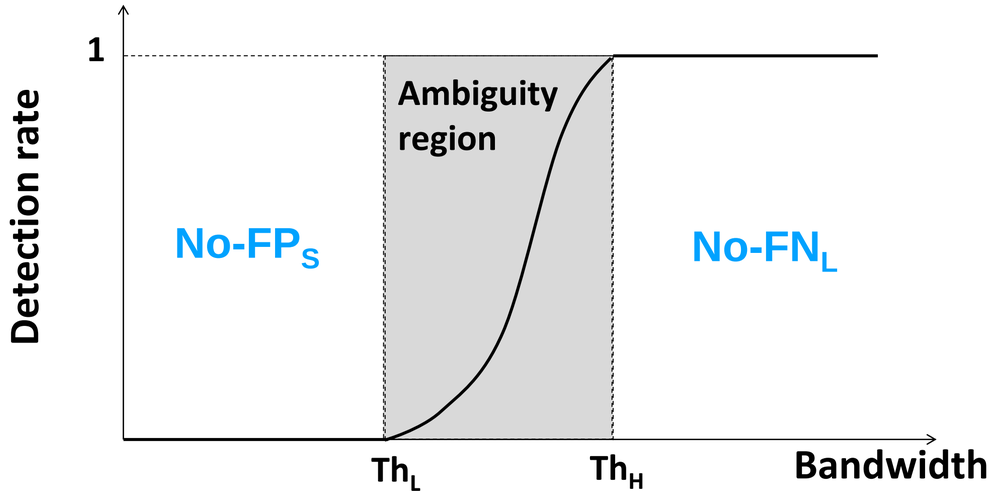
- Modify MG-Algorithm:
  - Virtual traffic:
    - Use “virtual flows” for each idle period
    - Each virtual flow is at most equal to a *low-bandwidth threshold* (Th_L)



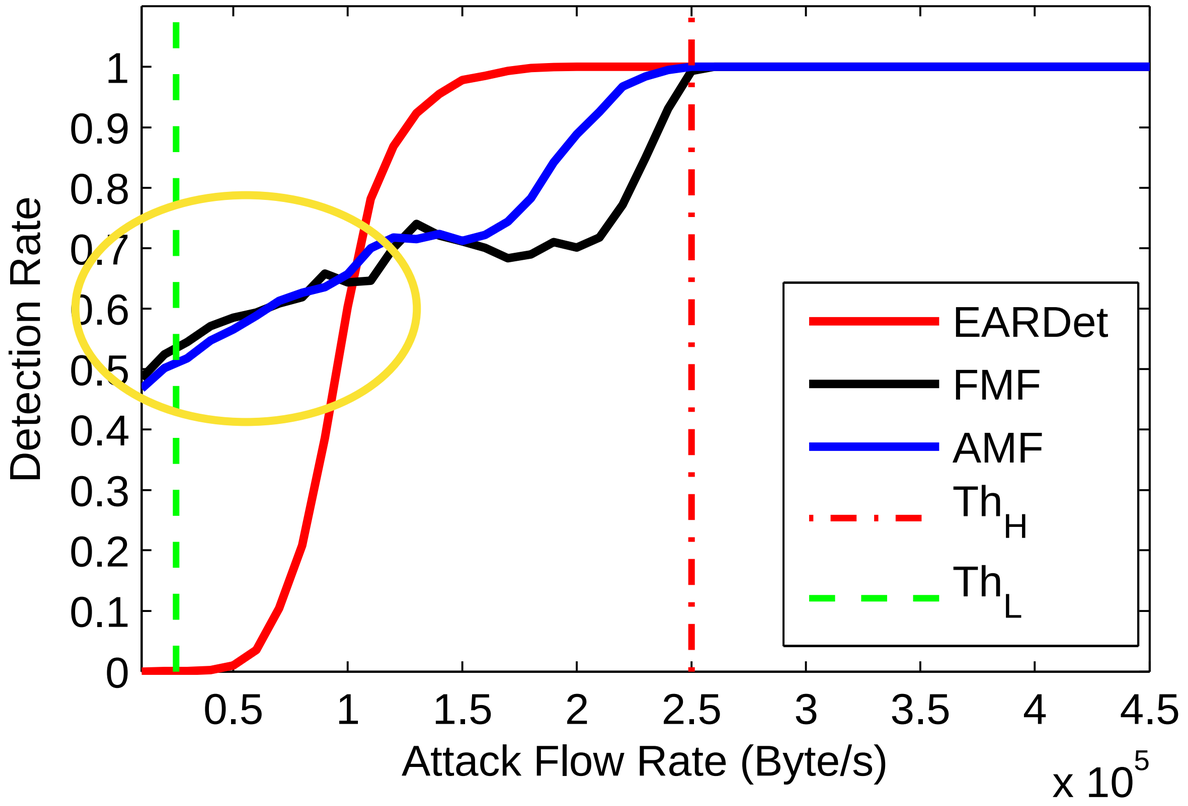
https://www.csie.ntu.edu.tw/~hchsiao/pub/2014_ACM_IMC.pdf

