Internet Measurement
& Analysis (IMA)

Traffic Flows

Slides based also on material by
Prof. Anja Feldmann

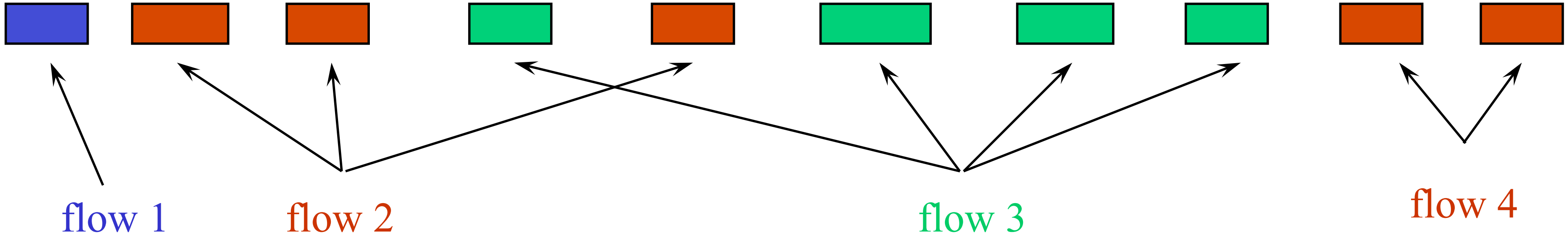
Prof. Georgios Smaragdakis, Ph.D.

What is a flow?

From some Working Groups of Flow Technology (IPFIX):

- A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval.
- All packets belonging to a particular flow have a set of common properties
  - A packet is defined to belong to a flow if it completely satisfies all the defined properties of the flow.

IP flows: What is it?



- Set of packets that "belong together" and share common characteristics

Passive measurement capabilities: Full Packet Monitors

- Available data:
  - All protocol information
  - All content
- Possible analysis:
  - Application performance
  - User behavior
  - Application usage (e.g., P2P)
  - Abuse detection (intrusion detection system)
- Disadvantages:
  - Sheer amount of data
    - Need for data aggregation
  - Bias of the Vantage Point
  - Needle in a haystack problem
  - Only captures on-network information
  - Usually needs fixed installations

Layer 3: IP

IP Header Format

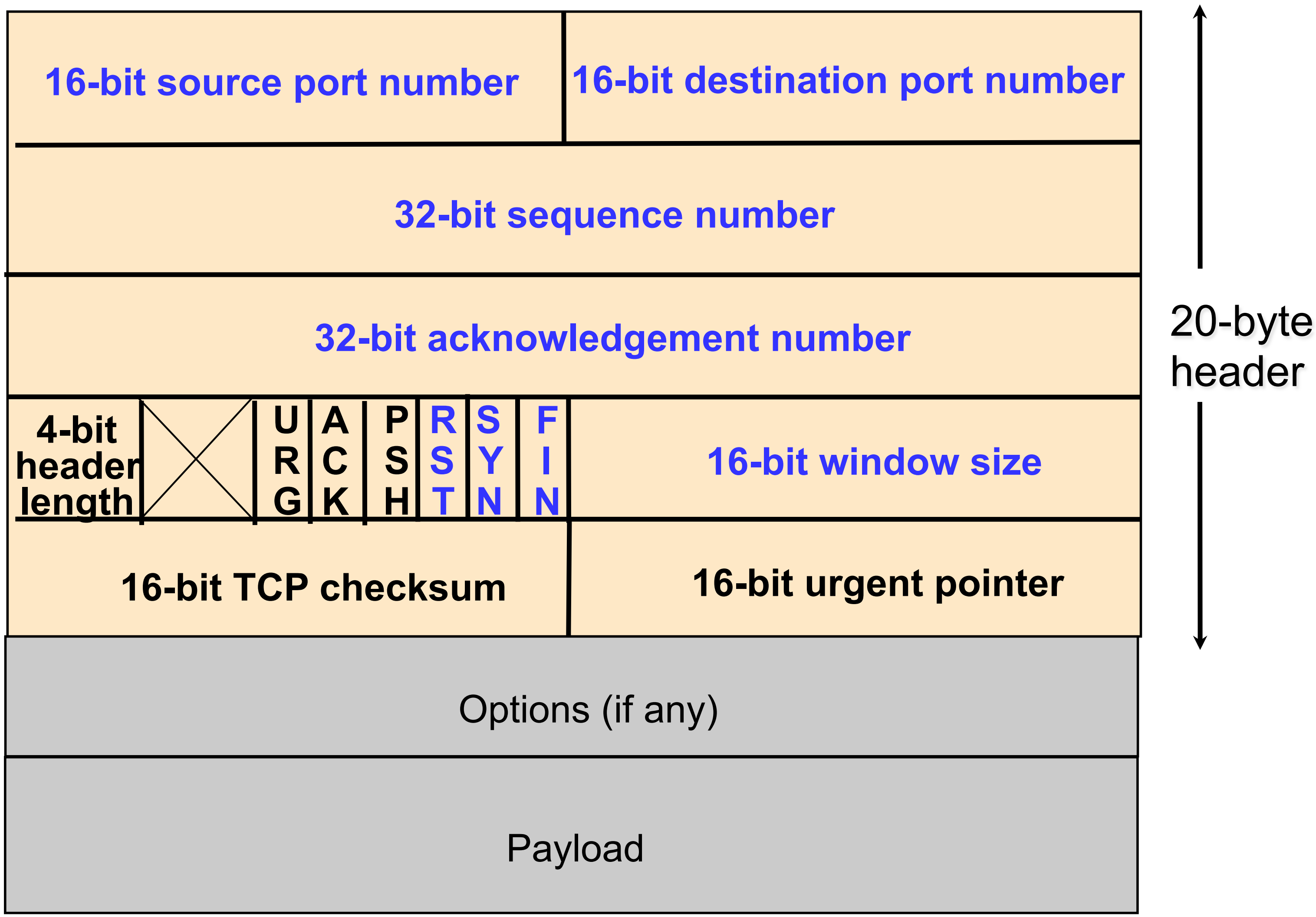
<table>
<tr><td>4-bit Version</td><td>4-bit Header Length</td><td colspan="2">8-bit Type of Service (TOS)</td><td colspan="4">16-bit Total Length (Bytes)</td><td rowspan="5">20-byte header</td></tr>
<tr><td colspan="4">16-bit Identification</td><td>3-bit Flags</td><td colspan="3">13-bit Fragment Offset</td></tr>
<tr><td colspan="2">8-bit Time to Live (TTL)</td><td colspan="2">8-bit Protocol</td><td colspan="4">16-bit Header Checksum</td></tr>
<tr><td colspan="8">32-bit Source IP Address</td></tr>
<tr><td colspan="8">32-bit Destination IP Address</td></tr>
<tr><td colspan="8">Options (if any)</td><td></td></tr>
<tr><td colspan="8">Payload</td><td></td></tr>
</table>

IP header analysis

- Source/destination addresses for traffic
  - Identity of popular Web servers & heavy customers (heavy hitters)
- Traffic breakdown by protocol (TCP/UDP/ICMP)
  - Amount of traffic not using congestion control
- Distribution of packet delay through the router
  - Identification of typical delays and anomalies
- Distribution of packet sizes
  - Workload models for routers and measurement devices
- Burstiness of the traffic on the link over time
  - Provisioning rules for allocating link capacity
- Throughput between each pair of src-dst addresses
  - Detection and diagnosis of performance problems

Layer 4: Transport; TCP

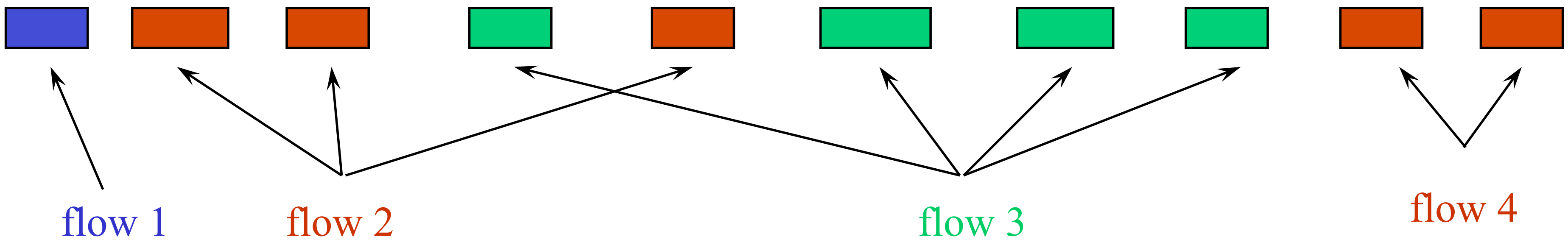
TCP header format



TCP header analysis

- Source and destination port numbers
  - Popular applications (HTTP, P2P, SMTP, DNS)
  - Number of parallel connections between src-dst pairs
- Sequence/ACK numbers and packet timestamps
  - Out-of-order/lost packets; violations of congestion control
  - Estimates of throughput and delay of Web downloads
- Number of packets/bytes per connection
  - Size of typical Web transfers; frequency of bulk transfers
- SYN flags from client machines
  - Unsuccessful connection requests; denial-of-service attacks
- FIN/RST flags from client machines
  - Frequency of Web transfers aborted by clients

IP flows: What is it?



