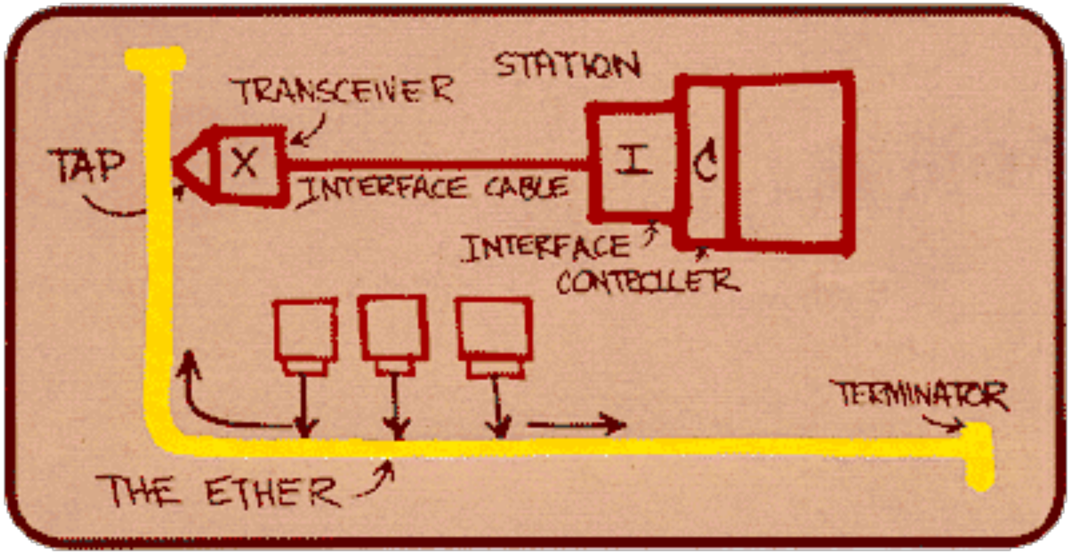
Network Monitoring in Practice

Luca Deri <deri@ntop.org>

Ethernet Switching

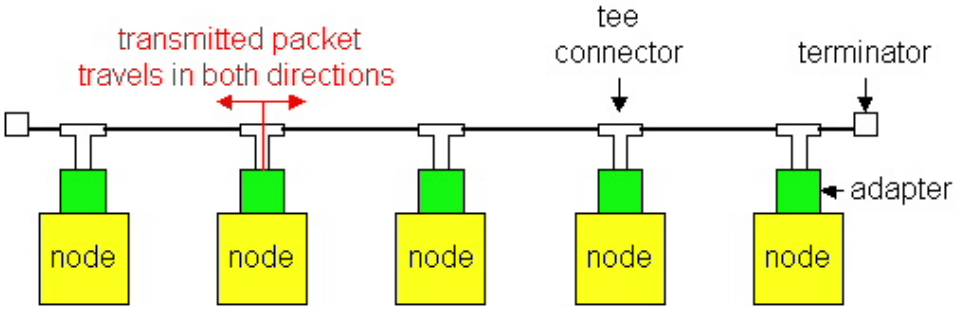
Ethernet [1/3]

- Simple, cheap

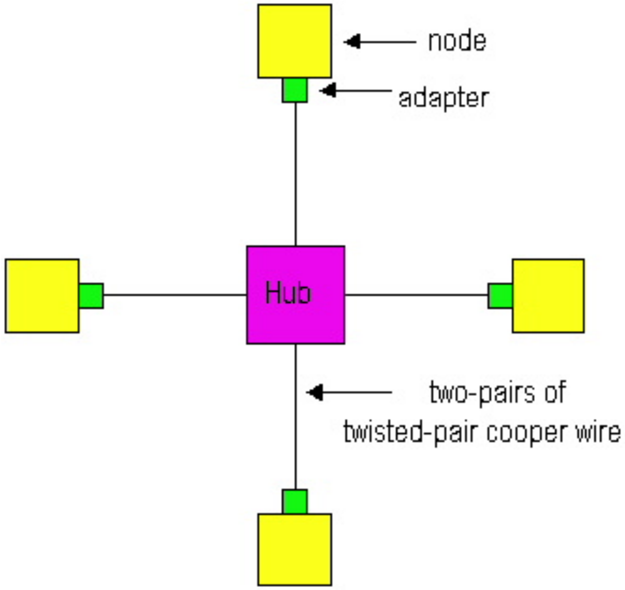


Bob Metcalfe, mid 1970s

Ethernet [2/3]



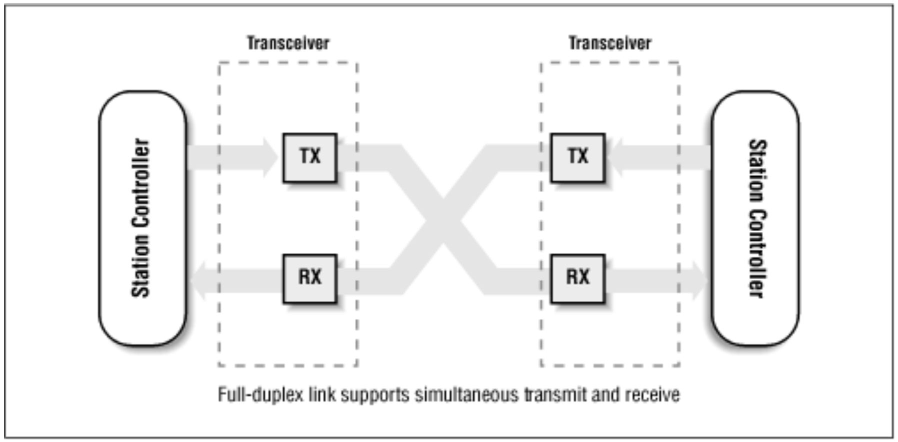
Bus Topology
(10Base2- 802.3a-1988)



Star topology
(10/100BaseT - 802.3i-1990)

Ethernet [3/3]

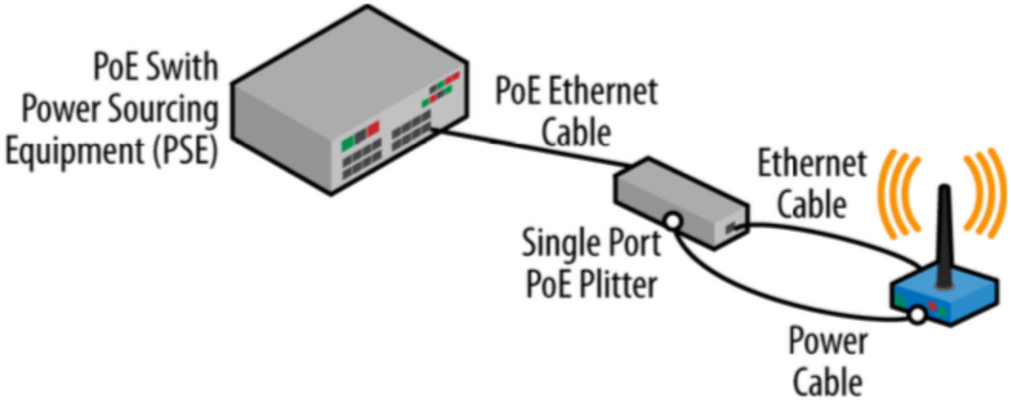
- Ethernet can operate in half-duplex (historical) and full duplex as specified in 802.3x (modern ethernet).
- In full duplex mode, stations can <u>simultaneously</u> both transmit <u>AND</u> receive since there is no contention for using the shared media.



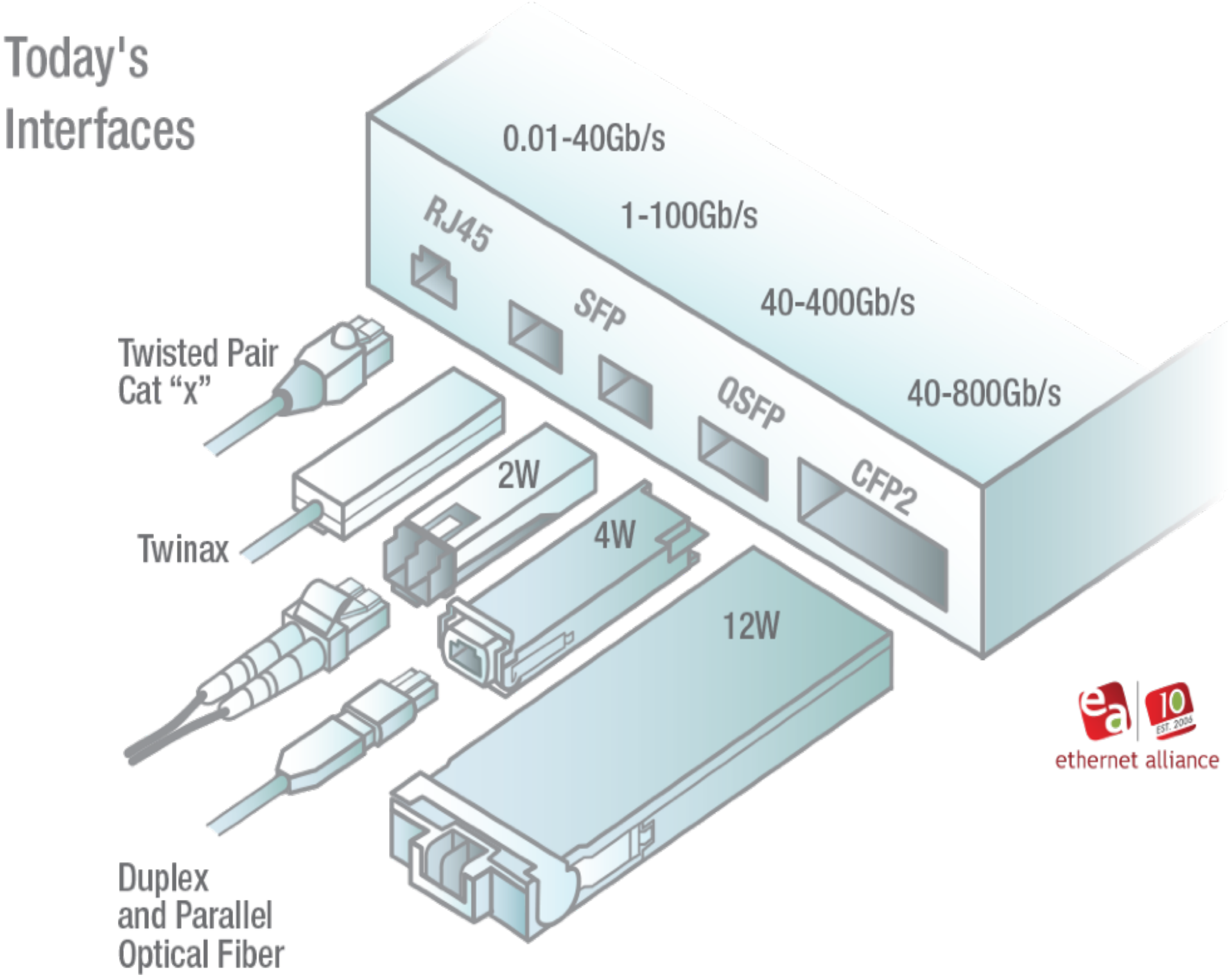
Full-duplex link supports simultaneous transmit and receive

PoE (Power over Ethernet)

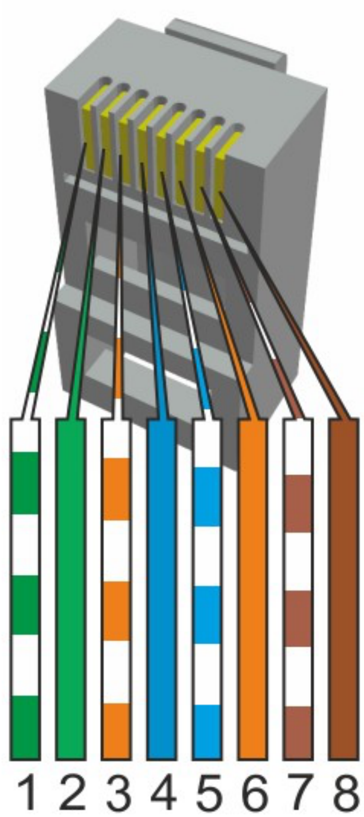
- Elegant solution for merging data and power used mostly for low power devices such as antennas and WiFi access points.
- PoE Type 2: 25.5 W, 600 mA.

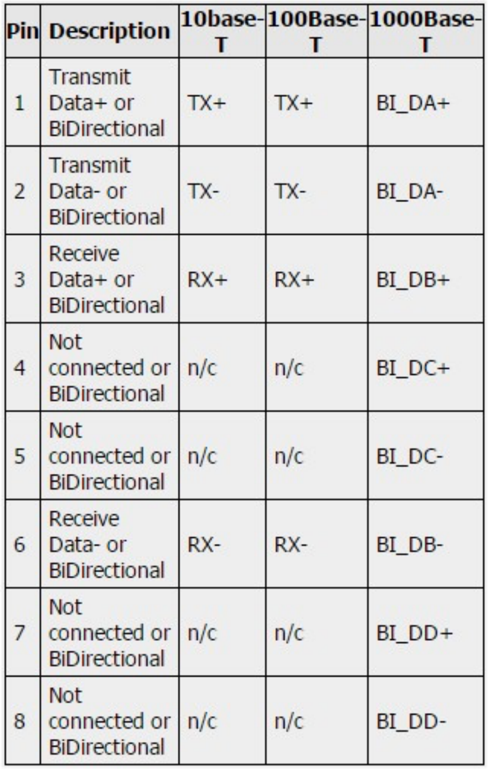


Ethernet Pinout [1/3]

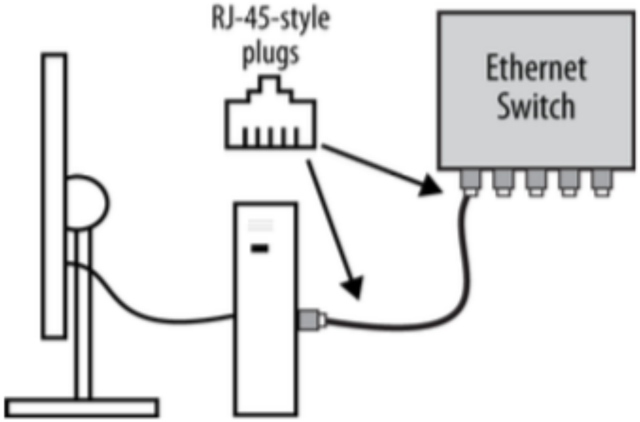


Ethernet Pinout [2/3]

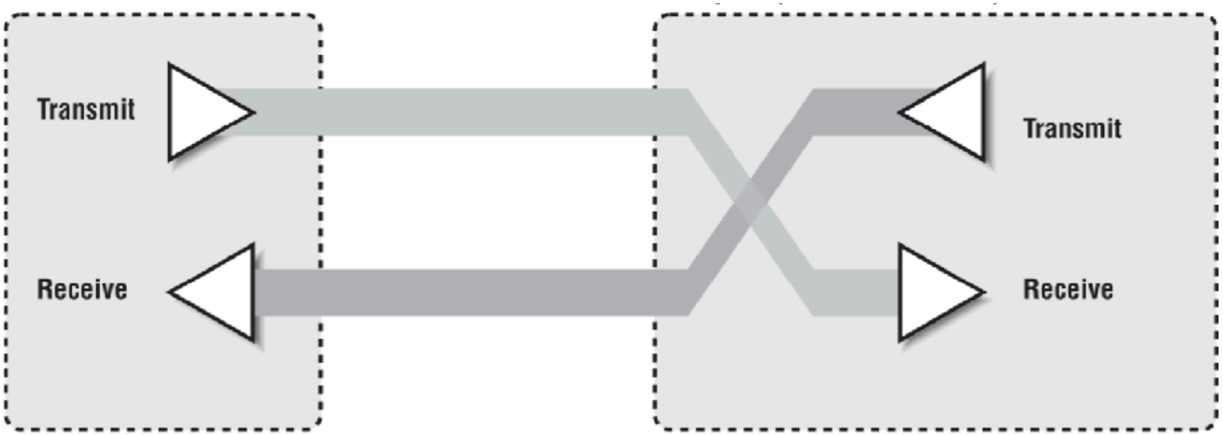




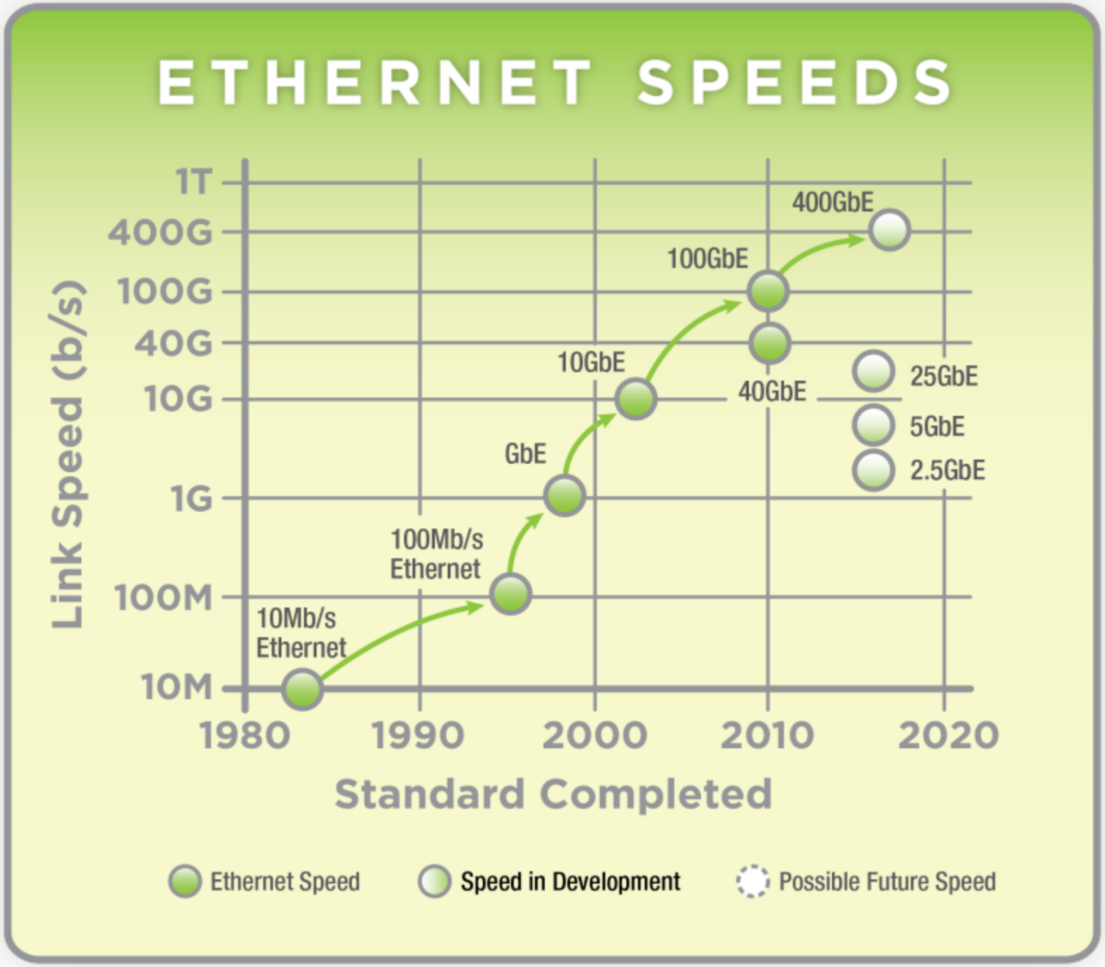
Pin	Description	10base-T	100Base-T	1000Base-T
1	Transmit Data+ or BiDirectional	TX+	TX+	BI_DA+
2	Transmit Data- or BiDirectional	TX-	TX-	BI_DA-
3	Receive Data+ or BiDirectional	RX+	RX+	BI_DB+
4	Not connected or BiDirectional	n/c	n/c	BI_DC+
5	Not connected or BiDirectional	n/c	n/c	BI_DC-
6	Receive Data- or BiDirectional	RX-	RX-	BI_DB-
7	Not connected or BiDirectional	n/c	n/c	BI_DD+
8	Not connected or BiDirectional	n/c	n/c	BI_DD-



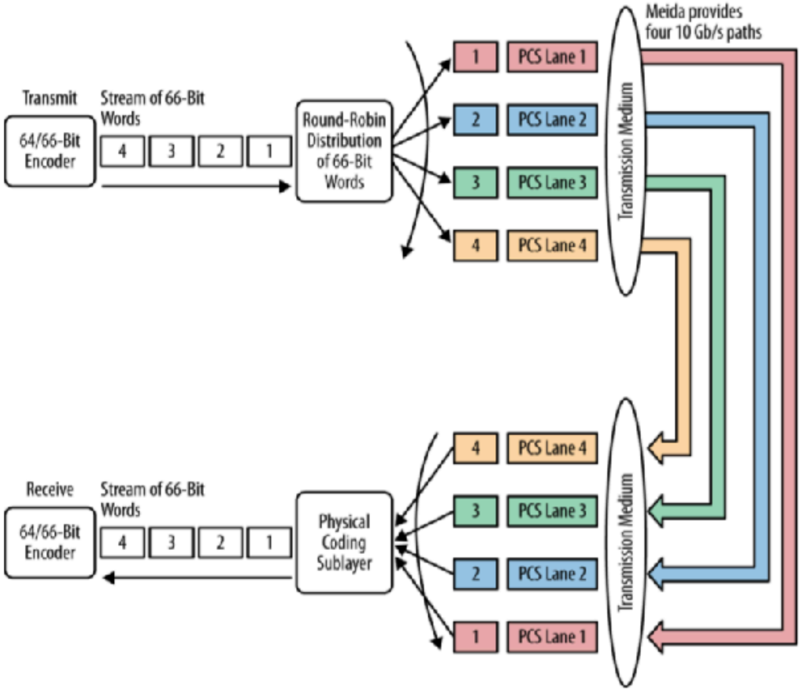
Ethernet Pinout [3/3]

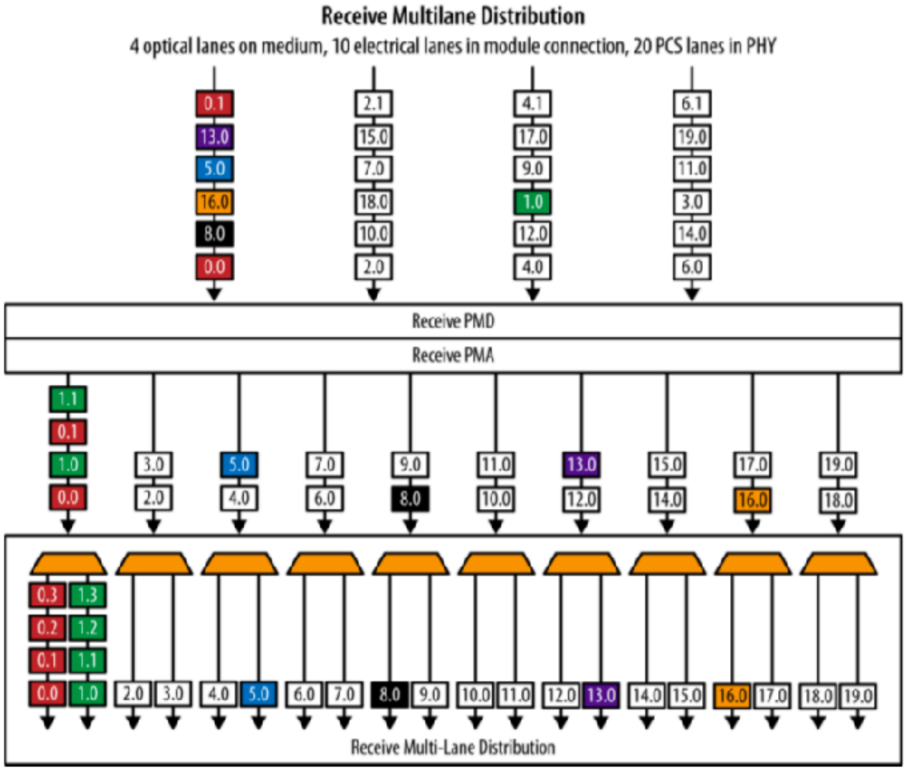


Ethernet Speed



Ethernet Aggregation



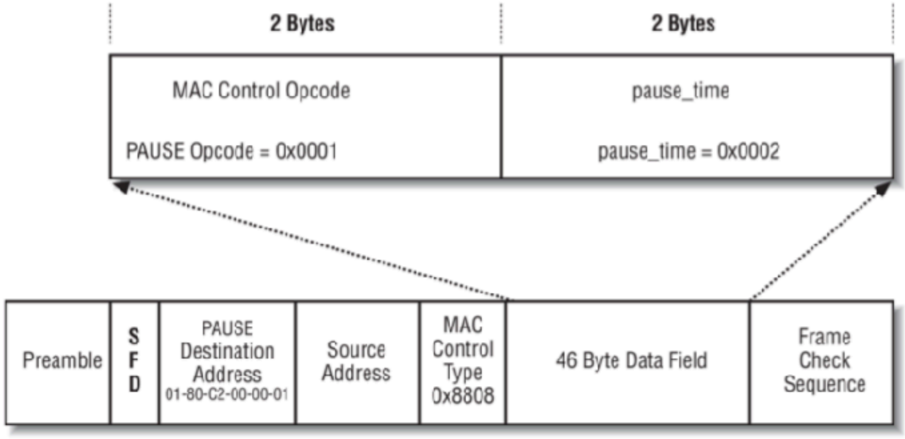


40 Gbit Ethernet

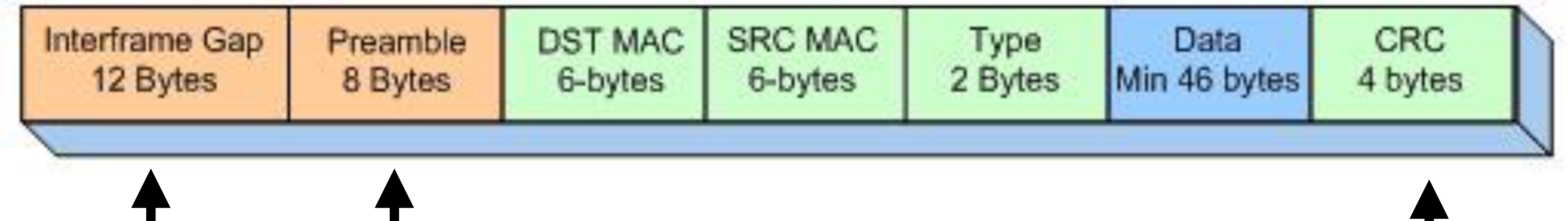
100 Gbit Ethernet

Ethernet Flow Control

Flow control is a mechanism used by the receiver (receiver is the king!) to request the sender a short pause in frame transmission sending a PAUSE command the destination multicast address 01:80:C2:00:00:01.



Ethernet Framing [1/3]



Frame check sequence based on CRC (Cyclic Redundancy Check) use to detect (but not to fix) transmission errors.

Sequence of 56 alternating 0 / 1 bits followed 8 1 bits, allowing the sender to sync with the receiver.

Minimum pause (time) between packets permitting the receiver to prepare for the next frame (96 nsec on Gbit Ethernet)

Ethernet Framing [2/3]

Minimum ethernet packet size is 60 bytes: shorter frames are padded.

IFG	12
Preamble	8
Min Ethernet Frame	60
CRC	4
Total (bytes)	84

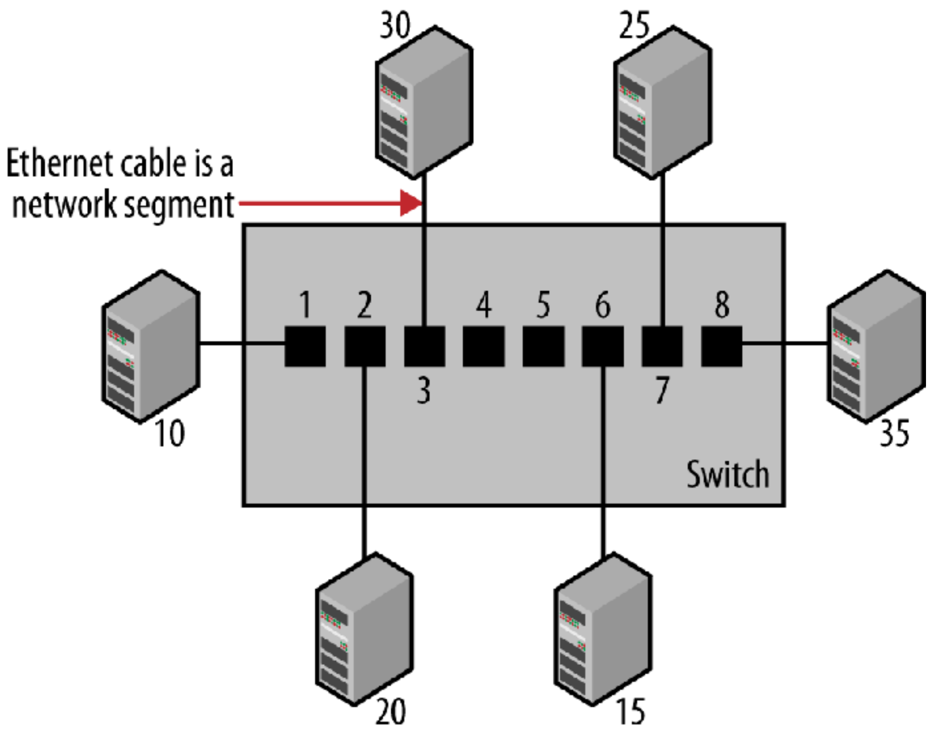
Ethernet Framing [3/3]

Speed	bits/sec	bytes/sec	PPS (bps/84)	Packet Rate (1/PPS) nsec
1 Gbps	1.000.000.000	125.000.000	1.488.095	672,00
10 Gbps	10.000.000.000	1.250.000.000	14.880.952	67,20
100 Gbps	100.000.000.000	12.500.000.000	148.809.524	6,72

Operation of Ethernet Switches [1/6]

- Ethernet switches are “invisible” devices to the network as they do not modify Ethernet frames bridged across ports.
- Switches do not require any configuration as they learn the network topology and configuration by analysing network traffic as described in the IEEE 802.1D bridging standard.

Operation of Ethernet Switches [2/6]



Port	Station
1	10
2	20
3	30
4	No station
5	No station
6	15
7	25
8	35

Forwarding Table

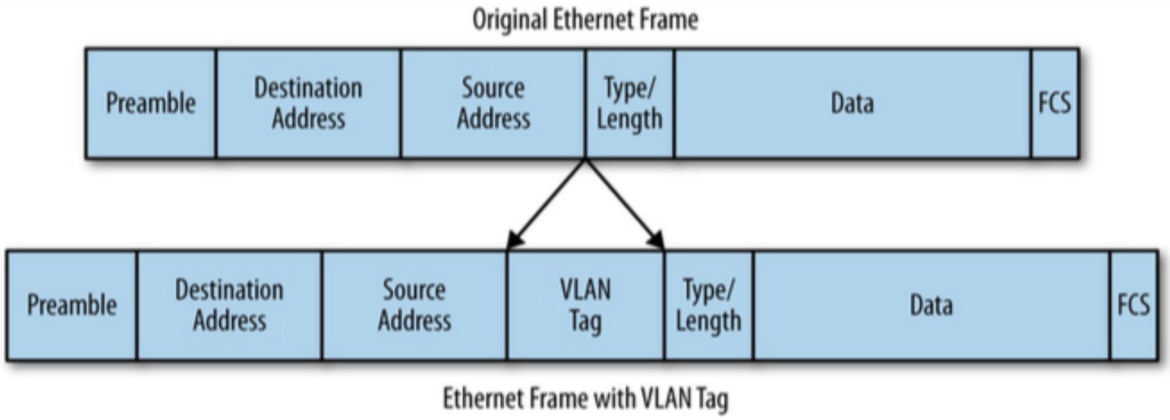
Operation of Ethernet Switches [3/6]

- Stations advertise their presence by sending ethernet frames.
- The forwarding table is filled up dynamically and automatically (no configuration) as incoming packets are received.
- As switches can be nested in a tree/mesh architecture, multiple stations can be attached to the same port.

Operation of Ethernet Switches [4/6]

- Traffic is forwarded based on the destination ethernet address:
  - When a frame is received the destination MAC is searched in the forwarding table.
  - If found, the frames is sent to the destination station (layer 2 anycast).
  - If not found the traffic is sent to all ports but the one from which the frame has been received (layer 3 broadcast).

Operation of Ethernet Switches [5/6]

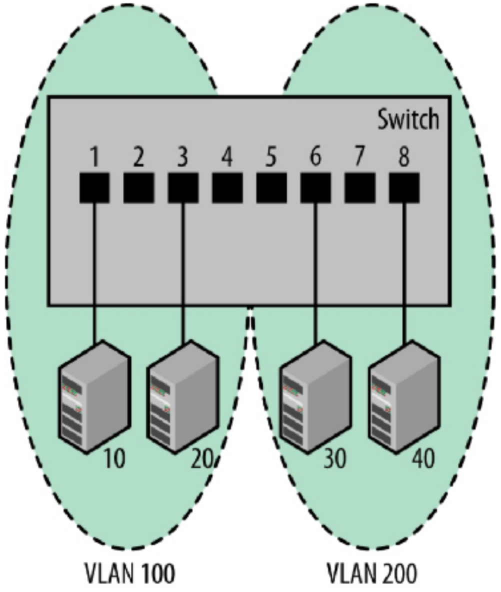


Caveat:

VLANs affect traffic forwarding

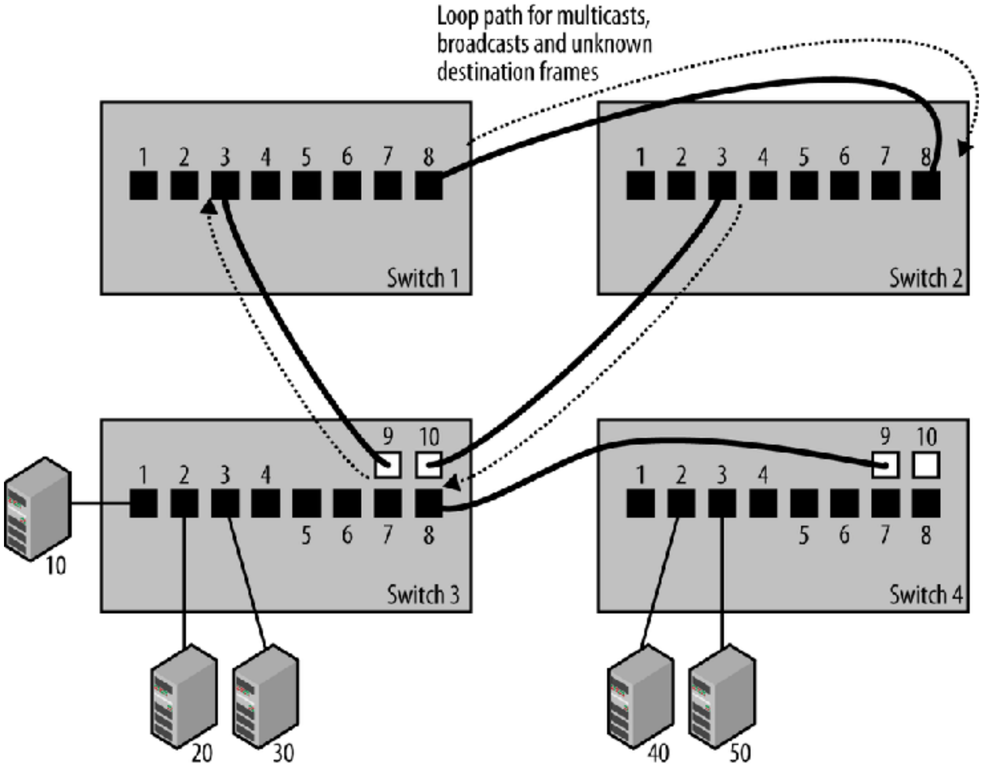
by creating multiple

broadcast domains



Operation of Ethernet Switches [6/6]

- Ethernet requires that only one path exists between any two stations.



