# Limitations of Packet Measurement

Collect and process less information:

- Only collect packet headers, not payload
- Ignore single packets (aggregate)
- Ignore some packets (sampling)

Make collection and processing faster:

- Move to kernel space
- Distributed collection & processing
- Dedicated hardware

# Sampling

- For packet-based measurements:
  - Systematic sampling (every *n*th) (bad!)
  - Random sampling: n-to-N sampling
  - Dynamic sampling: e.g. sample big flows more often
- For flow-based measurements, too!
  - Packet sampling
  - Flow sampling

SURFnet: netflow with 1:100 packet sampling

# Estimating Distributions From Sample Statistics

- How to reverse the effects of sampling?
- Example:
  1. Create flows based on packets sampled with ratio 1:10
  2. Observe flows with byte size $b'_1, b'_2, b'_3, \ldots$
  3. What were the original byte sizes $b_1, b_2, b_3, \ldots$ ?
  4. Estimation: $b_i = 10 \cdot b'_1$
  5. Right?

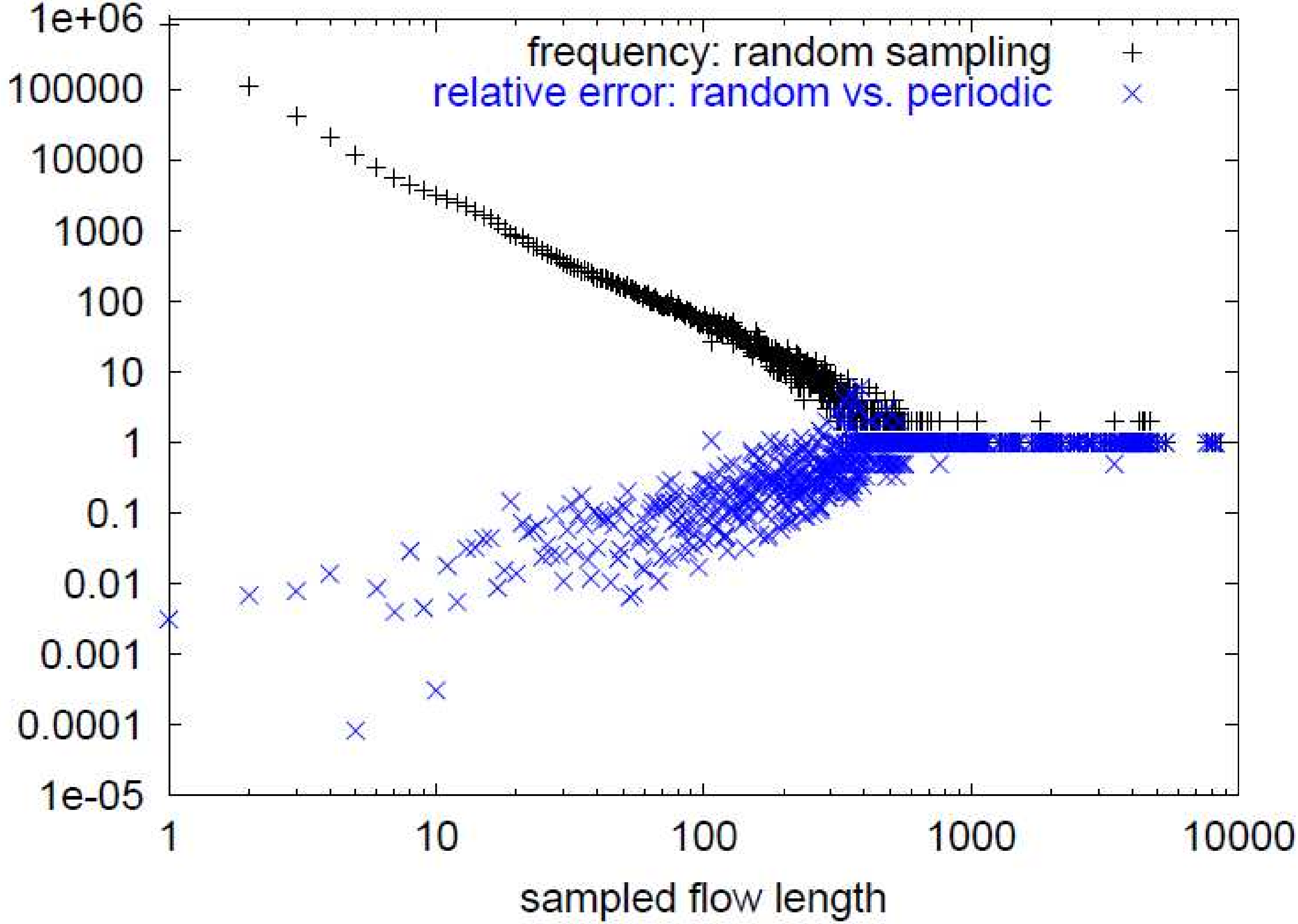
# Sampling Error

- How does sampling influence the results? (sampling error)
- Results are only statistical estimations with a certain confidence
- Depends on many factors:
  - Nature of data (packets or flows)
  - Sampling method (periodic, random,…)
  - Sampling rate
  - Characteristics of the sampled process (distribution,…)