- Set of packets that "belong together"
  - Source/destination IP addresses and port numbers
  - Same protocol, ToS bits, …
  - Same input/output interfaces at a router (if known)
- Packets that are "close" together in time
  - Maximum spacing between packets (e.g., 15 sec, 30 sec)
  - Example: Flows 2 and 4 are different flows due to time

5-tuple Flow Definition

- Available data:
  - Summary information about traffic flows
- Flow:
  - **<SRC_IP, DST_IP, SRC_port, DST_port, protocol>**

Application layer

Packet content: How about the payload?

- Application-layer header
  - HTTP and RTSP request and response headers
  - FTP, NNTP, and SMTP commands and replies
  - DNS queries and responses; OSPF/BGP messages
- Application-layer body
  - HTTP resources (or checksums of the contents)
  - User keystrokes in Telnet/Rlogin sessions
- Security/privacy
  - Significant risk of **violating user privacy**
  - More sensitive for information from higher-level protocols
  - Traffic analysis thwarted by use of end-to-end encryption

HTTP request and response message

```
GET /tutorial.html HTTP/1.1
Date: Mon, 27 Aug 2001 08:09:01 GMT
From: jrex@research.att.com
User-Agent: Mozilla/4.03
CRLF
```

Request

Response

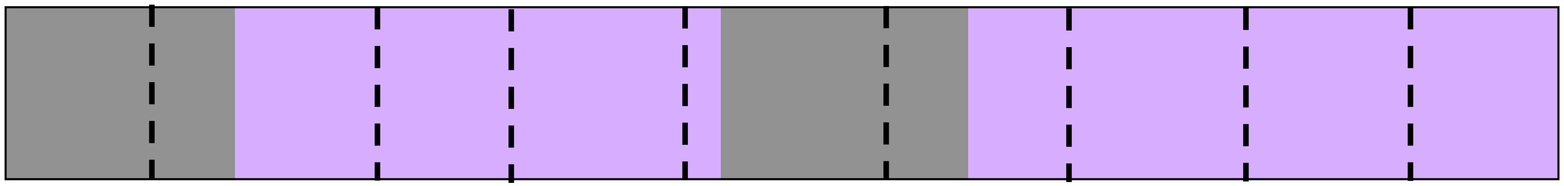
```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2001 10:09:03 GMT
Server: Netscape-Enterprise/3.5.1
Last-Modified: Sun, 12 Mar  2000 11:12:23 GMT
Content-Length: 23
CRLF
Traffic measurement talk
```

Application-layer analysis

- URLs from HTTP request messages
  - Popular resources/sites; potential benefits of caching
- Meta-data in HTTP request/response messages
  - Content type, cacheability, change frequency, etc.
  - Browsers, protocol versions, protocol features, etc.
- Contents of DNS messages
  - Common queries, frequency of errors, query latency
- Contents of Telnet/Rlogin sessions
  - Intrusion detection (break-ins, stepping stones)
- Routing protocol messages
  - Workload for routers; detection of routing anomalies
  - Tracking the current topology/routes in the backbone

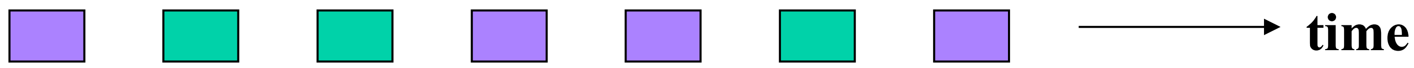
Mechanics: Application-level messages

- Reconstructing ordered, reliable byte stream
  - Sequence number and segment length in TCP header
  - Heap to store packets in correct order & discard duplicates
- Extraction of application-level messages
  - Parsing the syntax of the application-level header
  - Identifying the start of the next message (if any)

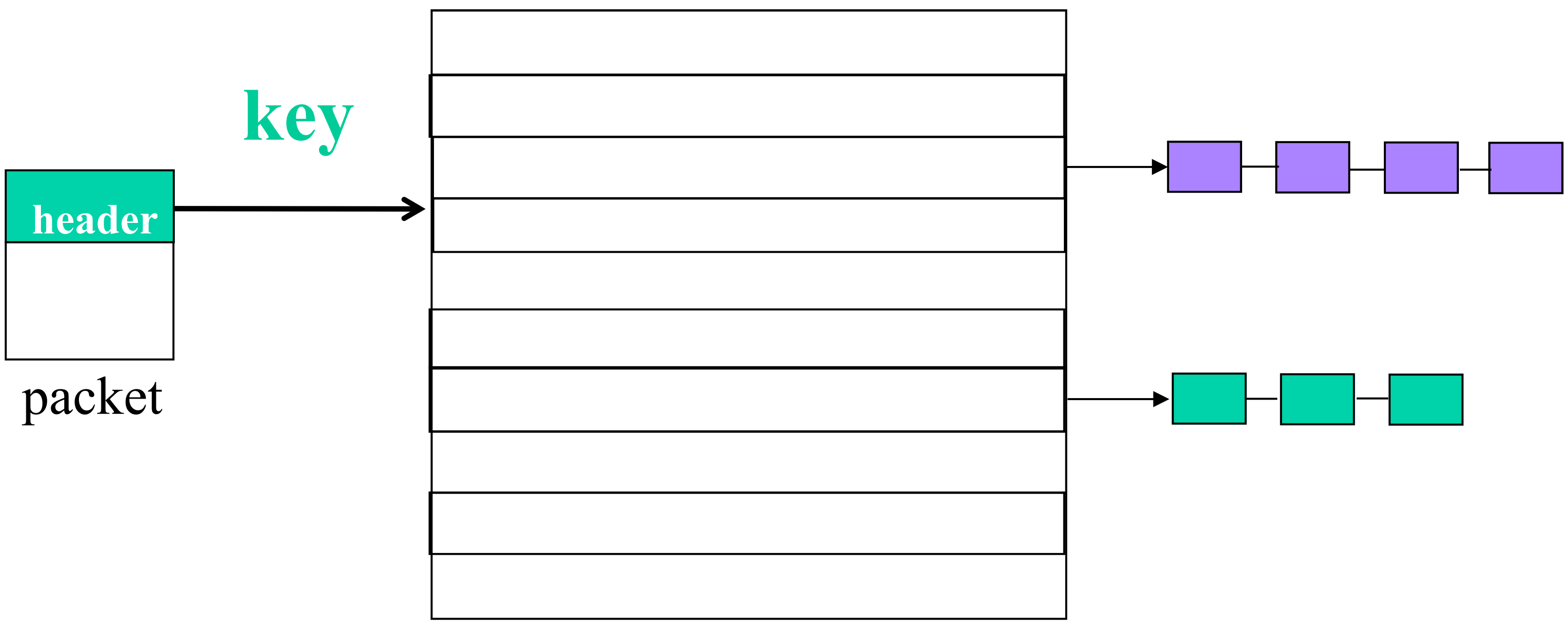


- Logging or online analysis of message
  - Record URL, header, body, checksum, timestamps, etc.
  - Copy traces or analysis result to separate machine