Traffic Mirror: Possible Solutions

Hardware:

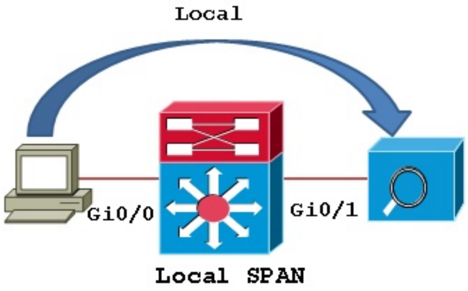
- Hub (Copper Ethernet, Token Ring)
- Optical Splitter (Optical Fibers)
- Tap (Copper/Fibre)

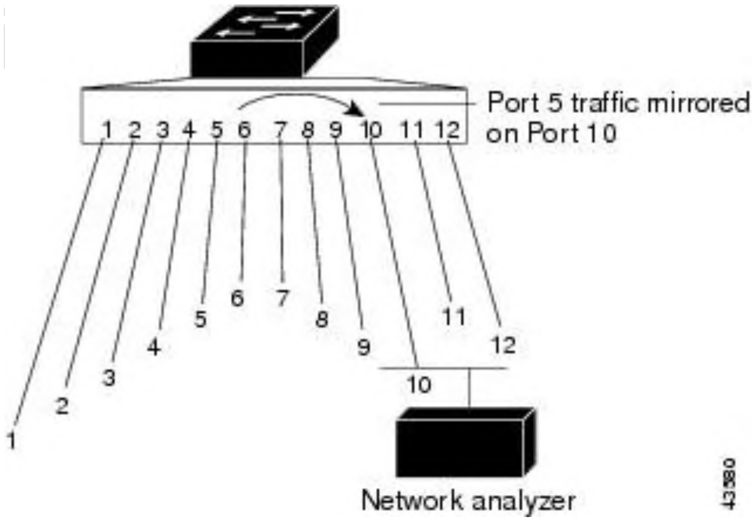
Software:

- Switch Port Mirror (1:1, 1:N)
- Switch VLAN Mirror (N:1)
- Switch Traffic Filter/Mirroring (Juniper)

Traffic Mirror: SPAN, RSPAN, ERSPAN [1/4]

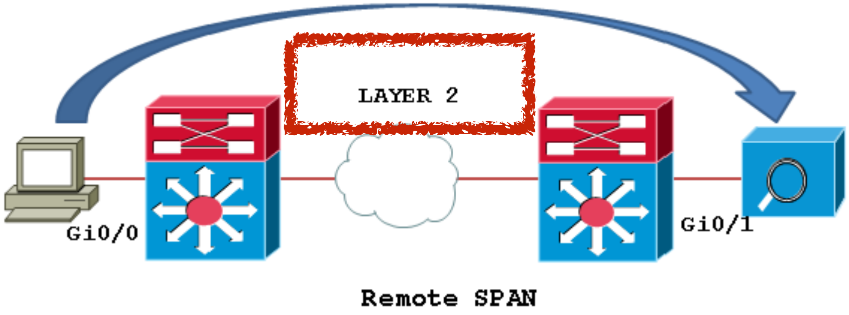
- Sometimes port mirror is named Switched Port Analyzer (SPAN) in the Cisco parlance.
- SPAN happens locally inside one switch (no cross switch mirror

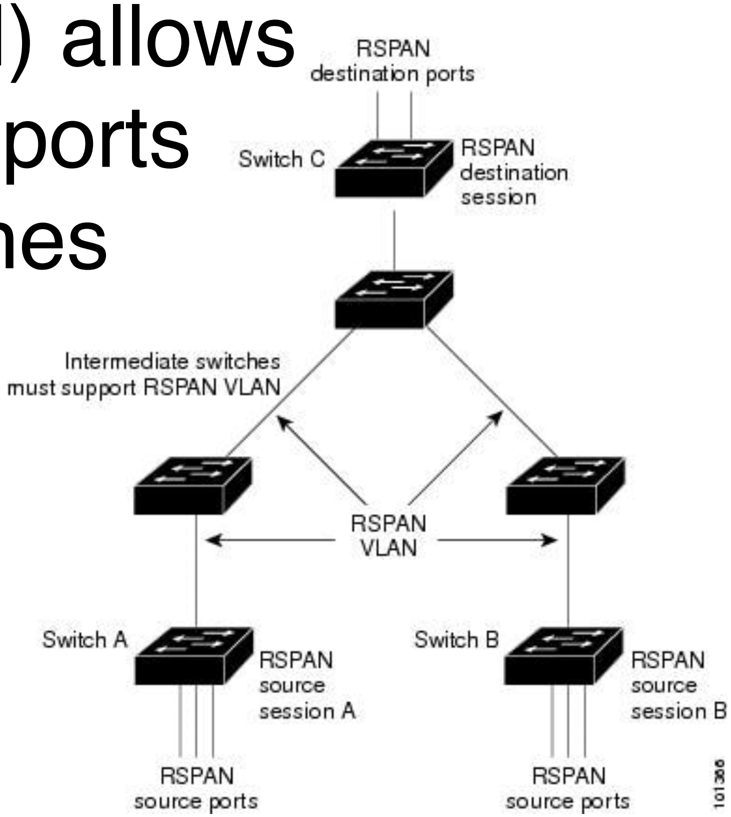




Traffic Mirror: SPAN, RSPAN, ERSPAN [2/4]

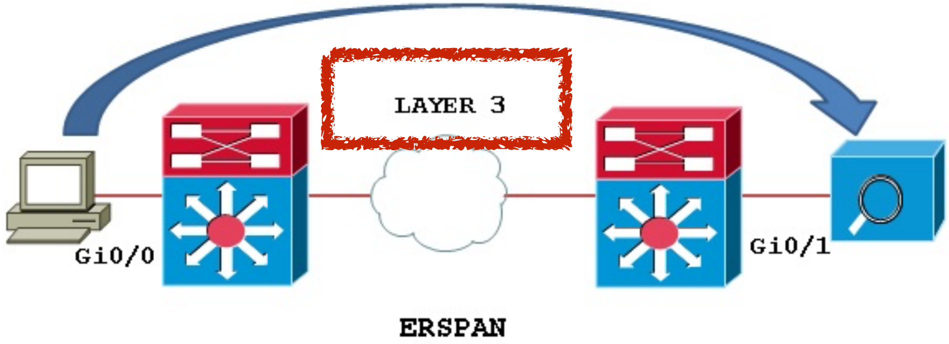
- Remote SPAN (RSPAN) allows source and destination ports to be on different switches transporting traffic on VLANs.





Traffic Mirror: SPAN, RSPAN, ERSPAN [3/4]

- Encapsulated remote SPAN (ERSPAN) allows mirrored traffic to be transport across the Internet via Generic Route Encapsulation (GRE) tunnels.



Traffic Mirror: SPAN, RSPAN, ERSPAN [4/4]

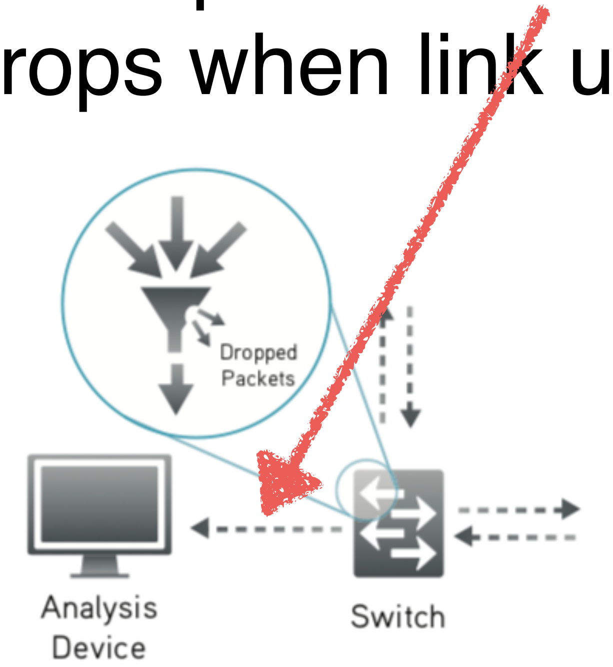
- You can achieve the same result with SSH.

```
ssh user@host "sudo /usr/sbin/
tcpdump -i ethX -U -s0 -w - " |
wireshark -k -i -
```

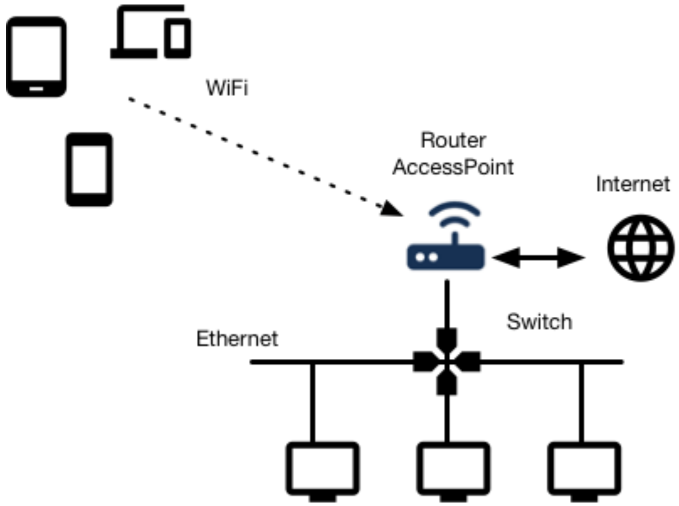
https://blog.packet-foo.com/2016/11/the-network-capture-playbook-part-4-span-port-in-depth/

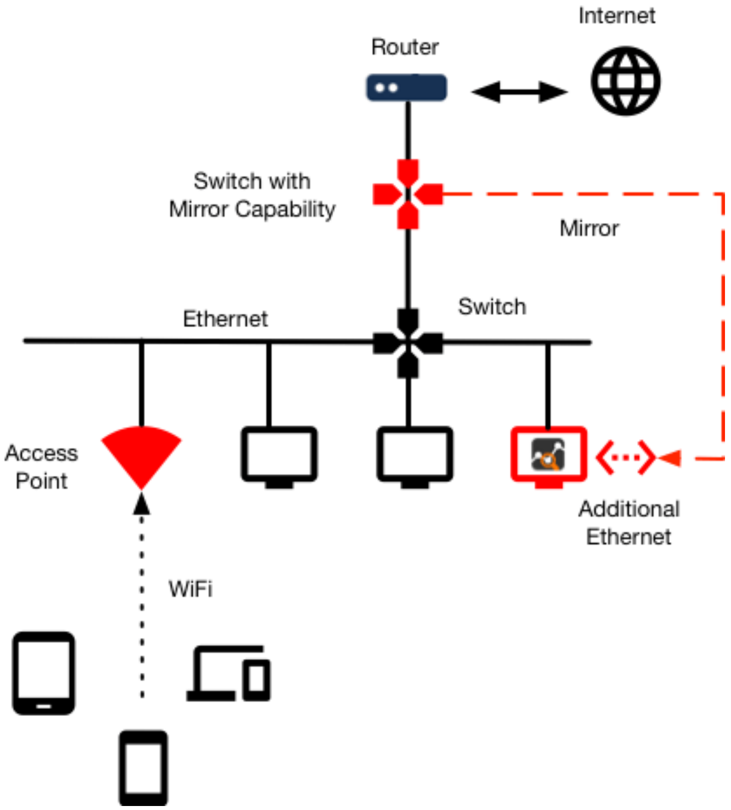
Port Mirror Shortcomings

- It is not guaranteed that the switch won’t drop frames when mirroring traffic.
- RX and TX are copied on a single TX signal resulting in drops when link utilisation goes about 50%.

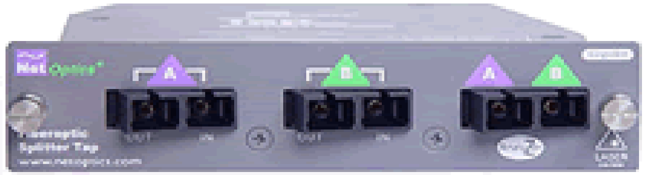


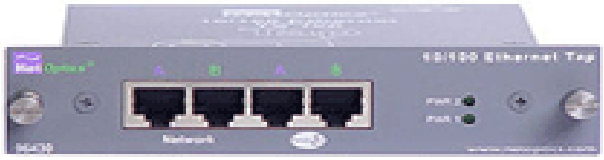
Use Case: Home Traffic Monitoring





Network Taps

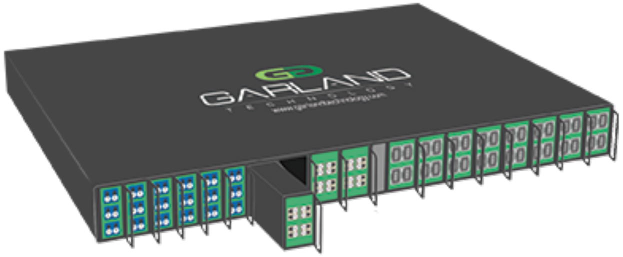




Tap Families

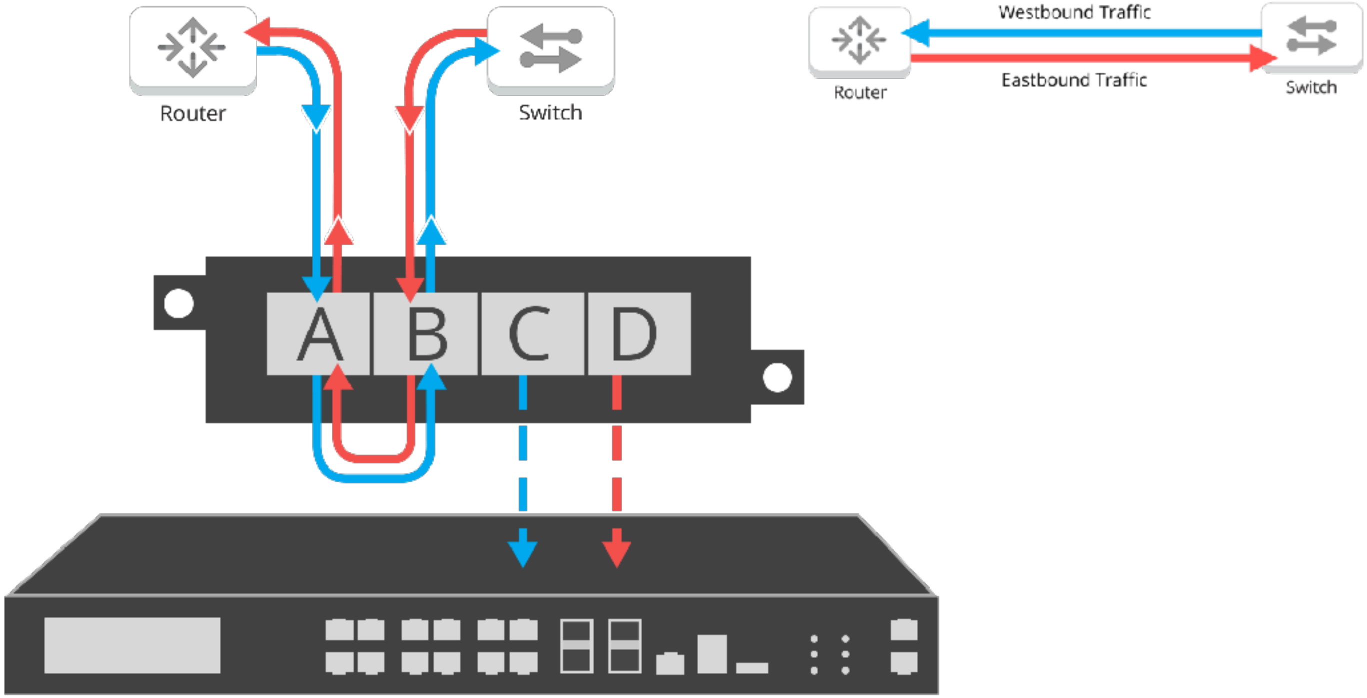
Portable

Modular



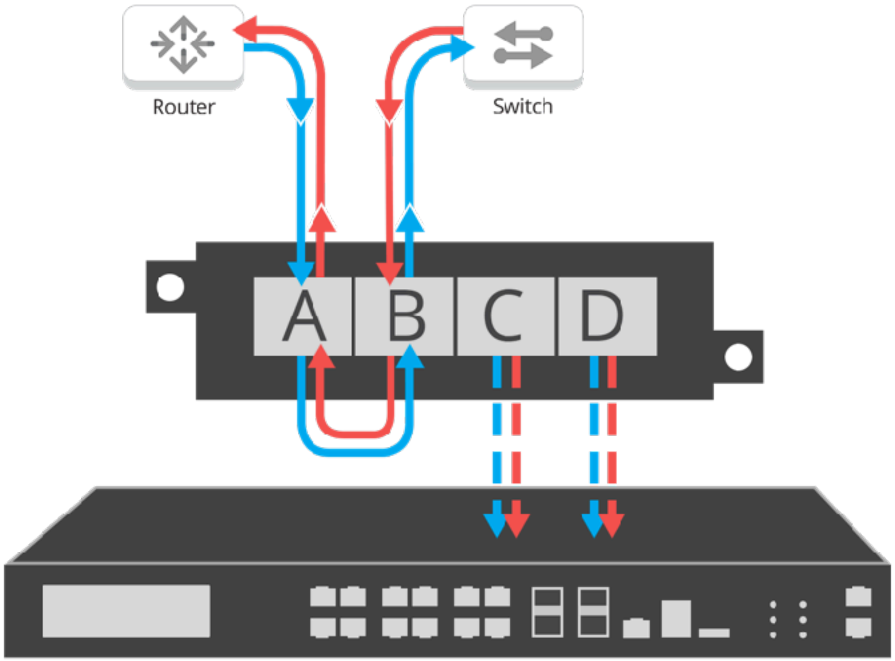
High-Density

Tap: Breakout Mode



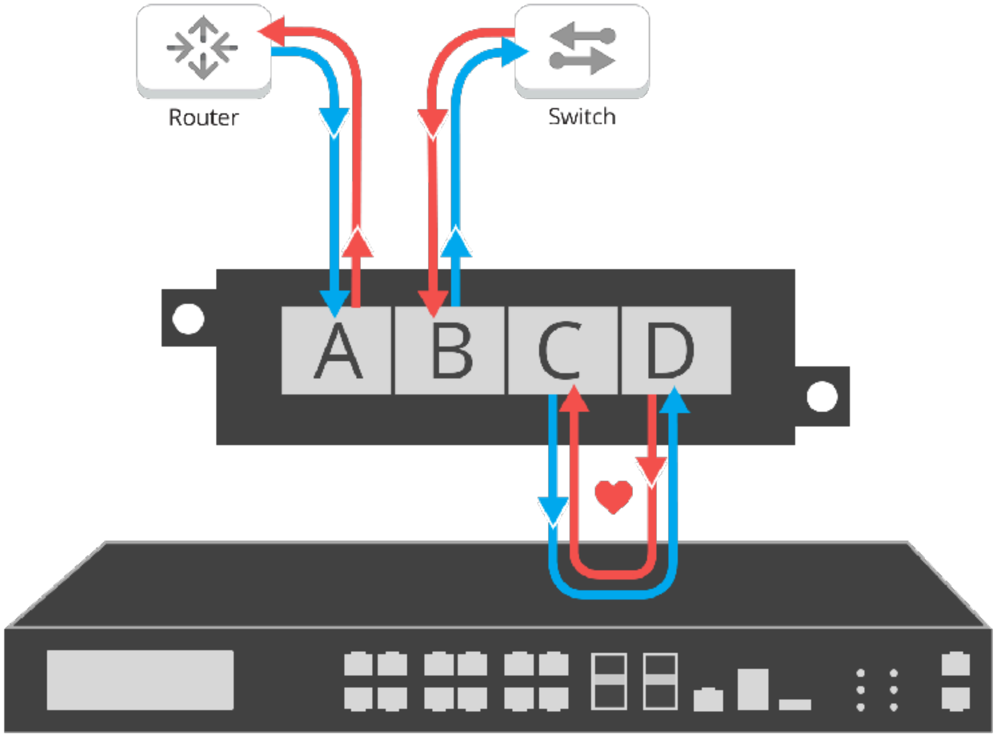
Sends each side of traffic to separate monitoring ports. Ensuring that no packet is lost to high-priority monitoring tools.

Tap: Aggregation Mode



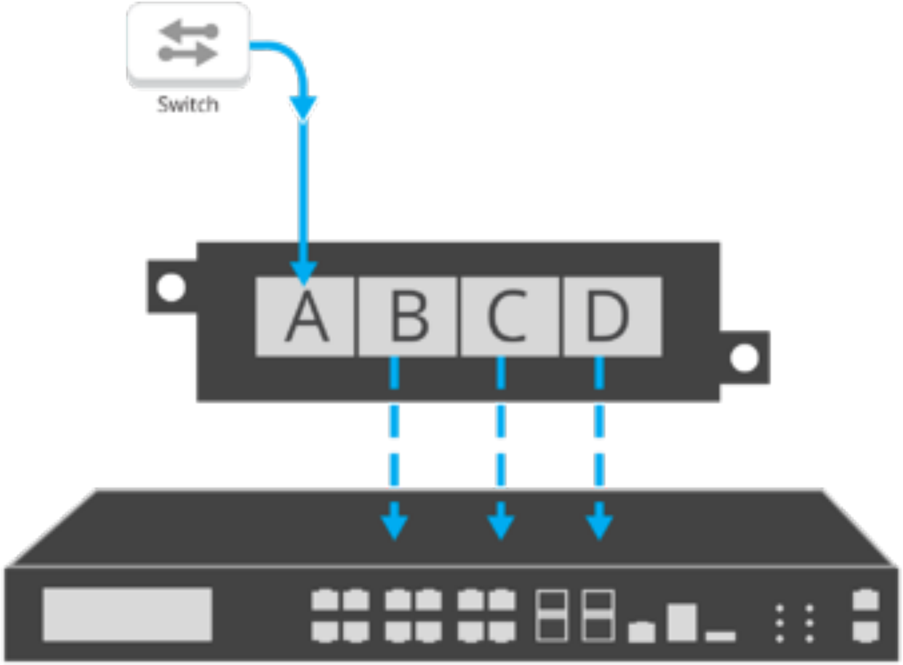
Merge traffic streams into one monitoring port to reduce appliance costs, often used in combination with filtering taps, ie: filter, aggregate data streams.

Tap: Bypass Mode



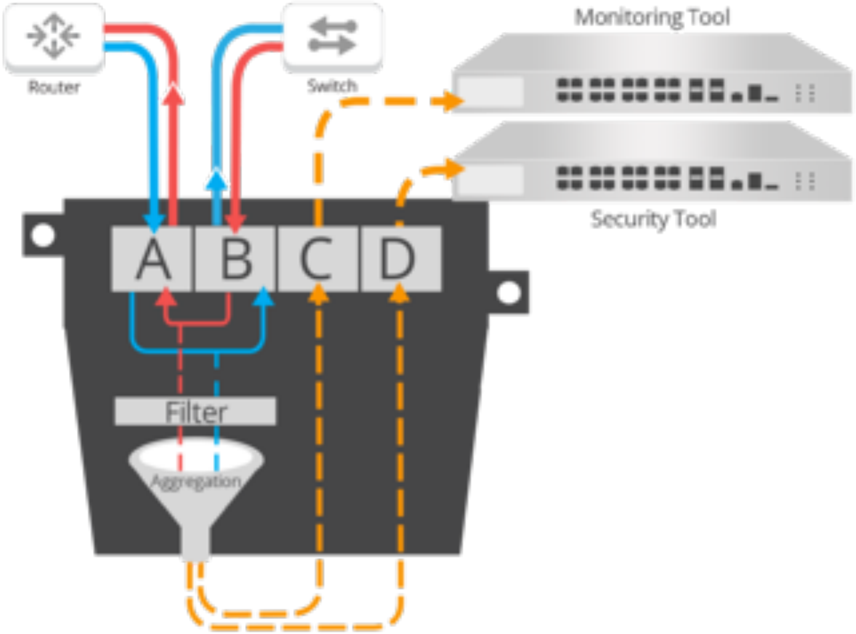
Prevent inline devices from causing network downtime by “bypassing” that device, in the event it fails or needs to be updated.

Regeneration/Span Mode



Create multiple copies of network data to support multiple devices from a single connectivity point.

Filtering



Allow you to set rules on what data is filtered and sent to monitoring or security tools. Filtering prevents ports from becoming oversubscribed.

Tap: Pros and Cons

- Pros
  - Eliminates the risk of dropped packets.
  - Monitoring device receives all packets, including physical errors.
  - Provides full visibility into full-duplex networks.
- Cons
  - Analysis device need dual-receive capture interface
  - Additional cost with purchase of TAP hardware
  - Cannot monitor intra-switch traffic

Introduction to Traffic Monitoring

Monitoring Requirements [1/4]

- Guarantee the availability of the function on the net.
  - Service maintenance (availability, response time) need to face with technological changes and big quota increase.
    - Security of the services through the control of security components.
    - (Human) Mistake prevention and bottleneck identification/ recovery.
  - Automatic or semiautomatic reaction on operation anomalies:
    - Real-time configuration modification in case of error.
    - Activation of redundant components in case of error.

Monitoring Requirements [2/4]

- Dynamic reactions to changes on the network and environment:
  - Changes regarding applications, users, components, services or fees.
  - Dynamic adaptation of the available transmission bandwidth according to requests originated by the management system.

Monitoring Requirements [3/4]

- Network Control:
  - Collection and (compressed) representation of relevant network information.
  - Definition and maintenance of a database of network configurations.
  - When applicable, centralisation of the control over peripherals and implemented functions (central management console).
  - Integration of management procedures on heterogeneous environments

Monitoring Requirements [4/4]

- Improvement of system/network administrators work conditions :
  - Improvement and standardisation of the available tools.
  - Identify and implement gradual automation of management functions.
  - Good integration of tools into the existing operational sequences.
- Progress through standardisation :
  - Transition of existing, often proprietary, solutions in a standardised environment.

Various Actors, Various Metrics

- End-Users vs. (Internet) Service Provider
  - Remote user (Dial-up or xDSL) vs. Facebook
    - Internet User (no services provided)
    - Mostly P2P, Email, WWW traffic
  - ntop.org vs. TIM
    - Provided Services (e.g. DNS, Mail, WWW)
    - Connected to a Regional ISP (no worldwide branches)
  - Sparkle/CenturyLink vs. TIM
    - Need to buy bandwidth for national customers
    - Need to sign SLA with customers influenced by the SLA signed with the global carrier.

End-Users Requirements

- Monitoring of Application performance:
  - Why this web page takes so long to load?
  - Why does the multicast video isn’t smooth?
- Check that the expected SLA can be provided by the available network infrastructure
  - Do I have enough bandwidth and network resources for my needs and expectations?
- Is poor performance “normal” or there’s an ongoing attack or suspicious activity?
  - Is there a virus that takes over most of the available resources?
  - Is anybody downloading large files at high priority (i.e. bandwidth monopolisation)?

Service Provider Requirements

- Monitor SLA (Service Level Agreements) and current network activities.
- Enforce committed SLA and monitor their violation (if any).
- Detection of network problems and faults.
- Redesign the network and its services based on the user feedback and monitoring outcome.
- Produce forecasts for planning future network usage hence implement extensions before it’s too late (digging out ground for laying cables/fibres takes a lot of time).

Problem Statement

- End-users and ISPs speak a different language
  - End-users understand network services
    - Outlook can’t open my mailbox.
    - Firefox isn’t able to connect to Google.
  - ISPs talk about networks
    - BGP announces contain wrong data.
    - The main Internet connection is 90% full.
    - We need to sign a peering contract with AS XYZ for

Measurements

Traffic Analysis Applications: Some Requirements [1/2]

- What: Volume and rate measurements by application host and conversation.
  Why: Identify growth and abnormal occurrences in the network.
- What: Customisable grouping of traffic by logic groups (e.g. company, class of users), geography (e.g. region), subnet.
  Why: Associate traffic with business entities and trend growth per grouping (aggregate data isn't very meaningful here: we need to drill-down the analysis at user level).

Traffic Analysis Applications: Some Requirements [2/2]

- What: Customisable filters and exceptions based on network traffic.
  Why: Filters can be associated with alarm notifications in the event of abnormal occurrences on the network.
- What: Customisable time-periods to support workday reporting.
  Why: Analyzing data based on the calendar helps identifying problems (e.g. the DHCP is running out of addresses every Monday morning between 9-10 AM, but the problem disappears for the rest of the week.)

Further Measurement Issues [1/2]

- Network appliances have very limited measurement capabilities (a router must switch packets first!).
  - Limited to few selected protocols
  - Aggregated measurements (e.g. per interface)
  - Only a few selected boxes can be used for network measurements (e.g. a router is too loaded for new tasks, this L2 switch isn’t SNMP manageable)
  - High-speed networks introduce new problems: measurement tools can’t cope with high speeds.

Further Measurement Issues [2/2]

- Need to constantly develop new services and applications (e.g. mobile video on 4/5G phones).
- Most of the services have not been designed to be monitored.
- Most of the internet traffic is consumed by applications (P2P) that are designed to make them difficult to detect and account.
- Modern internet services are:
  - Mobile hence not tight to a location and IP address
  - Encrypted and based on dynamic TCP/UDP ports (no fingerprinting, i.e. 1:1 port to service mapping)

Monitoring Capabilities in Network Equipment

- End-systems (e.g. Windows PC)
  - Completely under user control.
  - Simple instrumentation (just install new apps)

- Standard Network boxes (e.g. ADSL Router)
  - Access limited to network operators
  - Poor set of measurement capabilities
  - Only aggregated data (e.g. per interface)

- Custom Boxes (Measurement Gears)
  - Instrument-able for collecting specific data.
  - Issues in physical deployment so that they can analyse the traffic where it really flows

Problem Statement

- Users demand <u>services</u> measurements.
- Network boxes provide simple, aggregated <u>network</u> measurements.
- You cannot always install the measurement box wherever you want (cabling problems, privacy issues).
- New protocols appear every month, measurement protocols are very static and slow to evolve.

Benchmarking Terminology

- Traffic metrics are often not standardised contrary to everyday life metrics (kg, litre etc.).
- Vendors measurements are often performed in slightly different ways making the results not easy to compare.
- RFC 1242 “Benchmarking Terminology for Network Interconnection Devices” defines some common metrics used in traffic

RFC 1242: Some Definitions

- Throughput
- Latency
- Frame Loss Rate
- Datalink Frame Size
- Back-to-back
- Etc.

Very general RFC, not very “precise/formal”.

RFC 2285: Further Definitions

- “Benchmarking Terminology for LAN Switching Devices”
- It extends the 1242 RFC by adding new definitions that will be used in other RFCs, including:
  - Traffic burst
  - Network load/overload
  - Forwarding rate
  - Errored frames
  - Broadcasts

RFC 2432: Multicast Terminology

- “Terminology for IP Multicast Benchmarking”
- Very peculiar RFC targeting multicast traffic measurement.
- Some metrics:
  - Forwarding and Throughput (e.g. Aggregated Multicast Throughput)
  - Overhead (e.g. Group Join/Leave Delay)

RFC 1944: Benchmarking Methodology for Network Interconnect Devices

- It defines how to perform network traffic measurements:
  - Testbed architecture (where to place the system under test)
  - Packet sizes used for measurements
  - IP address to assigned to SUT (System Under Test)
  - IP protocols used for testing (e.g. UDP vs TCP)
  - Use of traffic bursts during measurements (burst vs. constant traffic)
- In a nutshell it defines the test environment to be used for network traffic testing.

Other Benchmarking Methodology (BM) RFCs

- 2285: BM for LAN Switching Devices
- 2544: BM for for Network Interconnect Devices
- 2647/3511: BM for Firewall Performance
- 2761/3116: BM for ATM Benchmarking
- 2889: BM for LAN Switching Devices
- 3918: BM for IP Multicast

Other Benchmarking Methodology (BM) ETSI

- Speech and multimedia Transmission Quality (STQ); QoS and network performance metrics and measurement methods; Part 3: Network performance metrics and measurement methods in IP networks
  ETSI EG 202 765-3

https://www.etsi.org/deliver/etsi_eg/202700_202799/20276503/01.01.01_60/eg_20276503v010101p.pdf

- Speech and multimedia Transmission Quality (STQ); QoS Parameter Measurements based on fixed Data Transfer Times
  ETSI TR 102 678

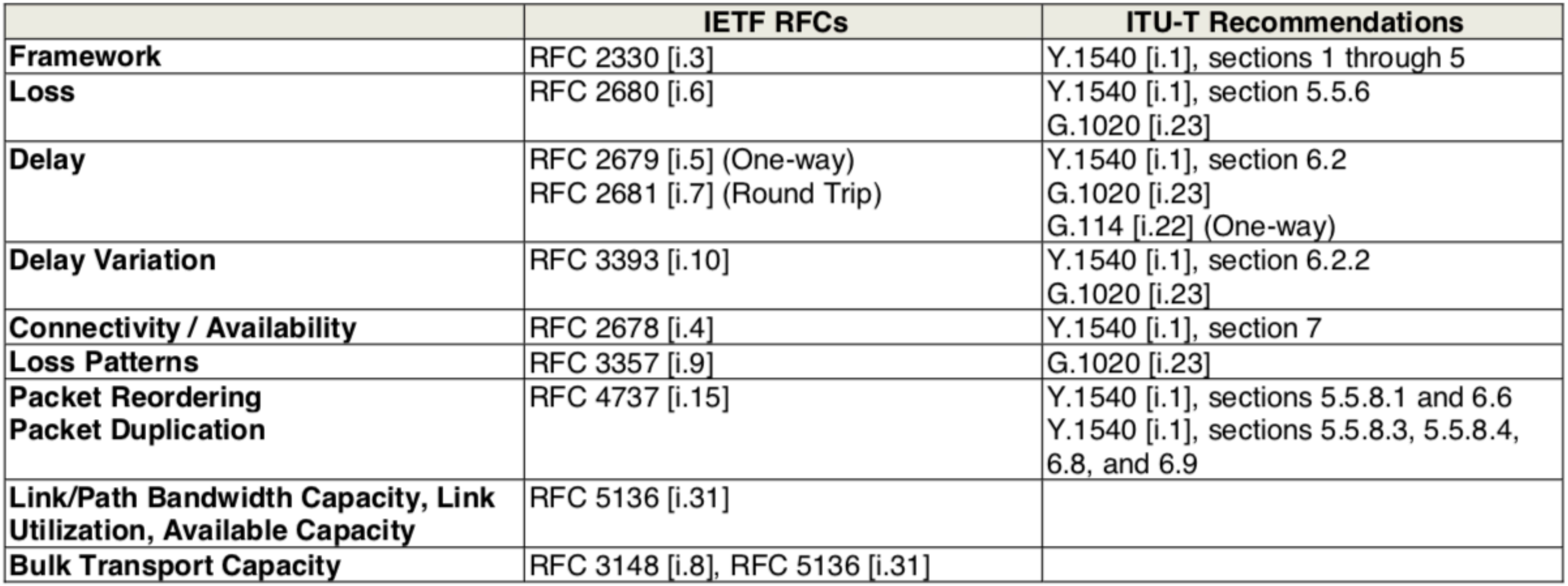
https://www.etsi.org/deliver/etsi_tr/102600_102699/102678/01.02.01_60/tr_102678v010201p.pdf

RFC 2544: Benchmarking Methodology for Network Interconnect Devices

It defines and specifies how to:

- Verify and evaluate the test results
- Measure common metrics defined in RFC 1242 such as:
  - Throughput
  - Latency
  - Frame Loss
- Handle “test modifiers” such as
  - Broadcast traffic (how can this traffic affect results)
  - Trial duration (how long the test should last)

Common Measurement Metrics [1/2]



	IETF RFCs	ITU-T Recommendations
Framework	RFC 2330 [i.3]	Y.1540 [i.1], sections 1 through 5
Loss	RFC 2680 [i.6]	Y.1540 [i.1], section 5.5.6 G.1020 [i.23]
Delay	RFC 2679 [i.5] (One-way) RFC 2681 [i.7] (Round Trip)	Y.1540 [i.1], section 6.2 G.1020 [i.23] G.114 [i.22] (One-way)
Delay Variation	RFC 3393 [i.10]	Y.1540 [i.1], section 6.2.2 G.1020 [i.23]
Connectivity / Availability	RFC 2678 [i.4]	Y.1540 [i.1], section 7
Loss Patterns	RFC 3357 [i.9]	G.1020 [i.23]
Packet Reordering **Packet Duplication**	RFC 4737 [i.15]	Y.1540 [i.1], sections 5.5.8.1 and 6.6 Y.1540 [i.1], sections 5.5.8.3, 5.5.8.4, 6.8, and 6.9
Link/Path Bandwidth Capacity, Link Utilization, Available Capacity	RFC 5136 [i.31]	
Bulk Transport Capacity	RFC 3148 [i.8], RFC 5136 [i.31]	

Common Measurement Metrics [2/2]

- Performance measurement
  - Availability
  - Response time
  - Accuracy
  - Throughput
  - Utilisation
  - Latency and Jitter

Measurement Metrics: Availability [1/2]

- Availability can be expressed as the percentage of time that a network system, component or application is available for a user.
- It is based on the reliability of the individual component of a network.

Measurement Metrics: Availability [2/2]

$$\% \text{ Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

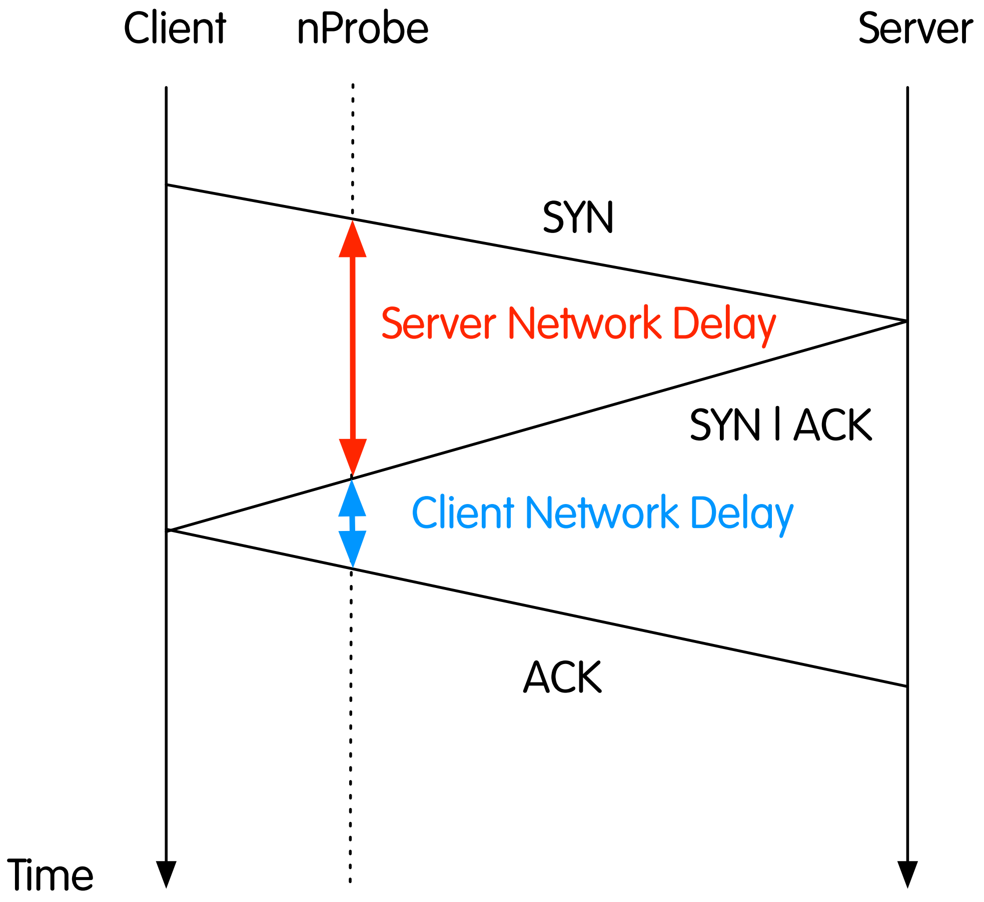
MTBF = mean time between failures

MTTR = mean time to repair following a failure.