EARDet Main Properties

- No false negative for large flows (No-FN_L)
- No false positive for small flows (No-FP_S)
- Deterministic: keep performance regardless of input traffic or attack pattern
- Relatively small storage cost, but many counters are needed
- Disadvantage: per-packet counter update is an expensive operation



EARDet Evaluation



- Detection rate under flooding DoS attack
- FMF: fixed-window multistage filter
- AMF: arbitrary-window multistage filter (using leaky bucket counters)
- EARDet:
  - No FP for small flow
  - No FN for large flow

Finding Duplicates: Bloom Filters

Finding Duplicates: Bloom Filter

- Problem: identify if an element is a duplicate
- Challenge: cannot store all previous elements
- Solution: Bloom filter provides probabilistic data structure for set membership testing

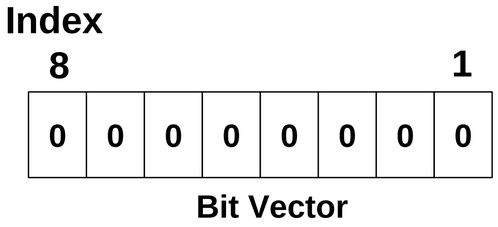
Bloom Filter

- Setup: bit vector V with m bits, $V = 0$
- Insert element e:
  - Compute k hash functions $h_i = H_i(e)$, where $0 < h_i \leq m$
  - Set all bits $V[h_i] = 1$
- Test if element e' has already been seen:
  - Recompute k hash functions, check all $V[h_i]$
  - If all bits are 1 → "seen before", otherwise "new"

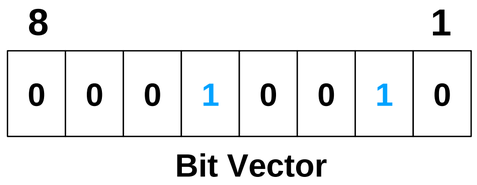
Advantages of Bloom filters: no FN, constant time insertion and test

Bloom Filter

- Initialization:



- Insertion of e: $H_1(e) = 5$, $H_2(e) = 2$



- Test of $e' = e$: $H_1(e') = 5$, $H_2(e') = 2$ ⇒ seen before
- Test of $e'' \neq e$: $H_1(e'') = 5$, $H_2(e'') = 1$ ⇒ new
- Test of $e''' \neq e$: $H_1(e''') = 2$, $H_2(e''') = 5$ ⇒ seen before ?!

Bloom Filter Parameters

- *m*: # of bits in BF
- *k*: # of hash functions
- *n*: # of inserted elements
- P(FP) = $\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$
- *k* to minimize P(FP): *k* = $\frac{m}{n} \ln 2 \approx 0.7\frac{m}{n},$
- For that choice of *k*, resulting P(FP) = *p* = $2^{-k} \approx 0.6185^{m/n}.$
- Given optimal *k*, choice of optimal $m = -\frac{n \ln p}{(\ln 2)^2}.$ → O(*n*) space