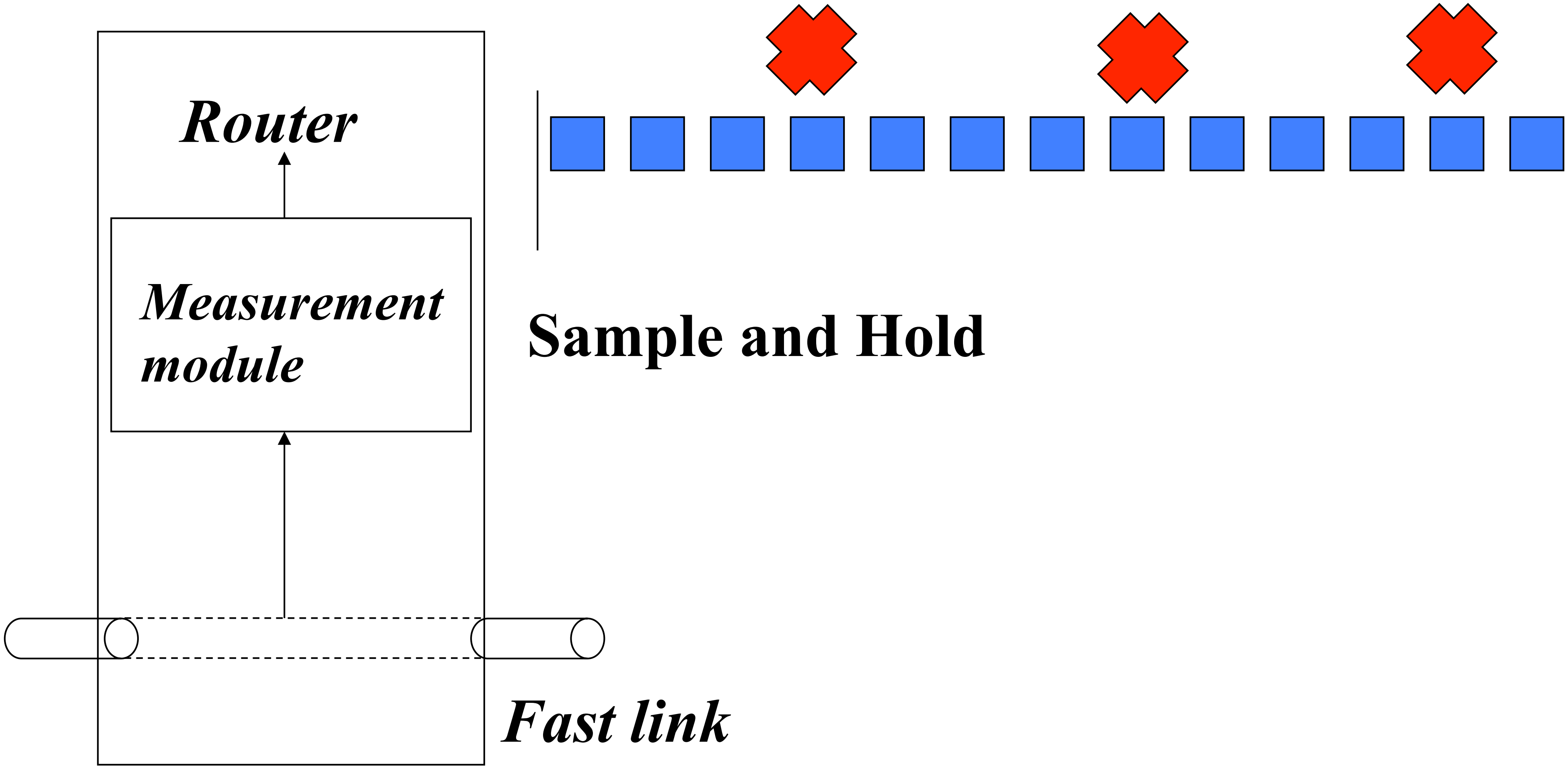
Mechanics: Application-level messages



- Application-level transfer may span multiple packets
  - Demultiplex packets into separate "flows"
  - Key of **source/dest IP addresses, port, and protocol**
  - Hash table to store packets from different flows



Measurements at the Router



Standard Bloom Filters

A bloom filter for representing a set $S=\{x_1, x_2, \cdots, x_n\}$ of n elements is described by an array of m bits (m<<n)

Standard Bloom Filters

A bloom filter for representing a set $S=\{x_1, x_2, \cdots, x_n\}$ of n elements is described by an array of m bits (m<<n)

0	0	0	0	0	0	0	0	0	0	0	0

Initially all the m bits are set to '0'

Standard Bloom Filters

A bloom filter for representing a set $S=\{x_1, x_2, \cdots, x_n\}$ of n elements is described by an array of m bits (m<<n)

Insertion:

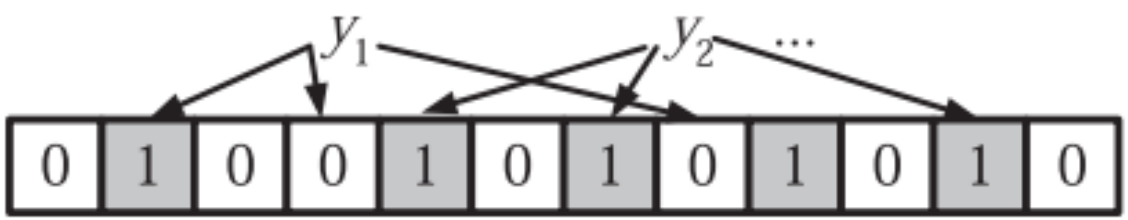
- Bloom Filter used k independent hash functions $h_1, h_2, \cdots, h_k$

With range $\{1,\cdots,m\}$.

- Each hash function maps an element x_i to a random number uniform over the range $\{1,\cdots,m\}$
- For each item x_i the bits $h_j(x_i)$ are set to 1 (if it is 0), j in [1,k]

Standard Bloom Filters

A bloom filter for representing a set S={x_1, x_2, ⋯, x_n} of n elements is described by an array of m bits (m<<n)



Query:

- To check if an item y_i is in set S, we check if **all** $h_j(y_i)$, j in [1,k] are set to 1
- If not, then y_i is not in S
- If yes, then y_i is **probably** in S with some (high) probability

Standard Bloom Filters

Bloom Filter may yield a **false positive**

Consider (perfect) hash functions. Then each bit can be marked with probability 1/m.

Standard Bloom Filters

Bloom Filter may yield a **false positive**

Consider (perfect) hash functions. Then each bit can be marked with probability 1/m.

When the item is hashed using one hash function, the probability that a specific bit is still '0' is:

$$1 - 1/m$$

Standard Bloom Filters

Bloom Filter may yield a **false positive**

Consider (perfect) hash functions. Then each bit can be marked with probability 1/m.

When the item is hashed using **k** hash function, the probability that a specific bit is still '0' is:

$$(1- 1/m)^k$$

Standard Bloom Filters

Bloom Filter may yield a **false positive**

Consider (perfect) hash functions. Then each bit can be marked with probability 1/m.

After **all** n items are hashed using k hash functions, the probability that a specific bit is still '0' is:

$$(1- 1/m)^{kn}$$

Standard Bloom Filters

Bloom Filter may yield a **false positive**

Consider (perfect) hash functions. Then each bit can be marked with probability 1/m.

After **all** n items are hashed using k hash functions, the probability that a specific bit is still '0' is:

$$(1- 1/m)^{kn} \approx e^{-kn/m}$$

Counting Bloom Filters

Inserting elements to Bloom Filters is easy, but to delete them can not be done by reversing the process!

Counting Bloom Filters

To support deletion of items (in addition to insertions) we can not rely on standard Bloom Filters.

In Counting Bloom Filters, each entry in the Bloom Filter is not a single bit but rather a small counter.

- When an item is inserted, the corresponding counters are incremented
- When an item is deleted the corresponding counters are decremented.
- To avoid counter overflow, the counters have to be sufficiently large, or delete all with the median value

Application: Recording Heavy Heaters

The goal is to determine heavy flows in a router.

Motivation: Denial of Service (DoS) Attacks, Traffic Matrix estimation.

- Cheap memories (DRAM) are **too slow** to count all packets
- Fast memories (SRAM) are **too small** to keep counters for all streams
- Opportunity: elephants matter, mice don't
- Problem: usually we don't know in advance **which** streams **are large**

Recording Heavy Heaters

The goal is to determine heavy flows in a router.

Packets belong to flows.

Each packet entering a router is hashed k times into a Bloom Filter.

The counter in the Bloom filter is increased by the packet bytes. When the minimum count (e.g. 1% of link capacity) for a flow reaches a threshold the flow is tagged as heavy heater.

Recording Heavy Heaters

The goal is to determine heavy flo… r.

Packets belong to flows.

Each packet ent… k times into a Bloom Filte…

The … filter is increased by the packet bytes … imum count (e.g. 1% of link capacity) for a fl… es a threshold the flow is tagged as heavy heater.

What can go wrong?

Recording Heavy Heaters

False Positive in this situation corresponds to:

- Light flows that happens to hash into k locations that are also hashed into by heavy flows
- Light flow that happens to hash into locations hit by several other flows

Recording Heavy Heaters

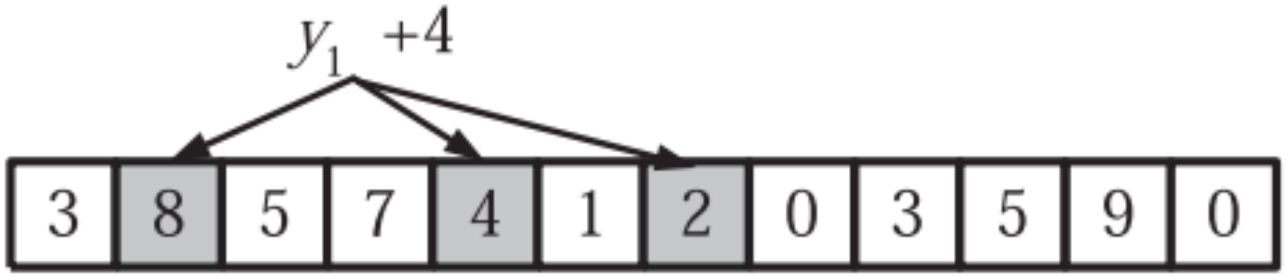
To reduce the false positives Estan and Varghese introduced the idea of “**conservative update**” a variation of Counting Bloom Filter.