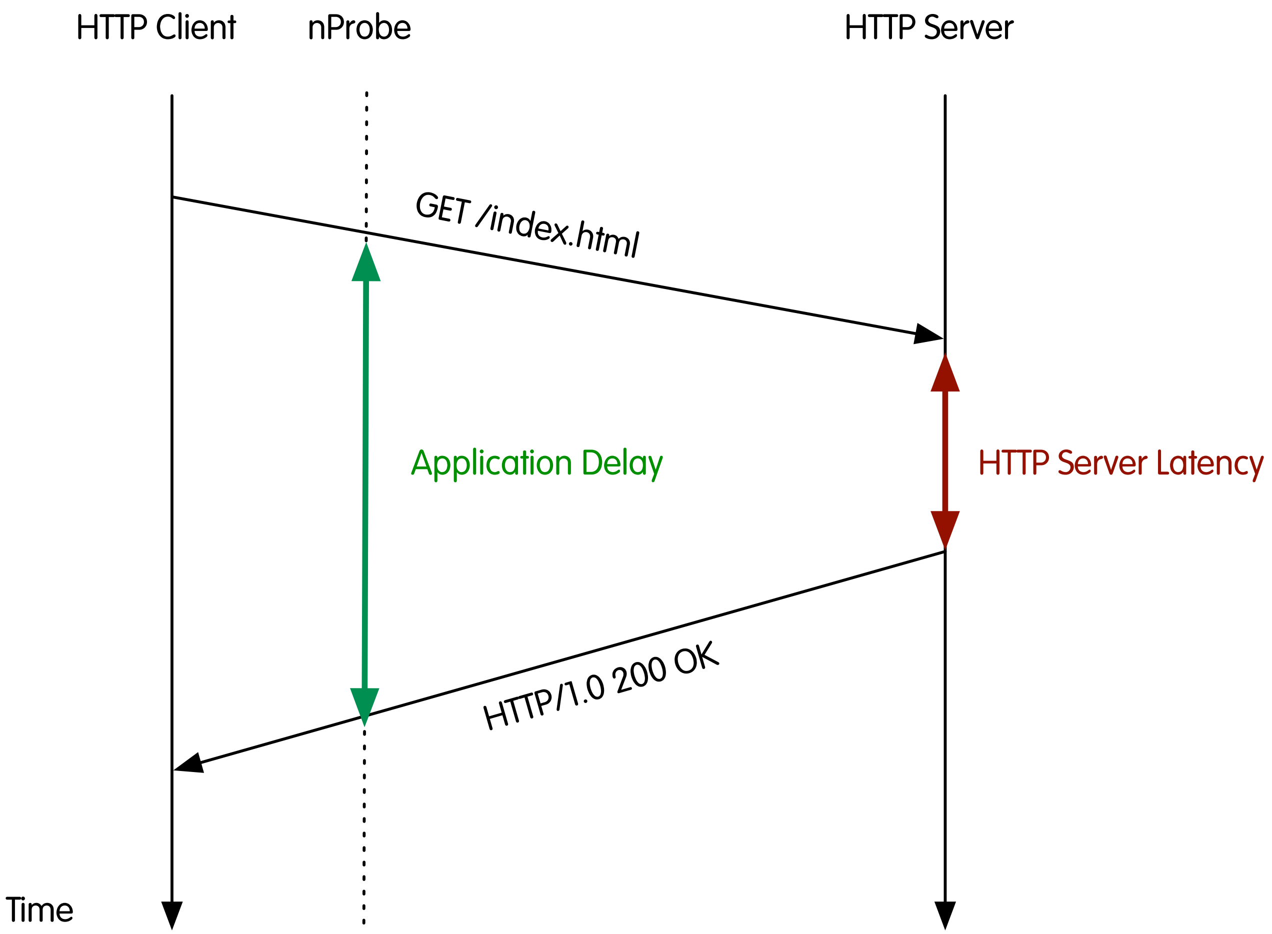
Measurement Metrics: Response Time

- Response time is the time it takes a system to react to a given input.
  - Example: In an interactive transaction, it may be defined as the time between the last keystroke by user and the beginning of the resulting display by the computer.
- Short response time is desirable.
- Necessary for interactive applications (e.g. telnet/ ssh) not very important for batch applications (e.g. file transfer).

Server and Client Network Delay (Latency)



Application Latency (Response Time)



HTTP Server Latency = Application Delay - Server Network Delay

Measurement Metrics: Throughput

- Metric for measuring the quantity of data that can be sent over a link in a specified amount of time.
- Often it is used for giving an estimation of an available link bandwidth.
- Note that bandwidth and throughput are very different metrics.
- Throughput is an application-oriented measurement.
- Examples:
  - The number of transactions of a given type for a certain period of time
  - The number of customer sessions for a given

Measurement Metrics: Goodput

- Same as throughput but considering only the packet payload (layer 7 only).
- It identifies the “real data” that is carried on the network not considering packet headers.
- It is useful to detect “stealth connections” (e.g. TCP connection that is up with no data exchanged beside dummy packets).

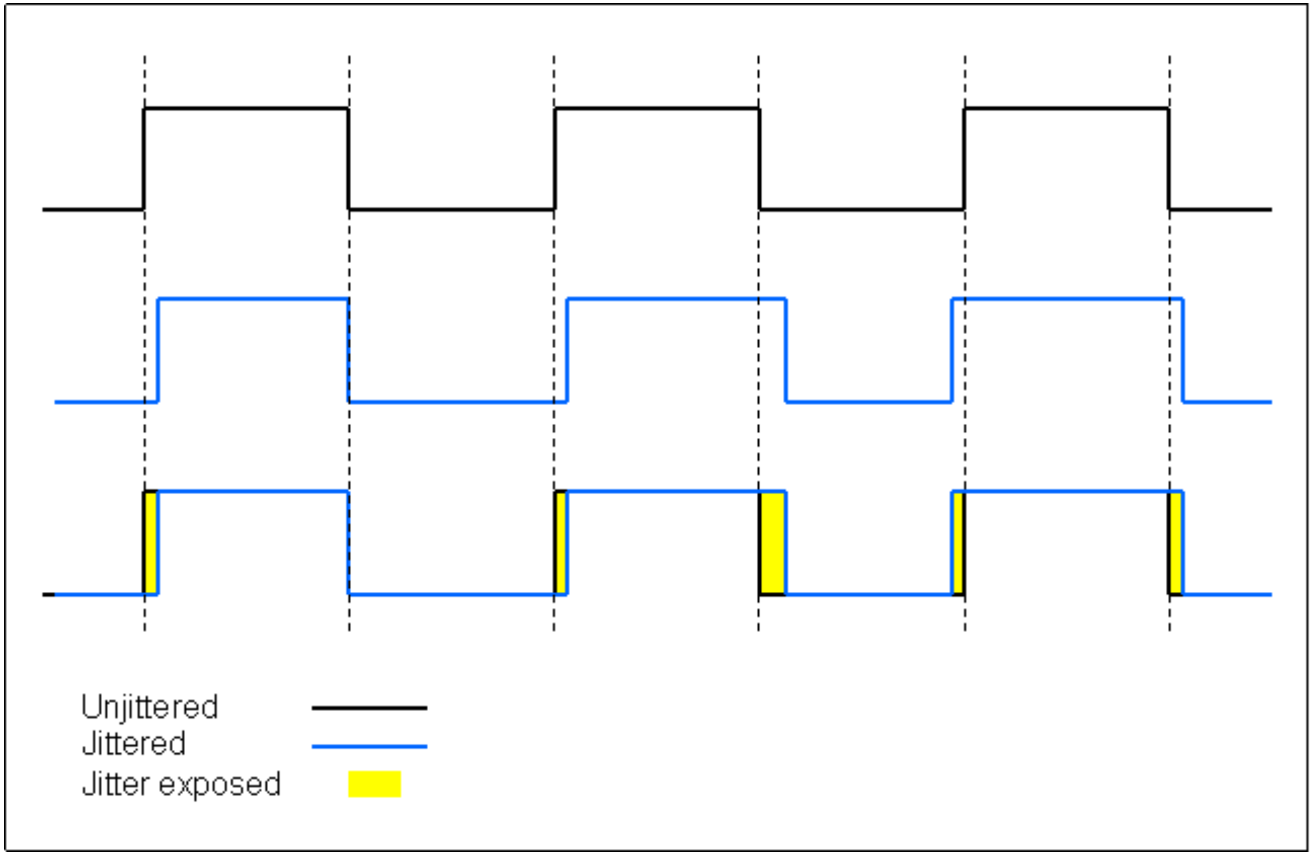
Measurement Metrics: Utilisation

- Utilisation is a more fine-grained measure than throughput. It refers to determining the percentage of time that a resource is in use over a given period of time.
- Often very little utilisation means that something isn’t working as expected (e.g. little traffic because the file server crashed).

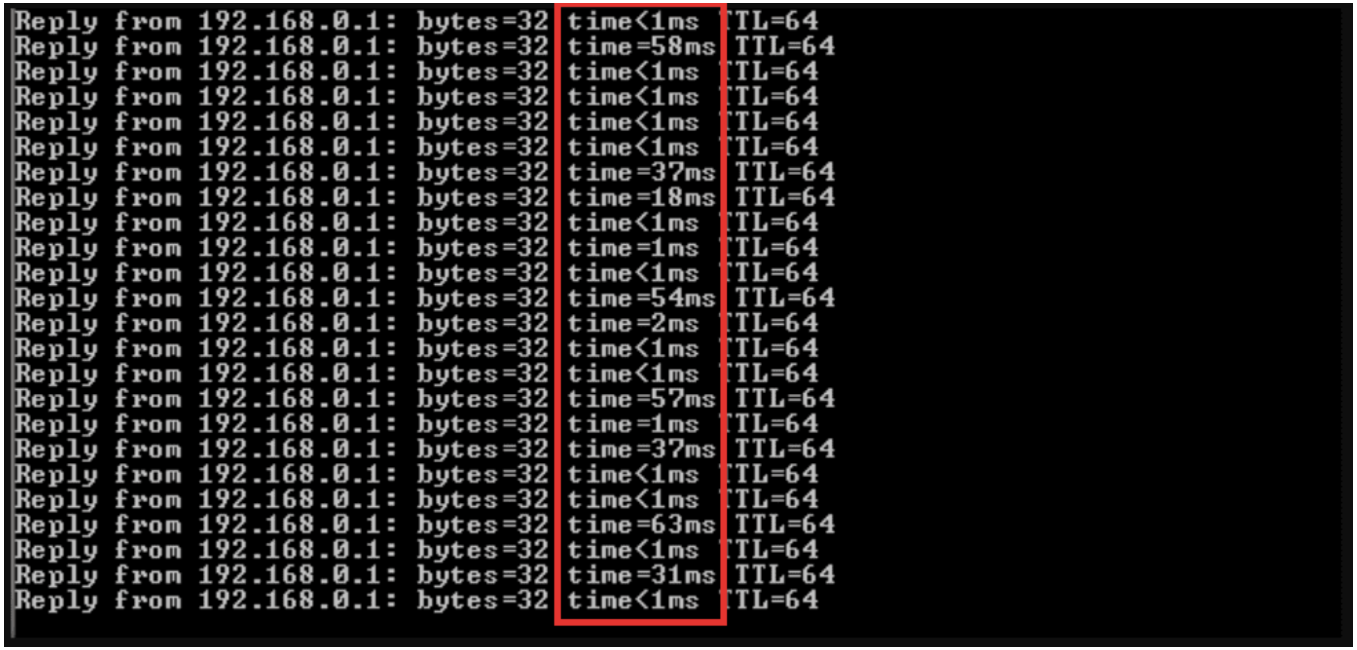
Measurement Metrics: Latency and Jitter

- Latency (msec): amount of time it takes a packet from source to destination. It is very important for interactive applications (e.g. online games).
- [RTT (Round Trip Time) / 2] is **NOT** the network latency, but the sum of c2s and s2c network latency is the RTT value.
- Jitter (msec): variance of intra-packet delay on a mono-directional link. It is very important for multimedia applications (e.g. Internet telephony or video broadcast).
- Jitter is a way to measure how a signal is irregular.

Jitter Explained



Jitter Calculation [1/2]



```
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=58ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=37ms TTL=64
Reply from 192.168.0.1: bytes=32 time=18ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=54ms TTL=64
Reply from 192.168.0.1: bytes=32 time=2ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=57ms TTL=64
Reply from 192.168.0.1: bytes=32 time=1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=37ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=63ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=31ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
```

```
jitter = sum(|x(i) - x(i-1)|) / (n-1))
```

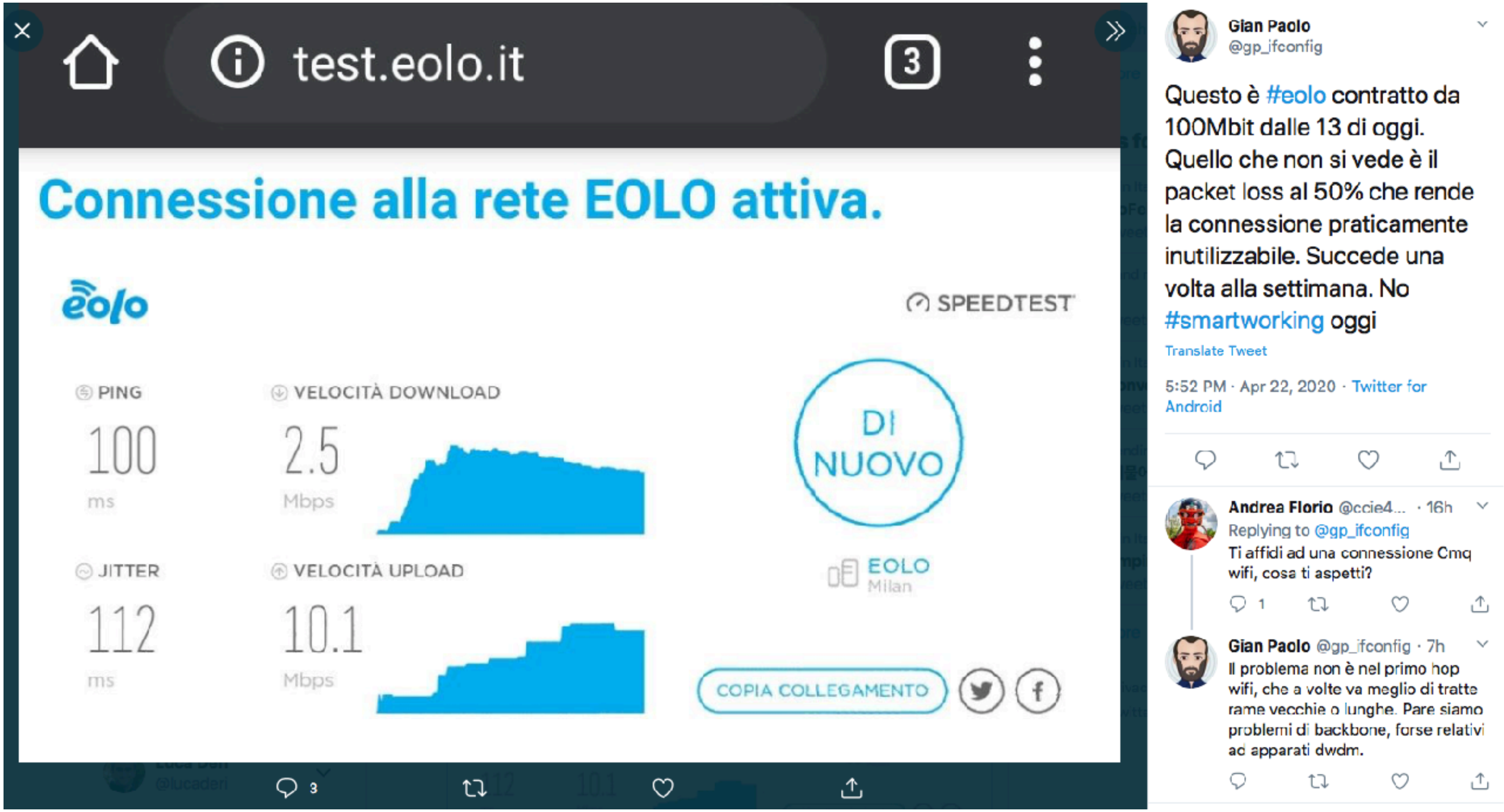
Note: Jitter takes into account the order of the events whereas standard deviation, average… do not.

Jitter Calculation [2/2]

Measurement	Value	Difference	
1	1		
2	58	57	
3	1	57	
4	1	0	
5	1	0	
6	1	0	
7	37	36	
8	18	19	
	Jitter	24,143	msec
	Average	14,750	msec
	StdDev	21,783	msec

NOTE:
Jitter is a generic metric that can be used in many other use cases

Jitter Evaluation



Connessione alla rete EOLO attiva.

PING 100 ms

VELOCITÀ DOWNLOAD 2.5 Mbps

JITTER 112 ms

VELOCITÀ UPLOAD 10.1 Mbps

Gian Paolo @gp_ifconfig

Questo è #eolo contratto da 100Mbit dalle 13 di oggi. Quello che non si vede è il packet loss al 50% che rende la connessione praticamente inutilizzabile. Succede una volta alla settimana. No #smartworking oggi

Translate Tweet

5:52 PM · Apr 22, 2020 · Twitter for Android

Andrea Florio @ccie4... · 16h

Replying to @gp_ifconfig

Ti affidi ad una connessione Cmq wifi, cosa ti aspetti?

Gian Paolo @gp_ifconfig · 7h

Il problema non è nel primo hop wifi, che a volte va meglio di tratte rame vecchie o lunghe. Pare siamo problemi di backbone, forse relativi ad apparati dwdm.

Measurement Metrics: Bandwidth

- Measurement Interval (Tc) - The time interval or “bandwidth interval” used to control traffic bursts.
- Burst Committed (Bc) - The maximum number of bits that the network agrees to transfer during any Tc.
- CIR (Committed Information Rate) - The rate at which a network agrees to transfer information under normal conditions, averaged over a minimum increment of time. CIR, measured in bits per second, is one of the key negotiated tariff metrics. CIR = Bc / Tc.
- Burst Excess (Be) - The number of bits to attempt to transmit after reaching the Bc value.
- Maximum Data Rate (MaxR) - Calculated value measured in bits per second. MaxR = ((Bc + Be)/Bc) * CIR = (Bc + Be)/Tc

Per-Link Measurements

- Metrics available on a link
  - – # packets, # bytes, # packets discarded on a specific interface over the last minute
  - – # flows, # of packets per flow
- It does not provide global network statistics.
- Useful to ISPs for traffic measurements.
- Examples:
  - – SNMP MIBs
  - – RTFM (Real-Time Flow Measurement)
  - – Cisco NetFlow

End-to-End Measurements

- Network performance != Application performance
  - Wire-time vs. web-server performance
- Most of network measurements are by nature end-to-end.
- Per path statistics
    - Are paths symmetric? Usually they are not. (Routing issue?)
    - How does the network behave with long/short probe packets?
- It is necessary for deducting per-link performance measurements.

Monitoring Approaches [1/2]

- Active Measurement
  - To inject network traffic and study how the network reacts to the traffic (e.g. ping).

- Passive Measurement
  - To monitor network traffic for the purpose of measurement (e.g. use the TCP three way handshake to measure network round-trip time).

Monitoring Approaches [2/2]

- Active measurements are often end-to-end, whereas passive measurements are limited to the link where the traffic is captured.
- There is no good and bad. Both approaches are good, depending on the case:
  - Passive monitoring on a switched network can be an issue.
  - Injecting traffic on a satellite link is often doable only by the satellite provider.
- Usually the best is to combine both approaches and compare results.

Inline vs. Offline Measurement

- Inline Measurements
  Measurement methods based on a protocol that flows over the same network where measurements are taken (e.g. SNMP).

- Offline Measurements
  Measurement methods that use different networks for reading network measurements (e.g. to read traffic counters from CLI using a serial port or a management network/VLAN).

Remote Monitoring

Networks are Changing… [1/2]

Force 1: the Internet

- network security will become even more critical in the future.
- the enterprise network will become a public network.

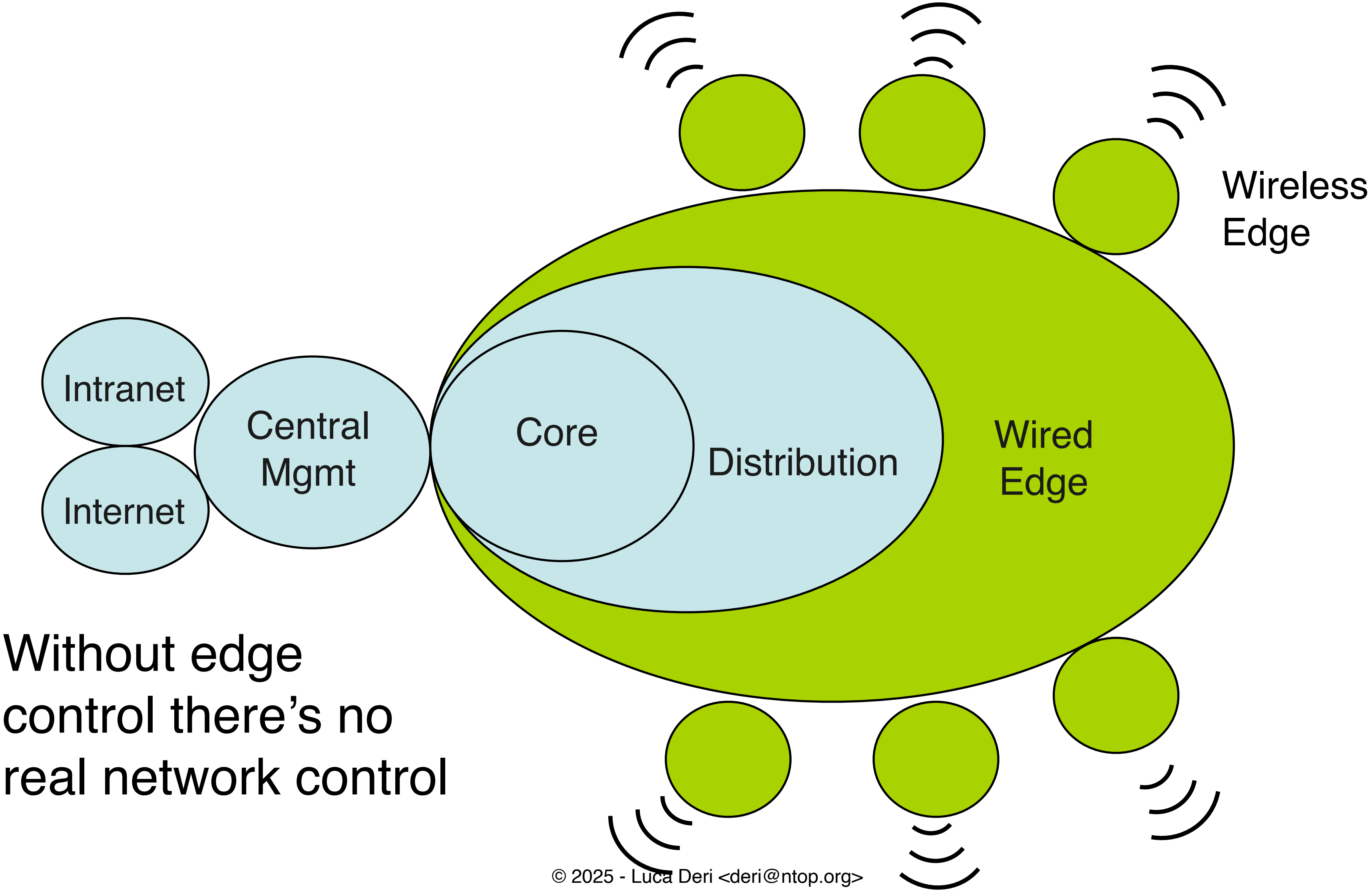
Force 2: Mobility

- supporting mobility across wired and wireless will be a key element of the network future.
- the network will become an anytime, anywhere resource.

Force 3: Dynamic Communications

- supporting a broad range of applications will be a key element in the future: convergence.
- the data network will become the only network.

Networks are Changing… [2/2]



Without edge control there's no real network control

Towards Remote Monitoring [1/3]

- Modern networks are distributed across various buildings, managed by different people with different skills (security, traffic engineer, DB administrator).
- It is necessary to collect traffic statistics on each network trunk, send them to a (limited number of) collector, in order to produce an aggregate network view.
- Some distributed analysis capabilities are necessary because a centralised network is not fault tolerant and scalable.

Towards Remote Monitoring [2/3]

- Deploying remote traffic analysers (e.g. pcap-based probes) are not always feasible because:
  - Server manufacturers do not always permit generic, untested software (e.g. the licence enforces that on a Oracle server you install only Oracle-certified apps) to be installed.
  - Modern servers often have several network interfaces (1Gb main+failover for data and 100 Mbit for server access), so a multi-interface probe is required.
  - Monitoring a 1 GE using a network tap requires 2 x GE (one RX for each direction of the original GE).

Towards Remote Monitoring [3/3]

- Solution: use traffic analysis capabilities provided by network appliances.
- Drawbacks:
  - Not all the appliances provide traffic analysis capabilities (e.g. most ADSL routers do not)
  - Even if supported, not always such capabilities can be enabled (strong impact on CPU and memory).
  - Basic monitoring capabilities provided by the default OS are rather limited so a custom card is necessary.
  - Custom cards for traffic analysis are not so cheap.

RMON: Remote Monitoring using SNMP

- Present in most mid-high end network appliances: often these are poor/limited implementations.
- Some vendors sell stand-alone probes: preferred case as
  - they are full implementations of the protocol.
  - They do not add additional load on the router.
- Not all the implementations (in particular those embedded in router/switches) support the whole standard but only selected SNMP groups.
- Together with Cisco NetFlow is the industrial, “trusted” monitoring standard.
- Two versions: RMON-1 (L2) and RMON-2 (L2/L3).

What can RMON do?

- Collect data and periodically report it to a more central management station, which potentially reduces traffic on WAN links and polling overhead on the management station.
- Report on what hosts are attached to the LAN, how much they talk, and to whom.
- "See" all LAN traffic, full LAN Utilisation, and not just the traffic to or through the router.
- Filter and capture packets (so you don't have to visit a remote LAN and attach a LAN Analyser) : it is basically a remote sniffer that can capture real-time traffic (until the integrated memory buffer is full).
- Automatically collect data, compare to thresholds, and send traps to your management station -- which offloads much of the work that might bog down the management station.

RMON vs. SNMP [1/2]

- The SNMP protocol is used to control and configure a probe. Usually GUI managers mask the complexity of SNMP-based configuration.
- Statistics and saved traffic are retrieved using SNMP by management applications to record statistics on a network and, possibly selected portions of the network traffic.

RMON Ethernet Statistics

- Packets: A unit of data formatted for transmission on a network.
- Multicast Packet: communication between a single sender and multiple receivers on a network.
- Broadcast Packet: a packet that is transmitted to all hosts on an Ethernet.
- Drop Events: An overrun at a port. The port logic could not receive the traffic at full line rate and had to drop some packets.
- Fragments: A piece of a packet. Sometimes a communications packet being sent over a network has to be temporarily broken into fragments; the packet should be reassembled when it reaches its destination.
- Jabbers: Packets received that were longer than 1518 octets and also contained alignment errors.
- Oversize Packets: Packets received that were longer than 1518 octets and were otherwise well formed.

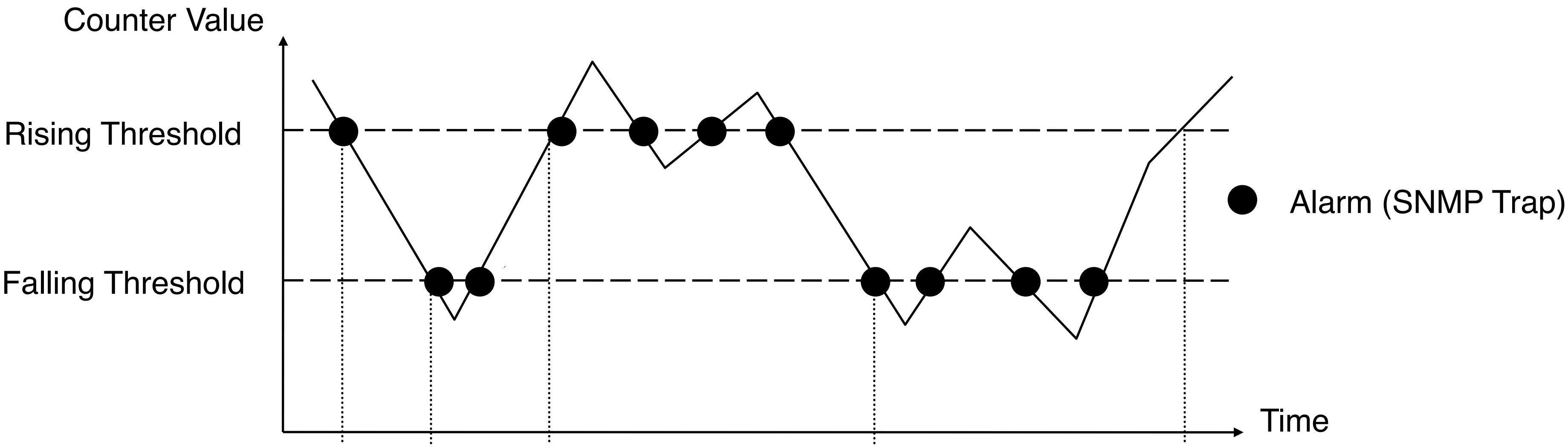
Network Utilisation with RMON

- Most RMON managers use RMON counters to compute network Utilisation.
- Network Utilisation can be calculated for all the ports of a given switch at regular intervals. This information can be gathered over the course of a day and be used to generate a network Utilisation profile of a switch or hub.

$$\%\ \text{Network Utilisation} = \frac{100 \times ((\#\ \text{packets} \times 160) + (\#\ \text{octets} \times 8))}{\text{port speed} \times \text{time (secs)}}$$

Note: 160 = 96 (# bit times (minimum) for the inter-frame gap) + 64 (# bits in the preamble + SFD (start frame delimiter). 8 is the number of bits in an octet.

RMON Alarm Group



- In case of exceeding a upper limit value, an event is produced each time the threshold is exceeded or when a value that used to be above a threshold returns inside the specified range. Similar considerations can be applied to lower threshold value.
- Thresholds can either be on to the measured value (absolute) or on the difference of the current value to the last measured value (delta

Case Study: Counter Sampling Interval [1/2]

Example 1: (10 seconds sampling interval, threshold value 20, 10 seconds test interval)

Time:	0	10	20
Value:	0	19	32
Delta:		19	13
Actual Threshold To Check:		19	13

Example 2: (5 seconds sampling interval, threshold value 20, 10 seconds test interval)

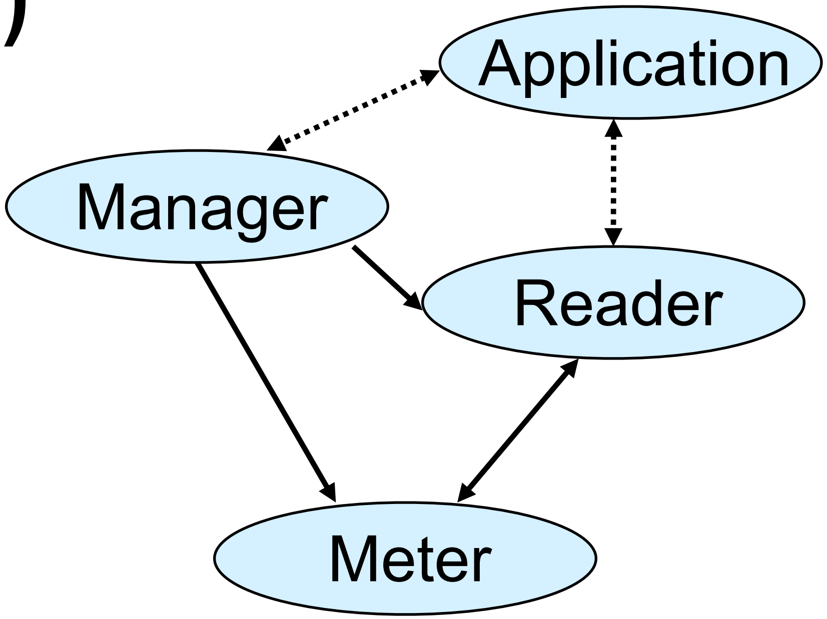
Time:	0	5	10	15	20
Value:	0	10	19	30	32
Delta:		10	9	11	2
Actual Threshold To Check :			19	20	13

Case Study: Counter Sampling Interval [2/2]

- MIB instance value sampling must be done twice per sampling interval, otherwise exceeded thresholds may be undetected for overlapping intervals.
- Fast polling has some drawbacks:
  - Much more data is collected.
  - Increased load on SNMP agents.
  - More data changes are detected (this can lead to false positives).
- Slow polling has some drawbacks too:
  - Some alarms can be missed (inaccuracy)

Flow Monitoring

Real-Time Flow Measurement (RTFM)



- Very flexible and powerful meter
  - programmable rule sets
  - can serve several readers
  - programmable overload behaviour
- Reader polls meter
- Implemented by SNMP Meter MIB
- Free software implementation NeTraMet
- No acceptance at manufacturers
- Complicated to use (too powerful)
- Specified by RFCs 2720 - 2724

SNMP vs. Network Flows [1/2]

- SNMP is based on the manager agent paradigm
  - The agent monitors the network and informs the manager (via traps) when something important happened (i.e. and interface changed state).
  - The manager keeps the whole system status by periodically reading (polling) variables (e.g. via SNMP Get) form the agent.
  - SNMP variables can be used for both element/device/system management (e.g. info about disk space and partitions) and traffic monitoring.

SNMP vs. Network Flows [2/2]

- Network flows are emitted by a probe towards one or more collectors according to traffic conditions.
  - Flows contain information about the analysed traffic (i.e. they do not contain device/probe information such as the MIB II variables).
  - Emitted flows have a well defined format (e.g. Cisco NetFlow v5) and often use UDP as transport (no specialised protocol like SNMP).
  - No concept of ‘alarm’ flows nor ability for the probe to perform actions based on flows: all the intelligence is in the collector.
  - Probe instrumentation is performed offline.
  - Probes are activated where the network traffic flows (e.g. inside routers and switches).

So What Do You Expect To Measure with Flows? [1/2]

- Where your campus exchanges traffic with by IP address, IP Prefix, or ASN.
- What type and how much traffic (SMTP, WEB, File Sharing, etc).
- What services running on campus.
- Department level traffic summaries.
- Track network based viruses back to hosts.

So What Do You Expect To Measure with Flows? [2/2]

- Track DoS attacks to the source(s), i.e. the 100 servers flooding XXX.com domain.
- Find busy hosts on campus (top host).
- How many destinations each campus host exchanges traffic with.
- Campus host counts by service, i.e. how many active web servers.

What You Can’t Measure with Flows?

- Non-IP traffic (e.g. NetBIOS, ARP).
- L2 information (e.g. interface up/down state changes).
- Filtered traffic (e.g. firewall policy counters).
- Per-link statistics (e.g. link usage, congestion, delay, packet loss).
- Application statistics (e.g. transaction latency, # positive/negative replies, protocol errors).

Network Flows: What Are They?

- “A flow is a set of packets with a set of common packet properties” (e.g. common IP address/port).
- A flow is (queued to be) emitted only when expired.
- Creation and expiration policy
  - What conditions start and stop a flow?
  - Maximum flow duration timeout regardless of the connection status (e.g. a TCP connection ends when both peers agreed on FIN/RST).
  - Emit a flow when there’s no flow traffic for a specified amount of time.

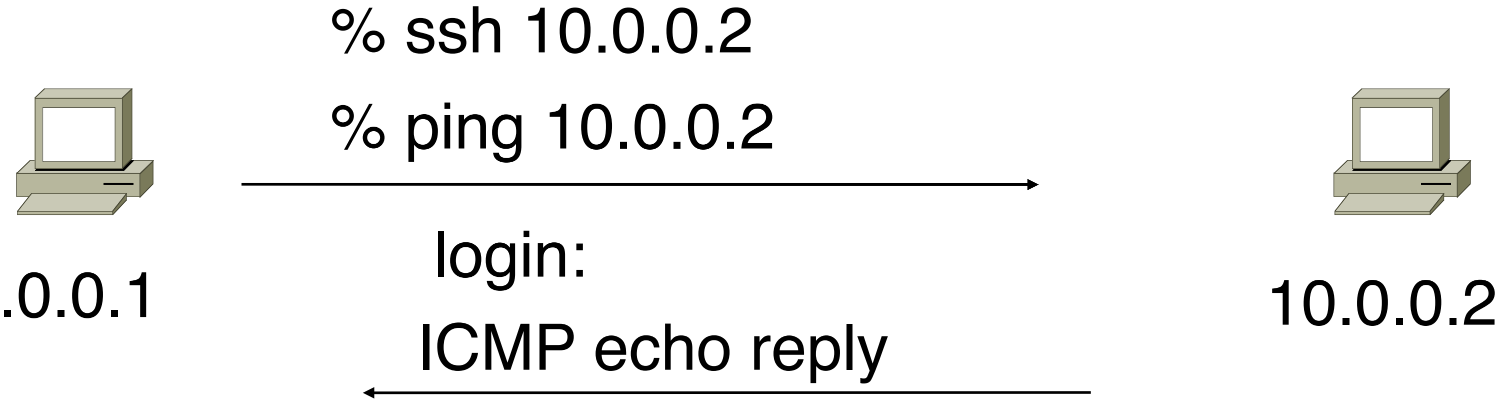
Network Flows Content

- Flow contain:
  - Peers: flow source and destination.
  - Counters: packets, bytes, time.
  - Routing information: AS, network mask, interfaces.
- Flows can be unidirectional (default) or bidirectional (v9/IPFIX only).
- Two, opposite, unidirectional flows are equivalent to one bidirectional flow.
- Bidirectional flows can contain other information such as round trip time, TCP behaviour.

Network Flows Issues

- Overhead vs. Accuracy
  - More measurement results in more collected data.
  - More flow aggregation, less granularity.
  - Overhead (e.g. CPU load) on routers, switches, end-hosts.
- Security vs. Data Sharing
  - Emitted flows must reach collectors on protected paths (e.g. using a different network/VLAN).
  - User privacy must be respected.
  - Traffic measurements must be kept protected in order not to disclosure important network information to third parties.