https://en.wikipedia.org/wiki/Bloom_filter

Bloom Filter Size Examples

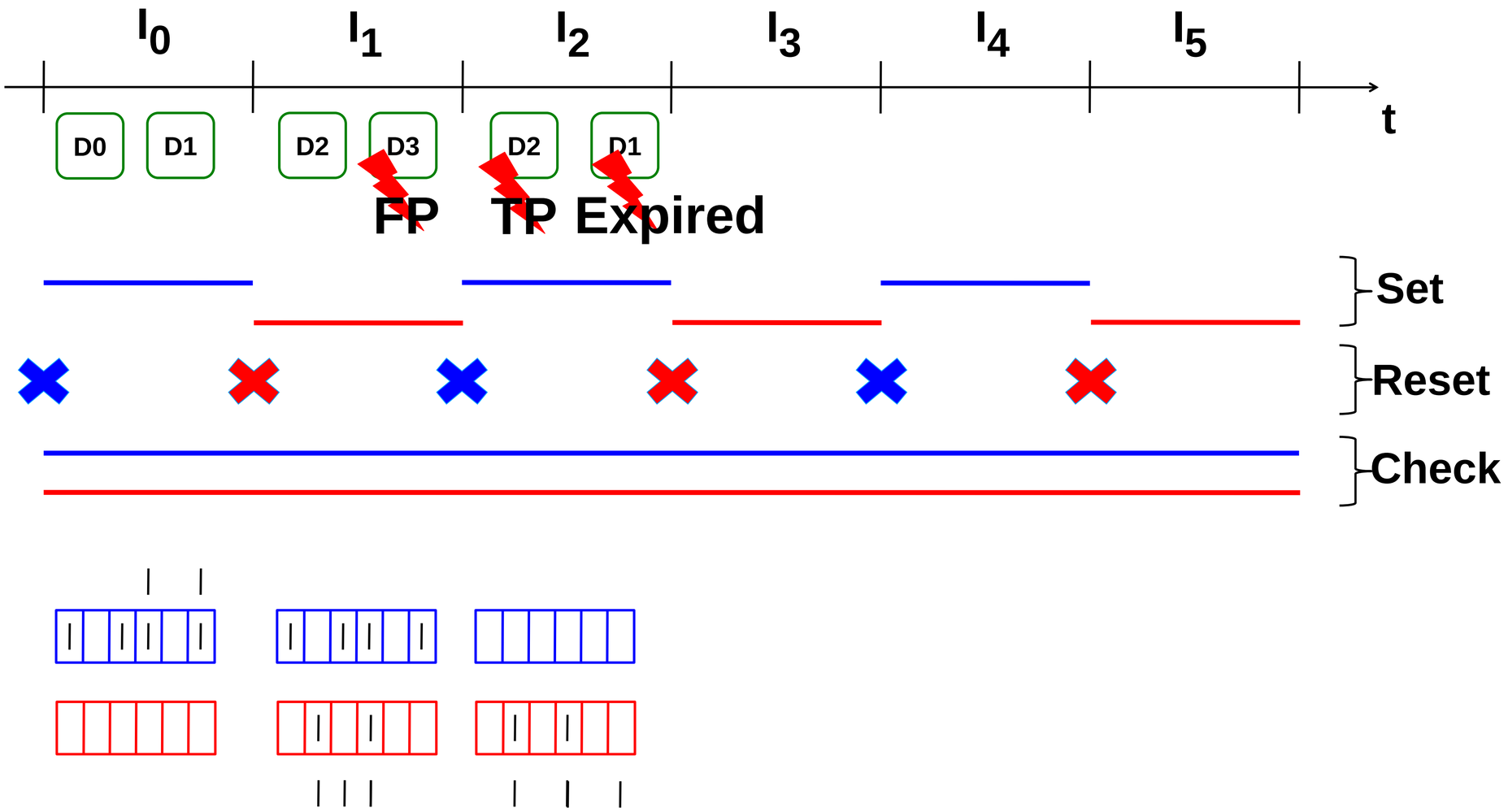
- $n = 10^6$
- $p = 1\%$
- $m = 9.6 \times 10^6$ bits ~ **1.2 MB**
- $k \sim 6.6 \rightarrow 7$
- With $k = 7$, $p \sim 1\%$

- $p = 0.1\%$
- $m = 14.4 \times 10^6$ bits ~ **1.8 MB**
- $k = 10$

How to Find Duplicates in Practice?

- Problem: Bloom filter “fills up”
- Simple approach: reset Bloom filter periodically
- Problems with simple approach?
  - Some false detections: fundamental problem with Bloom filters
  - Fail to provide “no FN” guarantee: some duplicates are not detected!
    - If duplicate arrives after reset, then it is not detected as duplicate

How to Find Duplicates in Practice?



How to Find Duplicates in Practice?

- Define maximum packet propagation time δ
- Require time synchronization, max sync error σ
- Split time up into time periods of duration $\Delta > \delta + 2\sigma$
- Keep two Bloom filters B_0 and B_1
- B_0 is reset at the beginning of time period i and filled during period i, where $i \bmod 2 = 0$
  - Analogous for B_1
- Each packet includes timestamp t_s it is sent, in time period T_s
- On packet arrival at router at t_c, in time period T_c
  - Check if $t_s - \sigma \leq t_c \leq t_s + \delta + \sigma$; otherwise drop packet
  - Router checks both B_0 and B_1 to detect duplicate
  - If packet is not a duplicate, insert it into B_i where $i = T_c \bmod 2$

Details of Setting Time Duration

- t_s^i: timestamp of packet i
- δ^i: propagation delay of packet i
- σ: max sync error