Recording Heavy Heaters

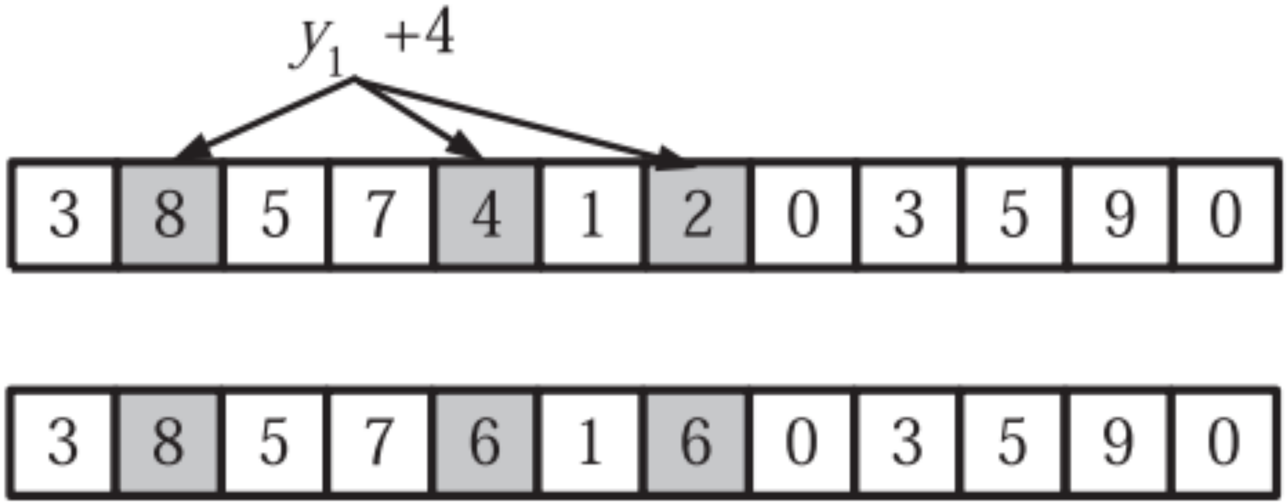
To reduce the false positives Estan and Varghese introduced the idea of "**conservative update**" a variation of Counting Bloom Filter.

When updating a counter upon arrival of a new packet, if B is the packet size, then update the associated counters values by no more than M_k+B where M_k is the minimum of the associated counters.

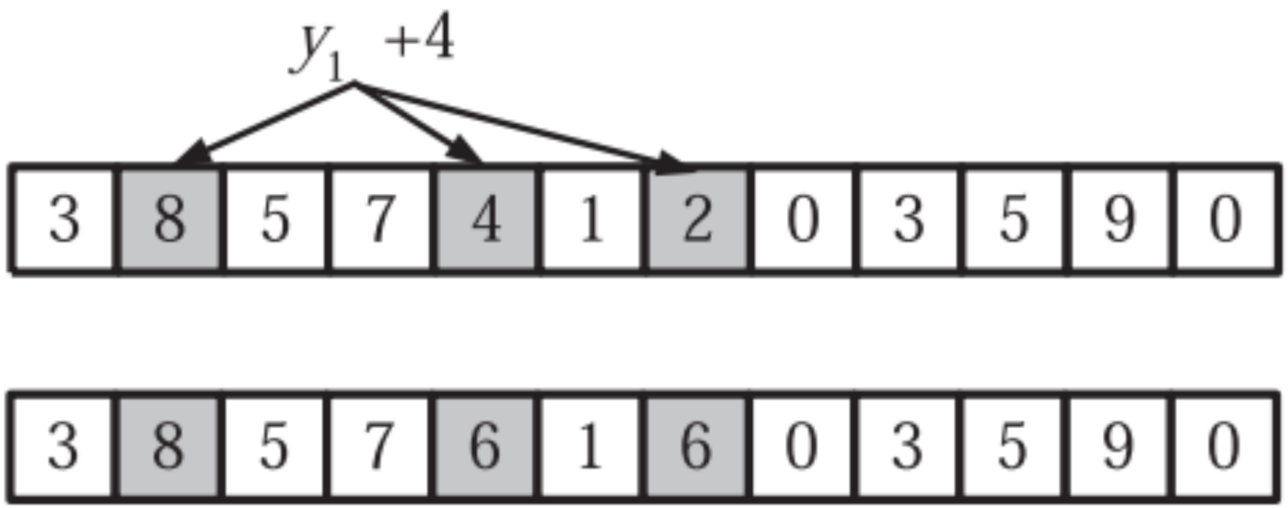
Recording Heavy Heaters



Recording Heavy Heaters

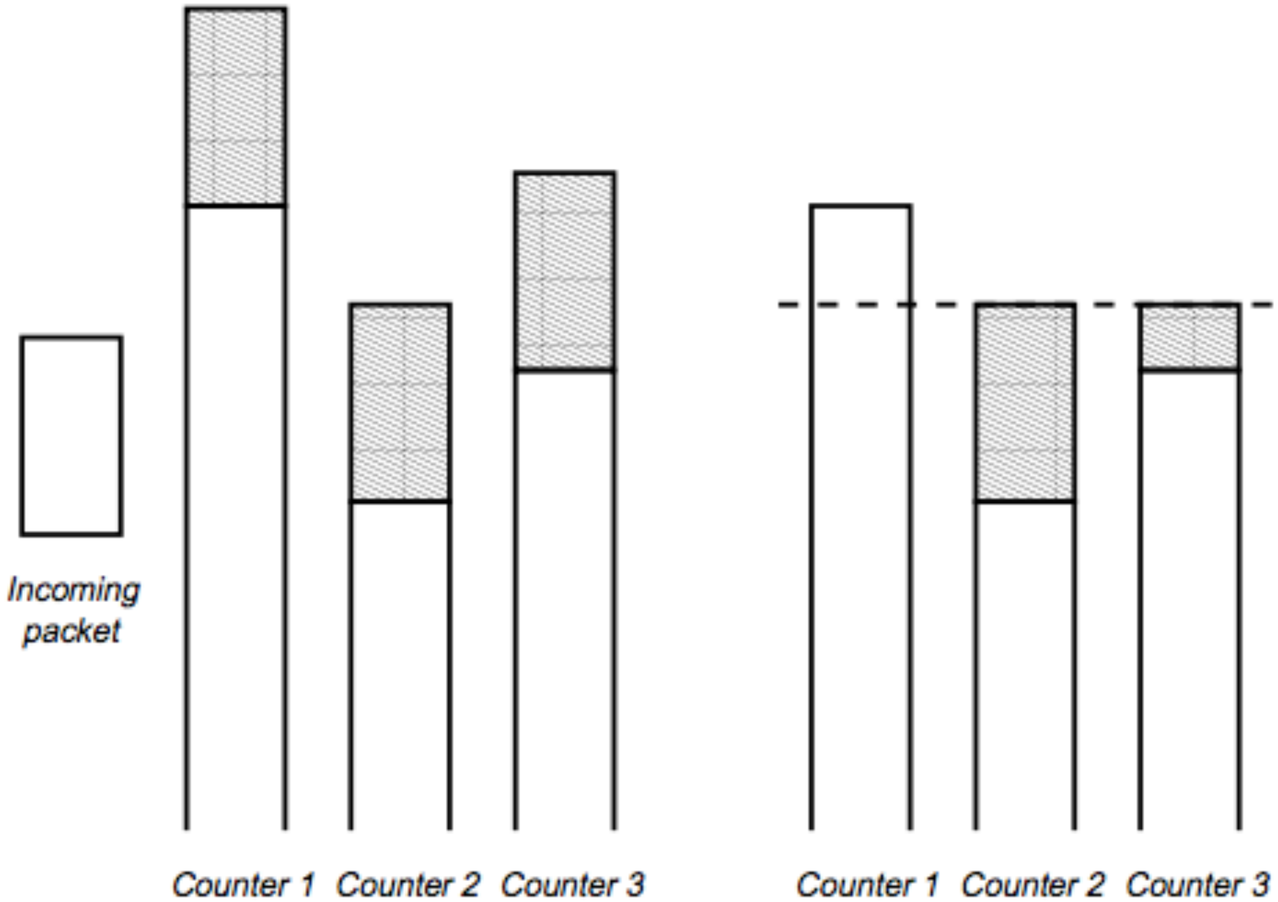


Recording Heavy Heaters



Much more efficient than keeping state for each individual flow!

Recording Heavy Heaters



Non-conservative Update

Conservative Update

System constraints

- High data rate
  - Bandwidth limits on CPU, I/O, memory, and disk/tape
  - Could monitor lower-speed links (near the edge of network)
- High data volume
  - Space limitations in main memory and on disk/tape
  - Could do online analysis to sample, filter, & aggregate
- High processing load
  - CPU/memory limits for extracting, counting, & analyzing
  - Could do offline processing for time-consuming analysis
- General solutions to system constraints
  - Sub-select the traffic (addresses/ports, first n bytes)
  - Kernel and interface card support for measurement
  - Efficient/robust software and hardware for the monitor

Passive measurement capabilities: Packet monitors (2.)