# Sampled Flows

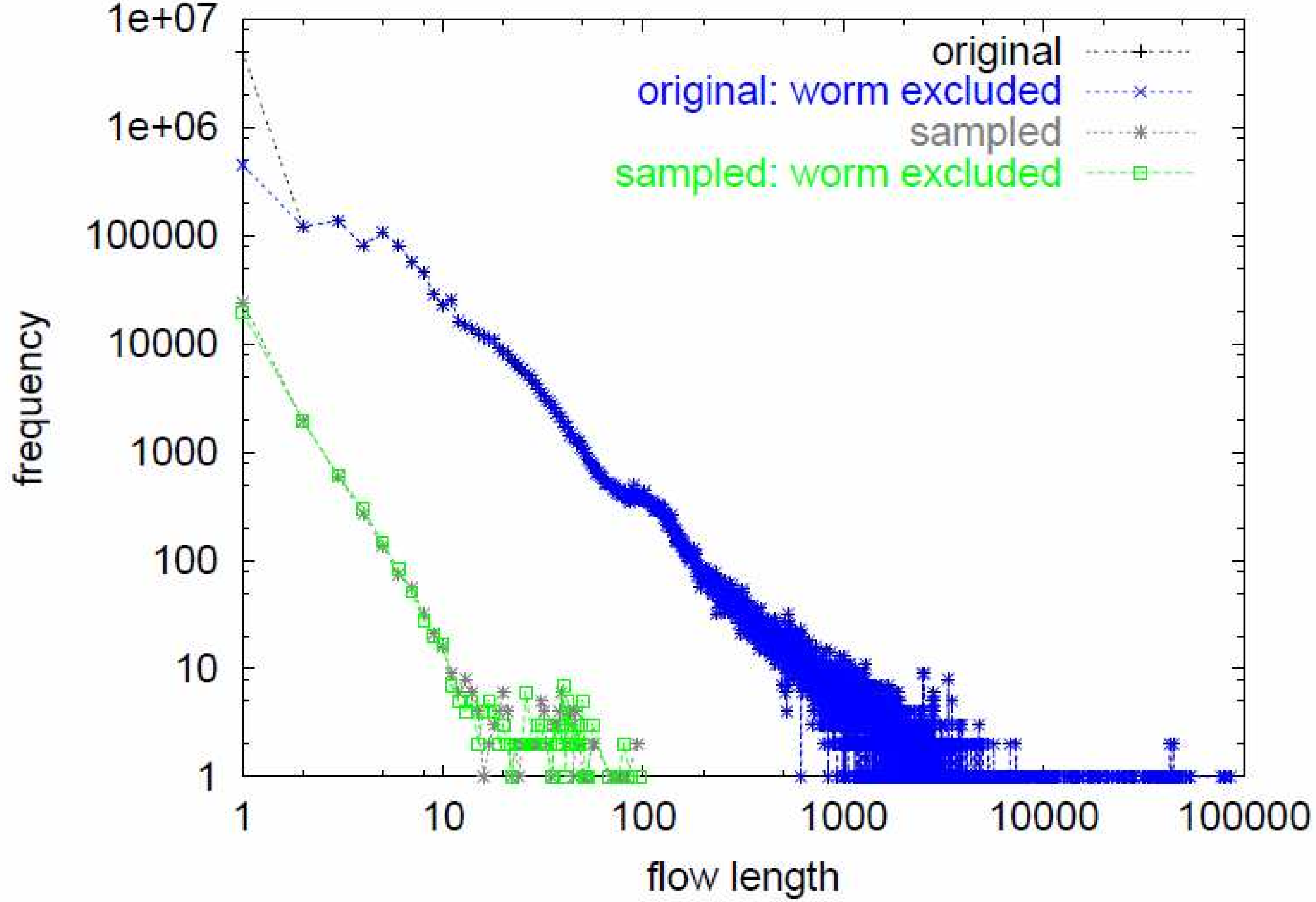
- A flow could be splitted into two:
  - Original packet sequence: a b c d e
  - Assume: Time between b and d larger than timeout used by flow collector to determine end of flow
  - If c is not sampled: two flows ab and de!
- High sampling ratio:
  - Small flows have a significant probability not to be sampled at all
  - Result is biased towards large flows
- Periodic sampling:
  - Fails if periodicities in the data
  - Example: back-to-back communications

# Example: Flows with Periodic Sampling




(from: Estimating Flow Distributions from Sampled Flow Statistics, 2003)

# Example: Sampling Small Flows




(from: Estimating Flow Distributions from Sampled Flow Statistics, 2003)

# Limitations of packet measurement

Collect and process less information:

- Only collect packet headers, not payload
- Ignore single packets (aggregate)
- Ignore some packets (sampling)

Make collection and processing faster:

- Move to kernel space
- Distributed collection & processing
- Dedicated hardware

# Dedicated Hardware

- Exist for packet measurements and flow measurements
- Jobs that could be handled by hardware:
  - Protocol analysis (analysis of IP packet payload)
  - Filtering (as a pre-processing step)
  - Analysis (run analysis algorithms over the packet)
  - For flows: building flow records

# How Flow Exporters Work 1

- When packet arrives:
  1. Calculate hash value for packet
  2. Lookup in hash table whether flow exists
     - Yes: update flow information (number of bytes, packets,…)
     - No: create new flow entry
- Entry from table is removed and flow record is exported if
  - Inactivity Timeout: no new packets for a flow since x seconds
  - Activity Timeout: flow is active but older than y seconds

    → Flow record ≠ Flow

# How Flow Exporters Work 2

- Usually more complex:
  - Out of memory: forced flow export
  - Don’t export single flow records: wait until several flows records are ready for export
  - Cache levels
- Can be implemented in software or hardware
- Loss of data:
  - Packet arrival rate too high
  - Flow export rate too high

# Example: Hardware Flow Exporter

From: www.invea-tech.com

Hardware-accelerated flow exporter FlowMon:

- *„The accelerated model can capture 6 million packets/s providing full 4 x 1 Gbps throughput under all conditions."*

# Storing and Processing Traffic Measurements

# Storing and Processing Traffic Measurements

- UT, 2007:
  - Average bandwidth: 652 Mb/s
  - Maximum bandwidth: 1010 Mb/s
  - Volume (packets): 21.6 TB in 2 days
  - Flows (no sampling): 983 Million flows in 2 day
- Surfnet, 2007:
  - Average bandwidth: 7730 Mb/s
  - Maximum bandwidth: 10500 Mb/s
  - Volume (packets): 162.3 TB in 2 days
  - Flows (1:100 sampling): 523.7 Million flows in 2 days

# Online vs. Offline Analysis

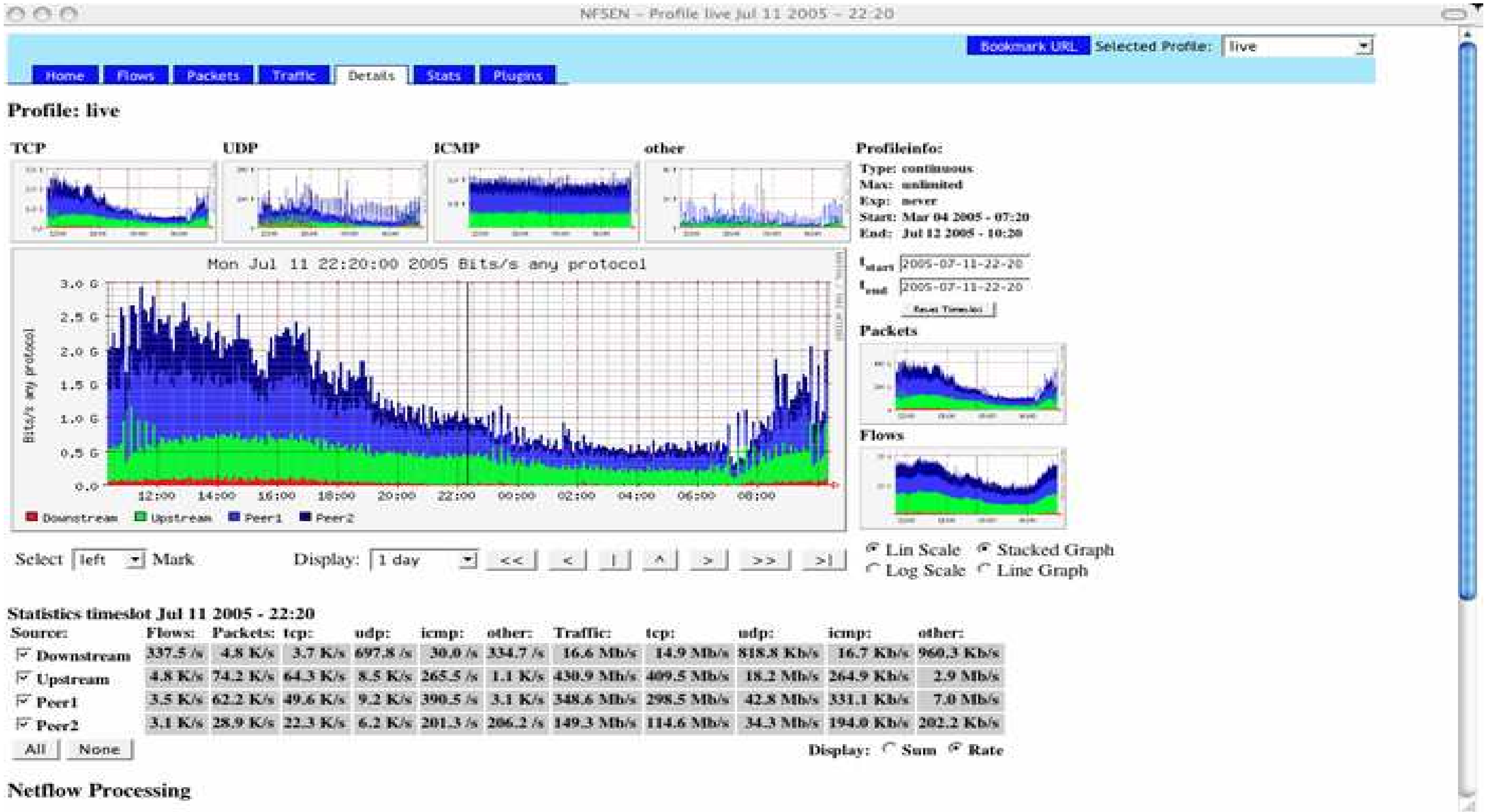
- Online: process data continuously, faster than arrival rate.
  - Often, only a portion of the data (last *n* seconds) or no data at all is stored
  - Simple example: protocol usage statistics

- Offline: store data and analyze it afterwards
  - Needed if restriction on time (analysis algorithm too complex) or space (large amounts of data to be analyzed)

# Nfsen

- Combination of offline and online approach
- Graphical web-based front end for nfdump netflow tools
- Flow data exported by router(s) is collected by capture deamon(s) and stored in round robin database(s) in pieces of several minutes
- Nfdump/nfsen allow to browse, search, filter, etc. (syntax similar to tcpdump)
- Profiles supported
- Extendable by plugins