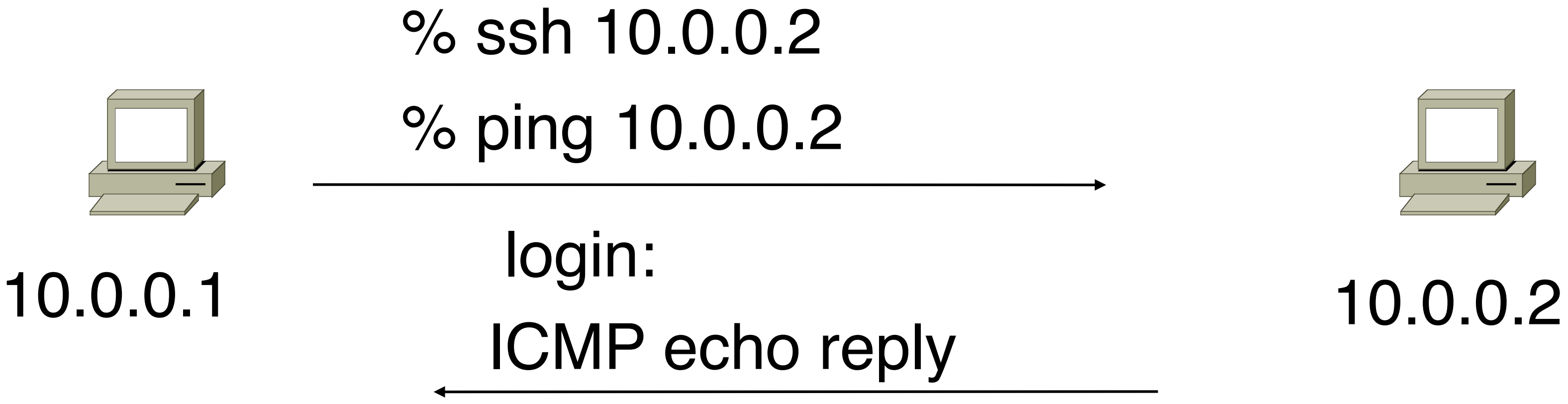
Unidirectional Flow with Source/ Destination IP Key



Active Flows

Flow	Source IP	Destination IP
1.	10.0.0.1	10.0.0.2
2.	10.0.0.2	10.0.0.1

Unidirectional Flow with IP, Port, Protocol Key



% ssh 10.0.0.2

% ping 10.0.0.2

login:

ICMP echo reply

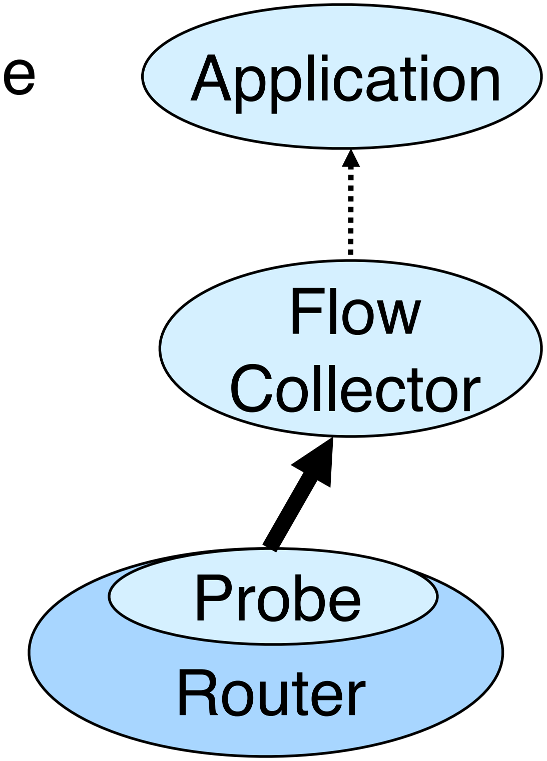
10.0.0.1

10.0.0.2

Active Flows

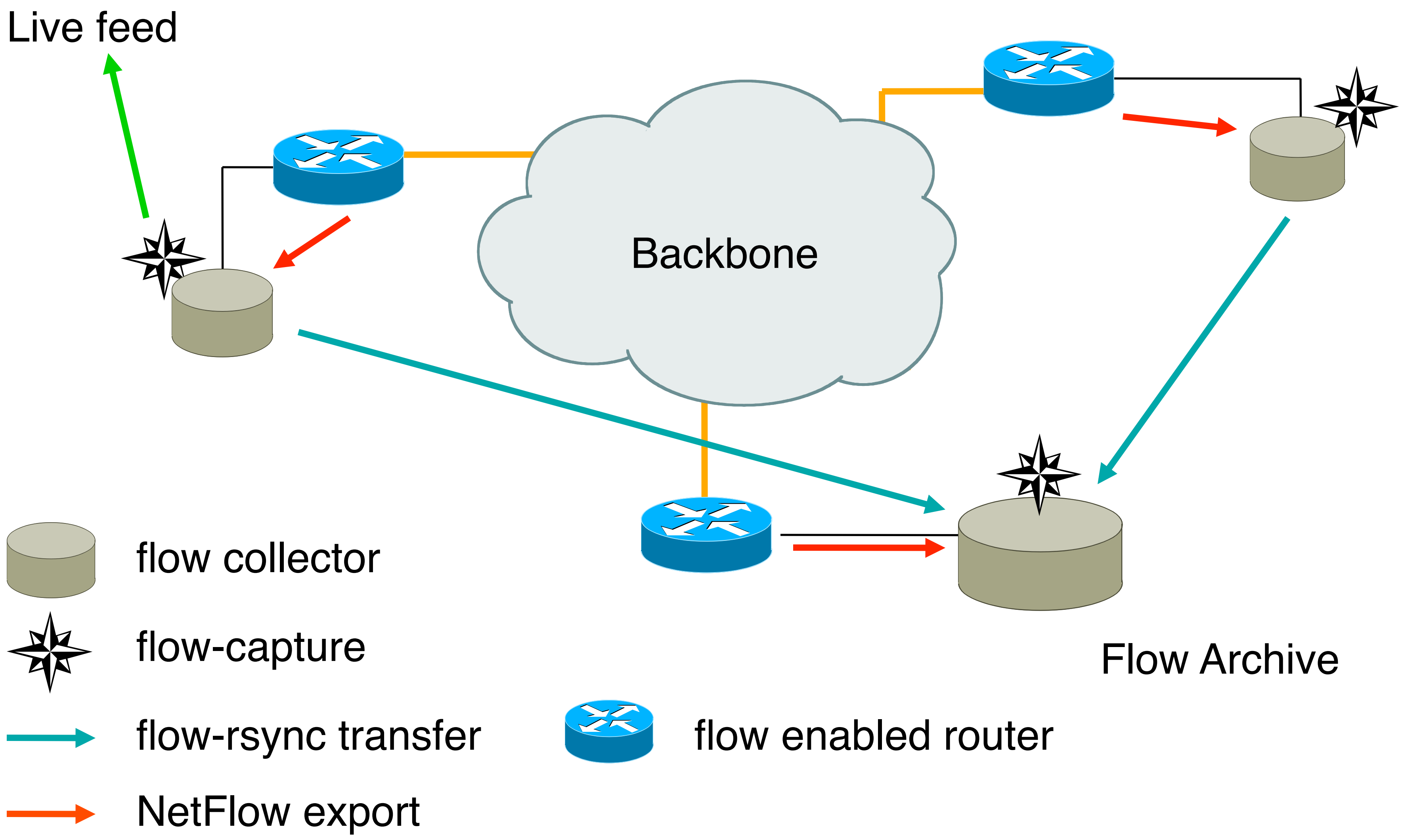
Flow	Source IP	Destination IP	Proto	srcPort	dstPort
1.	10.0.0.1	10.0.0.2	TCP	32000	22
2.	10.0.0.2	10.0.0.1	TCP	22	32000
3.	10.0.0.1	10.0.0.2	ICMP	0	0
4.	10.0.0.2	10.0.0.1	ICMP	0	0

NetFlow Architecture



- Flows are exported (push) by the probe when expired, contrary to SNMP where the manager polls the agent periodically.
- The flow transport protocol is NetFlow (no SNMP).
- Probe/collector configuration protocol is not specified by the NetFlow protocol.
- The NetFlow collector has the job of assembling and understanding the exported flows and combining or aggregating them to produce the valuable reports used for traffic and security analysis.

A Collection Architecture [1/2]



Live feed

Backbone

Flow Archive

flow collector

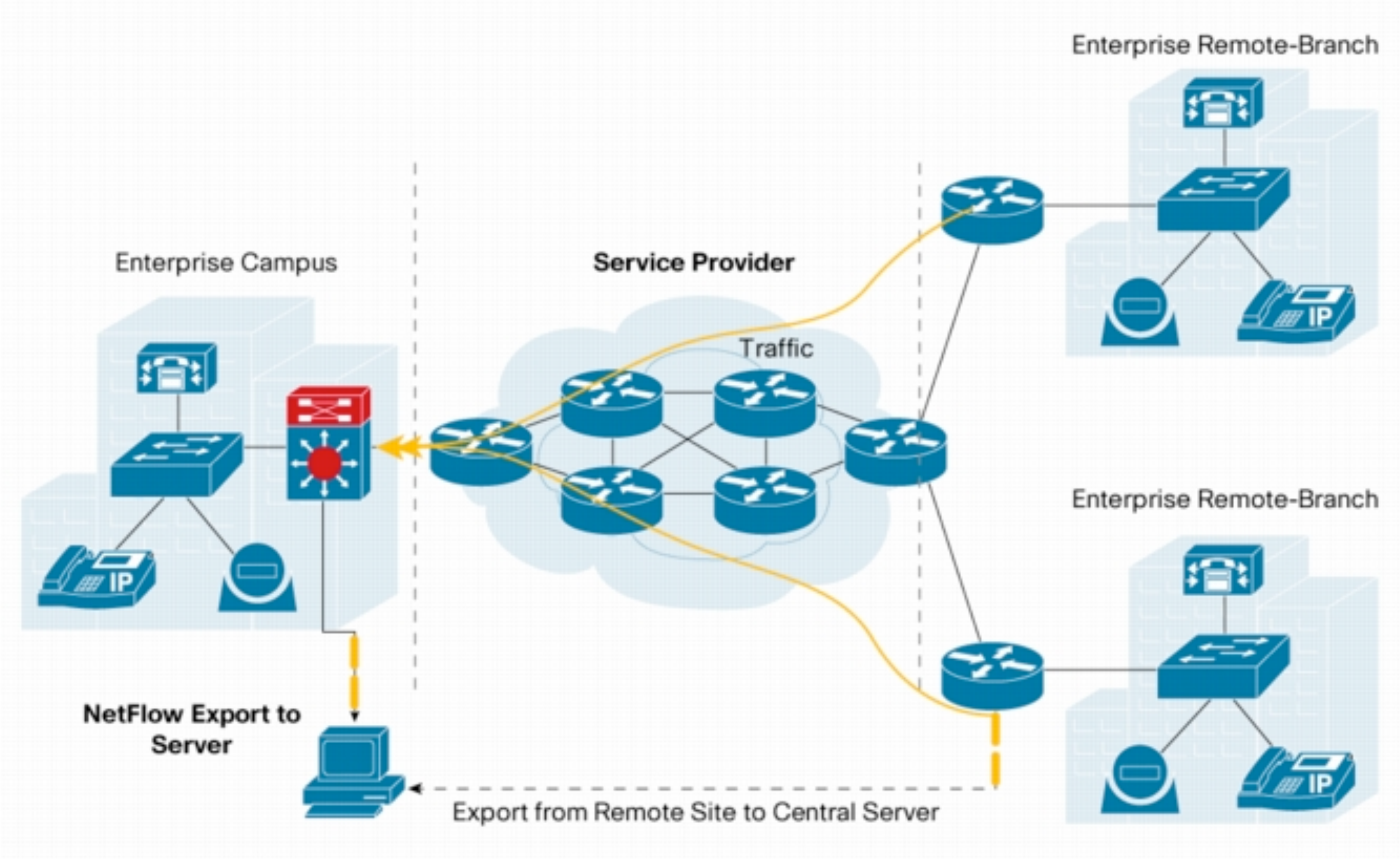
flow-capture

flow-rsync transfer

flow enabled router

NetFlow export

A Collection Architecture [2/2]



Collection Space Constraints

- Space required depends on traffic
- Some average figures:
  - 67.320 octets/flow, 92 packets/flow
  - Busy router: 397 GB of traffic/day, 548,000,000 packets/day == 5.900.000 flows/day
  - At 60 bytes/flow, this is 350 MB of logs/day
  - With level 6 compression we get 4.3:1
  - Which works out to 82 MB/day for this

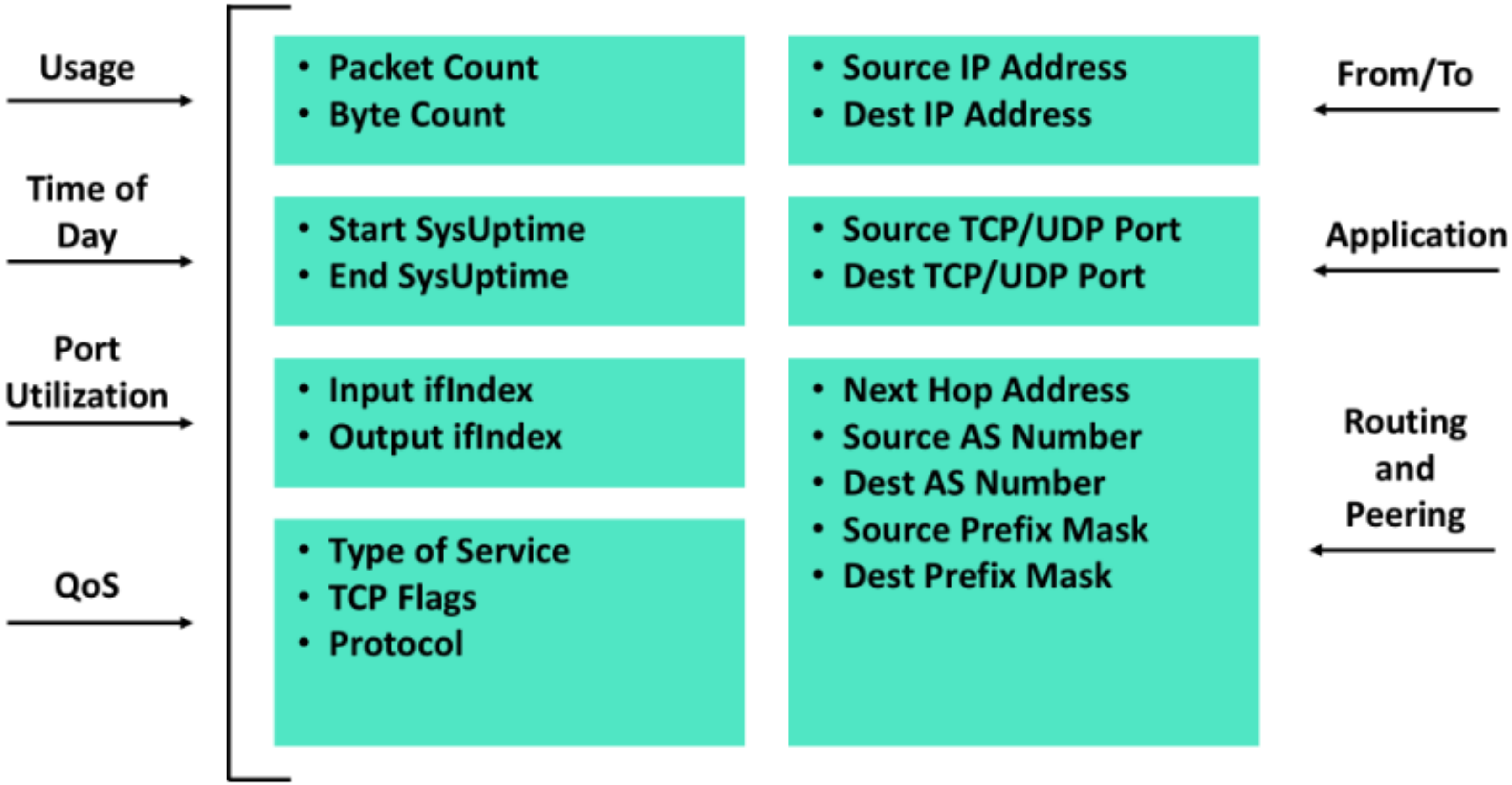
Cisco NetFlow Basics

- Unidirectional flows (up to v8), bidirectional on v9.
- Several versions v 1,5,6,7,8,9. The most common is v5, the latest version is v9.
- Traffic analysis only on inbound (i.e. the traffic that enters the router) IP-only traffic (not on all platforms).
- IPv4 unicast and multicast: all NetFlow versions. IPv6 is supported only by v9.
- Open protocol defined by Cisco and supported on IOS and CatIOS platforms (no NetFlow support on PIX firewalls) as well as on on-Cisco platforms

Cisco NetFlow Versions

- Each version has its own packet format
  - v1,5,6,7,8 have a fixed/closed, specified format.
  - v9 format is dynamic and open to extensions.
- Sequence Numbers:
  - v1 does not have sequence numbers (no way to detect lost flows).
  - v5,6,7,8 have flow sequence numbers (i.e. keep track of the number of emitted flows).
  - v9 has packet (not flow) sequence number (I.e. easy to know the number of lost packets but not of lost flows).
- The “version” defines what type of data is in the flow.
- Some versions (e.g. v7) specific to Catalyst platform.

Using Flows



Usage

- Packet Count
- Byte Count

Time of Day

- Start SysUptime
- End SysUptime

Port Utilization

- Input ifIndex
- Output ifIndex

QoS

- Type of Service
- TCP Flags
- Protocol

From/To

- Source IP Address
- Dest IP Address

Application

- Source TCP/UDP Port
- Dest TCP/UDP Port

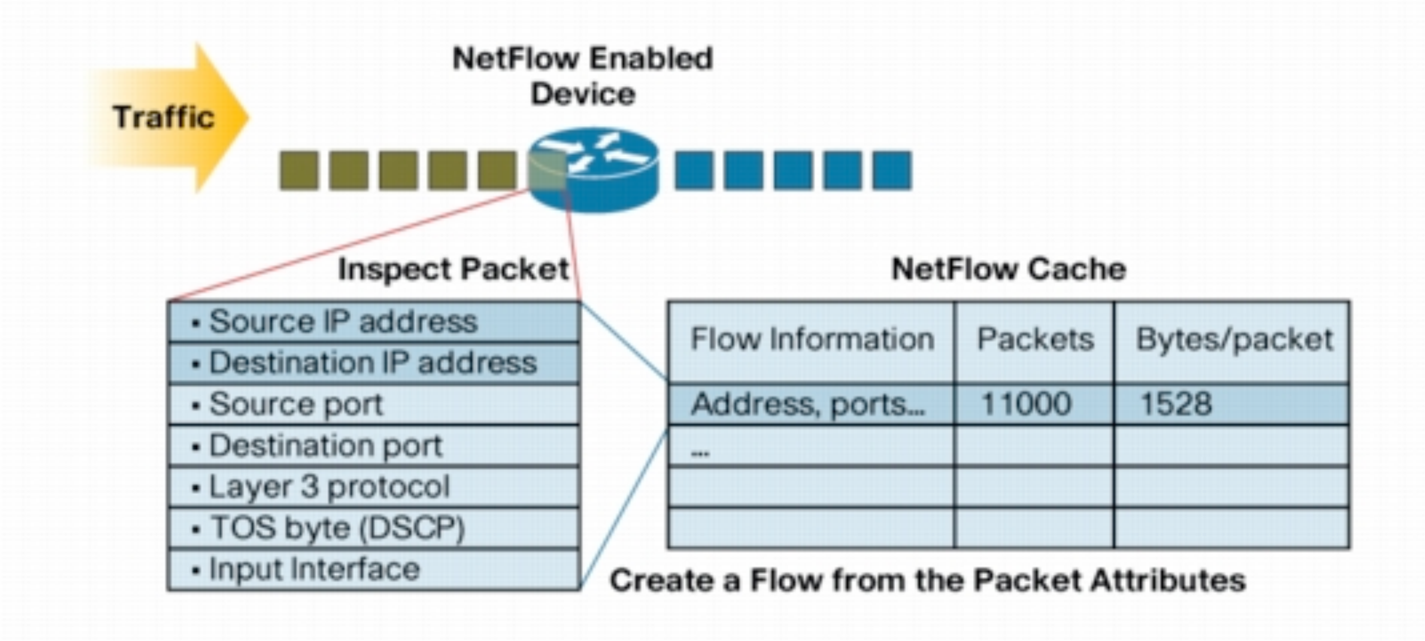
Routing and Peering

- Next Hop Address
- Source AS Number
- Dest AS Number
- Source Prefix Mask
- Dest Prefix Mask

NetFlow: Flow Birth and Death [1/5]

- Each packet that is forwarded within a router or L3 switch is examined for a set of IP packet attributes.
- All packets with the same source/destination IP address, source/destination ports, protocol interface are grouped into a flow and then packets and bytes tallied.
- Active flows are stored in memory in the so-called NetFlow cache.

NetFlow: Flow Birth and Death [2/5]



Inspect Packet

- Source IP address
- Destination IP address
- Source port
- Destination port
- Layer 3 protocol
- TOS byte (DSCP)
- Input Interface

NetFlow Cache

Flow Information	Packets	Bytes/packet
Address, ports...	11000	1528
...		

Create a Flow from the Packet Attributes

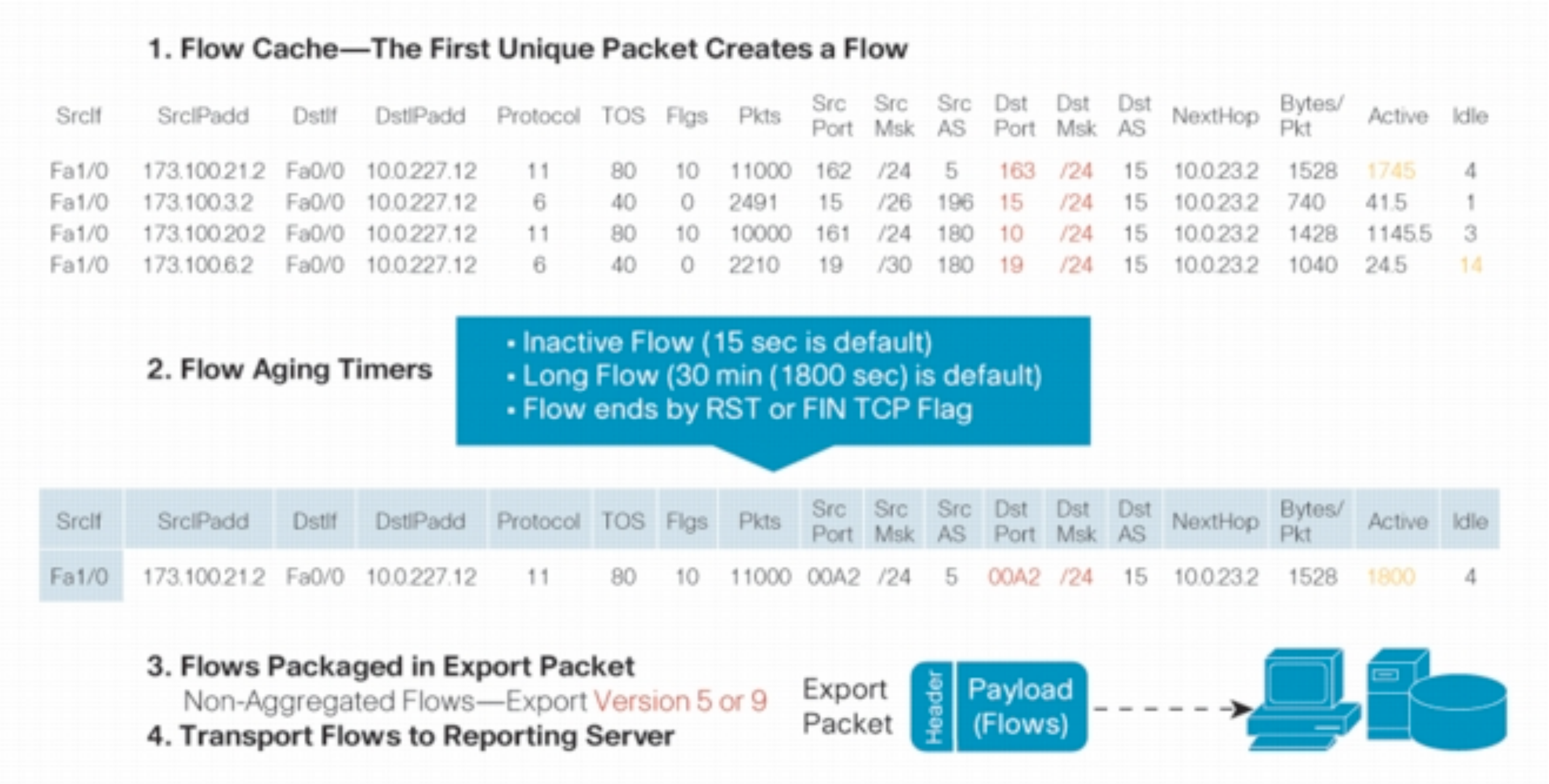
NetFlow: Flow Birth and Death [3/5]

Flows are terminated when one of these conditions are met:

- The network communication has ended (e.g. a packet contains the TCP FIN flag).
- The flow lasted too long (default 30 min).
- The flow has been not active (i.e. no new packets have been received) for too long (default 15 sec).
- The flow cache was full and the cache manager had to purge data.

Note that the flow cache has a limited size, hence it’s often not possible to accommodate all flows.

NetFlow: Flow Birth and Death [4/5]



1. Flow Cache—The First Unique Packet Creates a Flow

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	162	/24	5	163	/24	15	10.0.23.2	1528	1745	4
Fa1/0	173.100.3.2	Fa0/0	10.0.227.12	6	40	0	2491	15	/26	196	15	/24	15	10.0.23.2	740	41.5	1
Fa1/0	173.100.20.2	Fa0/0	10.0.227.12	11	80	10	10000	161	/24	180	10	/24	15	10.0.23.2	1428	1145.5	3
Fa1/0	173.100.6.2	Fa0/0	10.0.227.12	6	40	0	2210	19	/30	180	19	/24	15	10.0.23.2	1040	24.5	14

2. Flow Aging Timers

- Inactive Flow (15 sec is default)
- Long Flow (30 min (1800 sec) is default)
- Flow ends by RST or FIN TCP Flag

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	Src Port	Src Msk	Src AS	Dst Port	Dst Msk	Dst AS	NextHop	Bytes/ Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.0.23.2	1528	1800	4

3. Flows Packaged in Export Packet

Non-Aggregated Flows—Export Version 5 or 9

4. Transport Flows to Reporting Server

NetFlow: Flow Birth and Death [5/5]

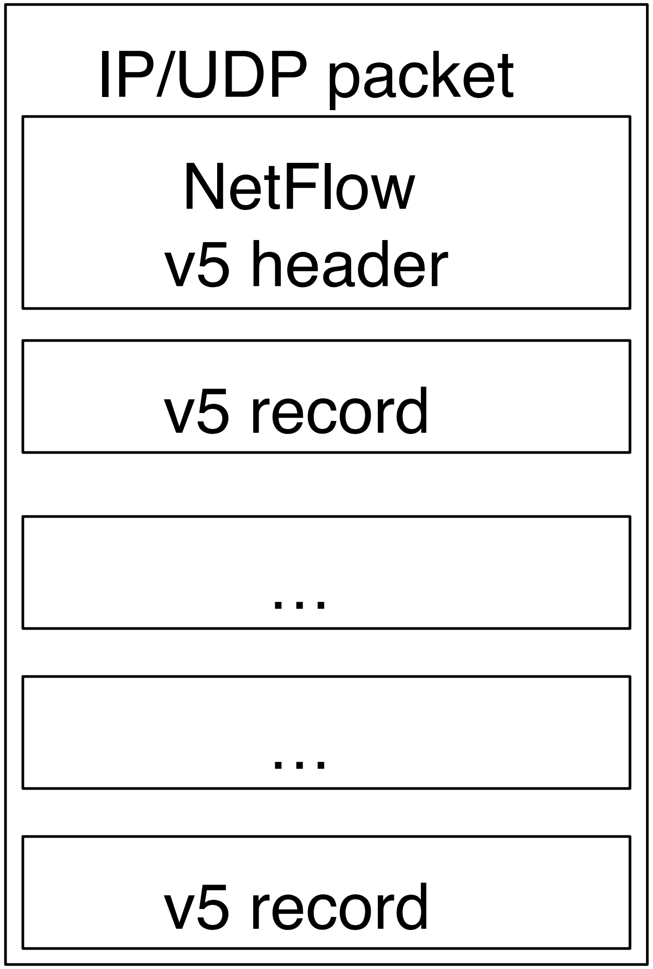
- The NetFlow cache is constantly filling with flows and software in the router or switch is searching the cache for flows that have terminated or expired and these flows are exported to the NetFlow collector server.
- The consequence is that network flows can be split in several netflow flows that, if necessary, are reassembled by the flow collector.

For more info see Cisco White Paper: “Introduction to Cisco IOS NetFlow - A Technical Overview”.

NetFlow Packet Format

- Common header among export versions.
- Version specific data field where N records of data type are exported.
- N is determined by the size of the flow definition (e.g. N=30 for v5). Packet size is kept under ~1480 bytes. No fragmentation on Ethernet.

Cisco NetFlow v5 [1/3]



Cisco NetFlow v5 [2/3]

```
struct netflow5_record  {
  struct flow_ver5_hdr flowHeader;
  struct flow_ver5_rec flowRecord[30];
} NetFlow5Record;

struct flow_ver5_hdr {
  u_int16_t version;                              /* Current version=5*/
  u_int16_t count;                                /* The number of records in PDU. */
  u_int32_t sysUptime;                  /* Current time in msecs since router booted */
  u_int32_t unix_secs;                  /* Current seconds since 0000 UTC 1970 */
  u_int32_t unix_nsecs;                 /* Residual nanoseconds since 0000 UTC 1970 */
  u_int32_t flow_sequence;              /* Sequence number of total flows seen */
  u_int8_t  engine_type;                /* Type of flow switching engine (RP,VIP,etc.)*/
  u_int8_t  engine_id;                  /* Slot number of the flow switching engine */
};
```

Cisco NetFlow v5 [3/3]

```
struct flow_ver5_rec {
  u_int32_t srcaddr;        /* Source IP Address */
  u_int32_t dstaddr;        /* Destination IP Address */
  u_int32_t nexthop;        /* Next hop router's IP Address */
  u_int16_t input;          /* Input interface index (SNMP) */
  u_int16_t output;         /* Output interface index (SNMP) */
  u_int32_t dPkts;          /* Packets sent */
  u_int32_t dOctets;        /* Octets sent */
  u_int32_t First;          /* (SNMP) SysUptime at start of flow */
  u_int32_t Last;           /* and of last packet of the flow */
  u_int16_t srcport;        /* TCP/UDP source port number (.e.g, FTP, Telnet, etc.,or equivalent) */
  u_int16_t dstport;        /* TCP/UDP destination port number (.e.g, FTP, Telnet, etc.,or equivalent) */
  u_int8_t pad1;            /* pad to word boundary */
  u_int8_t tcp_flags;       /* Cumulative OR of tcp flags */
  u_int8_t prot;            /* IP protocol, e.g., 6=TCP, 17=UDP, etc... */
  u_int8_t tos;             /* IP Type-of-Service */
  u_int16_t src_as;         /* source peer/origin Autonomous System */
  u_int16_t dst_as;         /* dst peer/origin Autonomous System */
  u_int8_t src_mask;        /* source route's mask bits */
  u_int8_t dst_mask;        /* destination route's mask bits */
  u_int16_t pad2;           /* pad to word boundary */
};
```

NetFlow v5 Flow Example [1/2]

```
Cisco NetFlow
    Version: 5
    Count: 30
    SysUptime: 1518422100
    Timestamp: May  7, 1993 08:49:48.995294598
        CurrentSecs: 736757388
        CurrentNSecs: 995294598
    FlowSequence: 9751
    EngineType: 0
    EngineId: 0
    SampleRate: 0
pdu 1/30
    [ …… ]
```

NetFlow v5 Flow Example [2/2]

```
pdu 1/30
        SrcAddr: 10.16.237.114 (10.16.237.114)
        DstAddr: 213.92.16.87 (213.92.16.87)
        NextHop: 10.158.100.1 (10.158.100.1)
        InputInt: 4
        OutputInt: 1
        Packets: 5
        Octets: 627
        StartTime: 1518415.920000000 seconds
        EndTime: 1518416.352000000 seconds
        SrcPort: 3919
        DstPort: 80
        padding
        TCP Flags: 0x1b
        Protocol: 6
        IP ToS: 0x00
        SrcAS: 0
        DstAS: 0
        SrcMask: 16 (prefix: 10.16.0.0/16)
        DstMask: 0 (prefix: 0.0.0.0/32)
        padding
pdu 2/30
  [ .......................... ]
```

Why Do We Need NetFlow v9?

- Fixed formats (v1-v8) for export are:
  - Easy to implement.
  - Consume little bandwidth.
  - Easy to decipher at the collector.
  - Not flexible (many proprietary hacks such as using TCP/UDP ports for transporting ICMP type/ code).
  - Not extensible (no way to extend the flow unless a new version is defined).
  - Some features are missing: L2, VLAN, IPv6, MPLS.

NetFlow v9 Principles [1/2]

- Open protocol defined by Cisco (i.e. it’s not proprietary) defined in RFC 3954.
- Flow Template + flow record
  - Template composed of type and length.
  - Flow record composed of template ID and value.
  - Templates are sent periodically and they are a prerequisite for decoding flow records.
  - Flow records contain the ‘flow meat’.
- Options templates + option records contain probe configuration (e.g. packet/flow sampling rate, interface packet counters).

NetFlow v9 Principles [2/2]

- Push model probe -> collector (as with past versions).
- Send the templates regularly: each X flows, each X seconds.
- Independent of the underlying protocol, ready for any reliable protocol.
- Can send both template and flow record in one export.
- Can interleave different flow records in one export packet.