- Deployment scenarios:
  - Needs cooperation of the network operator
  - Limited number
  - Specialized hardware/software
  - Data collection / aggregation infrastructure

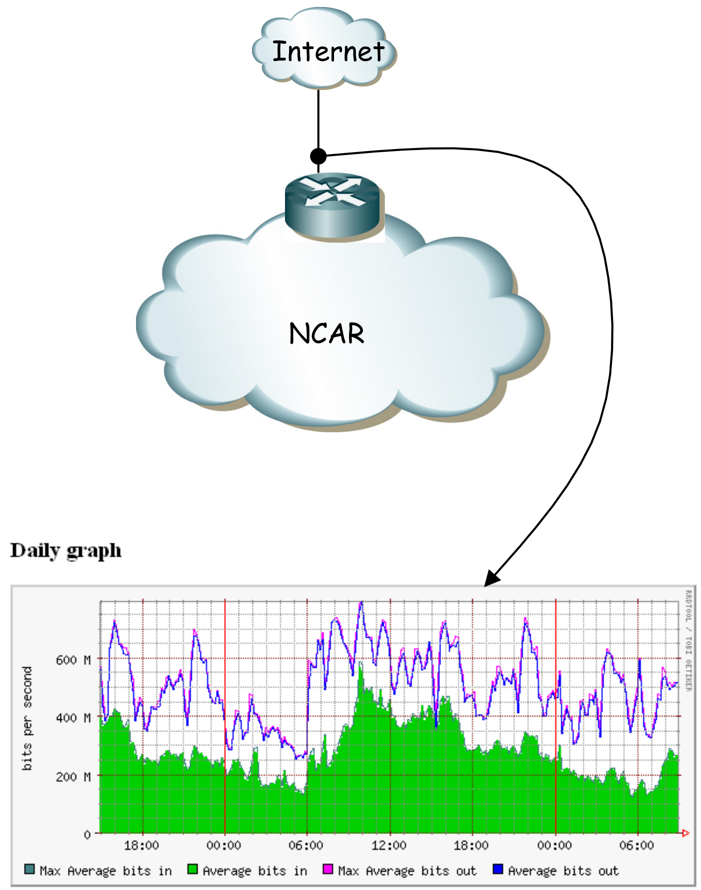
- Challenges
  - Data integrity
  - Incomplete data
  - User privacy & network security
  - Data correlation
  - Data privacy vs. data sharing
  - Data filtering
  - Data collection across network confederations

Netflow

- Developed and patented by Cisco in 1996
- Classifies network traffic into "flows" by inspecting packets at layers 2 – 4.
- Currently on standards track - IPFIX
- "Flows" can be analyzed to provide network and security monitoring, network planning, traffic analysis and IP accounting.

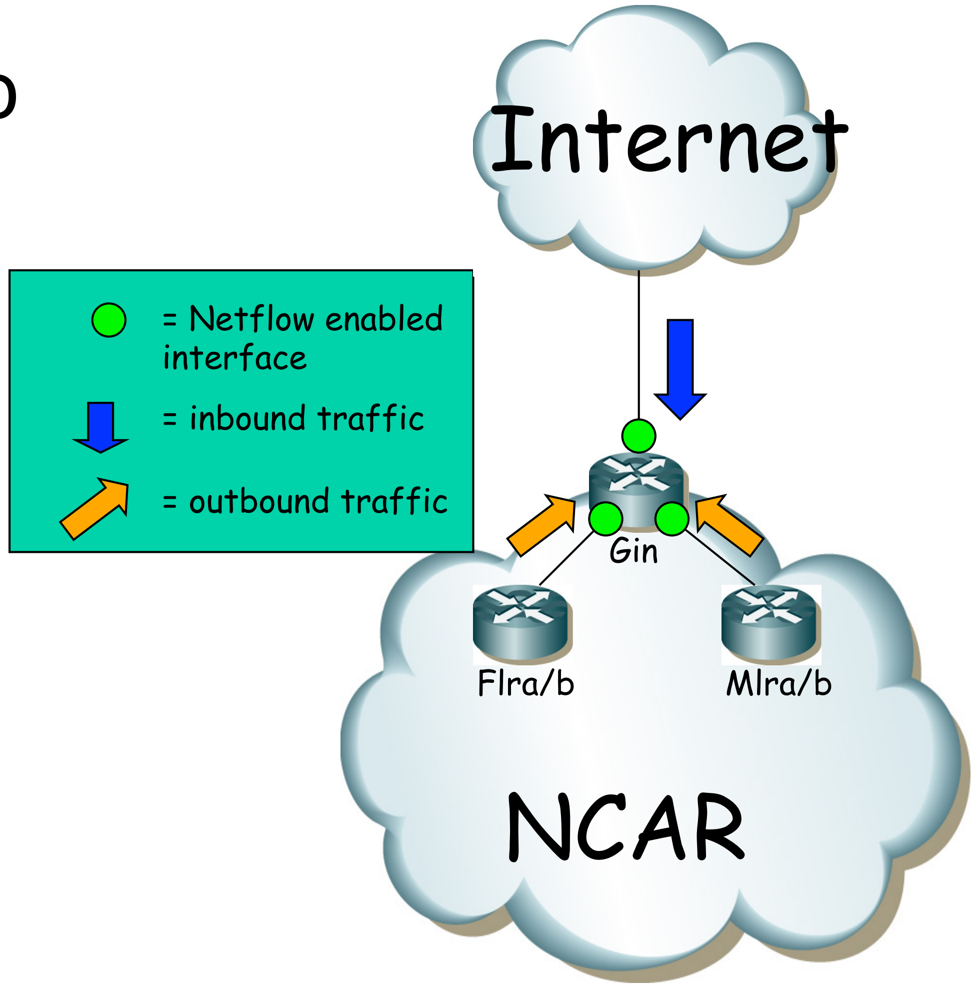
Why Use Netflow?

- Enterprise
  - protocol distribution
  - monitor users/applications
  - identify malicious traffic
- Service Provider
  - peering
  - Identify malicious traffic
  - planning
  - traffic engineering
  - accounting/billing



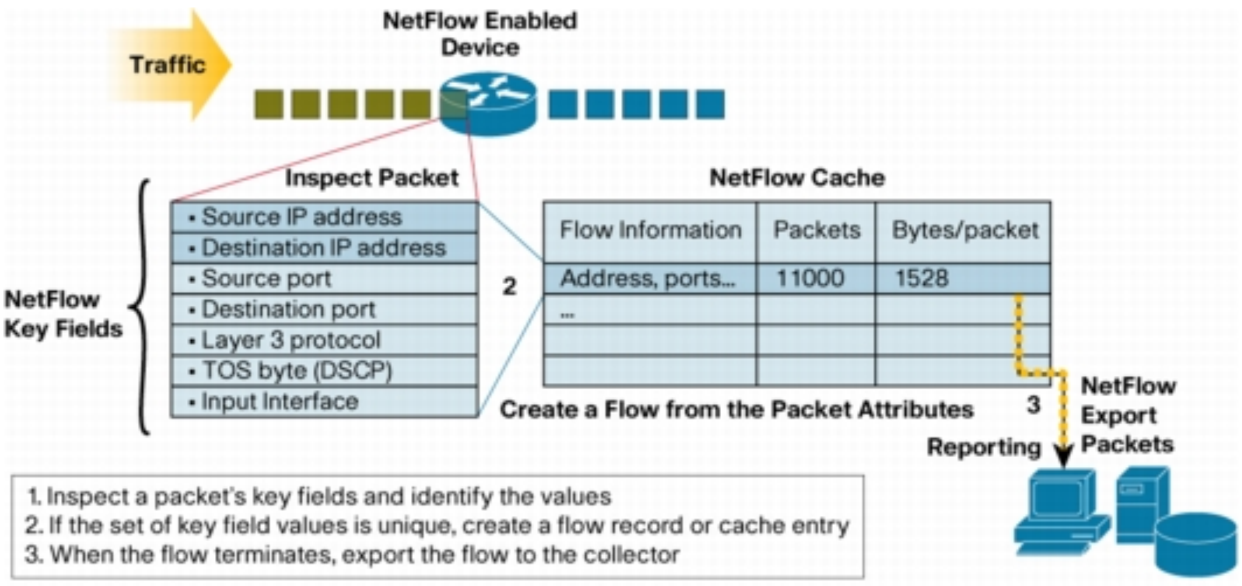
Determine which routers/interfaces to enable netflow

- Enable netflow on selected interfaces to capture all inbound/ outbound traffic
- Neflow only enabled inbound on an interface
- Avoid double counting!!



Example: Cisco Netflow

- Traffic monitoring system on switches and routers
  - Cache with 5-tuples: srcIP, srcPort, dstIP, dstPort, proto
  - Upon packet lookup, cache entry is created or updated
  - When cache full, flows are timed-out
  - Timers for flow time-outs