# Nfsen: Screenshot




| Source: | Flows: | Packets: | tcp: | udp: | icmp: | other: | Traffic: | tcp: | udp: | icmp: | other: |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Downstream | 337.5 /s | 4.8 K/s | 3.7 K/s | 697.8 /s | 30.0 /s | 334.7 /s | 16.6 Mb/s | 14.9 Mb/s | 818.8 Kb/s | 16.7 Kb/s | 960.3 Kb/s |
| Upstream | 4.8 K/s | 74.2 K/s | 64.3 K/s | 8.5 K/s | 265.5 /s | 1.1 K/s | 430.9 Mb/s | 409.5 Mb/s | 18.2 Mb/s | 264.9 Kb/s | 2.9 Mb/s |
| Peer1 | 3.5 K/s | 62.2 K/s | 49.6 K/s | 9.2 K/s | 390.5 /s | 3.1 K/s | 348.6 Mb/s | 298.5 Mb/s | 42.8 Mb/s | 331.1 Kb/s | 7.0 Mb/s |
| Peer2 | 3.1 K/s | 28.9 K/s | 22.3 K/s | 6.2 K/s | 201.3 /s | 206.2 /s | 149.3 Mb/s | 114.6 Mb/s | 34.3 Mb/s | 194.0 Kb/s | 202.2 Kb/s |

(from: nfsen.sourceforge.net)

# Nfsen: Plugins

# Relational Databases

- Advantages:
  - “Standard” software
  - Performant server-side processing of queries (stored procedures)
  - Use existing interfaces to programming languages
- Example of SQL query:

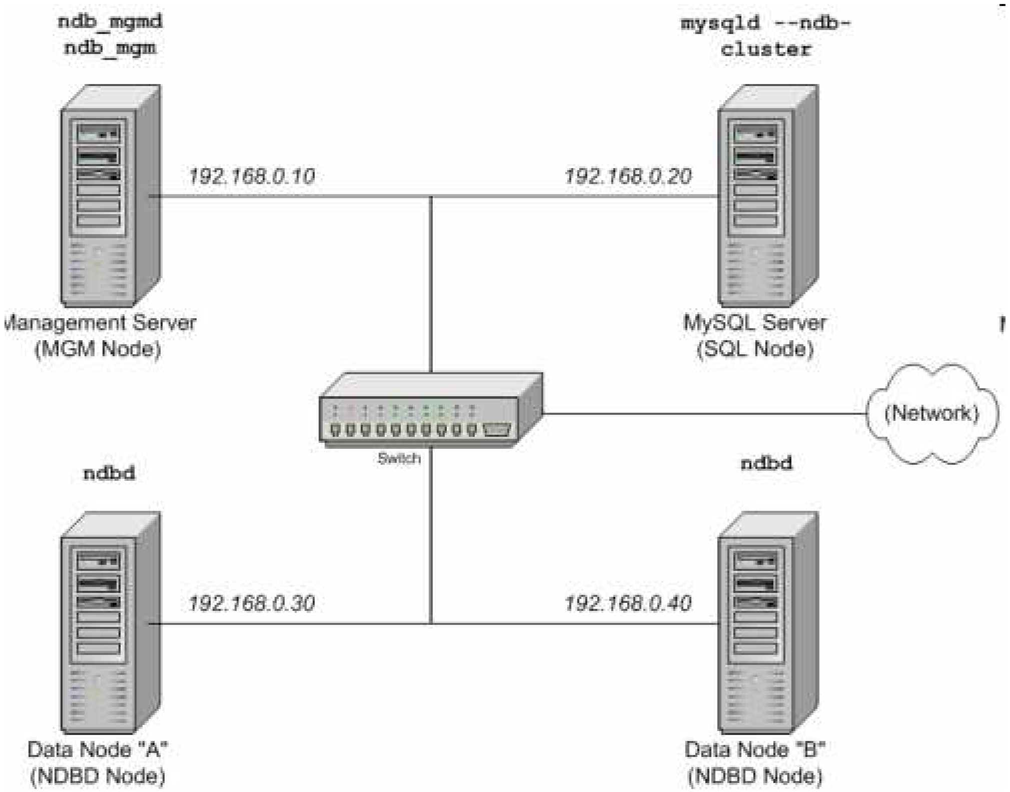
```
SELECT ipv4_dst,count(*) c FROM data
 WHERE port_dst=80 AND protocol=6
 GROUP BY ipv4_dst
 ORDER BY c DESC
```

# Relational Databases: Example

- Netflow data of UT measurement (2 days) stored in MySQL database
- MyISAM engine (transactions, constraints, etc. not needed)
- Table:
  - Columns: start/end time, IP src/dst, port src/dst, protocol,…
  - Data size: 49.6 GB ~ 53 Bytes/row
  - In total 10 indexes: start time, src, dst,…
  - Total index size: 86.9 GB

# Distributed Relational Databases

- MySQL cluster




(from: MySQL reference manual)

# Stream databases

- Continuous sequence of data instead of tables
- Queries operate on streams and return
  - a table by applying a sliding window to the input

  or
  - again a stream.
- Example: GSQL query in Gigascope

```
SELECT tb, srcIP, sum(len) FROM IPv4
  WHERE protocol=6
  GROUP BY time/60 as tb, srcIP
  HAVING count(*)>5
```