Some v9 Tags [1/4]

[1] %IN_BYTES	Incoming flow bytes
[2] %IN_PKTS	Incoming flow packets
[3] %FLOWS	Number of flows
[4] %PROTOCOL	IP protocol byte
[5] %SRC_TOS	Type of service byte
[6] %TCP_FLAGS	Cumulative of all flow TCP flags
[7] %L4_SRC_PORT	IPv4 source port
[8] %IPV4_SRC_ADDR	IPv4 source address
[9] %SRC_MASK	Source subnet mask (/<bits>)
[10] %INPUT_SNMP	Input interface SNMP idx
[11] %L4_DST_PORT	IPv4 destination port
[12] %IPV4_DST_ADDR	IPv4 destination address
[13] %DST_MASK	Dest subnet mask (/<bits>)
[14] %OUTPUT_SNMP	Output interface SNMP idx
[15] %IPV4_NEXT_HOP	IPv4 next hop address

Some v9 Tags [2/4]

[16] %SRC_AS	Source BGP AS
[17] %DST_AS	Destination BGP AS
[21] %LAST_SWITCHED	SysUptime (msec) of the last flow pkt
[22] %FIRST_SWITCHED	SysUptime (msec) of the first flow pkt
[23] %OUT_BYTES	Outgoing flow bytes
[24] %OUT_PKTS	Outgoing flow packets
[27] %IPV6_SRC_ADDR	IPv6 source address
[28] %IPV6_DST_ADDR	IPv6 destination address
[29] %IPV6_SRC_MASK	IPv6 source mask
[30] %IPV6_DST_MASK	IPv6 destination mask
[32] %ICMP_TYPE	ICMP Type * 256 + ICMP code
[34] %SAMPLING_INTERVAL	Sampling rate
[35] %SAMPLING_ALGORITHM	Sampling type (deterministic/random)
[36] %FLOW_ACTIVE_TIMEOUT	Activity timeout of flow cache entries

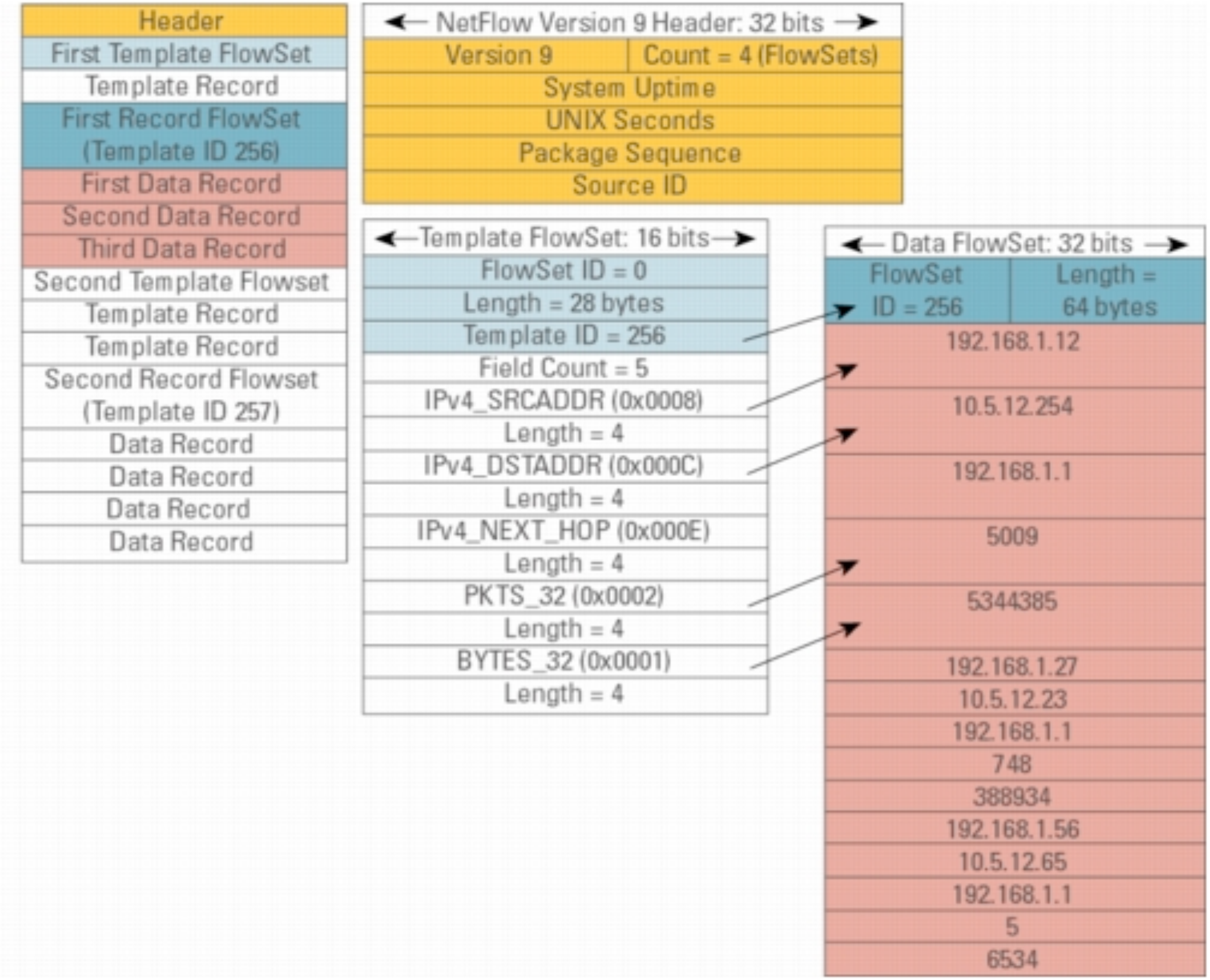
Some v9 Tags [3/4]

[37] %FLOW_INACTIVE_TIMEOUT	Inactivity timeout of flow cache entries
[38] %ENGINE_TYPE	Flow switching engine
[39] %ENGINE_ID	Id of the flow switching engine
[40] %TOTAL_BYTES_EXP	Total bytes exported
[41] %TOTAL_PKTS_EXP	Total flow packets exported
[42] %TOTAL_FLOWS_EXP	Total number of exported flows
[56] %IN_SRC_MAC	Source MAC Address
[57] %OUT_DST_MAC	Destination MAC Address
[58] %SRC_VLAN	Source VLAN
[59] %DST_VLAN	Destination VLAN
[60] %IP_PROTOCOL_VERSION	[4=IPv4][6=IPv6]
[61] %DIRECTION	[0=ingress][1=egress] flow

Some v9 Tags [4/4]

[70] %MPLS_LABEL_1	MPLS label at position 1
[71] %MPLS_LABEL_2	MPLS label at position 2
[72] %MPLS_LABEL_3	MPLS label at position 3
[73] %MPLS_LABEL_4	MPLS label at position 4
[74] %MPLS_LABEL_5	MPLS label at position 5
[75] %MPLS_LABEL_6	MPLS label at position 6
[76] %MPLS_LABEL_7	MPLS label at position 7
[77] %MPLS_LABEL_8	MPLS label at position 8
[78] %MPLS_LABEL_9	MPLS label at position 9
[79] %MPLS_LABEL_10	MPLS label at position 10
[80] %IN_DST_MAC	Source MAC Address
[81] %OUT_SRC_MAC	Destination MAC Address
[98] %ICMP_FLAGS	Cumulative of all flow ICMP types

v9 Flow Format



Header
First Template FlowSet
Template Record
First Record FlowSet (Template ID 256)
First Data Record
Second Data Record
Third Data Record
Second Template Flowset
Template Record
Template Record
Second Record Flowset (Template ID 257)
Data Record
Data Record
Data Record
Data Record

NetFlow Version 9 Header: 32 bits

Version 9	Count = 4 (FlowSets)
System Uptime	
UNIX Seconds	
Package Sequence	
Source ID	

Template FlowSet: 16 bits

FlowSet ID = 0
Length = 28 bytes
Template ID = 256
Field Count = 5
IPv4_SRCADDR (0x0008)
Length = 4
IPv4_DSTADDR (0x000C)
Length = 4
IPv4_NEXT_HOP (0x000E)
Length = 4
PKTS_32 (0x0002)
Length = 4
BYTES_32 (0x0001)
Length = 4

Data FlowSet: 32 bits

FlowSet ID = 256	Length = 64 bytes

192.168.1.12
10.5.12.254
192.168.1.1
5009
5344385
192.168.1.27
10.5.12.23
192.168.1.1
748
388934
192.168.1.56
10.5.12.65
192.168.1.1
5
6534

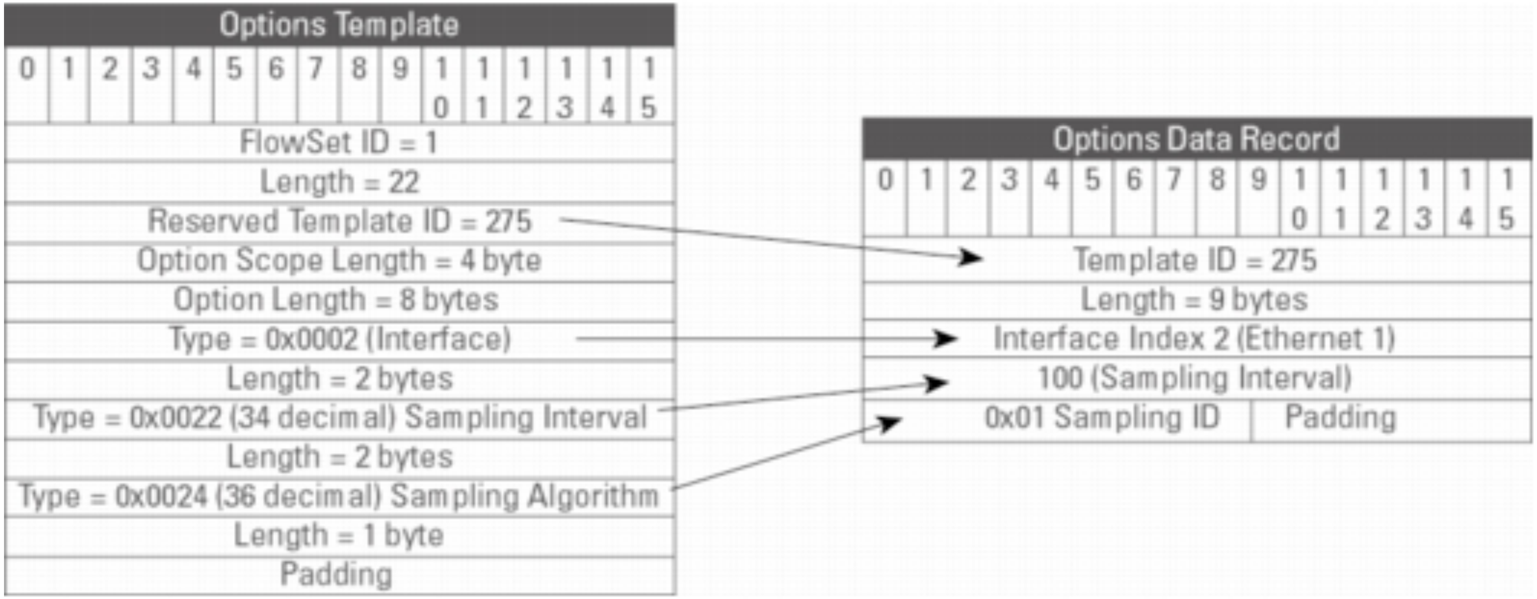
NetFlow v9 Flow Example [1/2]

```
Cisco NetFlow
    Version: 9
    Count: 4
    SysUptime: 1132427188
    Timestamp: Aug 18, 2000 23:49:25.000012271
        CurrentSecs: 966635365
    FlowSequence: 12271
    SourceId: 0
    FlowSet 1/4
 FlowSet 1/4
        Template FlowSet: 0
        FlowSet Length: 164
        Template Id: 257
        Field Count: 18
        Field (1/18)
            Type: LAST_SWITCHED (21)
            Length: 4
        Field (2/18)
            Type: FIRST_SWITCHED (22)
            Length: 4
        Field (3/18)
             [ ...... ]
```

NetFlow v9 Flow Example [2/2]

```
Cisco NetFlow
    Version: 9
    Count: 1
    SysUptime: 1133350352
    Timestamp: Aug 19, 2000 00:04:48.000012307
        CurrentSecs: 966636288
    FlowSequence: 12307
    SourceId: 0
    FlowSet 1/1
        Data FlowSet (Template Id): 257
        FlowSet Length: 52
        pdu 1
            EndTime: 1133334.000000000 seconds
            StartTime: 1133334.000000000 seconds
            Octets: 84
            Packets: 1
            InputInt: 15
            OutputInt: 0
            SrcAddr: 172.18.86.77 (172.18.86.77)
            DstAddr: 11.10.65.130 (11.10.65.130)
            Protocol: 1
            IP ToS: 0x00
            [ … ]
```

V9 Options Template



Options Template															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FlowSet ID = 1															
Length = 22															
Reserved Template ID = 275															
Option Scope Length = 4 byte															
Option Length = 8 bytes															
Type = 0x0002 (Interface)															
Length = 2 bytes															
Type = 0x0022 (34 decimal) Sampling Interval															
Length = 2 bytes															
Type = 0x0024 (36 decimal) Sampling Algorithm															
Length = 1 byte															
Padding															

Options Data Record															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Template ID = 275															
Length = 9 bytes															
Interface Index 2 (Ethernet 1)															
100 (Sampling Interval)															
0x01 Sampling ID									Padding						

NetFlow v5 vs. v9

	v5	v9
Flow Format	Fixed	User Defined
Extensible	No	Yes (Define new FlowSet Fields)
Flow Type	Unidirectional	Bidirectional
Flow Size	48 Bytes (fixed)	It depends on the format
IPv6 Aware	No	IP v4/v6
MPLS/VLAN	No	Yes

Cisco IOS Configuration [1/2]

- Configured on each input interface.
- Define the version.
- Define the IP address of the collector (where to send the flows).
- Optionally enable aggregation tables.
- Optionally configure flow timeout and main (v5) flow table size.
- Optionally configure sample rate.

Cisco IOS Configuration [2/2]

```
interface FastEthernet0/0/0
 ip address 10.0.0.1 255.255.255.0
 no ip directed-broadcast
 ip route-cache flow

interface ATM1/0/0
 no ip address
 no ip directed-broadcast
 ip route-cache flow

interface Loopback0
 ip address 10.10.10.10 255.255.255.255
 no ip directed-broadcast

ip flow-export version 5 origin-as
ip flow-export destination 10.0.0.10 5004
ip flow-export source loopback 0

ip flow-aggregation cache prefix
 export destination 10.0.0.10 5555
 enabled
```

JunOS Configuration [1/3]

- Sample packets with firewall filter and forward to routing engine.
- Sampling rate is limited to 7000 pps (sampling is compulsory).
- Fine for traffic engineering, but restrictive for DoS and intrusion detection.
- Juniper calls NetFlow cflowd (popular collector provided by CAIDA).

JunOS Configuration [2/3]

Firewall filter

```
firewall {
    filter all {
        term all {
            then {
                sample;
                accept;
            }
        }
    }
}
```

Enable sampling / flows

```
forwarding-options {
    sampling {
        input {
            family inet {
                rate 100;
            }
        }
        output {
            cflowd 10.0.0.16{
                port 2055;
                version 5;
            }
        }
    }
}
```

JunOS Configuration [3/3]

Apply firewall filter to each interface.

```
interfaces {
    ge-0/3/0 {
        unit 0 {
            family inet {
              filter {
                    input all;
                    output all;
                }
                address 192.148.244.1/24;
            }
        }
    }
```

PC-Based NetFlow Probes

- There are some PC-based probes.
- Most of them are based on the pcap library.
- Examples
  - nProbe
  - Softflowd

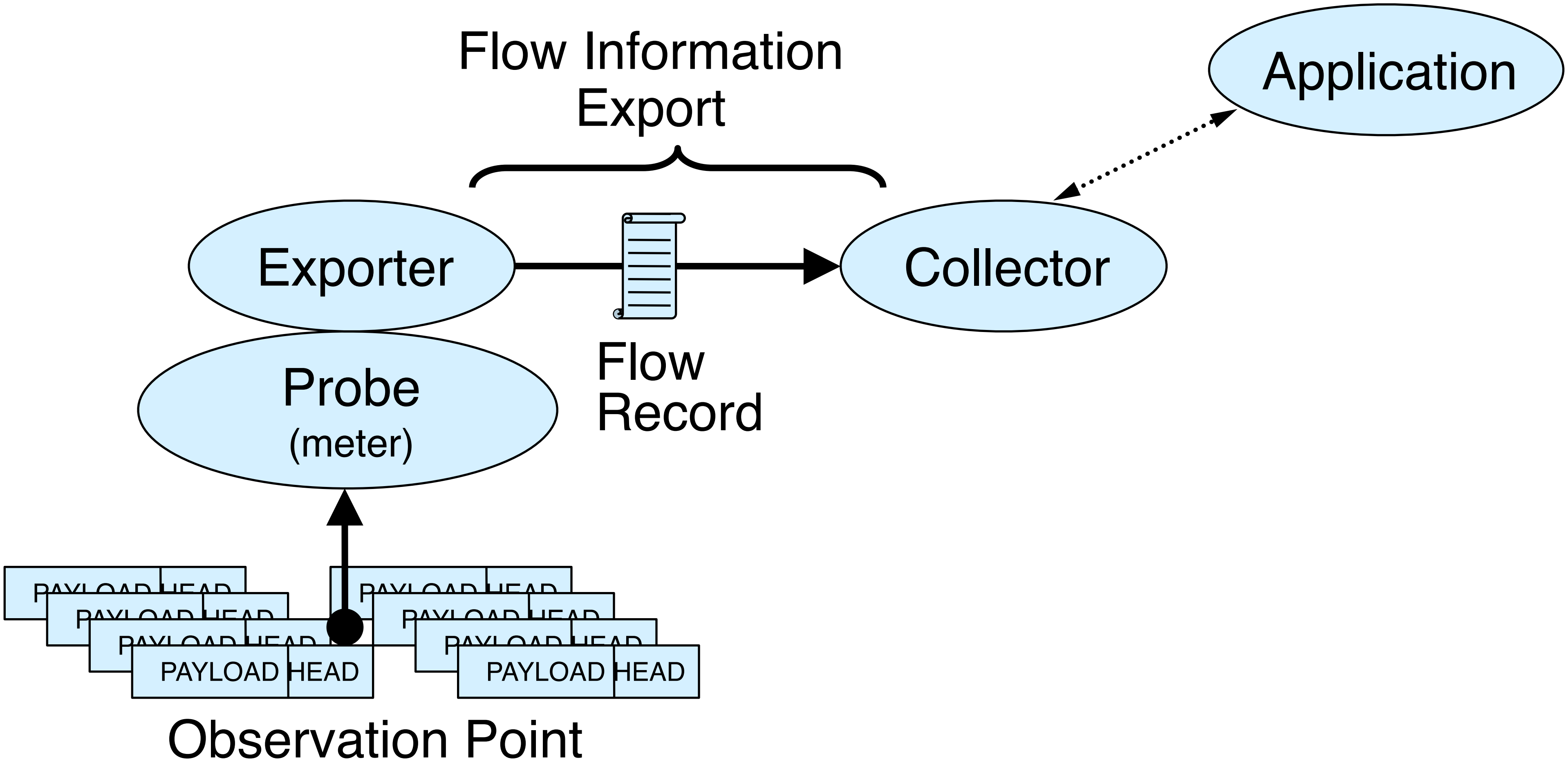
IPFIX Scope and General Requirements

- Goal: Find or develop a basic common IP Traffic Flow measurement technology to be available on (almost) all future routers.
- Fulfilling requirements of many applications.
- Low hardware/software costs.
- Simple and scalable.
- Metering to be integrated in general purpose IP routers and other devices (probes, middle boxes).
- Data processing to be integrated into various applications.
- Interoperability by openness or standardisation.

IPFIX in a Nutshell

- Strongly based on NetFlow v9.
- Ability to define new flow fields using a standard format based on PEN.
- Transport based on SCTP (Stream Control Transport Protocol), optional UDP/TCP support.
- Current status: international standard.
- Bottom Line: IPFIX = NetFlow v9 over SCTP (Note UDP is still the most used transport) with some extra little differences.

IPFIX Architecture



© 2025 - Luca Deri <deri@ntop.org>

Flow Aggregation [1/5]

Raw flows are useful but sometimes it’s necessary to answer to various questions:

- How much of our traffic is web, news, email, quake?
- How much traffic to/from departments?
- How much traffic to other departments, provider X, Google, etc.?
- Amount of traffic through interface X?

Flow Aggregation [2/5]

Main Active Flow Table

Flow	Source IP	Destination IP	Proto	srcPort	dstPort
1.	10.0.0.1	10.0.0.2	TCP	32000	23
2.	10.0.0.2	10.0.0.1	TCP	23	32000
3.	10.0.0.1	10.0.0.2	ICMP	0	0
4.	10.0.0.2	10.0.0.1	ICMP	0	0

Source/Destination IP Aggregation

Flow	Source IP	Destination IP
1.	10.0.0.1	10.0.0.2
2.	10.0.0.2	10.0.0.1

Flow Aggregation [3/5]

The same flow can be aggregated several times using different criteria. For instance from raw flows it’s possible to generate:

- List of protocols
- Conversation matrix (who’s talking to who)
- Top TCP/UDP ports

Aggregation flow early can save time/memory with respect to late aggregation (e.g. the conversation matrix is much easier to implement aggregating data on the probe instead of using raw/ unaggregated flows).

Flow Aggregation [4/5]

- Flows can be aggregated using “external” criteria and not just based on raw flow fields.
- Usually these external criteria are applied on “key” (not “value”) fields such as port, IP address, protocol etc. and are used to group values together.
- Criteria are added (don’t replace) to existing fields.
- Example: port-map, protocol-map, ip-address

	IP src	IP dst	Proto	Src port	Dst port
Before	10.0.0.1	10.0.0.2	UDP	32000	53
	10.0.0.2	10.0.0.1	TCP	34354	80

	IP src	IP dst	Proto	Src port	Dst port	Src port map	Dst port map
After	10.0.0.1	10.0.0.2	UDP	32000	53	udp_other	domain
	10.0.0.2	10.0.0.1	TCP	34354	80	tcp_other	http

Flow Aggregation [5/5]

- Flows can be aggregated according to:
  - TCP/UDP Port, ToS (Type of Service), Protocol (e.g. ICMP, UDP), AS (Autonomous System)
  - Source/Destination IP Address
  - Subnet, time of the day.
- Aggregation can be performed by the probe, the collector or both.
- Probe aggregation is very effective in terms of resource usage and network flow traffic.
- Collector aggregation is more powerful (e.g. aggregate flows produced by different probes) but rather costly (receive all the aggregate).

Flow Filtering

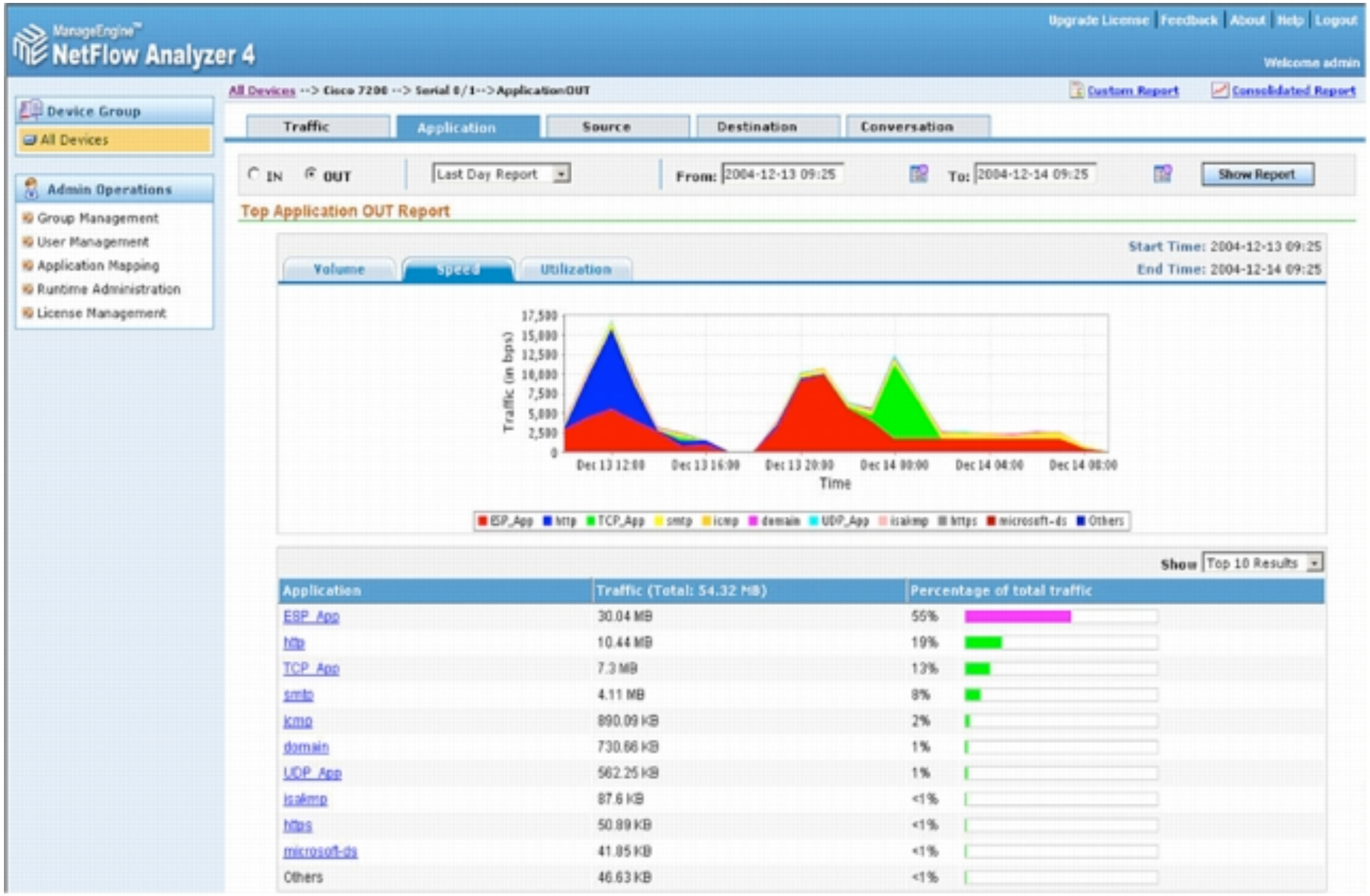
Filtering flows means: discard flows based on some criteria such as:

- Flow duration (discard flows that lasted less than X seconds)
- Flow src/dest (ignore flows containing broadcast addresses)
- Flow ports (ignore flows originated by port X)

Note that:

- Filtering != aggregation: they do two different jobs
- Filtering and aggregation can coexist
- Filtering is usually applied before aggregating flows and not after.

NetFlow Traffic Report Example



All Devices --> Cisco 7200 --> Serial 0/1-->ApplicationOUT

○ IN ● OUT

Last Day Report

From: 2004-12-13 09:25 To: 2004-12-14 09:25

Top Application OUT Report

Start Time: 2004-12-13 09:25

End Time: 2004-12-14 09:25

Show Top 10 Results

Application	Traffic (Total: 54.32 MB)	Percentage of total traffic
ESP_App	30.04 MB	55%
http	10.44 MB	19%
TCP_App	7.3 MB	13%
smtp	4.11 MB	8%
icmp	890.09 KB	2%
domain	730.66 KB	1%
UDP_App	562.25 KB	1%
isakmp	87.6 KB	<1%
https	50.89 KB	<1%
microsoft-ds	41.85 KB	<1%
Others	46.63 KB	<1%

Flows and Security

NetFlow/IPFIX can be used for security and not just for traffic accounting:

- Portscan/portmap detection
- Detect activities on suspicious ports
- Identify sources of spam, unauthorised servers (e.g. file servers)

Flows and Security: Portmap Scan

```
Start         SrcIPaddress      SrcP    DstIPaddress DstP  P Pkts

10:53:42.50  165.132.86.201     9781    128.146.0.76  111  6     1
10:53:42.54  165.132.86.201     9874    128.146.0.7   111  6     1
10:53:42.54  165.132.86.201     9982    128.146.0.80  111  6     1
10:53:42.54  165.132.86.201     9652    128.146.0.74  111  6     1
10:53:42.54  165.132.86.201     9726    128.146.0.75  111  6     1
10:53:42.54  165.132.86.201     9855    128.146.0.77  111  6     1
10:53:42.58  165.132.86.201    10107    128.146.0.82  111  6     1
```

Short timeframe, same IP source, different IP targets same port (111=RPC port).

Flows and Security: Backdoor Search

```
Start         SrcIPaddress   SrcP  DstIPaddress    DstP  P Pkts

19:08:40      165.132.86.201 8401  128.146.172.232 1524  6  19
19:08:40      165.132.86.201 8422  128.146.172.230 1524  6  16
19:08:40      165.132.86.201 8486  128.146.172.234 1524  6  19
19:08:40      165.132.86.201 8529  128.146.172.236 1524  6  10
19:08:41      165.132.86.201 8614  128.146.172.237 1524  6  16
19:08:41      165.132.86.201 8657  128.146.172.238 1524  6  22
```

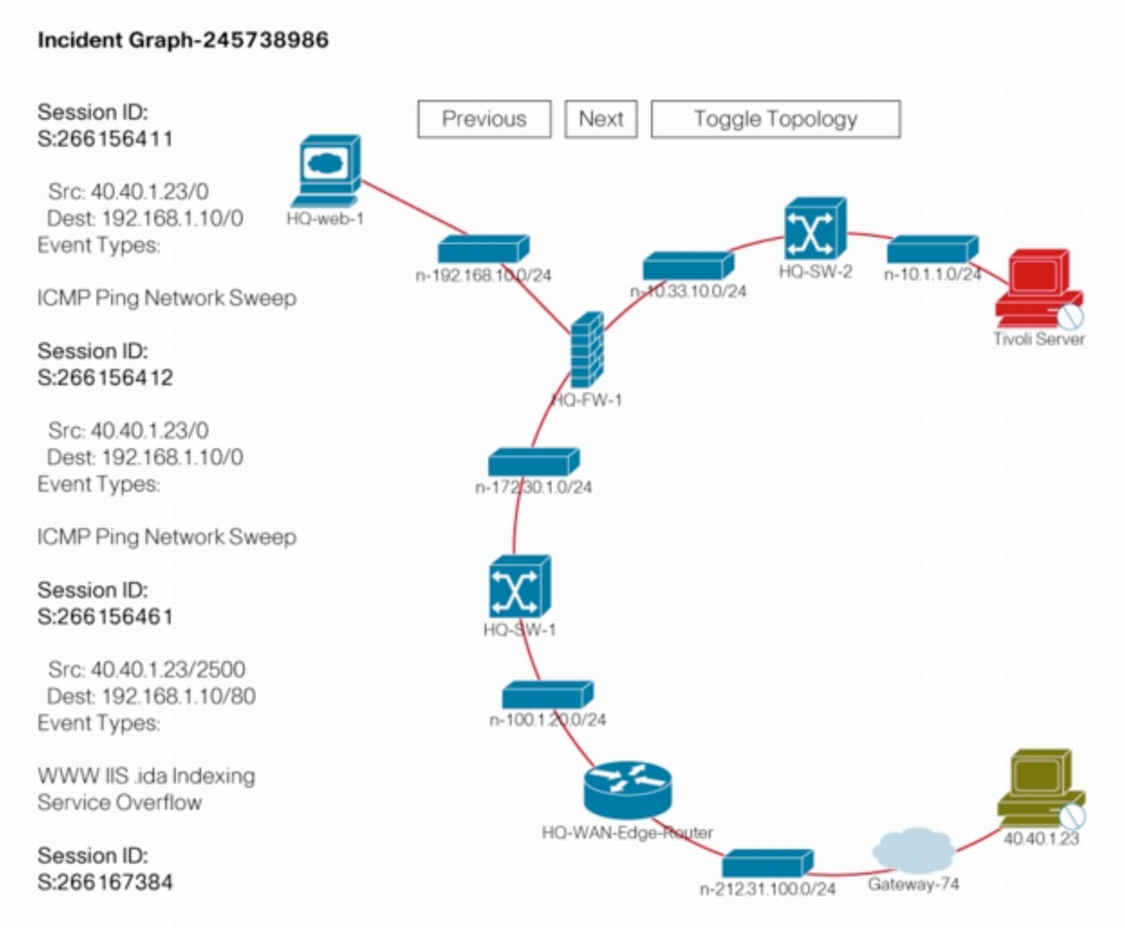
Same as portmap scan, targeting port 1524 (trinoo backdoor port [http://www.auditmypc.com/port/tcp-port-1524.asp]).

Flows and Security: Intrusion Detection

Simple flow-based IDS system:

- Flows with excessive octet or packet count (floods).
- IP sources contacting more than N destinations – host scanning.
- IP sources contacting more than M destination ports on a single host (for ports 0-1023) – port scanning.

Incident Report and NetFlow



Incident Graph-245738986

Session ID:
S:266156411

Src: 40.40.1.23/0
Dest: 192.168.1.10/0
Event Types:

ICMP Ping Network Sweep

Session ID:
S:266156412

Src: 40.40.1.23/0
Dest: 192.168.1.10/0
Event Types:

ICMP Ping Network Sweep

Session ID:
S:266156461

Src: 40.40.1.23/2500
Dest: 192.168.1.10/80
Event Types:

WWW IIS .ida Indexing
Service Overflow

Session ID:
S:266167384

sFlow

Driving Forces Towards sFlow

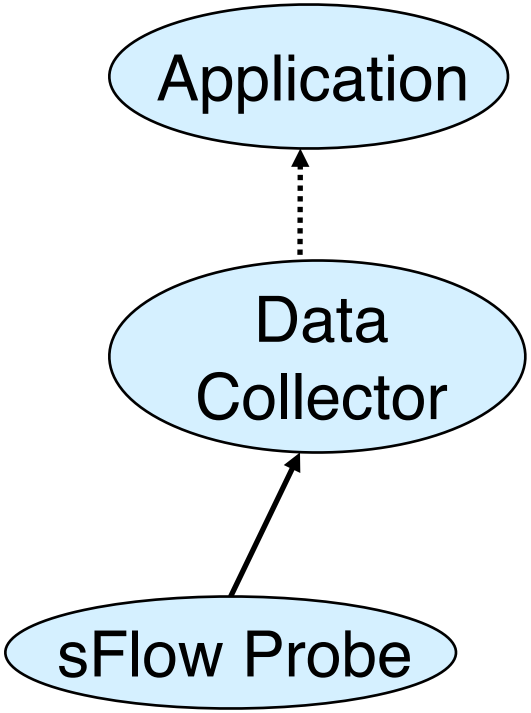
- Cost
  - Monitoring switched networks required multiple expensive probes.
  - Embedded monitoring solutions required extra hardware &/or software.
  - Traffic analysis solutions were costly.
  - Administrative costs of managing additional equipment.
- Impact to network performance
  - Switching performance impacted by measuring traffic flows.
  - Excessive network bandwidth used to export flow data.
- Poor scalability of monitoring system
  - Cannot keep up with Gigabit speeds.
  - Cannot build a network-wide traffic view for large, heavily used networks.

sFlow Principles

- Don’t pretend to be as fast as the monitored network: you will loose data anyway.
- Even if you can monitor everything you’ll run into trouble handling all the generated flows.
- Analyse 1 packet each X packets (sampling).

sFlow Architecture

- The probe samples traffic.
- Sampled packets are sent (in sFlow format) to the collector.
- Periodically the probe sends the collector interface statistics (SNMP MIB-II counters) inside sFlow packets. The packets are used to “scale” traffic.



sFlow Specification [1/4]

- Specified in RFC 3176 (Informational RFC) proposed by InMon Inc.
- It defines:
  - sFlow packets format (UDP, no SNMP).
  - A SNMP MIB per accessing sFlow collected data (http://support.ipmonitor.com/mibs/SFLOW-MIB/tree.aspx).
- The sFlow architecture is similar to NetFlow: the probe sends sFlow packets to the collector.

sFlow Specification [2/4]

- The sFlow probe is basically a sniffer that captures 1 out of X packets (e.g. 1:400).
- Such packets is sent to the collector coded in sFlow format.
- Periodically the probes sends other sFlow packets that contain network interface statistics (e.g. SNMP MIB-II interface traffic counters) used to scale collected data.

sFlow Specification [3/4]

- sFlow Packet Sample
  - Packet captured to the snaplen and complemented with metadata (e.g. port on which the packet has been captured).
- sFlow Counter Sample
  - SNMP MIB-II interface counters
  - Ethernet Counters

sFlow Specification [4/4]

- Using statistical formula it is possible to produce very precise traffic reports.
- % Sampling Error <= 196 * sqrt(1 / number of samples) [http://www.sflow.org/packetSamplingBasics/]
- sFlow is scalable (you just need to increase the sampling ration) even on 10 Gb networks or more.
- ntop.org is part of the sFlow.org consortium.

Caveat

- NetFlow != sFlow
- NetFlow flows have nothing in common with sFlow flows
  - In sFlow a packet sample is a flow
  - In NetFlow/IPFIX a flow is a 5-tuple with counters

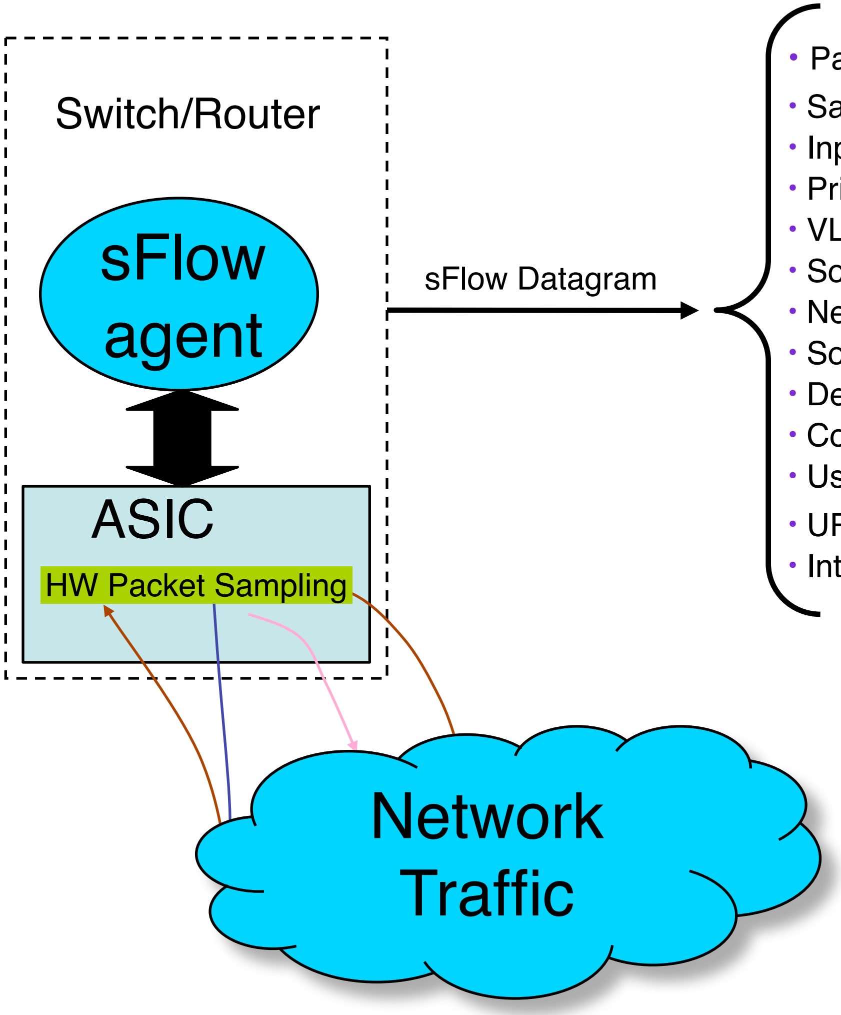
sFlow Packet [1/2]

```
struct sample_datagram_v5 {
   address agent_address          /* IP address of sampling agent,
                                     sFlowAgentAddress. */
   unsigned int sub_agent_id;     /* Used to distinguishing between datagram
                                     streams from separate agent sub entities
                                     within an device. */
   unsigned int sequence_number;  /* Incremented with each sample datagram
                                     generated by a sub-agent within an
                                     agent. */
   unsigned int uptime;           /* Current time (in milliseconds since device
                                     last booted). Should be set as close to
                                     datagram transmission time as possible.
                                     Note: While a sub-agents should try and
                                           track the global sysUptime value
                                           a receiver of sFlow packets must
                                           not assume that values are
                                           synchronised between sub-agents. */
   sample_record samples<>;        /* An array of sample records */
}
```

sFlow Packet [2/2]

```
struct flow_sample {
   unsigned int sequence_number;  /* Incremented with each flow sample
                                     generated by this source_id.
                                     Note: If the agent resets the
                                           sample_pool then it must
                                           also reset the sequence_number.*/
   sflow_data_source source_id;   /* sFlowDataSource */
   unsigned int sampling_rate;    /* sFlowPacketSamplingRate */
   unsigned int sample_pool;      /* Total number of packets that could have
                                     been sampled (i.e. packets skipped by
                                     sampling process + total number of
                                     samples) */
   unsigned int drops;            /* Number of times that the sFlow agent
                                     detected that a packet marked to be
                                     sampled was dropped due to
                                     lack of resources. The drops counter
                                     reports the total number of drops
                                     detected since the agent was last reset. */

   interface input;               /* Interface packet was received on. */
   interface output;              /* Interface packet was sent on. */

   flow_record flow_records<>;    /* Information about a sampled packet */
}
```

sFlow Summary



- Packet header (eg MAC,IPv4,IPv6,IPX,AppleTalk,TCP,UDP, ICMP)
- Sample process parameters (rate, pool etc.)
- Input/output ports
- Priority (802.1p and TOS)
- VLAN (802.1Q)
- Source/destination prefix
- Next hop address
- Source AS, Source Peer AS
- Destination AS Path
- Communities, local preference
- User IDs (TACACS/RADIUS) for source/destination
- URL associated with source/destination
- Interface statistics (RFC 1573, RFC 2233, and RFC 2358)

- Low cost
- No impact to performance
- Minimal network impact
- Scalable
- Quantitative measurements

Integrated Network Monitoring

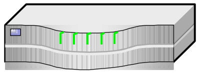
sFlow enabled switches

Core network switches

RMON enabled switches

L2/L3 Switches

NetFlow enabled routers

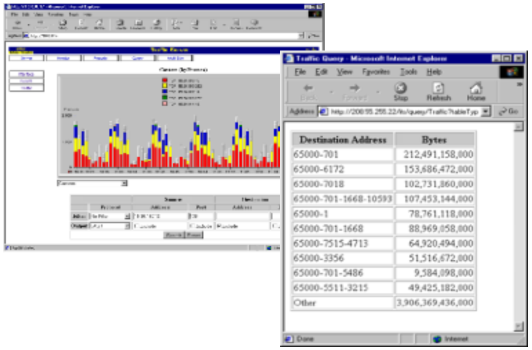


sFlow

RMON

NetFlow

Traffic Analysis & Accounting Solutions


Traffic Query - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Forward
Stop
Refresh
Home
Address
http://208.55.255.22/tto/query/Traffic?tableTyp
Go
Done
Internet

Destination Address	Bytes
65000-701	212,491,158,000
65000-6172	153,686,472,000
65000-7018	102,731,860,000
65000-701-1668-10593	107,453,144,000
65000-1	78,761,118,000
65000-701-1668	88,969,058,000
65000-7515-4713	64,920,494,000
65000-3356	51,516,672,000
65000-701-5486	9,584,098,000
65000-5511-3215	49,425,182,000
Other	3,906,369,436,000

- Network-wide, continuous surveillance
    - 20K+ ports from a single point
- Timely data and alerts
    - Real-time top talkers
    - Site-wide thresholds and alarms
- Consolidated network-wide historical usage data

sFlow vs. SNMP

	sFlow	SNMP
Model	Push	Poll
Interface Group MIB-II Counters	Yes (Counter Sample)	Yes
Packet Visibility	Yes (Packet Sample)	No

In essence sFlow is an evolution of SNMP that implements port traffic visibility.

sFlow vs. NetFlow

	sFlow	NetFlow
Native Environment	Switched	Routed
Operational Speed	MultiGigabit	1-10 Gbit
Sampling	Always	Sometimes
Monitoring	Statistical	Accurate (without packet loss)

Radius [RFC 2139, 1997]

Radius is acronym for Remote Authentication Dial In User Service (RADIUS) specified in the following RFC:

- Authentication Protocol

  Rigney, C., Rubens, A., Simpson, W, and Willens, S.; Remote Authentication Dial In User Service (RADIUS), RFC 2138, January 1997.