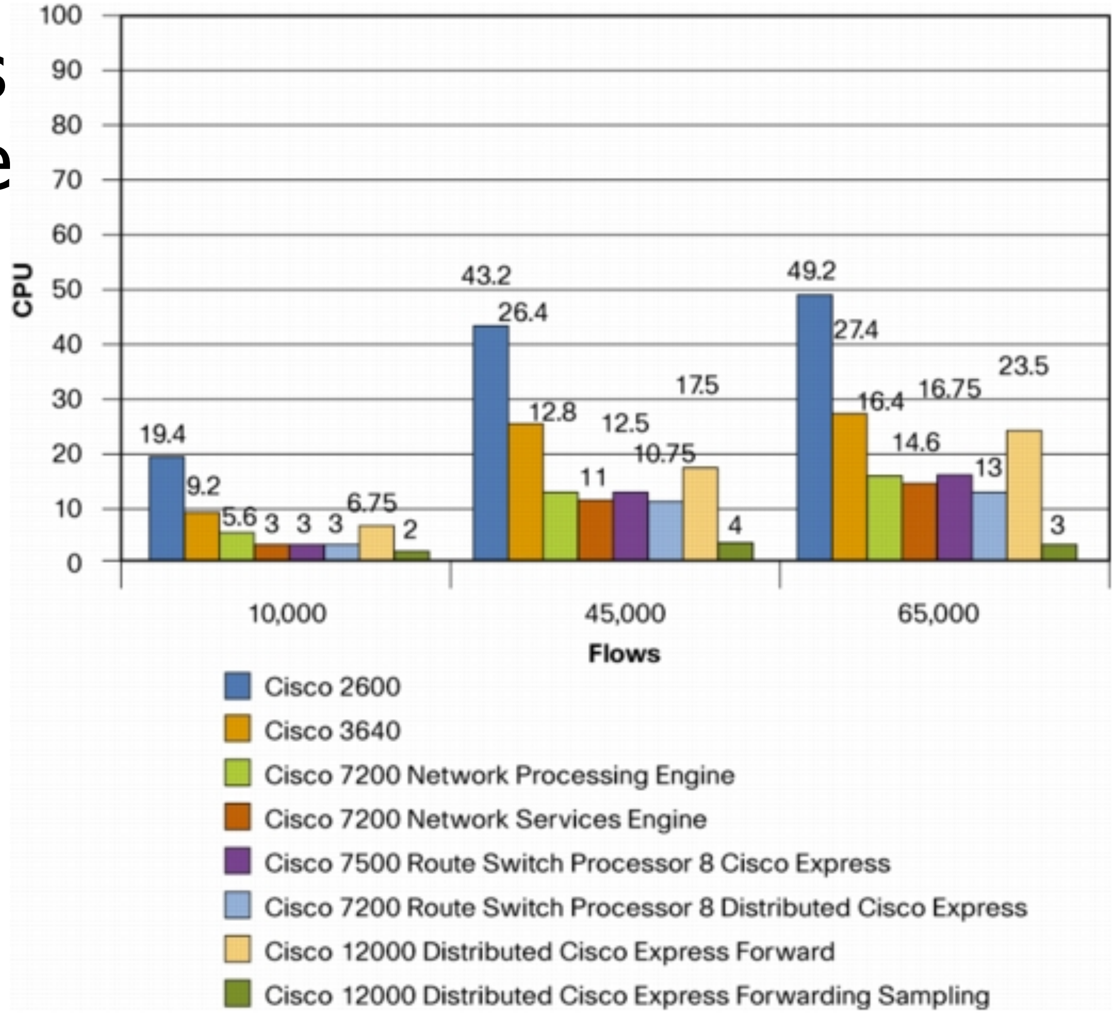
Netflow Example (https://mcfp.weebly.com/)

```
StartTime,Dur,Proto,SrcAddr,Sport,Dir,DstAddr,Dport,State,sTos,dTos,TotPkts,TotBytes,SrcBytes,Label
2011/08/10 09:46:53.047277,3550.182373,udp,212.50.71.179,39678,  <->,147.32.84.229,13363,CON,0,0,12,875,413,flow=Background-UDP-Established
2011/08/10 09:46:53.048843,0.000883,udp,84.13.246.132,28431,  <->,147.32.84.229,13363,CON,0,0,2,135,75,flow=Background-UDP-Established
2011/08/10 09:46:53.049895,0.000326,tcp,217.163.21.35,80,  <?>,147.32.86.194,2063,FA_A,0,0,2,120,60,flow=Background
2011/08/10 09:46:53.053771,0.056966,tcp,83.3.77.74,32882,  <?>,147.32.85.5,21857,FA_FA,0,0,3,180,120,flow=Background
2011/08/10 09:46:53.053937,3427.768066,udp,74.89.223.204,21278,  <->,147.32.84.229,13363,CON,0,0,42,2856,1596,flow=Background-UDP-Established
2011/08/10 09:46:53.056921,3086.547363,tcp,66.169.184.207,49372,  <?>,147.32.84.229,13363,PA_PA,0,0,591,45931,26480,flow=Background
2011/08/10 09:46:53.058746,3589.631348,udp,182.239.167.121,49649,  <->,147.32.84.229,13363,CON,0,0,12,1494,1122,flow=Background-UDP-Established
2011/08/10 09:46:53.058760,20.360268,tcp,147.32.3.93,443,  <?>,147.32.84.59,51790,FPA_FRPA,0,0,133,81929,67597,flow=Background-Established-cmpgw-CVUT
2011/08/10 09:46:53.062095,3118.470947,udp,24.117.206.20,8697,  <->,147.32.84.229,13363,CON,0,0,13,4328,840,flow=Background-UDP-Established
2011/08/10 09:46:53.068389,1065.003052,tcp,94.208.78.74,50687,  <?>,147.32.84.229,13363,FPA_RPA,0,0,156,14804,7699,flow=Background
2011/08/10 09:46:53.074655,2.210671,udp,79.129.201.26,56877,  <->,147.32.84.229,13363,CON,0,0,4,379,137,flow=Background-UDP-Established
2011/08/10 09:46:53.075905,0.187434,tcp,147.32.86.194,2065,   ->,217.163.21.35,80,FSPA_FSPA,0,0,11,3872,1147,flow=Background-TCP-Established
2011/08/10 09:46:53.078297,3599.972412,tcp,147.32.80.13,80,  <?>,147.32.84.162,51769,PA_A,0,0,72157,61638544,60214264,flow=From-Background-CVUT-Proxy
2011/08/10 09:46:53.082381,0.000307,tcp,74.200.246.228,80,  <?>,147.32.84.59,49382,FA_RA,0,0,3,180,60,flow=Background-Established-cmpgw-CVUT
2011/08/10 09:46:53.087248,0.000258,tcp,77.238.167.32,80,  <?>,147.32.86.194,2060,FA_A,0,0,2,120,60,flow=Background
2011/08/10 09:46:53.093292,37.925823,tcp,94.124.104.196,80,  <?>,147.32.84.59,49500,PA_FRA,0,0,1921,2636496,2625276,flow=Background-Established-cmpgw-CVUT
2011/08/10 09:46:53.098713,0.312088,tcp,98.127.111.126,51534,  <?>,147.32.84.229,13363,FRPA_FPA,0,0,10,750,407,flow=Background
2011/08/10 09:46:53.100496,2407.466797,udp,123.1.72.4,16562,  <->,147.32.84.229,13363,CON,0,0,4,268,148,flow=Background-UDP-Established
2011/08/10 09:46:53.104932,3495.295410,tcp,147.32.84.229,443,  <?>,212.217.56.83,58258,PA_PA,0,0,287,24672,10309,flow=Background
2011/08/10 09:46:53.104948,3591.918945,tcp,147.32.84.229,443,  <?>,213.142.200.29,10004,PA_PA,0,0,4360,339588,125248,flow=Background
2011/08/10 09:46:53.104954,3514.610352,tcp,147.32.84.229,13363,  <?>,93.45.94.195,44977,PA_PA,0,0,310,52117,9301,flow=Background
2011/08/10 09:46:53.104959,3599.977539,tcp,147.32.84.229,13363,  <?>,83.78.136.90,52573,PA_PA,0,0,164,12065,5378,flow=Background
2011/08/10 09:46:53.106431,507.347626,tcp,147.32.80.13,80,  <?>,147.32.85.112,10885,FPA_FA,0,0,162760,137136528,132816366,flow=From-Background-CVUT-Proxy
2011/08/10 09:46:53.107352,0.001105,udp,217.164.10.229,7797,  <->,147.32.84.229,13363,CON,0,0,2,582,77,flow=Background-UDP-Established
2011/08/10 09:46:53.107669,0.000000,tcp,199.59.148.20,443,   ?>,147.32.84.184,51855,A_,0,,1,60,60,flow=Background
2011/08/10 09:46:53.112416,1288.713379,udp,77.100.246.74,6430,  <->,147.32.84.229,13363,CON,0,0,18,1244,704,flow=Background-UDP-Established
2011/08/10 09:46:53.118569,2292.730469,udp,58.72.174.152,1769,  <->,147.32.84.229,13363,CON,0,0,16,3828,548,flow=Background-UDP-Established
2011/08/10 09:46:53.120880,273.880157,tcp,83.137.254.245,49455,  <?>,147.32.84.229,13363,PA_PA,0,0,121,14096,10430,flow=Background
2011/08/10 09:46:53.125002,3572.482422,tcp,147.32.84.59,49238,  <?>,74.125.232.215,443,PA_PA,0,0,1332,627474,282534,flow=Background-Established-cmpgw-CVUT
2011/08/10 09:46:53.127875,0.000368,udp,147.32.84.138,42315,  <->,147.32.80.9,53,CON,0,0,2,214,81,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/10 09:46:53.128036,0.000225,udp,147.32.84.138,42626,  <->,147.32.80.9,53,CON,0,0,2,214,81,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/10 09:46:53.131650,3133.970947,udp,186.204.215.229,12677,  <->,147.32.84.229,13363,CON,0,0,10,1218,908,flow=Background-UDP-Established
2011/08/10 09:46:53.132734,619.352722,udp,147.32.84.229,13363,   ->,31.9.113.254,23320,INT,0,,4,568,568,flow=Background-UDP-Attempt
2011/08/10 09:46:53.136258,0.000227,udp,147.32.84.138,58276,  <->,147.32.80.9,53,CON,0,0,2,214,81,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/10 09:46:53.136265,0.000272,udp,147.32.84.138,58867,  <->,147.32.80.9,53,CON,0,0,2,214,81,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/10 09:46:53.137179,536.390381,tcp,109.183.212.236,61775,  <?>,147.32.84.130,20,FPA_FA,0,0,23574,10855048,10056796,flow=Background
```

Example: Cisco Netflow (2)

- Impact of # of flows on router CPU usage

- Impact of sampling on average CPU utilization (Cisco 7505):
  - 1/100: -75%
  - 1/1000: -82%



Sampling Options

- At high speeds, traffic monitoring requires sampling
- How to sample?
  - **Systematic** sampling
    - Pick out every 100th packet and record entire packet/record header, e.g., Netflow
    - Ok only if no periodic component in process
  - **Random** sampling
    - Flip a coin for every packet, sample with prob. 1/100
  - **Time-based Sampling**: Record a link load every n seconds, e.g., SNMP
  - Systematic and Time-based Sampling may be biased to heavy flows

Sampling (2.)