- Speed: several Gbs

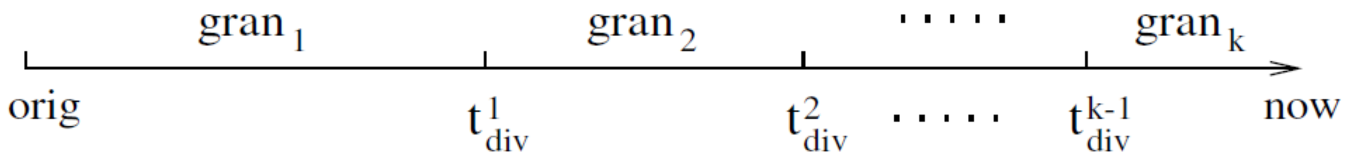
# Temporal Aggregation

- Temporal aggregation is a basic operation in data processing
- Example:
  A plot showing the number of transfered bytes for each month of the year
- Expensive operation if all data (every packet, every flow,…) is stored
- Simple solution: calculate aggregates (sums,…) online and only store the results
- But: which granularity to use?
  - Fine: too much data
  - Coarse: usefulness limited

# Temporal Aggregation Using Multiple Granularities

- From: *Temporal Aggregation over Data Streams using Multiple Granularities*, Zhang et al., 2002.
- Idea: Recent data more interesting than old data

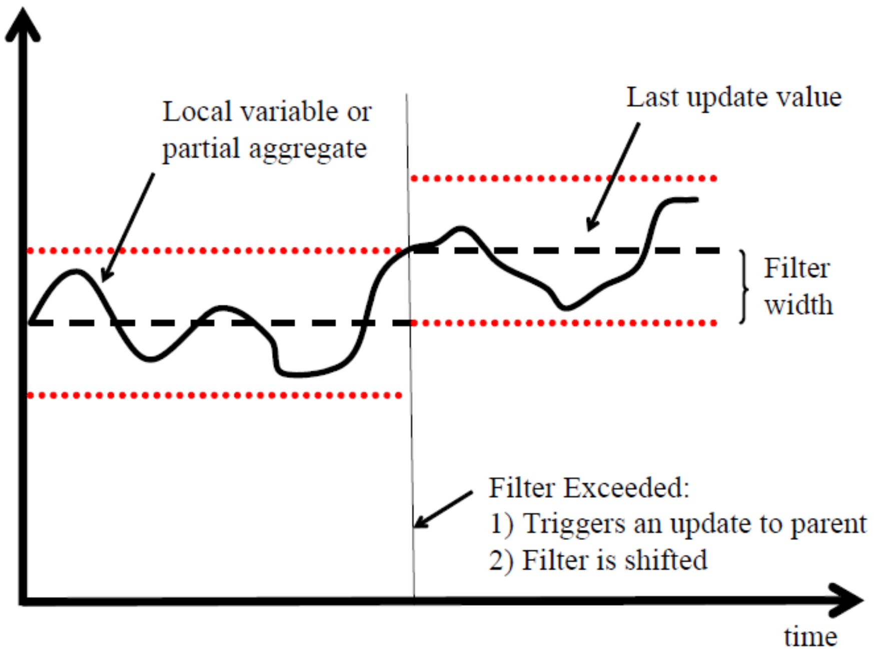
  → use different granularities for old and recent data




- Maintain different indexes for each segment, integrated as a unified index

# Adaptive Aggregation

- No fixed aggregation granularity
- A new measurement record is created if value aggregated so far differs from last measurement record by a given Δ
- Example:




(from: Adaptive Distributed Monitoring with Accuracy Objectives, 2006)

# Hadoop, Google's MapReduce

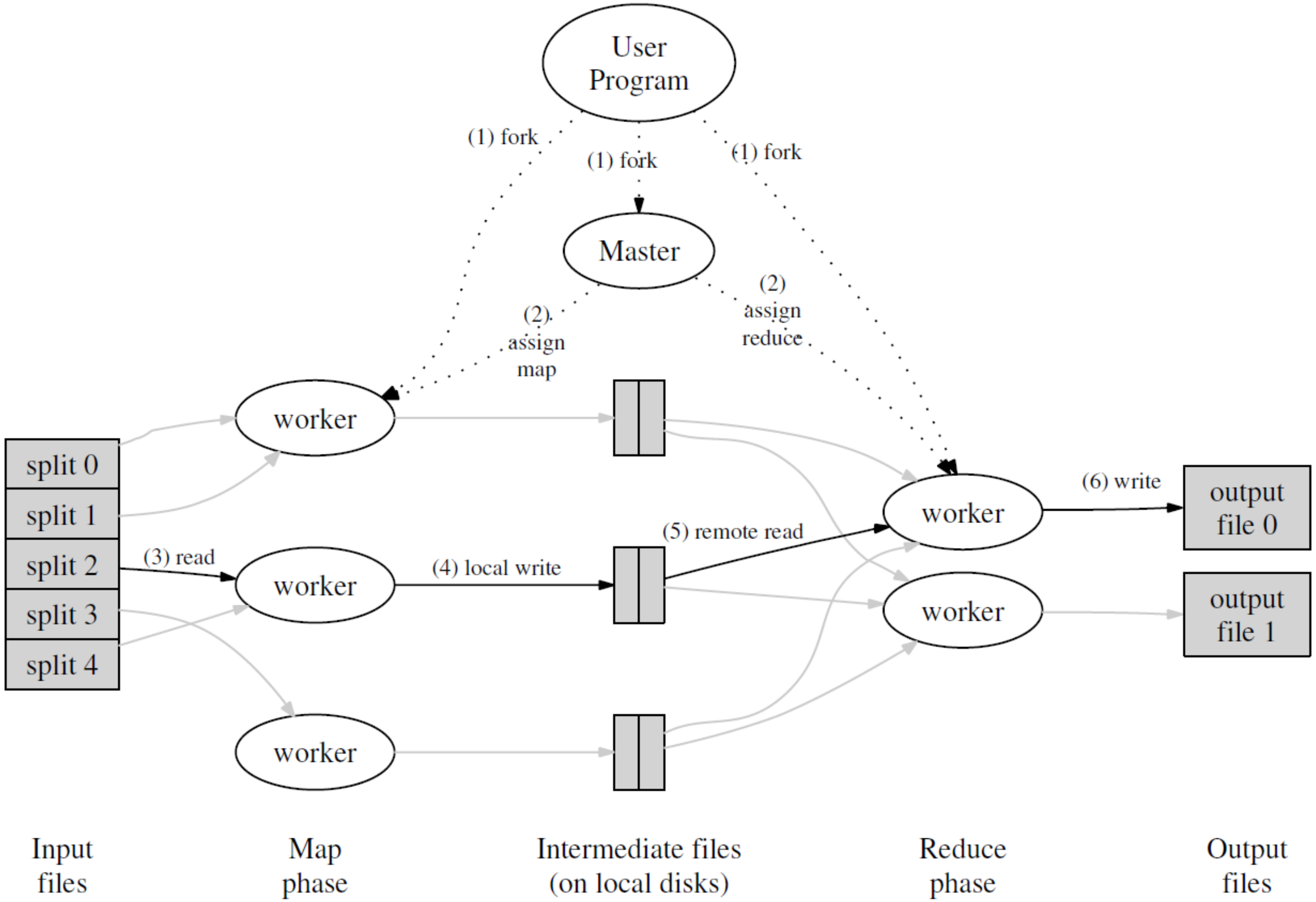
- Designed to run on hundred or thousands of nodes
- Execution:

1. Data is split into pieces and distributed over nodes ("splits")
2. Worker processes read data from splits and apply the "map" function. Result: (key,value) pairs.
3. Worker processes sort the intermediate data according to the key and apply the "reduce" function to each set of values with same key. Result: output data.
4. (Do another MapReduce call)

# MapReduce: Example

- Data: flow data
- Goal: count number of flows per source IP address
- Map: Emit a pair (key=source IP,value=1) for each flow record in a split
- Reduce: For all pairs with same source IP, sum up the values.
- Result data: list of pairs (source IP,total count)

# Hadoop, Google's MapReduce: Overview




(from: MapReduce: Simplified Data Processing on Large Clusters, OSDI'04)

# Many other approaches

- P2P-based distributed analysis
- Flow record query language (Jacobs University Bremen)
- …

# Network Tomography and Geolocation

# Network Tomography

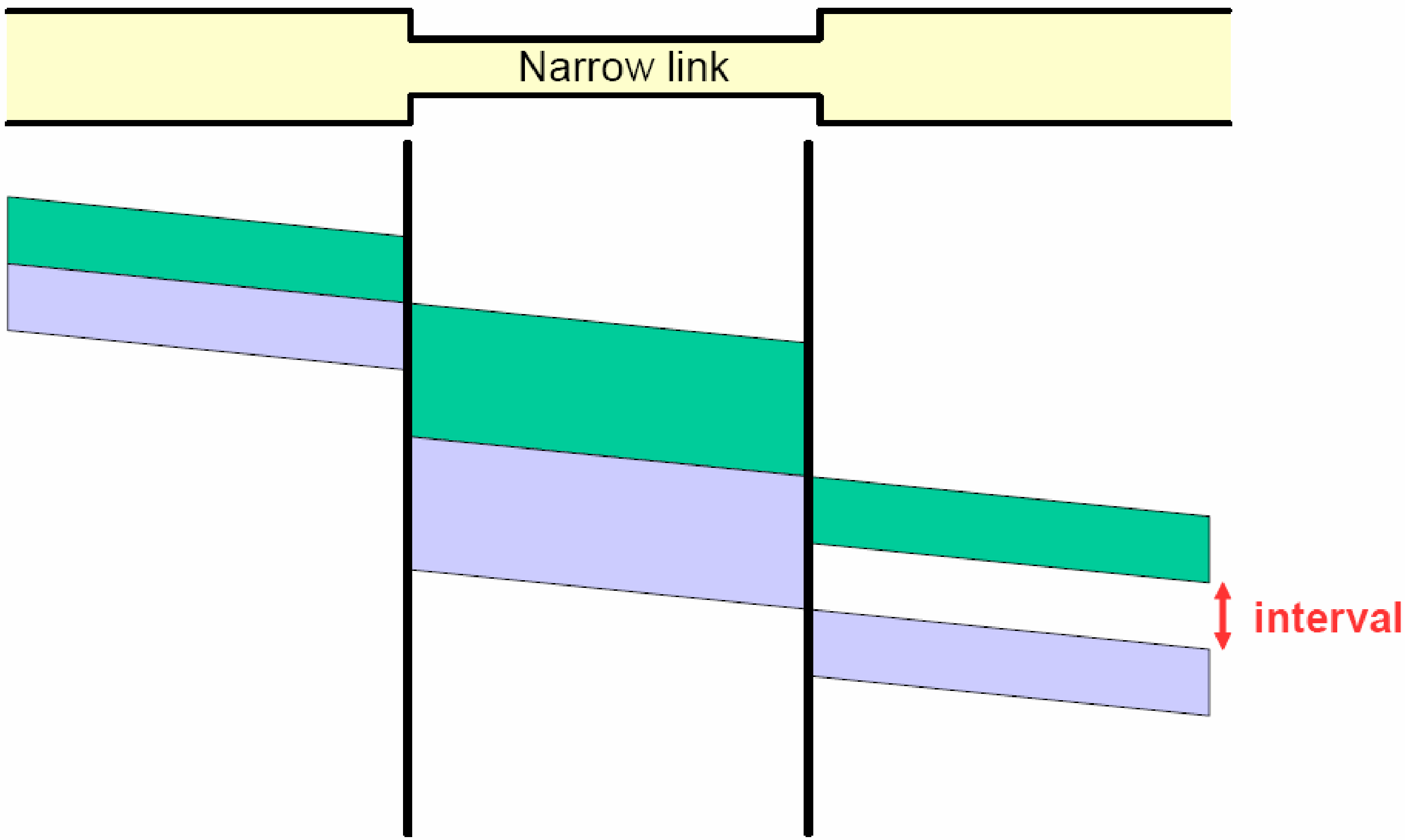
- Learn the *internal* characteristics of a network from *external* observations
- Why in that way? Internal data is often not available. Example:Internet
  - Decentralized management
  - Splitted up in subnetworks
- Examples:
  - Bandwidth estimation
  - Topology identification
  - …