$\Rightarrow |(t_s^i + \delta^i) - t_c| \leq \sigma$

$\Rightarrow t_s^i + \delta^i - \sigma \leq t_c \leq t_s^i + \delta^i + \sigma$

$\Rightarrow t_s - \sigma \leq t_c \leq t_s + \delta + \sigma$ (because $0 \leq \delta^i \leq \delta$)

$\Rightarrow$ For any claimed t_s, packet is valid at router for a duration of at most $\delta + 2\sigma$

$\Rightarrow$ If duration of time period $\Delta > \delta + 2\sigma$, packet is valid for at most two intervals (current and the next one)

Quick Summary of Bloom Filters

- Basic operations
  - Insert an element
  - Membership test
- Properties
  - No FN, low FP
  - Time efficient: O(1) insertion and membership checking
  - Space efficient: constant bits per element given FP rate
    - But still O(*n*) memory overhead
- Challenges of duplicate detection in practice

Estimating the Number of Flows

First Ideas to Estimate the Number of Flows

- Record all distinct flow IDs
  - Often infeasible to keep one record for every distinct flow
  - O(N) memory overhead, where N is number of flows
- Increase a counter when the incoming packet belongs to a “new” flow
  - Use Bloom filter to test whether a flow has been recorded or is new
  - Bloom filter enables O(1) time membership checking but still O(N) memory overhead

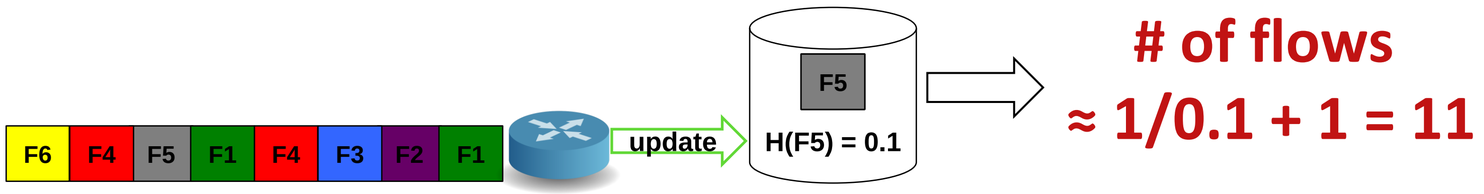
Estimating Number of Flows: Probabilistic Counting

- Example of a simple probabilistic counting:
  - Hash flow ID to generate a value in [0,1)
  - Keep the flow ID associated with the smallest hash value
  - Expectation value of smallest value is 1 / (number of flows + 1)
    - Assumption: Hash values are uniformly distributed in interval [0,1)

→ Estimate the number of flows by the smallest value v seen so far: $\tilde{n} \approx 1/v - 1$

Example:

H(F1) = 0.6
H(F2) = 0.85
H(F3) = 0.4
H(F4) = 0.92
H(F5) = 0.1
H(F6) = 0.26



Estimating Number of Flows: Probabilistic Counting

- Problems:
  - Minimum has very large variance → not very robust
  - An attacker controlling only 1 input can bias the estimation
- Proposal by Bar-Yossef et al. (2002):
  - Keep track of the k smallest hash values
  - Expectation value of k-th smallest value is $k/(n+1)$, variance is smaller
  - Estimate the number of flows by the k-th smallest value v_k that has been seen so far: $\tilde{n} \approx k / v_k - 1$
- Can additionally use sampling in order to reduce overhead due to hashing

Discussion and Summary

Tradeoffs of Probabilistic Monitoring

- Accuracy
  - False positive, false negative, error in estimates
- Efficiency
  - Computation: per packet processing or sampling
  - Storage: per flow or not
- Security assumptions and considerations
  - E.g., unpredictable sampling, private hash functions, etc.

Traffic Monitoring vs. Intrusion Detection

- Both can detect malicious activities such as DoS attacks & port scans at selected network vantage points
- Intrusion detection
  - Typically deployed at network edges
  - Destination-based diagnosis
  - Can analyze detailed payload data as well
- Traffic monitoring
  - Can be deployed at high-speed backbone routers
  - Diagnoses network-wide anomalies
  - Analyzes packet headers only

Challenges Remain…

- Monitoring schemes should be secure against attacks
- Traffic measurement increases the risk of DDoS attacks
- Require source authentication to prevent IP spoofing
- How to prevent or detect an attacker who crafts certain inputs that bias the final estimate?
- What if routers are compromised? Can legitimate hosts be framed?

Summary

- Efficient & accurate traffic monitoring is important for security & management applications
- Probabilistic-monitoring algorithms
  - Provide a tradeoff between accuracy & efficiency
  - Process a large amount of traffic at line speed with limited memory
  - Depending on algorithm, they can introduce false positives, false negatives, inaccurate measurement on quantities