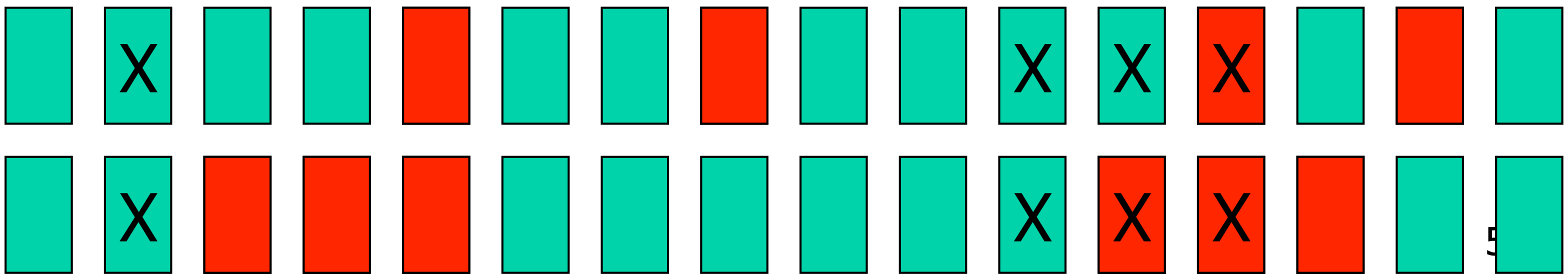
- What can we infer from samples?
- Easy:
  - Metrics directly defined over variables of interest, e.g., mean, variance etc.
  - Confidence interval = "error bar"
- Hard:
  - Small probabilities:
    "Number of SYN packets sent from A to B"
  - Events such as: "Has X received any packets"?

Sampling (3.)

- Hard:
  - Metrics over sequences
  - Example: “How often is a packet from X followed immediately by another packet from X?”
    - Higher-order events: probability of sampling i successive records is p^i
    - Would have to sample different events, e.g., flip coin, then record k packets

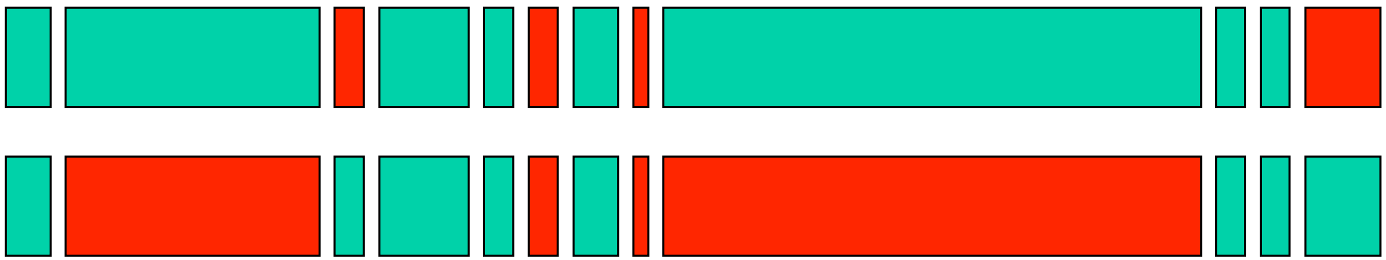
packet sampling

sequence sampling



Sampling (4.)

- Sampling objects with different weights
- Example:
  - Weight = flow size
  - Estimate average flow size
  - Problem: a small number of large flows can contribute very significantly to the estimator
- **Stratified sampling**: make sampling probability depend on weight (sampling proportionated to the bytes of a flow)
  - Sample "per byte" rather than "per flow"
  - Try not to miss the "heavy hitters" (heavy-tailed size distribution!)

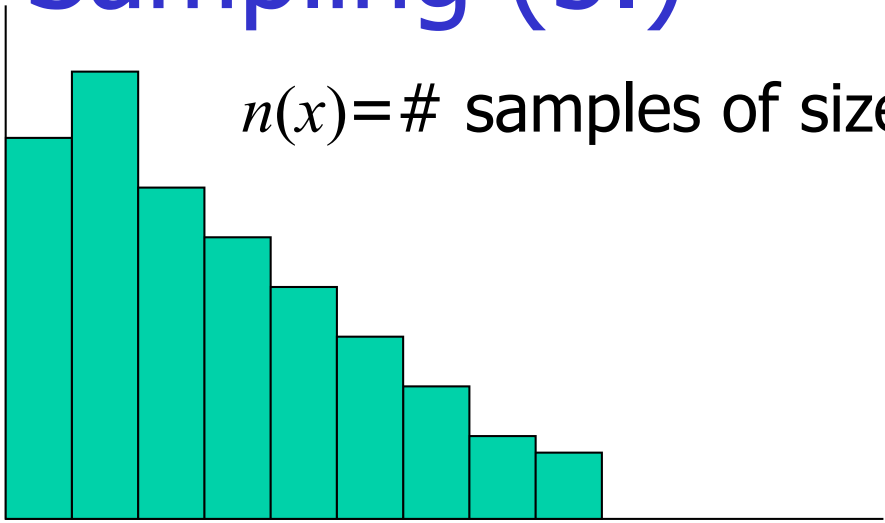


$p(x)$ constant

$p(x)$ increasing

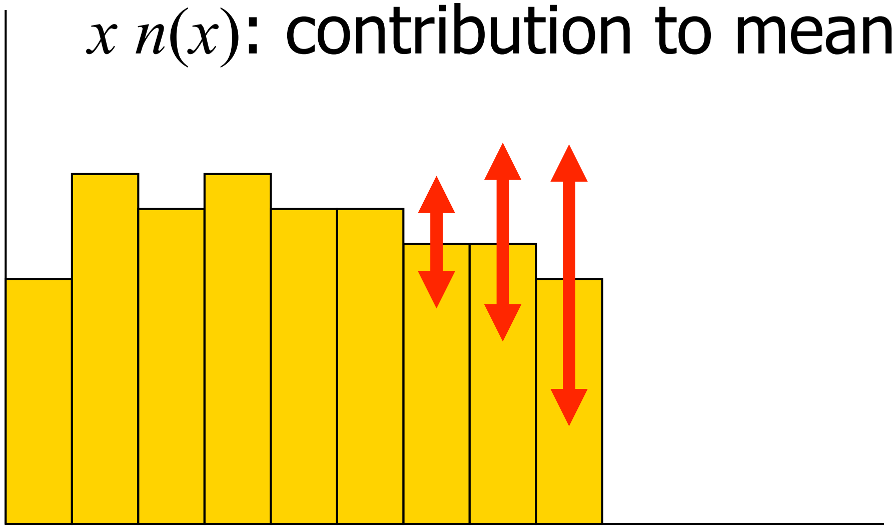
Sampling (5.)

Object size distribution



Estimated mean :

$$\hat{\mu} = \frac{1}{n} \sum_x x \cdot n(x)$$



Variance mainly due to large x

Better estimator: reduce variance
by increasing # samples of large objects

Basic Properties

	Filtering	Aggregation	Sampling
Precision	exact	exact	approximate
Generality	constrained a-priori	constrained a-priori	general
Local Processing	filter criterion for every object	table update for every object	only sampling decision
Local memory	none	one bin per value of interest	none
Compression	depends on data	depends on data	controlled

Combinations

- In practice, rich set of combinations of filtering, aggregation, sampling
- Examples:
  - Filter traffic of a particular type (ACLs), then sample packets, e.g. Netflow
  - Sample packets, then filter
  - Aggregate packets between different source-destination pairs (e.g. subnet), sample resulting records
  - When sampling a packet, sample also the next k packets, compute some aggregate metric over these k packets
  - ... etc.