# Bandwidth Estimation

Applications of bandwidth estimation:

- Network administrator: find bottlenecks, find corrupt lines,…
- Video streaming: adapt compression rate
- P2P: find best peer
- …

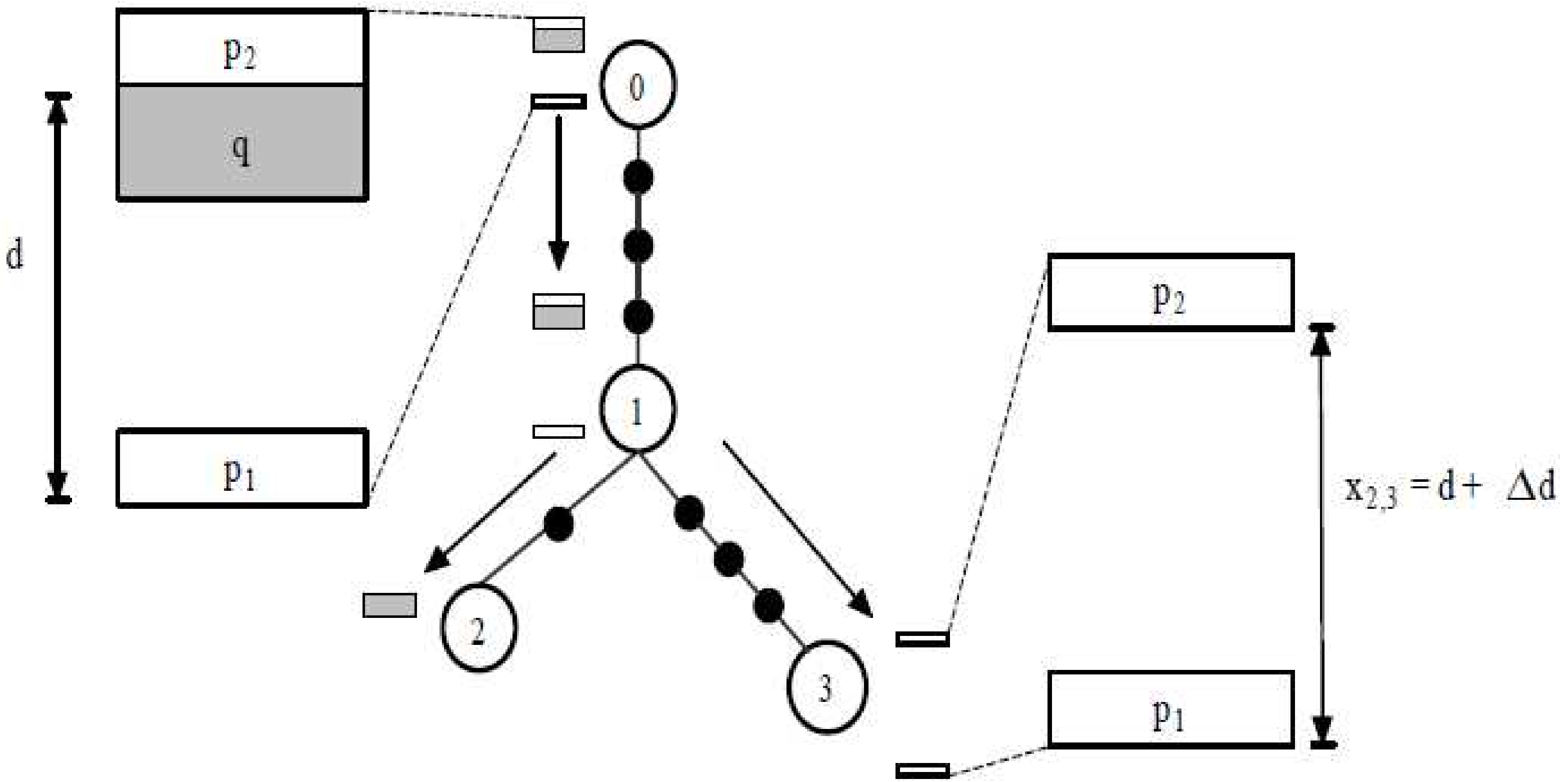
# Bandwidth Estimation with Packet Pair Probing



# Topology Identification

- Determine the topology of the network
- Applications of topology identification:
  - Optimize routing
  - Optimize virtual overlay networks (e.g. P2P)
  - Optimize data delivery: “What is the closest cache server to this particular client?”
  - Attacks
  - …
- Simple way: `traceroute` (only works if the network cooperates)