- Data Accounting

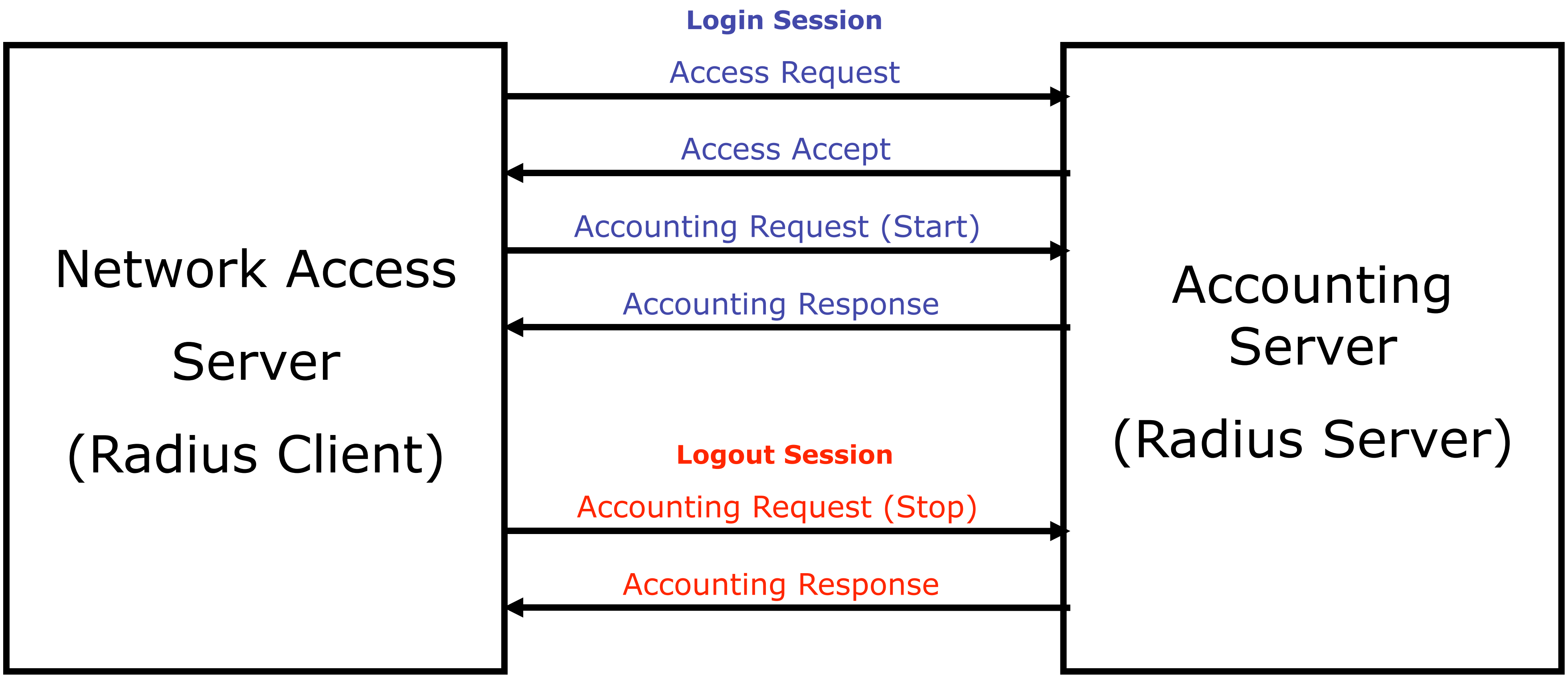
  Rigney, C.; RADIUS Accounting, RFC 2139, January 1997.

Radius

Radius is important because:

- It is the most used protocol for implementing authentication on network devices.
- Used for billing activities on wired lines (e.g. ADSL, Modem).
- Allows accounting for connection duration or data volumes.
- Supported by all the network devices (Low-end excluded).

The Radius Protocol



Login Session

Access Request

Access Accept

Accounting Request (Start)

Accounting Response

Logout Session

Accounting Request (Stop)

Accounting Response

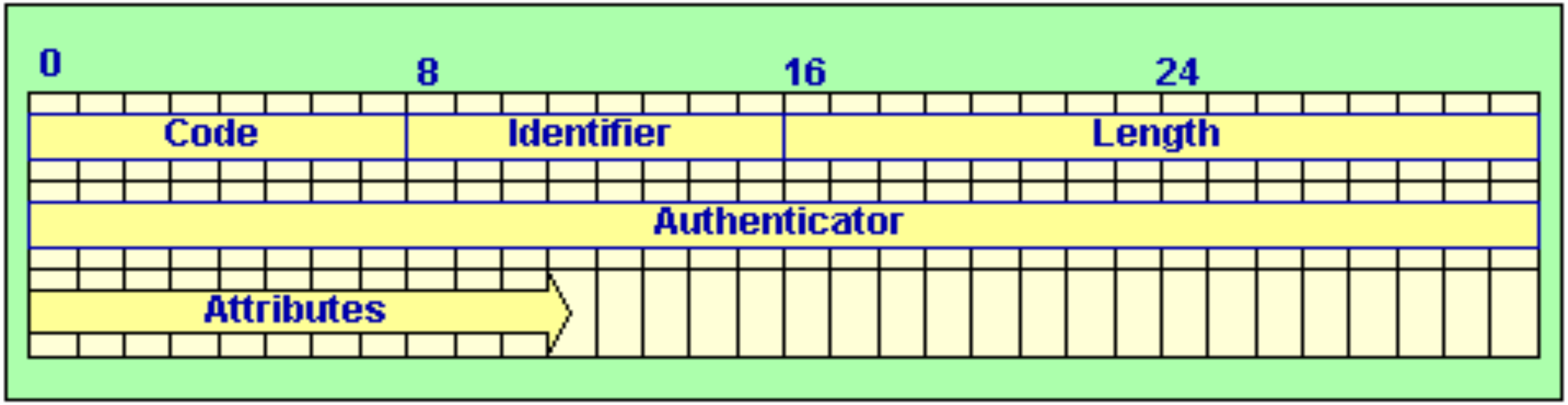
Network Access Server

(Radius Client)

Accounting Server

(Radius Server)

Radius Protocol: Messages



- *Code*: Byte containing command/reply RADIUS.
- *Identifier:* Byte identifies command/reply RADIUS.
- *Length*: Packet length.
- *Authenticator*: Value used for authenticating RADIUS server reply.
- *Attributes*: Attributes of command/reply.

Radius Protocol: Primitives

access-request, (client->server):

- Request to access to network services (e.g. user authentication).
- Possible reply:
  - access-accept, (server->client).
  - access-reject, (server->client).
  - access-challenge, (server->client): used for CHAP authentication.

accounting request, (client->server)

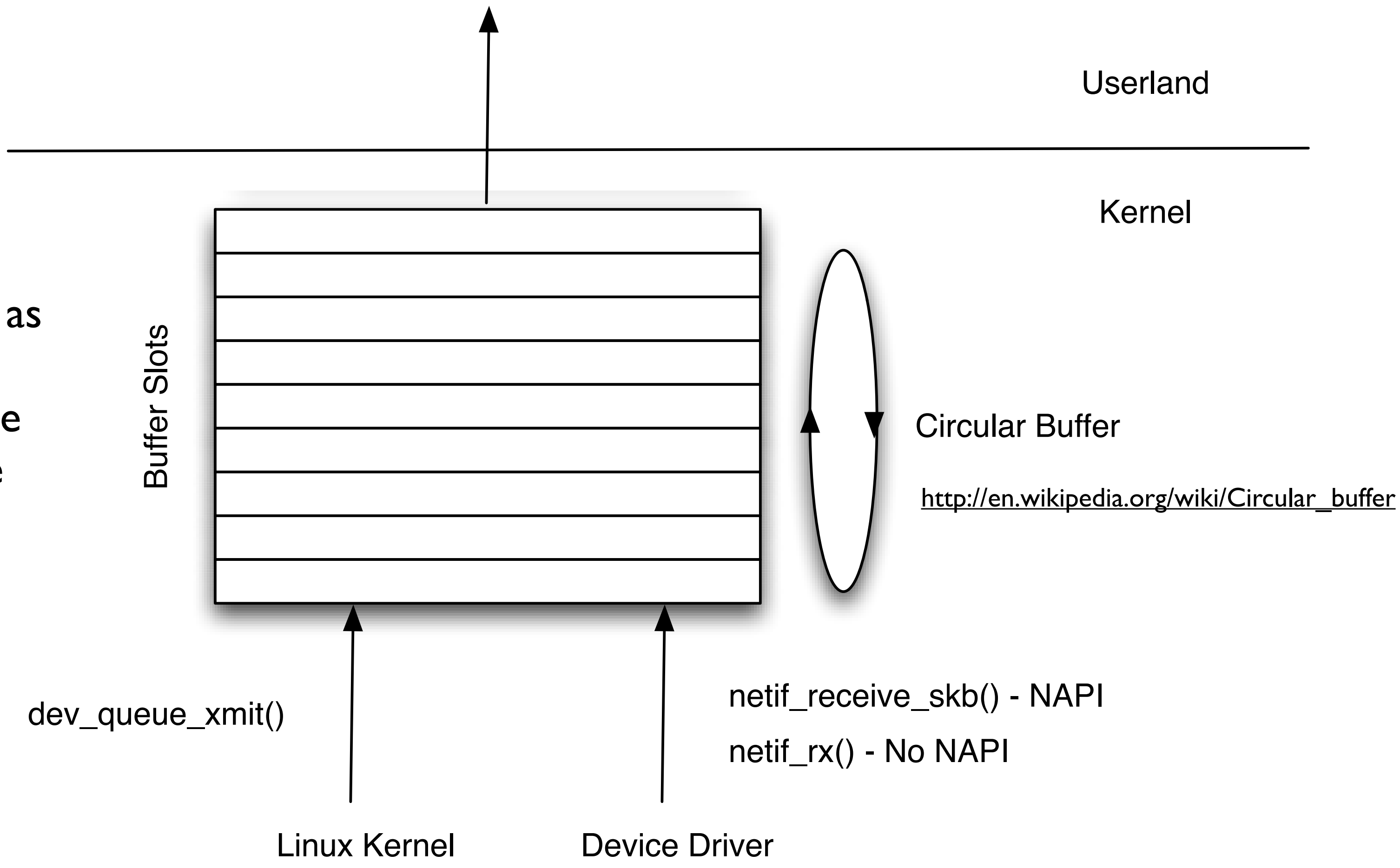
- Request to write accounting data on the accounting server.
- Replies:
  - Accounting response, (server->client)

Interim update (client->server)

- Send partial accounting updates

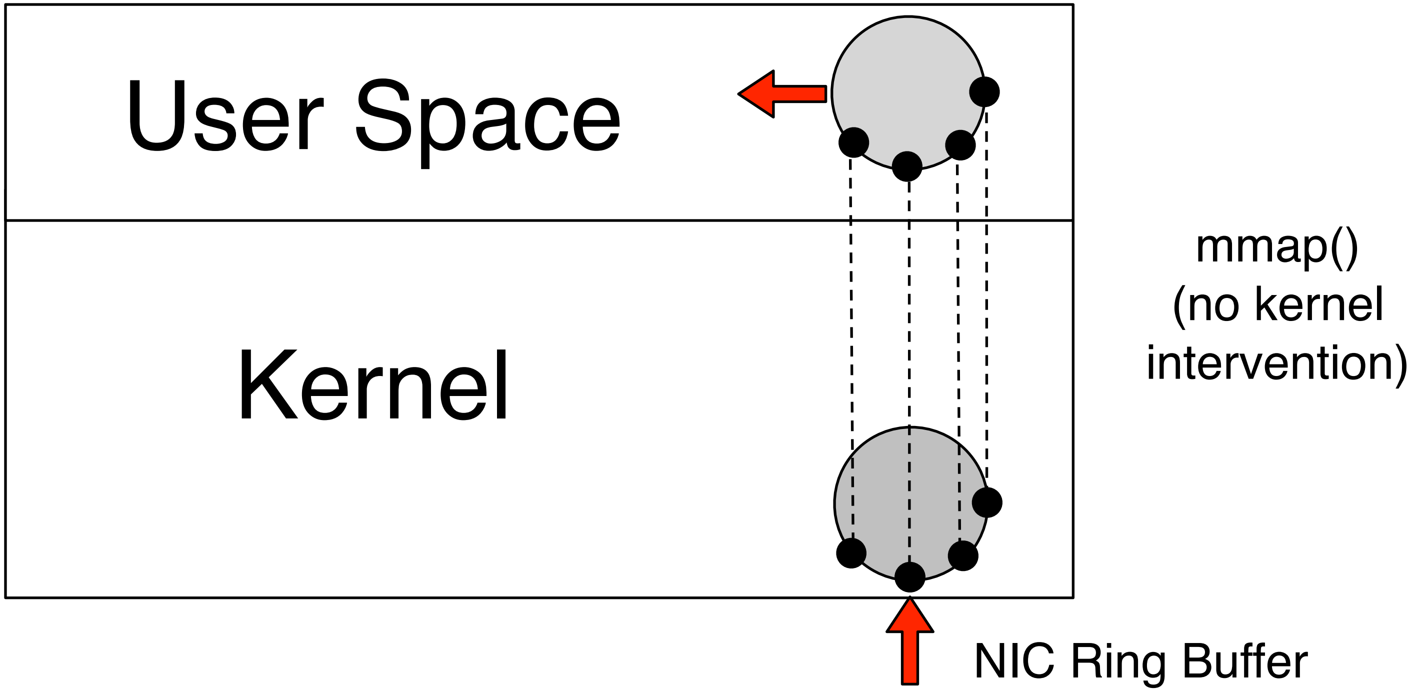
Packet Capture

Linux Packet Capture

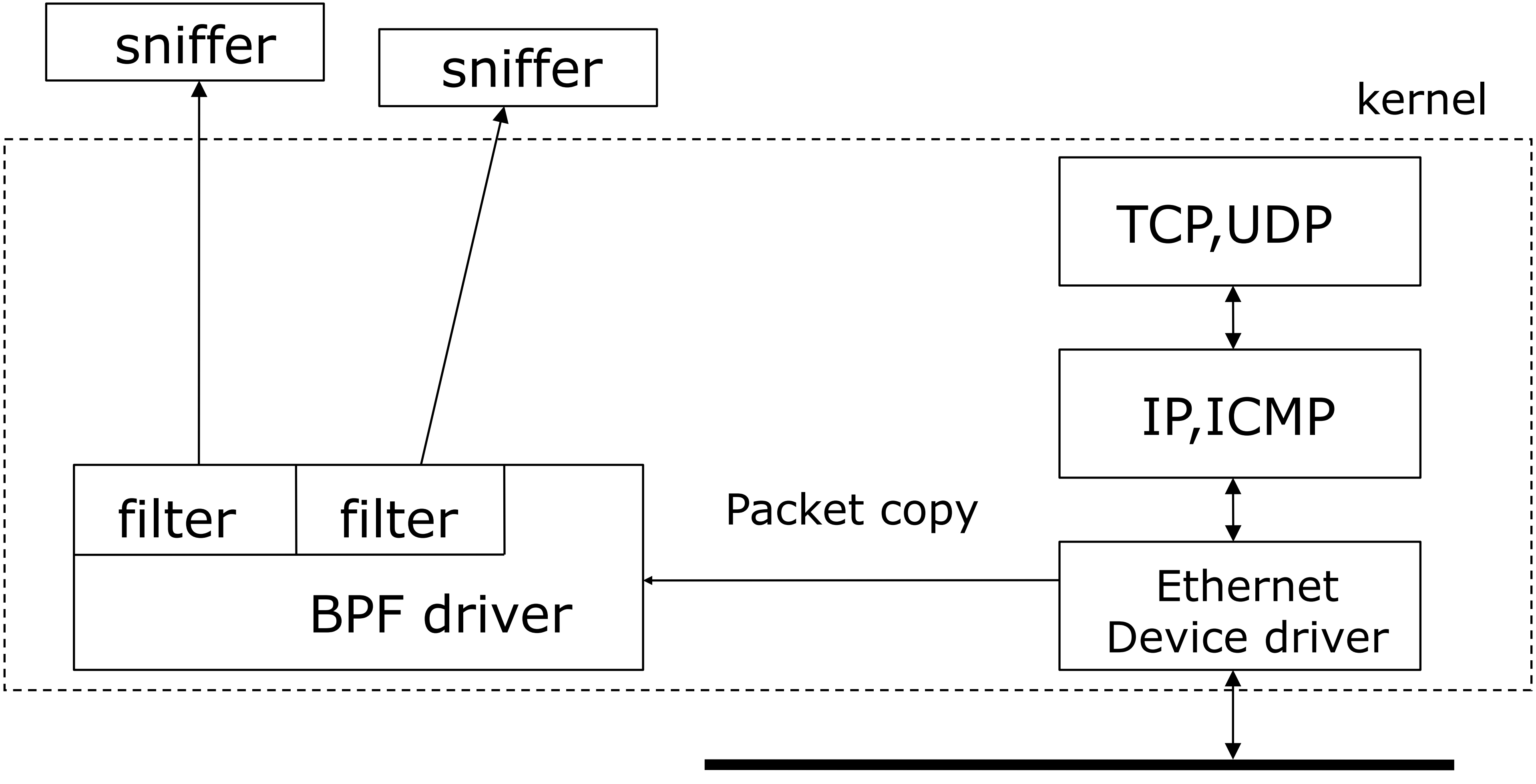


Slot size is dynamic as they are filled in according to the size of packets that have been received

Packet Capture: PF_RING



Packet Capture: libpcap



Libpcap: Usage Example [1/2]

```
pcapPtr = pcap_open_live(deviceName,
     maxCaptureLen, setPromiscousMode,
     pktDelay, errorBuffer);

while(pcap_dispatch(pcapPtr, 1,
          processPacket, NULL) != -1);

void processPacket(u_char *_deviceId,
     const struct pcap_pkthdr *h,
     const u_char *p) {
     …
}
```

See also: http://jnetpcap.sourceforge.net/

Libpcap: Usage Example [2/2]

```
int main(int argc, char* argv[]) {
 /* open a network interface */
descr = pcap_open_live(dev,BUFSIZ,0, 1,errbuf);

/* install a filter */
pcap_compile(descr,&fp,”dst port 80”,0,netp);
pcap_setfilter(descr,&fp);

while (1) {
  /* Grab packets forever */
  packet = pcap_next(descr,&hdr);
  /* print its length */
  printf("Grabbed packet of length %d\n", hdr.len);
 }
}
```

Scapy [1/2]

- Scapy is a python library (https://github.com/secdev/scapy) able to create, send/receive, manipulate network packets.
- It can be used as traffic generation tool to:
  - Test applications you develop
  - Forge packets for prototyping new protocols
- It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery …

Scapy [2/2]

```
 >>> send(IP(dst="10.1.99.2")/ICMP()/"HelloWorld") .
 Sent 1 packets.
>>> h.show()
###[ IP ]###
version= 4L
ihl= 5L
tos= 0x0
len= 38
..
ttl= 64
proto= icmp
chksum= 0x83d7
src= 10.1.99.2
dst= 10.1.99.25
\options\
###[ ICMP ]###
type= echo-reply
code= 0
chksum= 0x0
id= 0x0
seq= 0x0
###[ Raw ]###
load= 'HelloWorld'
###[ Padding ]###
load= '\x00\x00\x00\x00\xe7\x03N\x99' >>>
```

Common Problems with Packet Capture

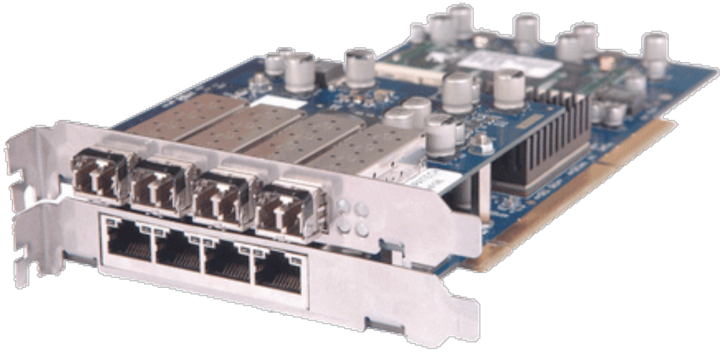
- Security issues
  - All the network traffic is captured and not just the one sent to the sniffing host
  - If there is a switched network it is captured only a part of traffic (ARP poisoning)
  - Usability limited to who have root capabilities
    NOTE: this append also with ICMP command (e.g. ping) and therefore they are set with the setuid.

- Performance
  - Sniffer implies also the cpu load because all the captured packets must be analysed by the program and not just those directed to the host

Packet Capture: Solutions

1. Use of NICs that feature an NPU (Network Process Unit). Every modern NIC has a limited NPU (multicast and ethernet).
2. Use a programmable card (e.g. Napatech).
3. Execution of traffic accounting/management code directly on the NIC.
4. High-speed access (via mmap()) to packets directly on the NIC via the PCI bus.

FPGA-based Packet Capture Cards



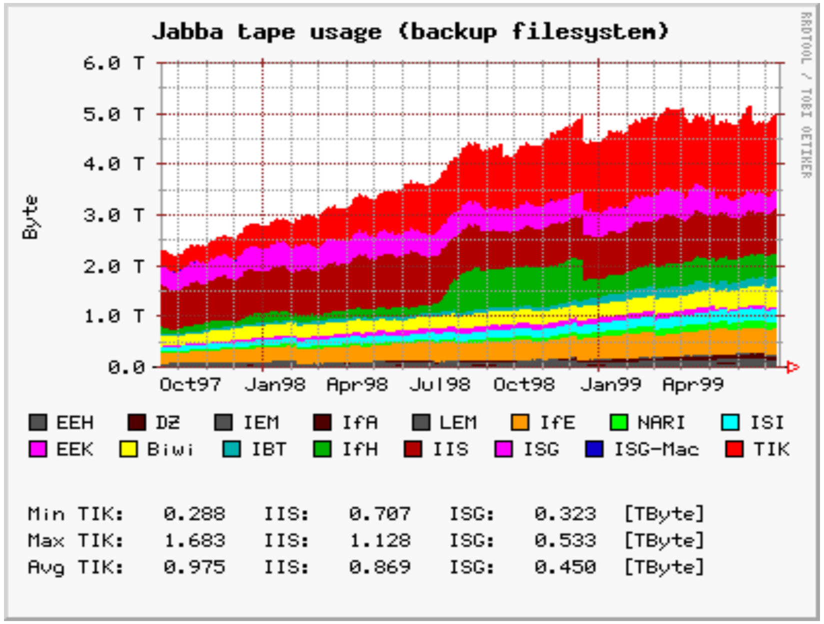
Capturing Traffic on WiFi

- Some WiFi features might prevent you from observing all traffic.
- Wireless Isolation: it confines and restricts clients connected to the Wi-Fi network. They can’t interact with devices connected to the wired network, nor can they communicate with each other. They can <u>only</u> access the Internet.

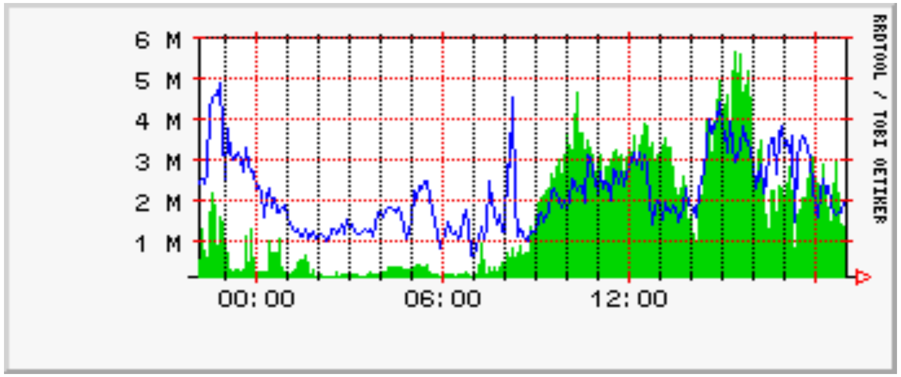
Data Collection: RRD

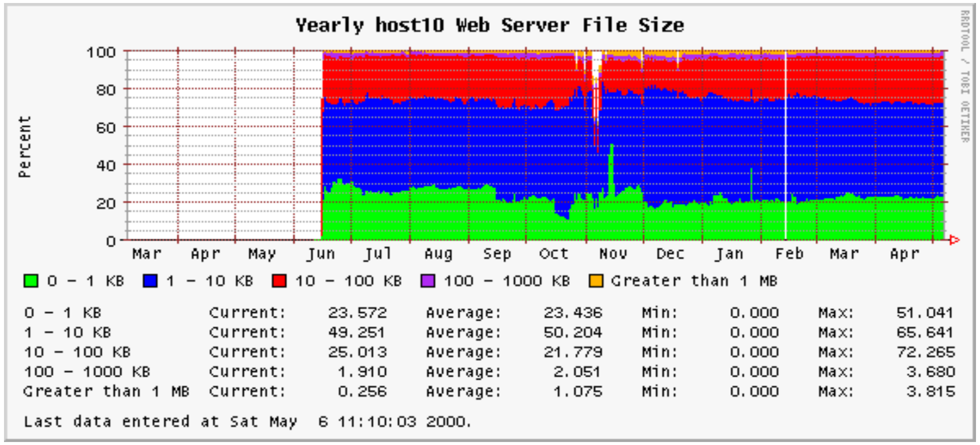
- RRD [http://www.rrdtool.org/]
  - Round Robin Database: Tool used to store and display data on which is based MRTG on.
  - Data are stored in “compress” format and they don’t grow with time (automatic data aggregation) and always equal file size.
  - Perl/C interface to access to data and produce graphs.

Data Collection: RRD Graphs



```
Min TIK:   0.288   IIS:   0.707   ISG:   0.323  [TByte]
Max TIK:   1.683   IIS:   1.128   ISG:   0.533  [TByte]
Avg TIK:   0.975   IIS:   0.869   ISG:   0.450  [TByte]
```





```
0 - 1 KB           Current:   23.572   Average:   23.436   Min:    0.000   Max:   51.041
1 - 10 KB          Current:   49.251   Average:   50.204   Min:    0.000   Max:   65.641
10 - 100 KB        Current:   25.013   Average:   21.779   Min:    0.000   Max:   72.265
100 - 1000 KB      Current:    1.910   Average:    2.051   Min:    0.000   Max:    3.680
Greater than 1 MB  Current:    0.256   Average:    1.075   Min:    0.000   Max:    3.815

Last data entered at Sat May  6 11:10:03 2000.
```

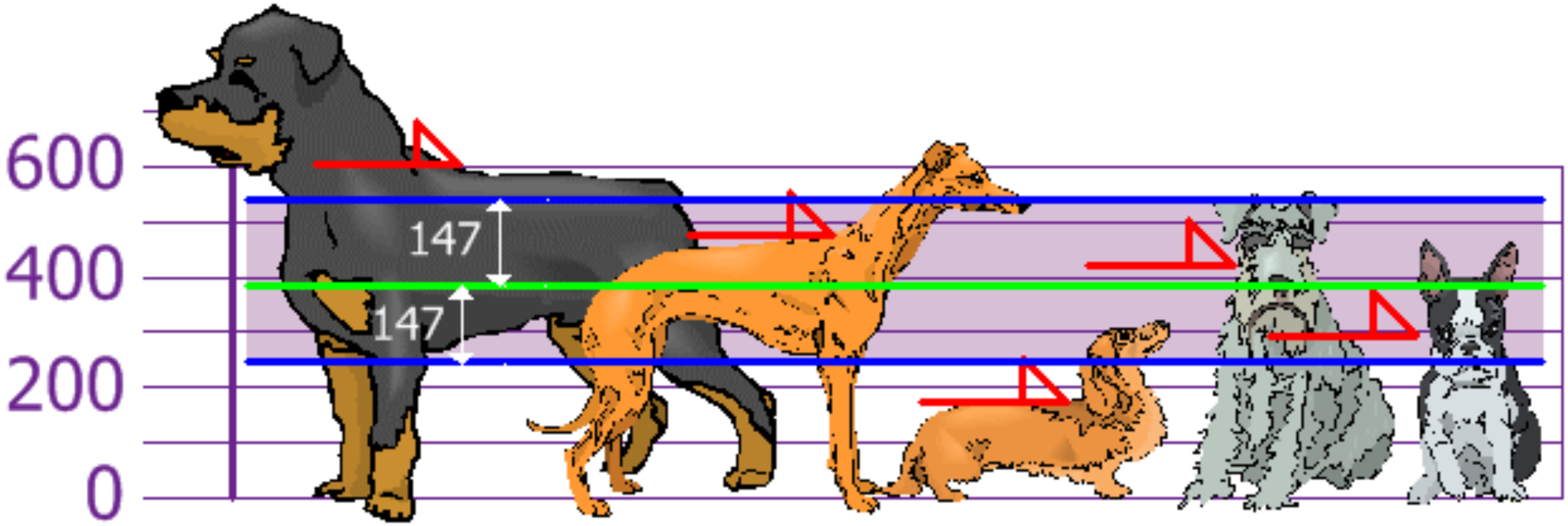
Some TimeSeries Databases

- InfluxDB (www.influxdb.org)
- Prometheus (https://prometheus.io)
- TimescaleDB (https://www.timescale.com/)
- QuestDB (https://questdb.io/)

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

Timeseries, Forecast, Anomaly Detection

Mean and Standard Deviation



https://www.mathsisfun.com/data/standard-deviation.html

Variance The average of the sum of **squared** differences from the mean

StdDev The square root of the variance

In the above example: Mean=394, StdDev=147

Using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large (> mean+stddev)) or extra small (< mean-stddev)).

Percentile [1/2]

- Percentile: the value below which a given percentage of observations in a group of observations falls.
- Example: 70th is the age under which 80% of the population falls (so not 80% of the max age).

- In networking a popular percentile is 95th

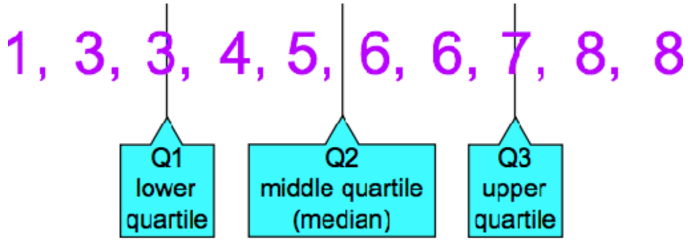
Percentile [2/2]

```
series = [13,43,54,34,40,56,34,61,34,23]

# Percentile we want to compute
percentile = 80

# Sort the data
sorted_series = sorted(series)

# Find the index in the sorted data that corresponds to the searched percentile
index = len(sorted_series)*(percentile/100)

# Round it to the nearest upper integer
rounded_index = int(index + 0.5)

# Find the element that identifies the percentile
pcentile = sorted_series[rounded_index-1]

print(sorted_series)
print("%uth percentile: %u" % (percentile, pcentile))
```

```
$ ./percentile.py
[13, 23, 34, 34, 34, 40, 43, 54, 56, 61]
80th percentile: 54
```

Quartiles

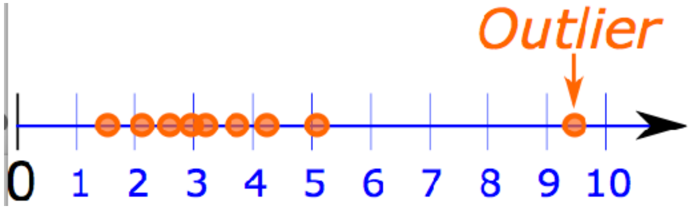
- A quartile is a split of series into quarters.



- Q_1 is defined as the 25th percentile
- Q_2 is defined as the 50th percentile
- Q_3 is defined as the 75th percentile

Outlier [1/2]

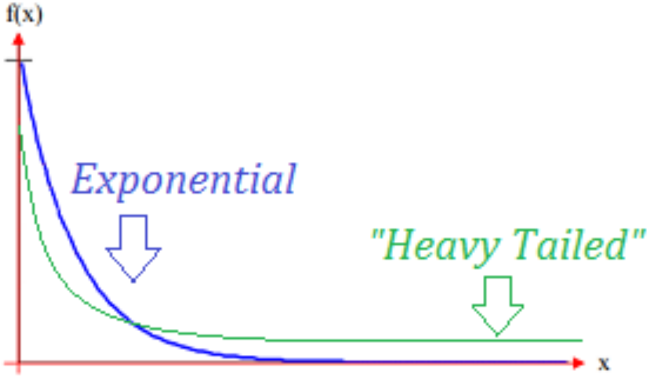
- Outliers are values that "lie outside" the other values.



- Interquartile Range (IQR): Q_3-Q_1
- In statistics a value is an outlier when it falls outside of the
  - lower fence: Q_1 - 1.5 * IQR
  - upper fence: Q_3 + 1.5 * IQR

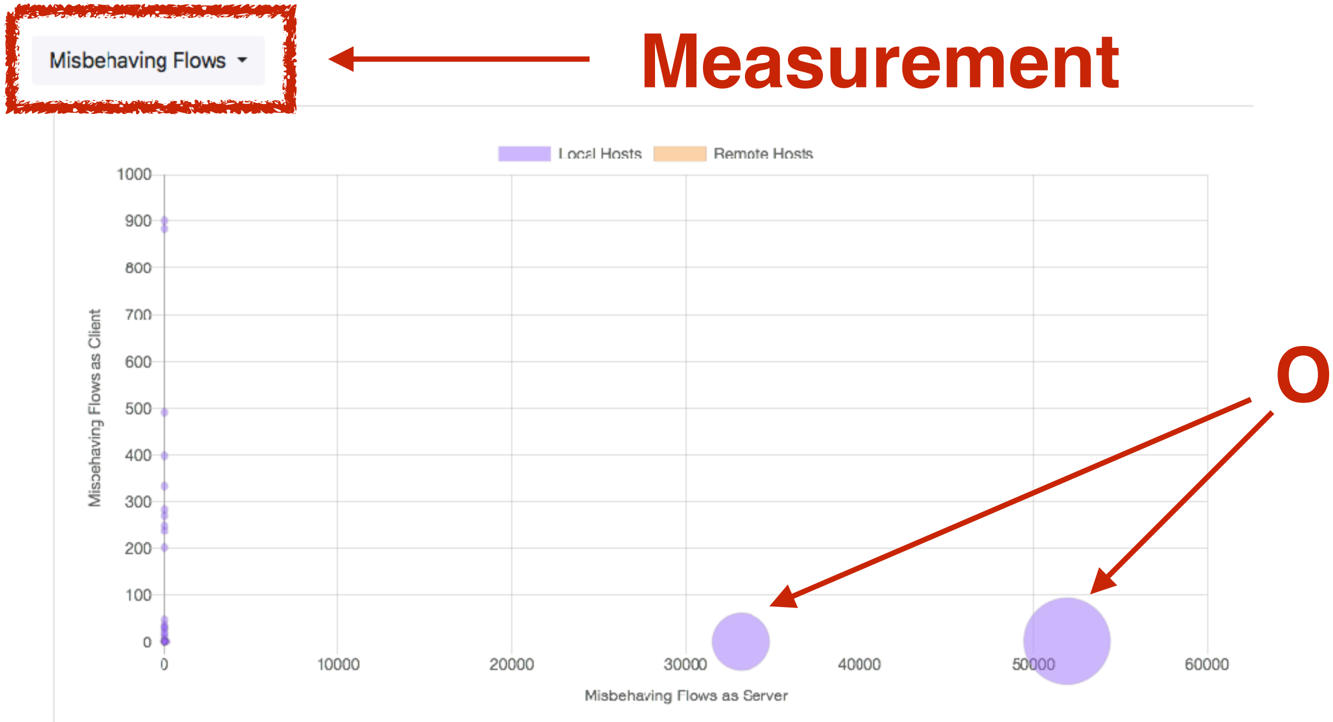
Outlier [2/2]

- Outliers are used to spot anomalous values often named as ‘aberrant’, i.e.departing from an accepted standard including:
  - Experimental/measurement errors
  - Heavy tailed distributions (i.e. that goes to zero very slowly) where one or more very large values will affect the statistics.



Finding Outliers [1/2]

- In networking, outliers can identify “special” entities for a given measurement.



Measurement

Outlier

Finding Outliers [2/2]

```
series = [ 0, 0, 0, 0, 6, 2, 0, 0, 0, 0, 0, 12, 0, 0, 33182, 51945 ]
sorted_series = sorted(series)

q1 = calc_percentile(sorted_series, 25)
q2 = calc_percentile(sorted_series, 50)
q3 = calc_percentile(sorted_series, 75)
iqr = q3-q1
k = 1.5
lower_outlier_limit = q1-k*iqr
upper_outlier_limit = q3+k*iqr

# Compute the mean
m = mean(series)

# Compute the standard deviation
std = stddev(series, m)

print("Original series:")
print(sorted_series)
print("Percentiles:     q1=%.1f, q2=%.1f, q3=%.1f" % (q1, q2, q3))
print("IQR:             %.1f" % iqr)
print("Mean +/- Stddev  [ %.1f ... %.1f ]" % ((m-std), (m+std)))
print("Outlier Formula: [ %.1f ... %.1f ]" % (lower_outlier_limit, upper_outlier_limit))
```

```
$ outlier.py
Original series:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 6, 12, 33182, 51945]
Percentiles:     q1=0.0, q2=0.0, q3=4.0
IQR:             4.0
Mean +/- Stddev  [ -9140.6 ... 19782.6 ]  <==== Best
Outlier Formula: [ -6.0 ... 10.0 ]        <==== Good
```

Finding Outliers Using Z-Score [1/2]

- Another method for find outliers is using the Z-Score defined as:
  - Z-Score = (Value - Mean) / StdDev
- A z-score tells you how many standard deviations a given value is from the mean: positive (above the mean), negative (below the mean).
- A typical outlier Z-Score threshold is 2.5:
  - Values < -2.5 or > 2.5 are considered outliers.
- Note: In statistics, there is <u>no difference</u> between the mean and average.