Tools

- *netstat*
  Good for quick monitoring
- *nettop*
  Good for quick ongoing monitoring
- *tcpdump*
  - Good for in depth details
  - Basis for wireshark
- *wireshark*
  - Good for visual inspection of in depth details
- *Bro*
  - Good for in depth scripted analysis
  - Security analysis
  - Application analysis

Netstat

$ netstat

Monitors the activity of connections

$ netstat –an

gives and overview of all the active connections

$ netstat -i tcp

summary of TCP connections

Nettop

$ nettop
overview

$ nettop -t wifi
   overview of wifi activity

$ nettop –m tcp
   overview of TCP connections

tcpdump

```
TCPDUMP(8)                          System Manager's Manual                          TCPDUMP(8)

NAME
       tcpdump - dump traffic on a network

SYNOPSIS
       tcpdump [ -AbdDefhHIJKlLnNOpqStuUvxX# ] [ -B buffer_size ]
               [ -c count ]
               [ -C file_size ] [ -G rotate_seconds ] [ -F file ]
               [ -i interface ] [ -j tstamp_type ] [ -m module ] [ -M secret ]
               [ --number ] [ -Q in|out|inout ]
               [ -r file ] [ -V file ] [ -s snaplen ] [ -T type ] [ -w file ]
               [ -W filecount ]
               [ -E spi@ipaddr algo:secret,...  ]
               [ -y datalinktype ] [ -z postrotate-command ] [ -Z user ]
               [ --time-stamp-precision=tstamp_precision ]
               [ --immediate-mode ] [ --version ]
               [ expression ]

DESCRIPTION
       Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression; the description is pre-
       ceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight.  It can also  be  run  with
       the  -w  flag,  which causes it to save the packet data to a file for later analysis, and/or with the -r flag, which causes it to read from a
       saved packet file rather than to read packets from a network interface.  It can also be run with the -V flag, which causes it to read a  list
       of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.

       Tcpdump  will, if not run with the -c flag, continue capturing packets until it is interrupted by a SIGINT signal (generated, for example, by
       typing your interrupt character, typically control-C) or a SIGTERM signal (typically generated with the kill(1) command); if run with the  -c
       flag, it will capture packets until it is interrupted by a SIGINT or SIGTERM signal or the specified number of packets have been processed.
```

Tcpdump output
(three-way TCP handshake and HTTP request message)

timestamp

client address and port #

Web server (port 80)

```
23:40:21.008043 eth0 > 135.207.38.125.1043 > lovelace.acm.org.www: S
617756405:617756405(0) win 32120 <mss 1460,sackOK,timestamp 46339
0,nop,wscale 0> (DF)
```

sequence number

TCP options

SYN flag

```
23:40:21.036758 eth0 < lovelace.acm.org.www > 135.207.38.125.1043: S
2598794605:2598794605(0) ack 617756406 win 16384 <mss 512>

23:40:21.036789 eth0 > 135.207.38.125.1043 > lovelace.acm.org.www: .
1:1(0) ack 1 win 32120 (DF)

23:40:21.037372 eth0 > 135.207.38.125.1043 > lovelace.acm.org.www: P
1:513(512) ack 1 win 32256 (DF)

23:40:21.085106 eth0 < lovelace.acm.org.www > 135.207.38.125.1043: .
1:1(0) ack 513 win 16384

23:40:21.085140 eth0 > 135.207.38.125.1043 > lovelace.acm.org.www: P
513:676(163) ack 1 win 32256 (DF)

23:40:21.124835 eth0 < lovelace.acm.org.www > 135.207.38.125.1043: P
1:179(178) ack 676 win 16384
```

wireshark – tshark

- Network packet analyzer for Unix/Windows
  - Displays detailed packet stats
  - GUI (wireshark) or command-line (tshark)
- Intended audience:
  - Network admins (troubleshooting)
  - Security engineers (security problems)
  - Developers (debugging protocols)
- What it is not:
  - Not for large packet traces:
    - Not for high-speed links
    - Out of memory => crash!

Wireshark

```
Usage: wireshark [options] ... [ <infile> ]

Capture interface:
  -i <interface>           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>             packet snapshot length (def: appropriate maximum)
  -p                       don't capture in promiscuous mode
  -k                       start capturing immediately (def: do nothing)
  -S                       update packet display when new packets are captured
  -l                       turn on automatic scrolling while -S is in use
  -I                       capture in monitor mode, if available
  -B <buffer size>         size of kernel buffer (def: 2MB)
  -y <link type>           link layer type (def: first appropriate)
  --time-stamp-type <type> timestamp method for interface
  -D                       print list of interfaces and exit
  -L                       print list of link-layer types of iface and exit
  --list-time-stamp-types  print list of timestamp types for iface and exit

Capture stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ...  duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                           files:NUM – stop after NUM files
```

tshark

```
Usage: tshark [options] ...

Capture interface:
  -i <interface>          name or idx of interface (def: first non-loopback)
  -f <capture filter>     packet filter in libpcap filter syntax
  -s <snaplen>            packet snapshot length (def: 65535)
  -p                      don't capture in promiscuous mode
  -D                      print list of interfaces and exit
  -L                      print list of link-layer types of iface and exit
  -r <infile>             set the filename to read from (no pipes or stdin!)

Processing:
  -R <read filter>        packet filter in Wireshark display filter syntax
  -n                      disable all name resolutions (def: all enabled)
  -d <layer_type>==<selector>,<decode_as_protocol> ...
                          "Decode As", see the man page for details
                          Example: tcp.port==8888,http
-z <statistics>         various statistics, see the man page for details

Miscellaneous:
  -h                      display this help and exit
  -v                      display version info and exit
```

Capturing with tshark

- Capturing the trace:
  - `tshark -i en0 -w trace-1.pcap`
- Capturing the trace for 10 seconds:
  - `> tshark -i en0 -w trace-1.pcap -a duration:10`

Basic stats with tshark

- Protocol summary of the trace:

```
> tshark -z io,phs -r trace-1.pcap
```

- All traffic from/to a host every minute:

```
> tshark -z io,stat,60,ip.addr==xxx -r trace-1.pcap
```

- All TCP conversations of the trace:

```
> tshark -z conv,tcp -r trace-1.pcap
```

- All Telnet conversations of the trace:

```
> tshark -z conv,tcp,telnet -r trace-1.pcap
```

- All UDP conversations of the trace:

```
> tshark -z conv,udp -r trace-1.pcap
```

- All ICMP conversations of the trace:

```
> tshark -z conv,tcp -r trace-1.pcap -R ‘icmp’
```

Basic stats with wireshark

- General summary of the trace
- Protocol hierarchy stats
  - IP-level protocols
  - Transport protocols
  - ARP
  - ICMP
- „Conversations"
  - Follow a telnet session
  - Follow a DNS flow
  - Check IGMP messages
- Endpoints
  - Heavy-hitters
  - Low-hitters (scans)
- Packet size distribution

The Bro system

- Real-time network analysis framework
  - Unix-based network intrusion detection system
  - Misused for traffic analysis
- Emphasis on
  - Application-level semantics
    - Manipulating packets is uncommon/painful, e.g., wireshark
  - Tracking information over time
    - Within and across flows
    - Archiving for post-mortem analysis
  - Scalability, i.e. Gbit/second links

The Bro system (2)

- Analyzing data means programming the analysis
  - No specification
  - No magic in Bro: The user has to specify what has to be detected
- Programming the analysis ~ behavioral analysis
  - No good/evil
  - But matched/unmatched

Connection summaries

- One line summary for all connections
- Basic, but saves a lot of time

```
> bro -r trace-1.pcap tcp (output in conn.log)
```

Time	Duration	Source	Destination	Service
964953011	0.063756	10.20.12.187	207.126.127.69	http

SrcPort	DstPort	Proto	SrcBytes	DstBytes	State
9002	80	tcp	0	?	RSTR X

- Try for UDP and ICMP

Connection summaries (2)

- Connection states

SF	Normal establishment and termination
REJ	Connection attempt rejected
S0	Connection attempt seen, no reply
OTH	No SYN seen, partial connection
RSTO	Connection established, originator aborted
…	…

Connection summaries (3)

- Fraction of connections with a given state?
  - TCP: SF, REJ, S0, OTH, RSTO
  - UDP: SF, REJ, S0, OTH, RSTO
  - ICMP: OTH

Weird activity

- Network traffic contains lots of weirdoes
  - Activity which does not conform to standard but is not an attack
  - Example: data being sent after RST

```
> bro -r trace-1.pcap weird
```

- Scans

```
> bro -r trace-1.pcap scan
```

Protocol analyzer

- Protocol-specific analysis
  - Log activity
  - Check for protocol-specific attacks
- Bro ships with analyzers for many protocols:
  - FTP, HTTP, POP3, IRC, SSL, DNS, NTP, …
- Example: FTP analyzer

```
> bro -r trace-1.pcap ftp
> cat ftp.log
```

Packet filter

- Bro analyzes only the packets required by scripts´ analysis
  - Builds dynamically packet filter
- Seeing packet filter:

```
> bro tcp ftp smtp print-filter
```

- Packet filter can be changed
- Bro skips whatever traffic does not match filters!

Dynamic protocol detection

- How does Bro know the analyzer for a connection?
- Default mechanism: examine the ports
- Problem: well-known ports are unreliable
- Bro can analyze protocols independent of ports
  - Dynamic protocol detection
  - Current support for HTTP, IRC, SMTP, SSH, FTP, POP3, BITTORRENT
  - Identifies potential protocol usage with signatures and then validates by parsing

```
> bro -r trace-1.pcap -f "tcp" http-request http-
  reply dpd

> bro -r trace-1.pcap -f "tcp" http-request http-
  reply brolite
```