# Topology Identification with Sandwich Probe Measurement




(from: Maximum Likelihood Network Topology Identification from Edge-based Unicast Measurements, 2002)

# Topology Identification with Sandwich Probe Measurement

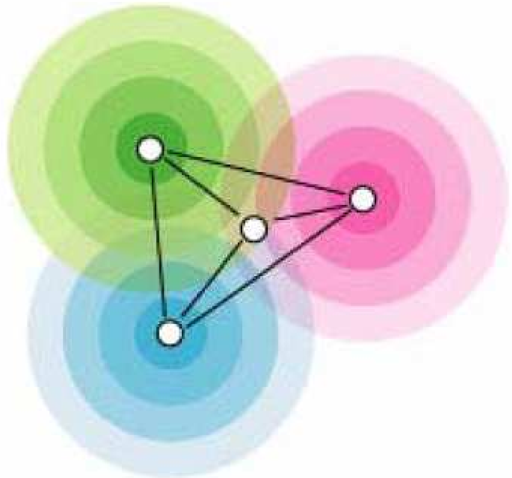
- Every link shared by the paths from the start node to the two destination nodes will increase $\Delta d$
- Approach:
  1. Make many measurements from one start node to several destination nodes
  2. Use statistical techniques to calculate the *most probable* tree connecting the start nodes with the destination nodes
- Assumption: cross-traffic has zero-mean effect

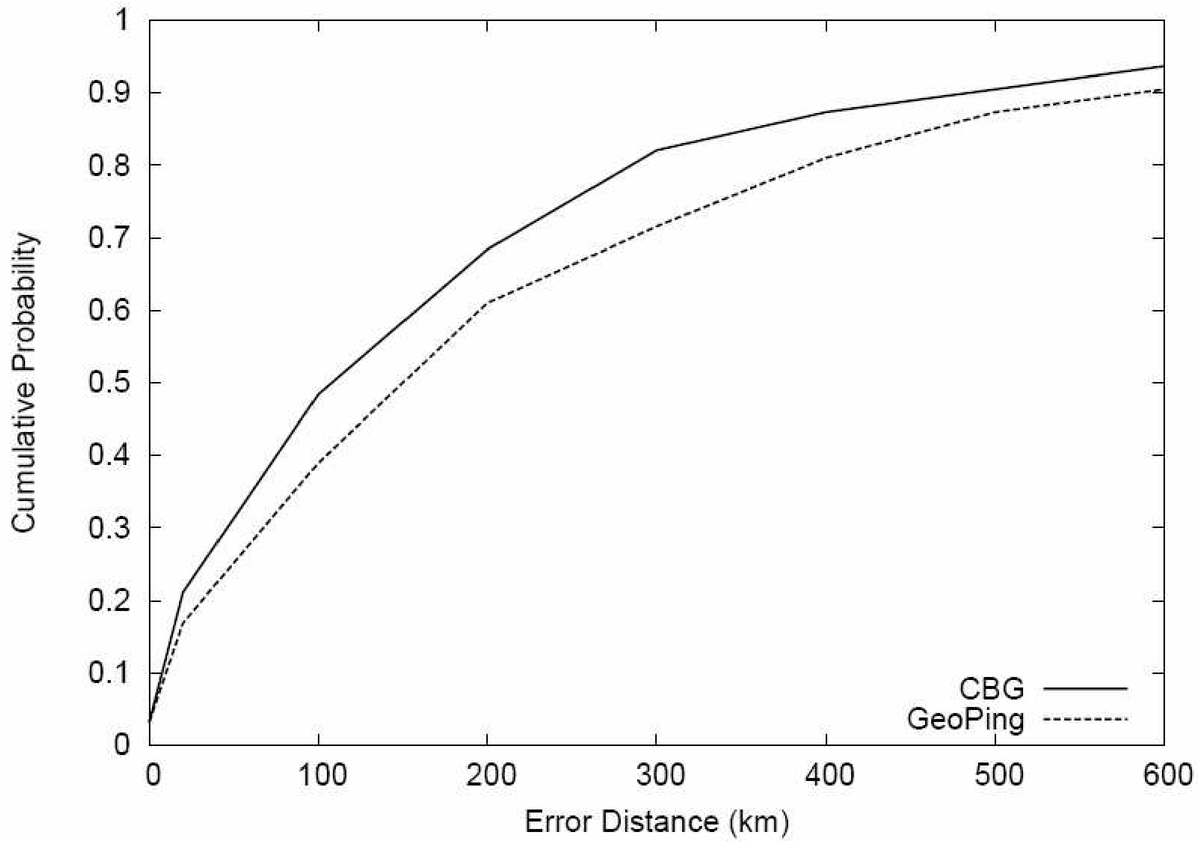
# Geolocation

- Identify the real-world geographical location of a network host
- Sources of information: IP address, GPS (if available), WiFi
- Applications:
  - Provide location-specific information to user (where is the next ...?)
  - Tracking (parcel tracking, criminal prosecution,…)
  - Advertisement
  - …
- Simple approach: map IP address to geographical location

# Geolocation with CBG

Idea: measure delays to reference hosts with known position






(from: Constraint-Based Geolocation of Internet Hosts, 2004)

# WiFi-based Geolocation of Internet Hosts

Approach:

- Companies collect information on geographical positions of WiFi networks (Skyhook, Google Streetview cars,…)
- Client sends list of WLANs that it currently receives to the service provider
- Geographical position is calculated

+ also works inside buildings

+ no special hardware (GPS) required

- requires dense deployment of WiFi