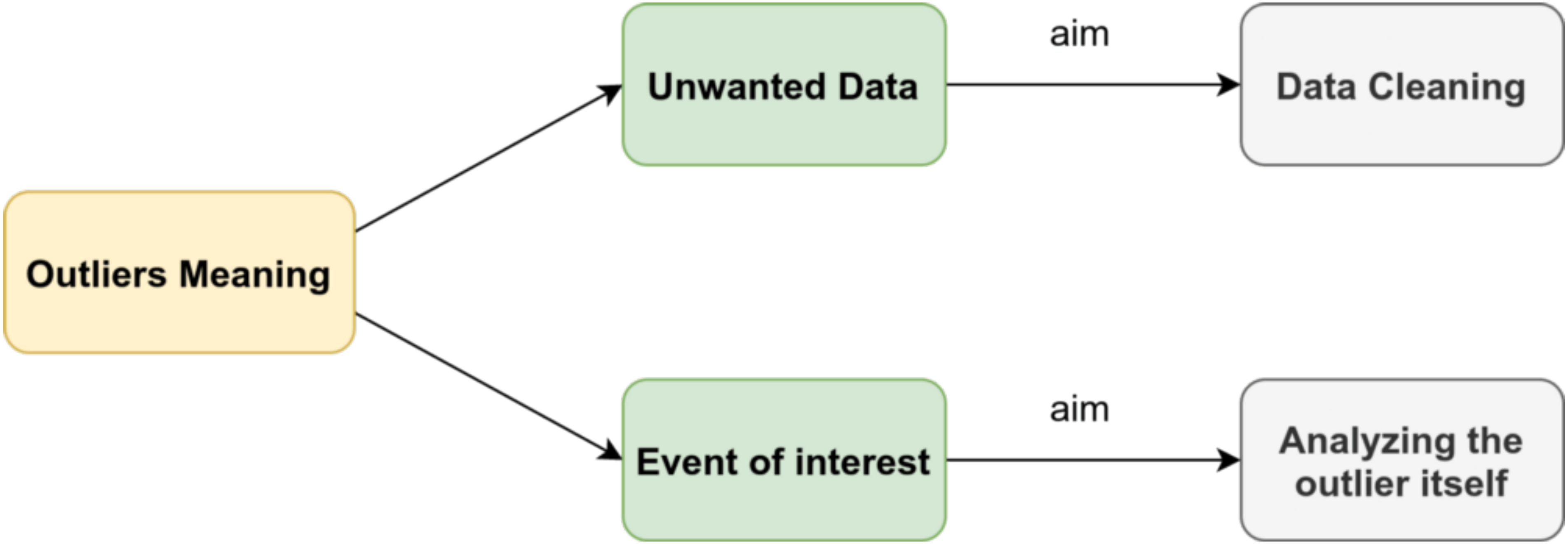
Finding Outliers Using Z-Score [2/2]

	Value	Z-Score	Is Outlier ?
	1	-1,03	
	3	-0,77	
	3	-0,77	
	4	-0,64	
	5	-0,51	
	2	-0,90	
Z-Score = (X – µ) / σ	6	-0,38	
	7	-0,25	
	30	**2,74**	Yes
	16	0,92	
Mean (µ)	7,70		
Standard Deviation (σ)	8,90		
Z-Score Threshold	2,50		

Anomaly Detection: Goal

Anomaly: an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.

Anomaly Detection: Process



Definitions [1/2]

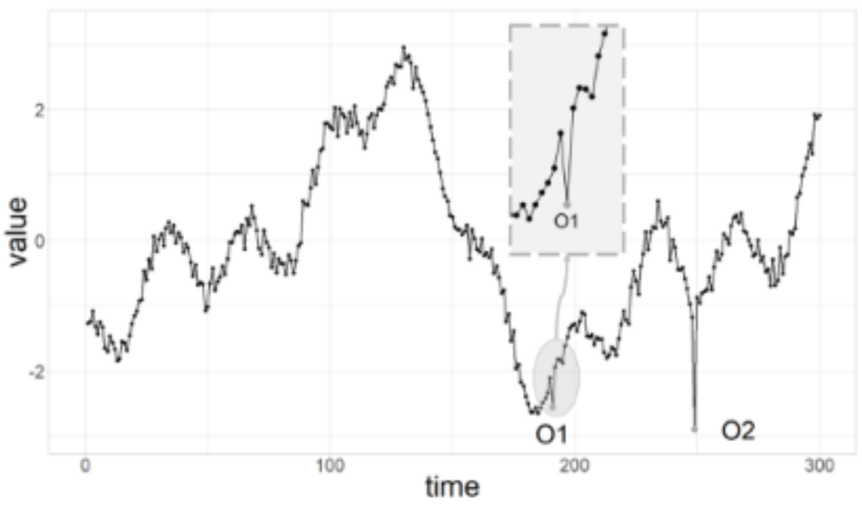
- Series: an ordered sequence of numbers.
- Order: the index of a number in the series.
- Timeseries: a series of data points in time order.
- Observation: the numeric value observed (in reality) at a specified time.
- Forecast: estimation of an expected value (that we don’t know yet) at a specific time.

https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/

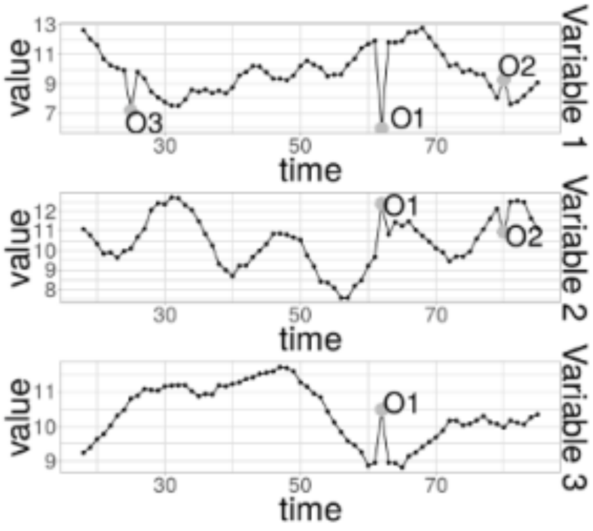
Definitions [2/2]

- Forecast Error: positive/negative difference of the observation with respect to the forecast. Usually the error is reported as square to obtain an always positive number.
- SSE: the sum of squared errors of a series SUM((observation$_i$ - forecast$_i$)^2)

Univariate vs Multivariate



(a) Univariate time series.



(b) Multivariate time series.

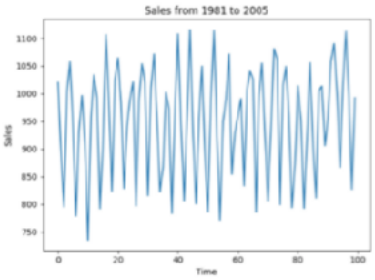
The term "univariate time series" refers to a time series that consists of single (scalar) observations recorded sequentially over equal time increments. Example: daily temperature.

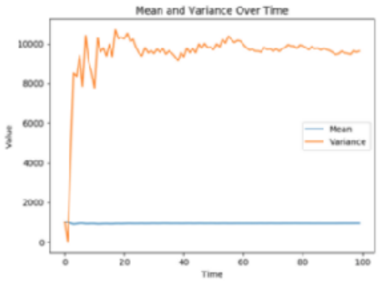
A multivariate time series has more than one time-dependent variable. Each variable depends not only on its past values but also has some dependency on other variables. This dependency is used for forecasting future values. Example: weather forecast that depends on temperature, wind, humidity, cloud cover.

Stationary Time Series [1/2]

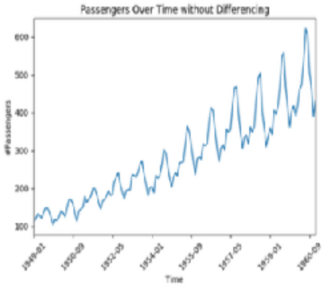
- A time series is stationary when its statistical properties (e.g. mean and variance) do not change overtime, i.e. if they have no trend or seasonality.

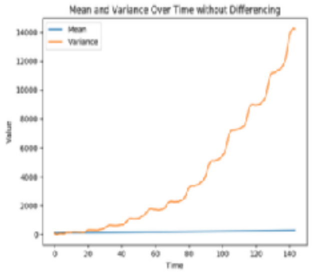
Stationary





Non Stationary

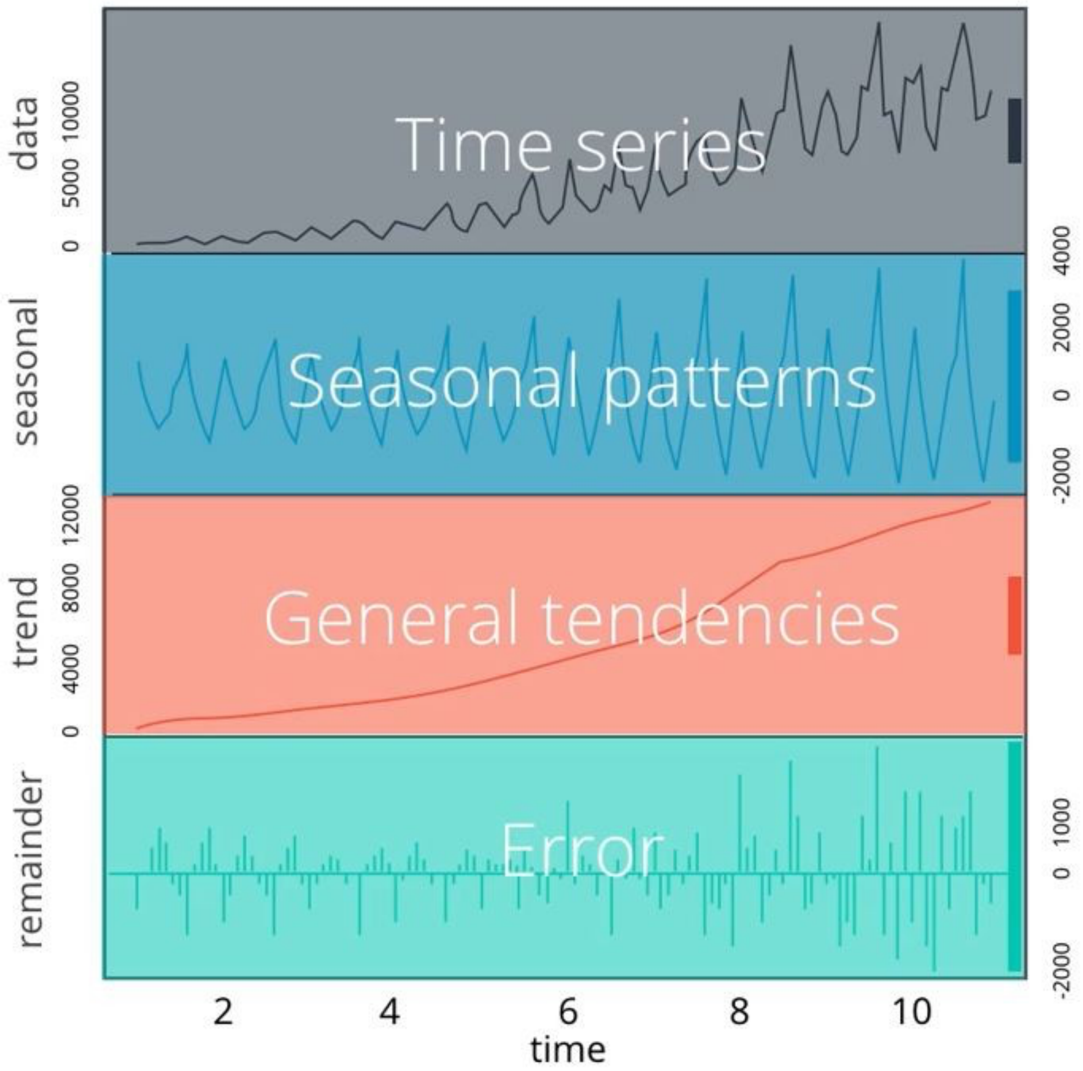




Stationary Time Series [2/2]

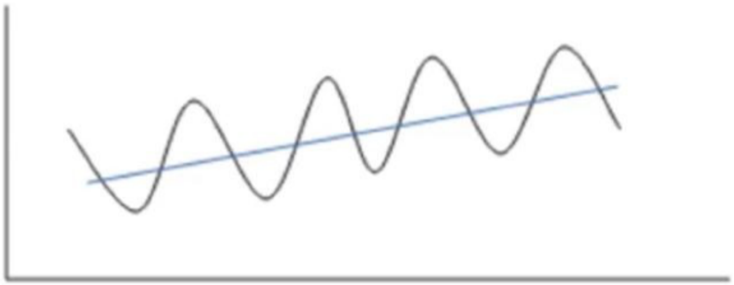
- A counter is non-stationary. It can be made stationary writing the time series as observation(t) - observation(t-1) [gauge].
- Stationarity is important as for forecasting we need some sort of invariance.
- Intuitively a time series variable is stationary about some equilibrium path if after a shock it tends to return to that path. A series is non-stationary if it moves to a new path after a shock. It is very hard to model the path of a variable that changes path if it is subject to some shock.

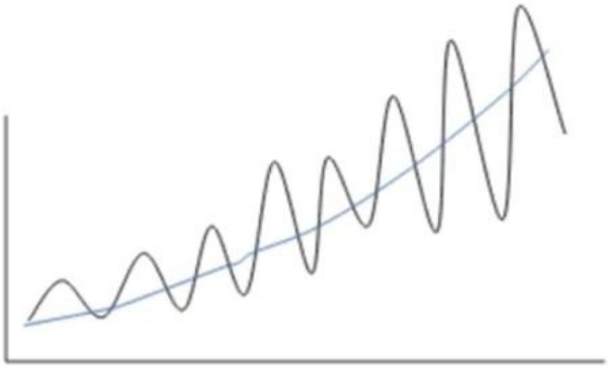
Timeseries Decomposition



Additive vs Multiplicative Methods

- The additive method is useful when the *trend and seasonal* variation is relatively constant over time $y_t = S_t + T_t + R_t,$
- The multiplicative method (right) is useful when the *trend and seasonal* variation increases or decreases in magnitude over time $y_t = S_t \times T_t \times R_t.$





Timeseries Prediction Methods

- ETS (Error, Trend and Seasonality) are applied in the smoothing method calculation.
- ARIMA (AutoRegressive Integrated Moving Average)
  - AR: a changing variable regresses to prior values
  - I: use differencing in raw value to implement stationarity
  - MA: incorporates the residual error between an observation and its real value.
- Other methods: Prophet (Meta), TimeGTP (AI-based).

"From ARIMA to TimeGPT — A New Era in Time Series Prediction", https://medium.com/@andrepedrinho/from-arima-to-timegpt-a-new-era-in-time-series-prediction-part-i-f65602fead04

Averages and Forecasts [1/3]

Given a series, there are several ways to predict (^) the value (y) of point x+1 ($y^{\wedge}_{x+1}$)

- Simple Average: the next point is the average of apps points of the series

```
def average(series):
    return float(sum(series))/len(series)
```

- Moving Average: same as simple average but computed only on the last n points that are more relevant than old ones

```
def moving_average(series, n):
    return average(series[-n:])
```

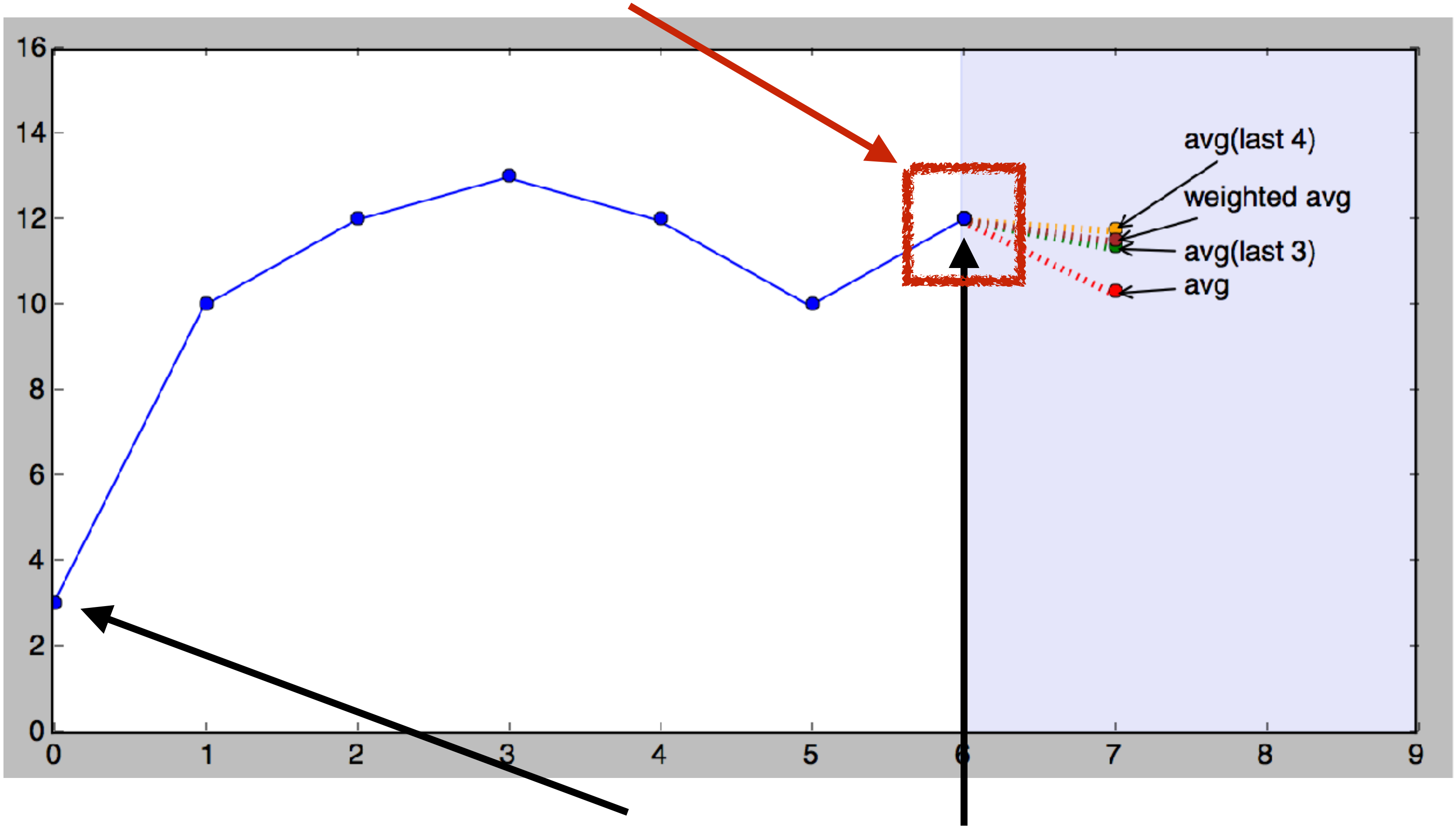
Averages and Forecasts [2/3]

– Weighted Moving Average: same as moving average with a weight assigned to the each point according to their age

```
# weighted average, weights is a list of weights
def weighted_average(series, weights):
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result

# >>> weights = [0.1, 0.2, 0.3, 0.4]
# >>> 10%, 20%, 30% and 40% to the last 4 points
# >>> weighted_average(series, weights)
# 11.5
```

Averages and Forecasts [3/3]

Last Observation



series = [3,10,12,13,12,10,12] (order is 7)

Single Exponential Smoothing [1/3]

- Exponential shooting was born in 1950s
- The formula below (Poisson, Holts or Roberts) states that the expected/smoothed value for y ($y^{\wedge}_{x}$) is the sum of two products (recursive formula):

$$y^{\wedge}_{x}=\alpha * y_{x} + (1-\alpha) * y^{\wedge}_{x-1}$$

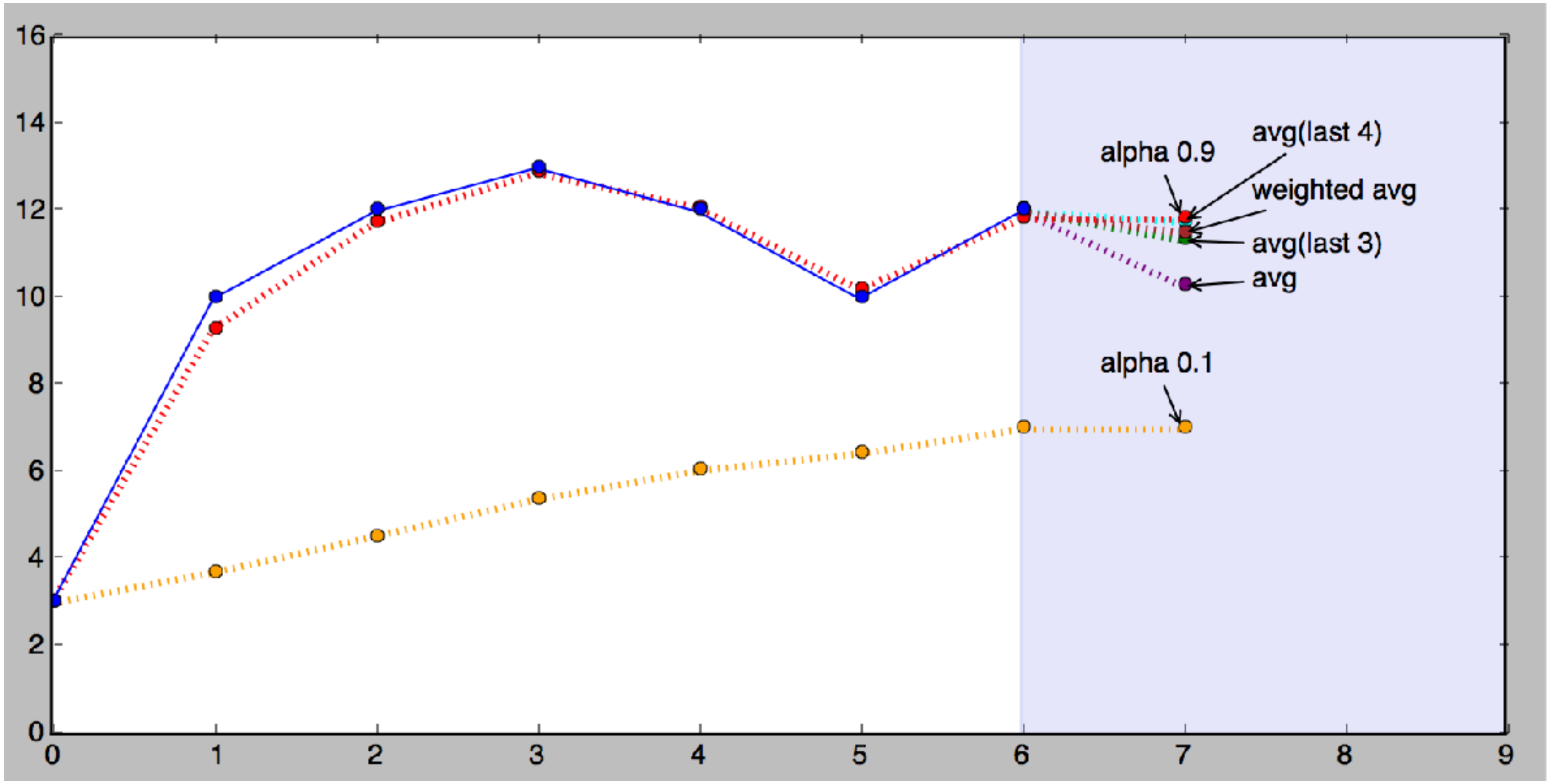
←NOTE: this is in essence a smoothing function rather than a prediction.

- α multiplied to the value of y_{x}
- and (1-α) multiplied by the expected value $y^{\wedge}_{x-1}$
- α: smoothing factor or “memory decay rate”: the higher α, the faster the method forgets.
- This formula in essence “predicts” the next value (1

Single Exponential Smoothing [2/3]

```
# given a series and alpha, return series of smoothed points
def exponential_smoothing(series, alpha):
    prediction = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        prediction.append(alpha * series[n] + (1 - alpha) * prediction[n-1])

    # Now append the “prediction”
    prediction.append(alpha * series[n] + (1 - alpha) * prediction[n])
    return prediction
# >>> exponential_smoothing(series, 0.1)
# [3, 3.7, 4.53, 5.377, 6.0393, 6.43537, 6.991833]
# >>> exponential_smoothing(series, 0.9)
# [3, 9.3, 11.73, 12.873000000000001, 12.0873, 10.20873, 11.820873, 11.9820873]
```

Single Exponential Smoothing [3/3]



- The process of finding the best value for α is named *fitting* (e.g. Nelder-Mead algorithm).

More Terminology

- Level: $\hat{y}_x$ is also called level ℓ_x

- Trend (or slope) ’b’ for two consecutive points
  $b_x = y_x - y_{x-1}$

Smoothing Factor

- Level: $\ell_x = \alpha * y_x + (1 - \alpha) * (\ell_{x-1} + b_{x-1})$

- Trend: $b_x = \beta * (\ell_x - \ell_{x-1}) + (1 - \beta) * b_{x-1}$

- Forecast: $\boxed{\hat{y}_{x+1}} = \ell_x + b_x$

Trend Factor

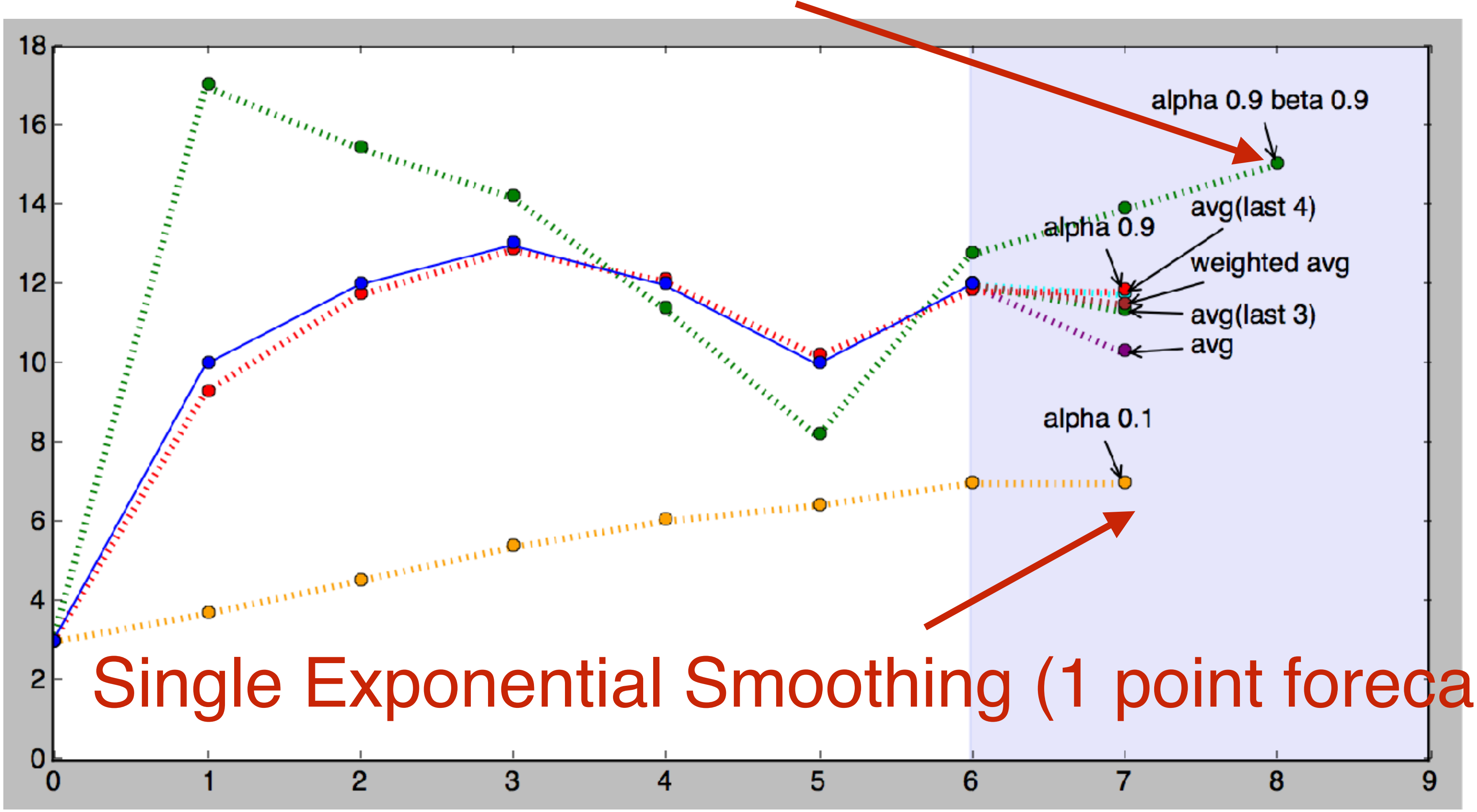
Two number prediction

Double Exponential Smoothing [1/2]

```
# given a series and alpha, return series of smoothed points
def double_exponential_smoothing(series, alpha, beta):
    result = [series[0]]
    for n in range(1, len(series)+2):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # we are forecasting
          value = result[-1]
        else:
          value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result

# >>> double_exponential_smoothing(series, alpha=0.9, beta=0.9)
# [3, 17.0, 15.45, 14.210500000000001, 11.396044999999999, 8.183803049999998, 12.753698384500002,
13.889016464000003, 15.0243345435]
```

Double Exponential Smoothing [2/2]

Double Exponential Smoothing (2 points forecast)



Single Exponential Smoothing (1 point forecast)

Triple Exponential Smoothing (a.k.a Holt-Winters Method) [1/2]

- Season: when a series is repetitive at regular intervals, it is defined seasonal.
- Season Length: the number of data points in a season.
- Seasonal Component: a deviation from level+trend (of the double exponential smoothing) that repeats itself into the season. For every season datapoint there is a seasonal component s defined. Example if you have a weekly season (length 7) every day has a seasonal component.

Triple Exponential Smoothing (a.k.a Holt-Winters Method) [2/2]

Arbitrary number of forecasted points

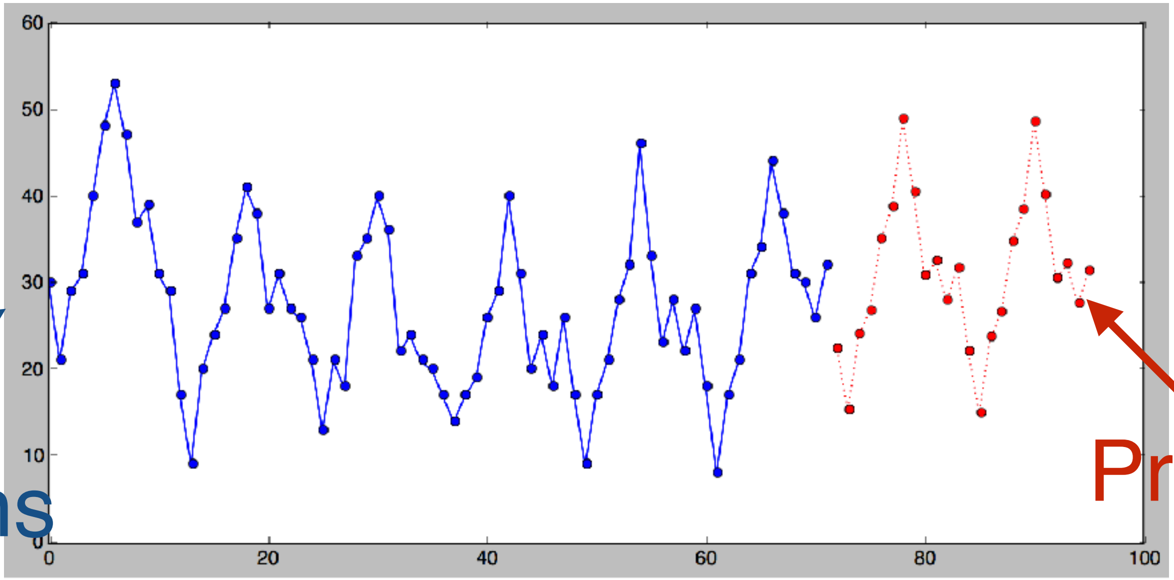
- Level: $\ell_x = \alpha * y_x + (1 - \alpha) * (\ell_{x-1} + b_{x-1})$
- Trend: $b_x = \beta * (\ell_x - \ell_{x-1}) + (1 - \beta) * b_{x-1}$
- Seasonal: $s_x = \gamma * (y_x - \ell_x) + (1 - \gamma) * s_{x-L}$
- Forecast: $y\hat{}_{x+m} = \ell_x + m * b_x + s_{\boxed{x-L+1+(m-1) \% L}}$

Seasonal Smoothing Factor

The seasonal factor has to be repeated in a loop for every season length

Triple Exponential Smoothing (a.k.a Holt-Winters Method) [3/2]

- As the timeseries has a season, the number of prediction is arbitrary, allowing to predict an arbitrary number of points (with increasing error)



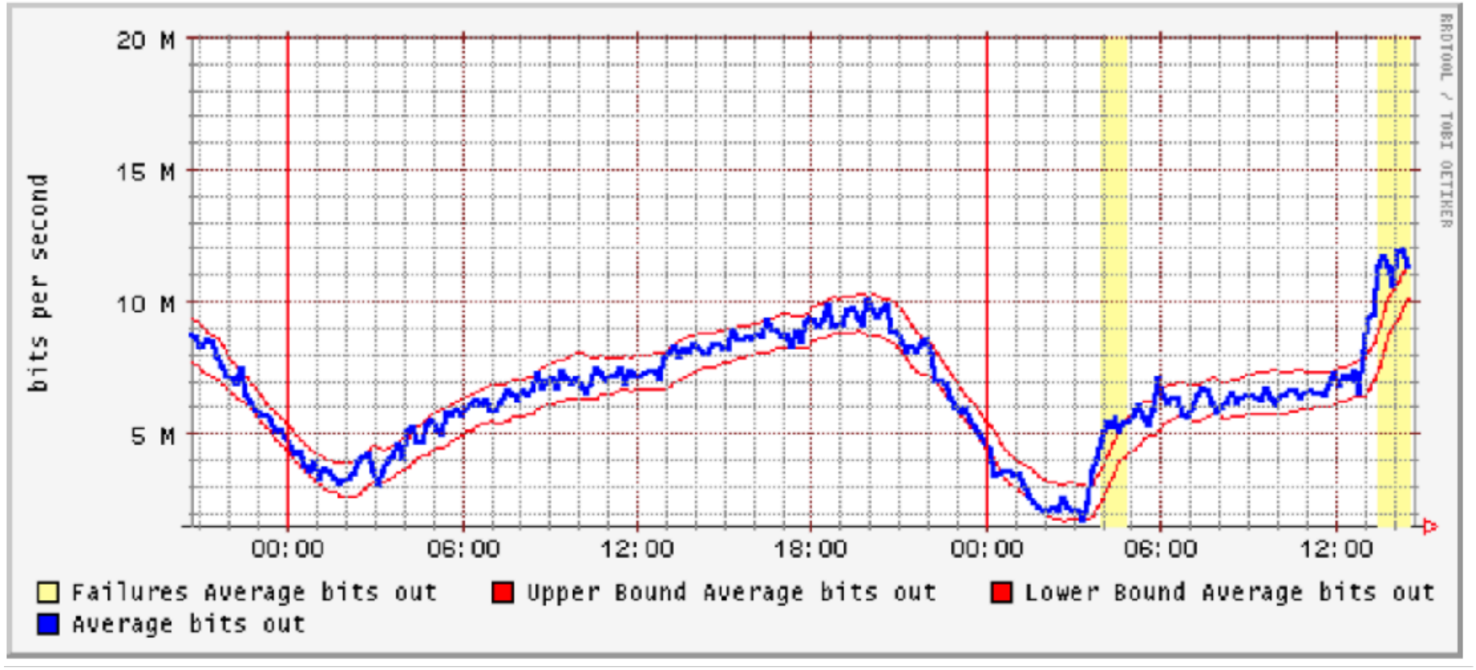
Observations

Predictions

Triple Exponential Smoothing (a.k.a Holt-Winters Method) [4/4]

- α, β, and γ are the adaptation parameters of the algorithm and 0 < α, β, γ < 1. Larger values mean the algorithm adapts faster and predictions reflect recent observations in the time series; smaller values means the algorithm adapts slower, placing more weight on the past history of the time series.
- The value for α, β and γ can be determined trying to determine the smallest SSE (sum of squared errors) with an iterative process called fitting.

Measuring Deviation [1/4]



- If Holt-Winters can predict a point, we can detect anomalies when the observation deviates too much from the prediction.

Measuring Deviation [2/4]

- Deviation can be detected when the observation falls outside of the min and max confidence bands for a given point.
- $d_t = \gamma * \text{abs}(y_t - y^\wedge_t) + (1 - \gamma) * d_{t-m}$

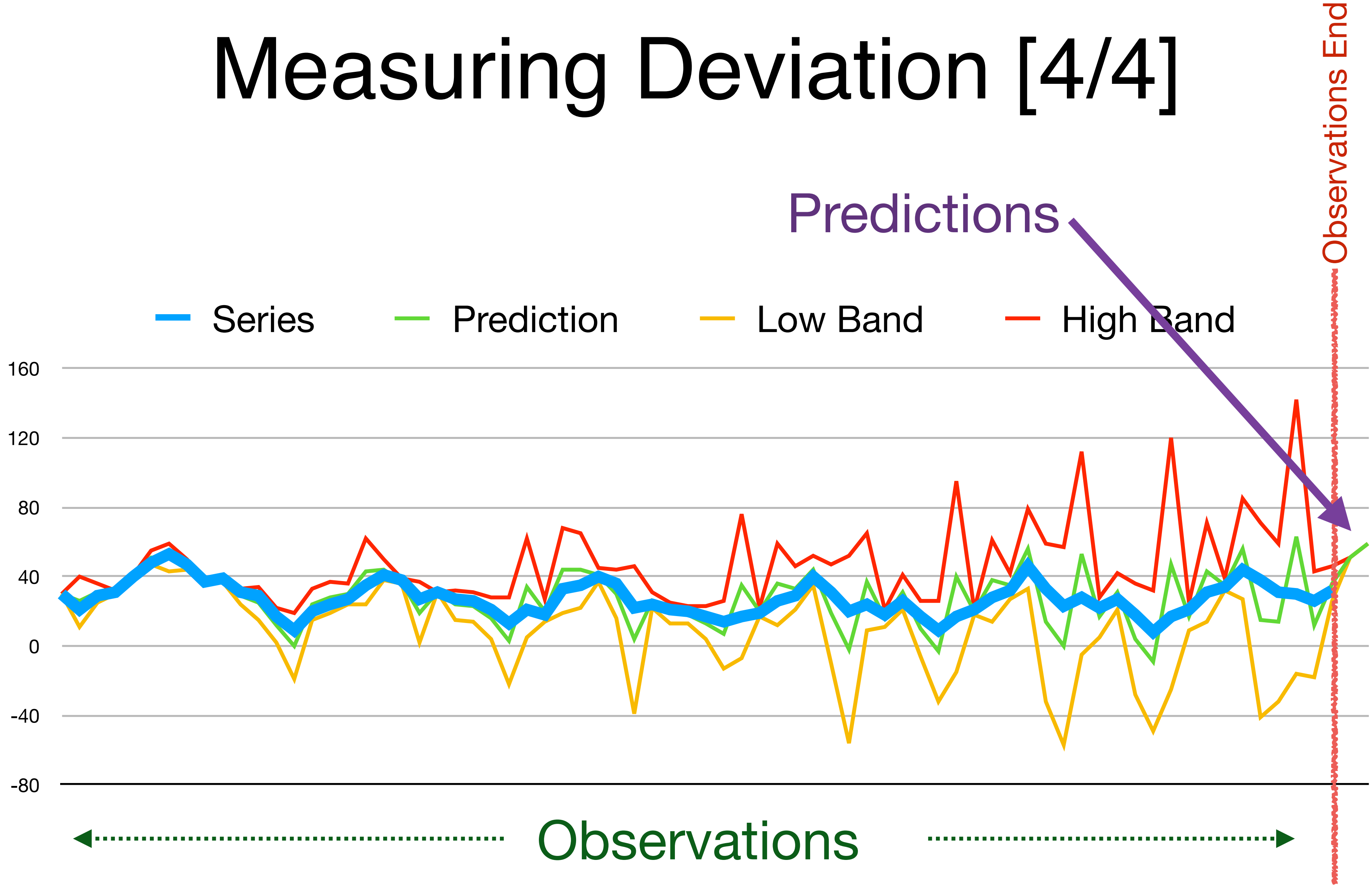
Seasonal Smoothing Factor

- Confidence Bands: Confidence Scaling Factor
  - Upper Band: $y^\wedge_t - \delta * d_{t-m}$ (value in 2..3 range)
  - *Lower Band:* $y^\wedge_t + \delta * d_{t-m}$

Measuring Deviation [3/4]

```
def triple_exponential_smoothing(series, slen, alpha, beta, gamma, n_preds):
    result = []
    deviation = []

    seasonals = initial_seasonal_components(series, slen)
    deviations = seasonals
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
            result.append(series[0])
            deviation.append(0)
            continue

        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append((smooth + m*trend) + seasonals[i%slen])
            deviation.append(0) # Unknown as we've not predicted yet
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-alpha)*(smooth+trend)
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            prediction = smooth+trend+seasonals[i%slen]
            result.append(prediction)

            deviations[i%slen] = gamma*abs(val-prediction) + (1-gamma)*deviations[i%slen]
            deviation.append(abs(deviations[i%slen]))

    return result,deviation
```

Measuring Deviation [4/4]



https://www.real-statistics.com/wp-content/uploads/2020/07/Real-Statistics-Time-Series-Examples.xlsx

Measuring Correlation [1/3]

- Typical correlation questions:
  - Number of SYN packets vs active flows.
  - Number of TCP retransmissions vs Throughput
  - SNMP Errors on port X and port Y.

- Are two timeseries correlated, i.e. they have some sort of statistical relationship?

Measuring Correlation [2/3]

- The Pearson coefficient is the ratio between the covariance of two variables and the product of their standard deviations.
- $-1 < x < 0$: Negative correlation (when one changes the other series changes in opposite direction).
- $x = 0$: No correlation (no relationship between the series).
- $0 < x < 1$: Positive correlation (when one changes the other series changes in the same direction).

Measuring Correlation [3/3]

```c
#include <stdio.h>
#include <math.h>

double calculate_mean(int data[], int n) {
    double sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += data[i];
    }
    return sum / n;
}

double calculate_variance(int data[], int n, double mean) {
    double sum_squared_diff = 0;
    for (int i = 0; i < n; ++i) {
        sum_squared_diff += pow(data[i] - mean, 2);
    }
    return sum_squared_diff / n;
}

double calculate_covariance(int data1[], int data2[], int n, double mean1, double mean2) {
    double sum_product_diff = 0;
    for (int i = 0; i < n; ++i) {
        sum_product_diff += (data1[i] - mean1) * (data2[i] - mean2);
    }
    return sum_product_diff / n;
}

double calculate_pearson_correlation(int data1[], int data2[], int n) {
    double mean1 = calculate_mean(data1, n);
    double mean2 = calculate_mean(data2, n);

    double variance1 = calculate_variance(data1, n, mean1);
    double variance2 = calculate_variance(data2, n, mean2);

    double covariance = calculate_covariance(data1, data2, n, mean1, mean2);

    return covariance / sqrt(variance1 * variance2);
}
```

```c
int main() {
    // Example data
    int data1[] = {1, 2, 3, 4, 5};
    int data2[] = {1002, 10013, 1004, 1005, 1006};
    int n = sizeof(data1) / sizeof(data1[0]);

    // Calculate Pearson correlation coefficient
    double correlation = calculate_pearson_correlation(data1, data2, n);

    // Print the result
    printf("Pearson correlation coefficient: %lf\n", correlation);

    return 0;
}
```

See ndpi_pearson_correlation()

Data Prediction [1/3]

- In capacity planning it is important to predict how the usage of a certain resource will be in the future.
- Examples:
  - If my bandwidth usage won't change significantly, how many Gbit will I need in two years?
  - Storing data at the current pace, how many TB will I need in 6 months from now?

Data Prediction [2/3]

- Predicting data can be a complex activity but it can greatly simplified using linear prediction, a mathematical operation where future values of a discrete-time signal (i.e. a time series consisting of a sequence of quantities) are estimated as a linear function ($y = \alpha + \beta * x$) of previous samples.
- In other words: how can I predict the future if I know my past?

Data Prediction [3/3]

```
int ndpi_predict_linear(u_int32_t *values, u_int32_t num_values,
                        u_int32_t predict_periods, u_int32_t *prediction) {
  u_int i;
  float m, c, d;
  float sumx = 0, sumx_square = 0, sumy = 0, sumxy = 0;

  for(i = 0; i < num_values; i++) {
    float y = values[i];
    float x = i + 1;

    sumx   = sumx+x;
    sumx_square = sumx_square + (x * x);
    sumy   = sumy + y;
    sumxy  = sumxy + (x * y);
  }

  d = (num_values * sumx_square) - (sumx * sumx);

  if(d == 0) return(-1);

  m = ((num_values * sumxy) - (sumx * sumy))  / d; /* beta  */
  c = ((sumy * sumx_square) - (sumx * sumxy)) / d; /* alpha */

  *prediction = c + (m * (predict_periods + num_values - 1));

  return(0);
}
```

References

- Forecasting: Principles and Practice, https://otexts.com/fpp2/
- https://github.com/lucaderi/HoltWinters
- https://en.wikipedia.org/wiki/Exponential_smoothing
- Jake D. Brutlag, Aberrant Behavior Detection in Time Series for Network Monitoring , Proceedings of LISA 2000 Conference.
- Amy Ward, Peter Glynn, and Kathy Richardson, Internet service performance failure detection, Performance Evaluation Review, 1998.

Data Structures

Bitmap [1/2]

- Array of bits (a.k.a. bit vector).
- Bitmaps can have arbitrary lengths.
- Elements can added/removed in O(1) with simple bit operations.

Bitmap [2/2]

```
#define bitmap64_t(name, n) u_int64_t name[n / 64]
#define bitmap64_ptr_t u_int64_t *
#define bitmap64_reset(b)   memset(b, 0, sizeof(b))
#define bitmap64_set_all(b) memset(b, 0xFF, sizeof(b))
#define bitmap64_clone(b1, b2) memcpy(b1, b2, sizeof(b1))
#define bitmap64_set_bit(b, i)   b[i >> 6] |=  ((u_int64_t) 1 << (i & 0x3F))
#define bitmap64_clear_bit(b, i) b[i >> 6] &= ~((u_int64_t) 1 << (i & 0x3F))
#define bitmap64_isset_bit(b, i) !!(b[i >> 6] &   ((u_int64_t) 1 << (i & 0x3F)))
#define bitmap64_or(b1, b2) for (size_t __i = 0; __i < (sizeof(b1)/8); __i++) b1[__i] |= b2[__i]
```

```
Example:
bitmap64_t(tot_tcp_flags_combinations, 256);   /* Define the variable (256 bit) */
bitmap64_reset(tot_tcp_flags_combinations);    /* Reset the variable            */
bitmap64_set(tot_tcp_flags_combinations, 67);  /* Set a bit                     */
```

Compressed Bitmap [1/3]

- A bitmap is an array of bit (usually) with a predefined length.
- For specific domains, bitmaps can be <u>sparse</u> with a few bit sets.
- Example : [0,12,23,500,510,522,10000] requires at least 10k bits even though most bit are 0, with 7 bits at 1.
- As most bits are 0, this data structure is inefficient.

Compressed Bitmap [2/3]

- The simplest way to compress this bitmap is by means of counting repetitions:
  - [0,12,23,500,510,522,10000]
  - 1(1), 11(0), 1(1), 10(0), 1(1), 476(0)….
- There are many algorithms to compress the bitmap such as WAH, EWAH, COMPAX…
- Anyway: if you can “increasingly” set bits, you can build the bitmap at runtime without static memory costs.

Compressed Bitmap [3/3]

- The nice thing about compressed bitmap is that you can perform operations such as AND, OR, NOT… on the compressed bitmap without decompressing it at all.
- One of the most popular is http://roaringbitmap.org

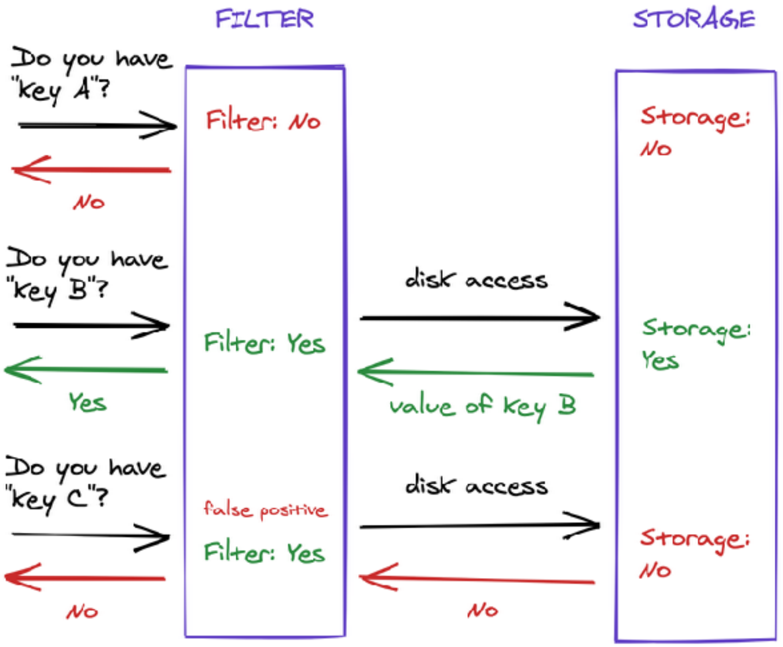
Compressed Bitmap Indexes

Row Id	Value	Col. bit0	Col. bit1	Col. bit2	Col. bit3	Col. bit4
0	1	1	0	0	0	0
1	4	0	0	1	0	0
2	6	0	1	1	0	0
3	15	1	1	1	1	0
4	28	0	1	1	1	1
5	1	1	0	0	0	0
6	3	1	1	0	0	0

- In order to know the rowId’s of records with value 1: C_0=1 AND C_1=0 AND C_2=0 AND C_3=0 AND C_4=0.
- When bitmaps are compressed the above operation is very fast fast

Bloom Filters [1/5]

- Bloom filters are probabilistic data structure that answers a simple question: is an element part of a set ?



Bloom Filters [2/5]

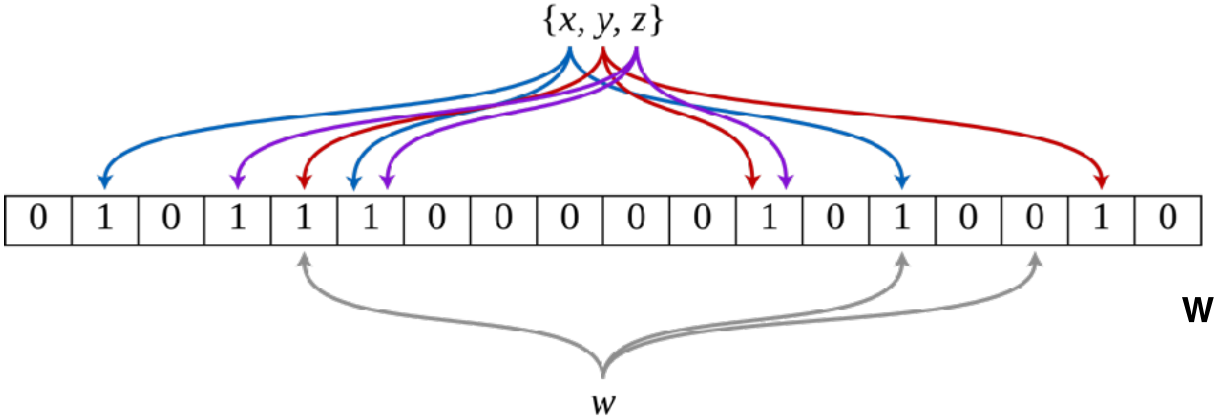
- The idea is to do a quick comparison so see if a given input can match our needs. In case of match we can do an exact, yet costly, exact match with a different algorithm.
- You can check the set presence with many other data structures (e.g. a hash or a very-long bitmap) at a higher cost:
  - How much memory do you need to keep a bitmap 2^{32} bits long, to know what IP contacted your PC?
  - What if you want to move to IPv6 (2^{128} bits)?

Bloom Filters [3/5]

- A bloom filter is a bit vector of length m.
- Select two or more independent and uniformly distributed hash h1() and h2() functions.
- Add an element α: set the bits corresponding to h1(α) and h2(α).
- Check for presence of β: if h1(β) and h2(β) are set, then β will likely (now you understand why is a probabilistic datastructure) be part of set.

Bloom Filters [4/5]

- Note: being probabilistic, it cannot be used to know for sure if an element belongs to the set.



Wikipedia

- Counting bloom filter: replace the bit vector with an integer to know how many bits have been set
- Advantage: you can remove elements (bloom can't)
- Disadvantage: more memory used.

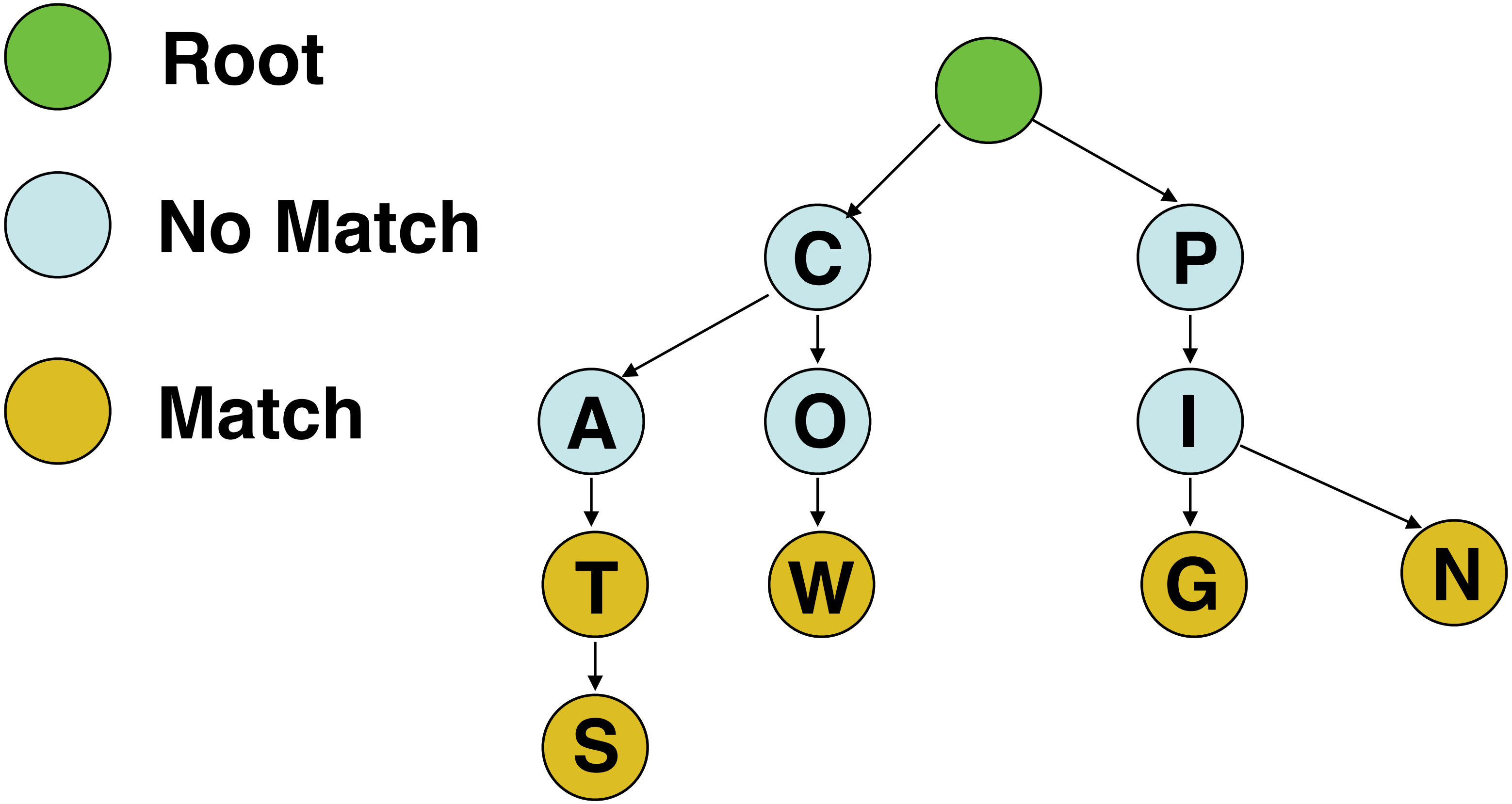
Bloom Filters [5/5]

- Good hashes include murmur, nfv, and md5.
- Test membership error rate is $(1-e^{-kn/m})^k$ where:
  - m: number of vector bits
  - k: number of hash functions used
  - n: the number of elements inserted
- Typical usage example: use the bloom filter to a “quick” check for presence and if so, do the “slow” check.

Tries [1/3]

- A trie (pronounce as try, “pun on re**trie**val and tree”) is tree not based on comparisons (<,>)
  - – Each node has a letter.
  - – In case of multiple options per letter a list is used for each possible tree branch.
  - – Nodes with letters in the set are “marked”.

Tries [2/3]



{ Cats, Cat, Cow, Pig, Pin }

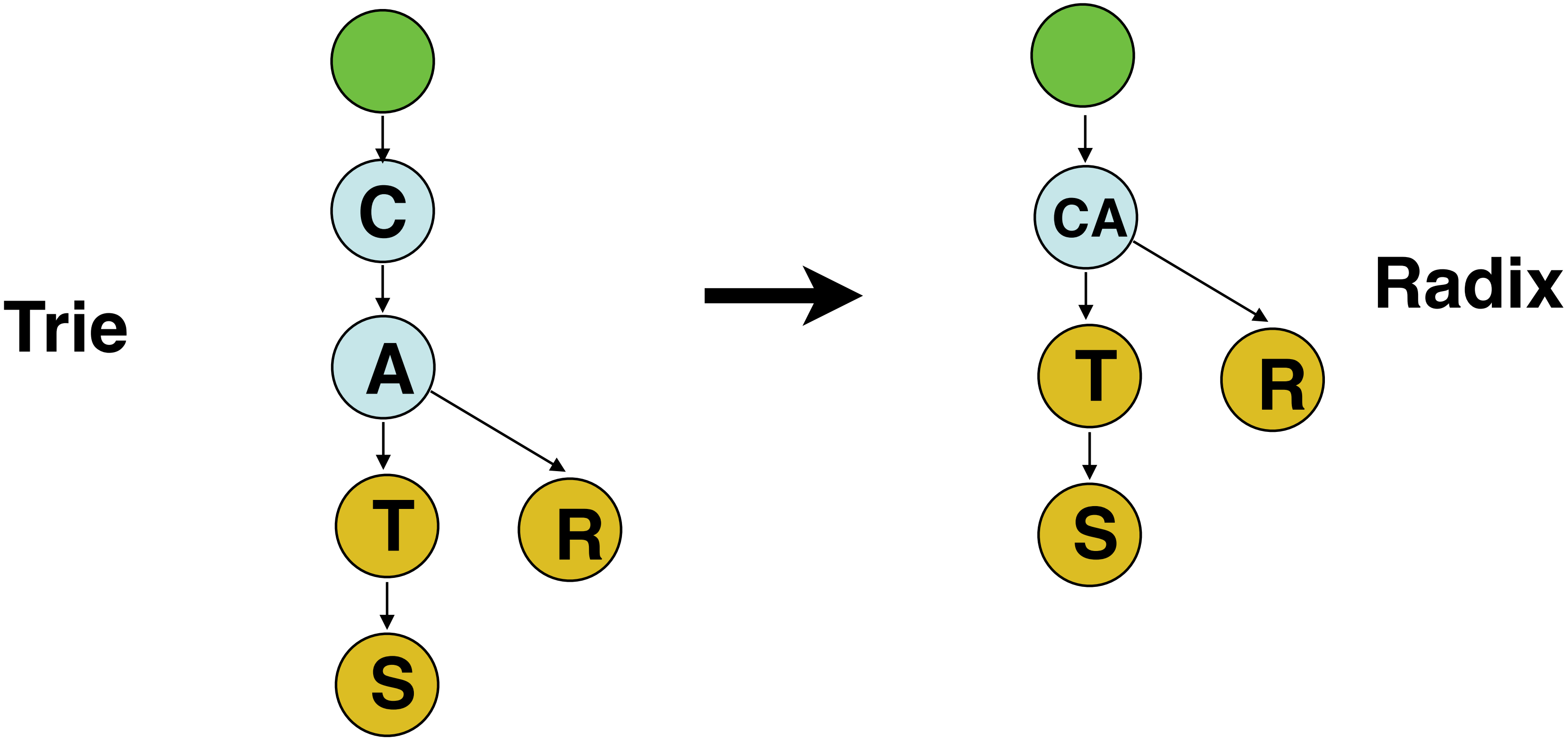
Tries [3/3]

- In tries nodes can be added/removed/searched.
- Features
  - Ability to search strings starting with a given prefix.

    Sounds familiar?
  - Ability to generate string in dictionary order (if links in nodes are alphabetically sorted).
- Performance
  - insert O(w), where w is the length of the string to be inserted, regardless of the number of stored strings

Radix Tree

- Same as trie where nodes have a set of strings



Patricia Tree

- Patricia: Practical Algorithm to Retrieve Information Coded in Alphanumeric, D.R. Morrison (1968).
- Radix tree where numbers are used instead of strings.
- Efficient for subnet matching, IPv4/IPv6.
- You can search partial matches (e.g. /24) and if you keep searching and found a match for finding a narrower match (e.g. /32)
- Code: https://github.com/ntop/nDPI/blob/dev/src/lib/third_party/src/ndpi_patricia.c

Substring Matching [1/3]

- In networking this is used very often:
  - Extract domain name from a hostname (i.e. bbc.co.uk from www.bbc.co.uk, or google.com from www.google.com).
  - Match malware/porn/advertisement/... domain names.
- In other words: check if a domain name belonging to a category matches a hostname (i.e. www.youtube.com -> streaming)

Substring Matching [2/3]

- Tries are a solution but they use a lot of memory (at least the size of the dataset).
- Aho-Corasick (and successor Commentz-Walter algorithm) is a string matching algorithm has a space complexity of O(total length of keywords + number of nodes in the trie).
- Alternatives?

Substring Matching [3/3]

- nDPI includes support for binary fuse filters that are an improvement of Bloom filters.
- Using 64 bit hashes it is possible to create fast probabilistic filters that have a space complexity of ~13% of the original dataset size.

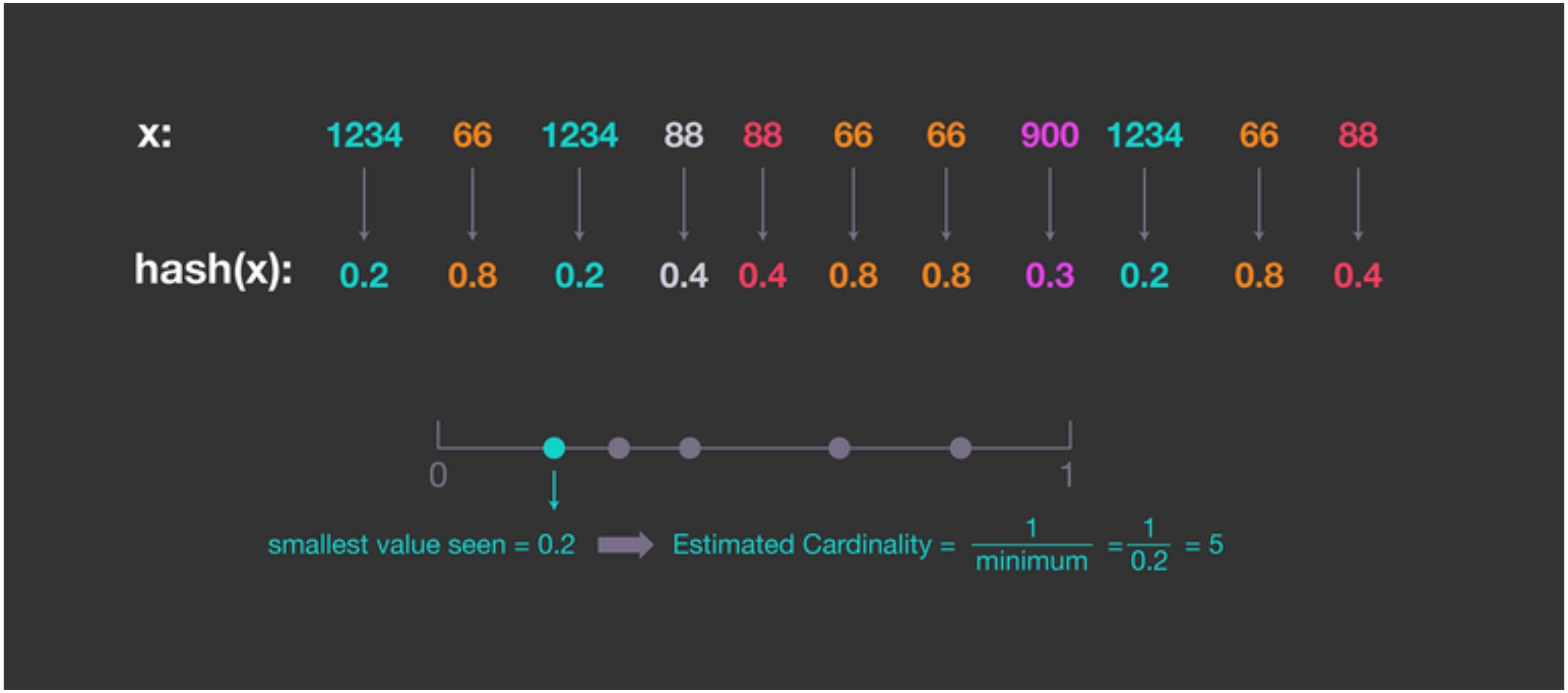
Probabilistic Counting [1/4]

- How can I get an estimate (i.e. approximate) of a number of unique set elements ? For instance: how many IP addresses has my host contacted in the past 5 minutes? Of course you can do this in many ways (e.g. a hash table) but at a higher memory cost.
- Assume that you have a string of length m which consists of {0, 1} with equal probability. What is the probability that it will start with 0, with 2 zeros, with k zeros? It is 1/2, 1/4 and $1/2^k$.

Probabilistic Counting [2/4]

- Therefore, having a list of elements that are evenly distributed between 0 and 2^k - 1 you can count the max number of the biggest prefix of zeros in binary representation and this will give you a reasonable estimate.
- Even distribution can be achieved with a good hashing function in the range 0 and 2^{k_max} - 1.
- Example: SHA1 has k_max = 160 that is pretty large.

Probabilistic Counting [3/4]


x:
1234
66
1234
88
88
66
66
900
1234
66
88
hash(x):
0.2
0.8
0.2
0.4
0.4
0.8
0.8
0.3
0.2
0.8
0.4
0
1
smallest value seen = 0.2
Estimated Cardinality = $\frac{1}{\text{minimum}} = \frac{1}{0.2} = 5$

To ensure that the entries are evenly distributed, we can use a hash function and estimate the cardinality from the hashed values instead of from the entries themselves. The picture above illustrates a simple example in which the hashed values are normalised and uniformly distributed between 0 and 1.

Probabilistic Counting [4/4]

- Unfortunately one random occurrence of high frequency 0-prefix element can spoil everything.
- A possible solution is to use multiple hashing functions and report as estimation the average of all the estimations, but this is not an optimal solution.

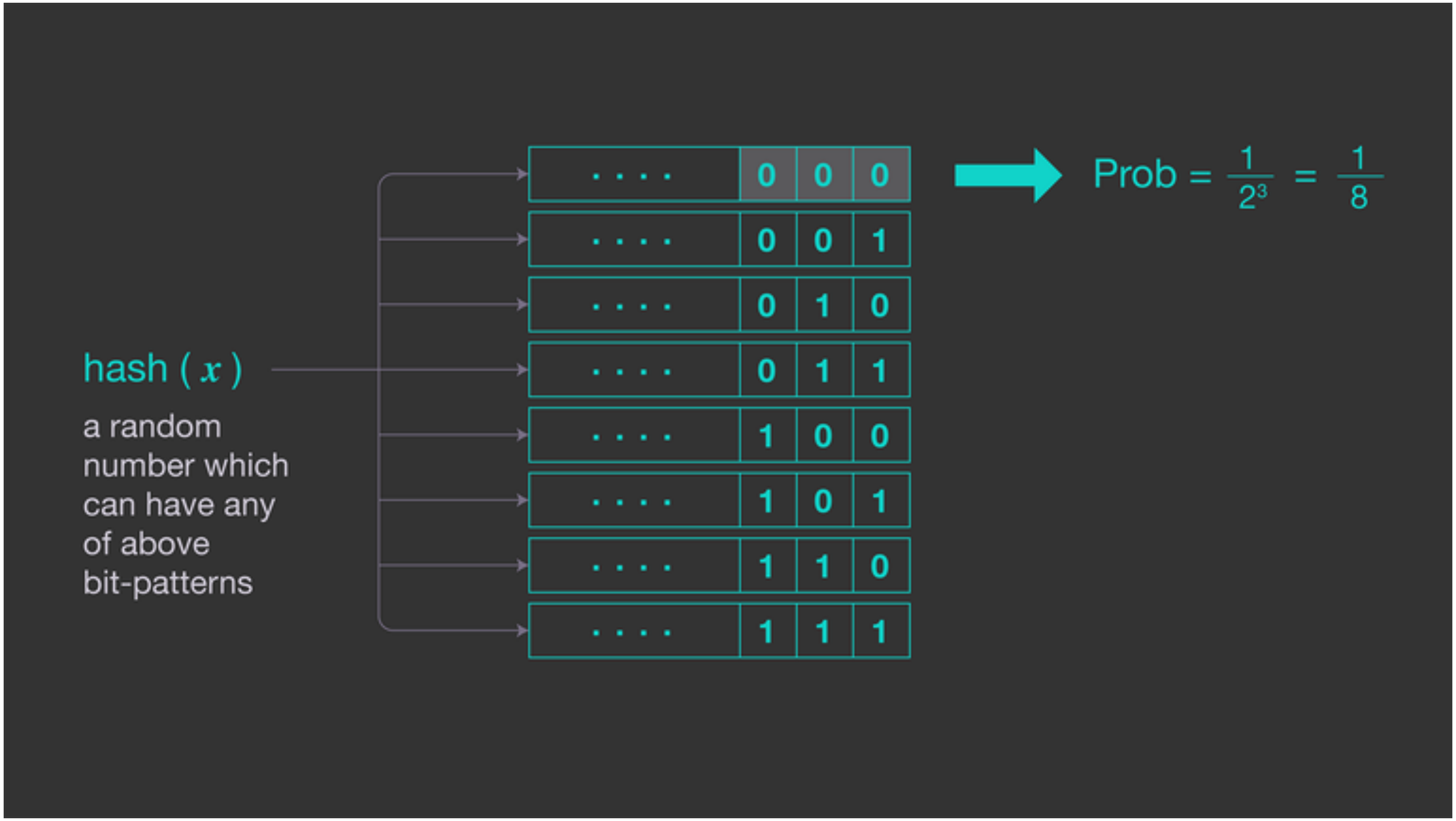
HyperLogLog [1/4]

- HyperLogLog is a probabilistic data structure used to estimate the cardinality of a set.
- It improves probabilistic counting by hashing every element, and counting the amount of 0s to the left of such hash.
- The HLL algorithm divides the hash result in 2, using the first section to address a **bucket**, and the second one to count the 0s to the left and store the largest amount so far.

HLL Paper: http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf

HyperLogLog [2/4]

Simplified HLL Algorithm



hash (x)

a random number which can have any of above bit-patterns

```
. . . . . 0 0 0
. . . . . 0 0 1
. . . . . 0 1 0
. . . . . 0 1 1
. . . . . 1 0 0
. . . . . 1 0 1
. . . . . 1 1 0
. . . . . 1 1 1
```

$$\text{Prob} = \frac{1}{2^3} = \frac{1}{8}$$

HyperLogLog [3/4]

```
int hll_init(struct ndpi_hll *hll, u_int8_t bits) {
  if(bits < 4 || bits > 20) {
    errno = ERANGE;
    return -1;
  }

  hll->bits = bits; /* Number of bits of buckets number */
  hll->size = (size_t)1 << bits; /* Number of buckets 2^bits */
  hll->registers = ndpi_calloc(hll->size, 1); /* Create the bucket register counters */

  return 0;
}

/* Add a new hash value to the HLL */
static __inline void _hll_add_hash(struct ndpi_hll *hll, u_int32_t hash) {
  if(hll->registers) {
    u_int32_t index = hash >> (32 - hll->bits);   /* Use the first 'hll->bits' bits as bucket index */
    u_int8_t rank   = _hll_rank(hash, hll->bits); /* Count the number of leading 0 */

    if(rank > hll->registers[index])
      hll->registers[index] = rank; /* Store the largest number of leading zeros for the bucket */
  }
}

double hll_count(const struct ndpi_hll *hll) {
  if(hll->registers) {
    …
    return estimate;
  } else
    return(0.);
}
```

HyperLogLog [4/4]

```
HyperLogLog Memory and StandardError Notes

StdError = 1.04/sqrt(2^i)

[i: 4] 16 bytes      [StdError: 26%  ]
[i: 5] 32 bytes      [StdError: 18.4%]
[i: 6] 64 bytes      [StdError: 13%  ]
[i: 7] 128 bytes     [StdError: 9.2% ]
[i: 8] 256 bytes     [StdError: 6.5% ]
[i: 9] 512 bytes     [StdError: 4.6% ]
[i: 10] 1024 bytes   [StdError: 3.25%]
[i: 11] 2048 bytes   [StdError: 2.3% ]
[i: 12] 4096 bytes   [StdError: 1.6% ]
[i: 13] 8192 bytes   [StdError: 1.15%]
[i: 14] 16384 bytes  [StdError: 0.81%]
[i: 15] 32768 bytes  [StdError: 0.57%]
[i: 16] 65536 bytes  [StdError: 0.41%]
[i: 17] 131072 bytes [StdError: 0.29%]
[i: 18] 262144 bytes [StdError: 0.2% ]
[i: 19] 524288 bytes [StdError: 0.14%]
```

https://github.com/ntop/nDPI/blob/dev/src/lib/third_party/src/hll/hll.c

Measuring Excess: Leaky Bucket [1/2]

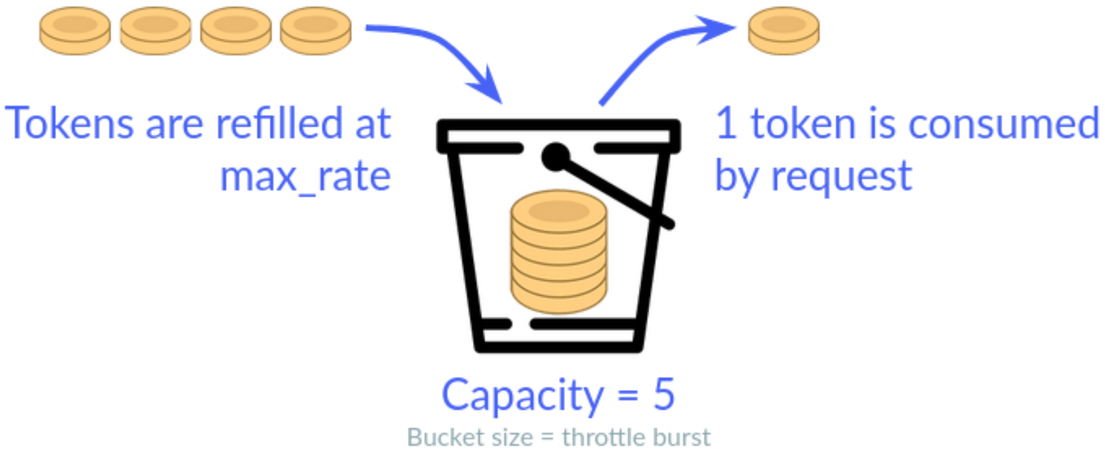
- How do I detect a port or network scan ?
- How can I detect traffic bursts?

A simple solution is to continuously monitor an event (e.g. number of new/unsuccessful connections) and monitor its rate over time.

Example when more than X connections/sec are spotted -> alert

Measuring Excess: Leaky Bucket [2/2]

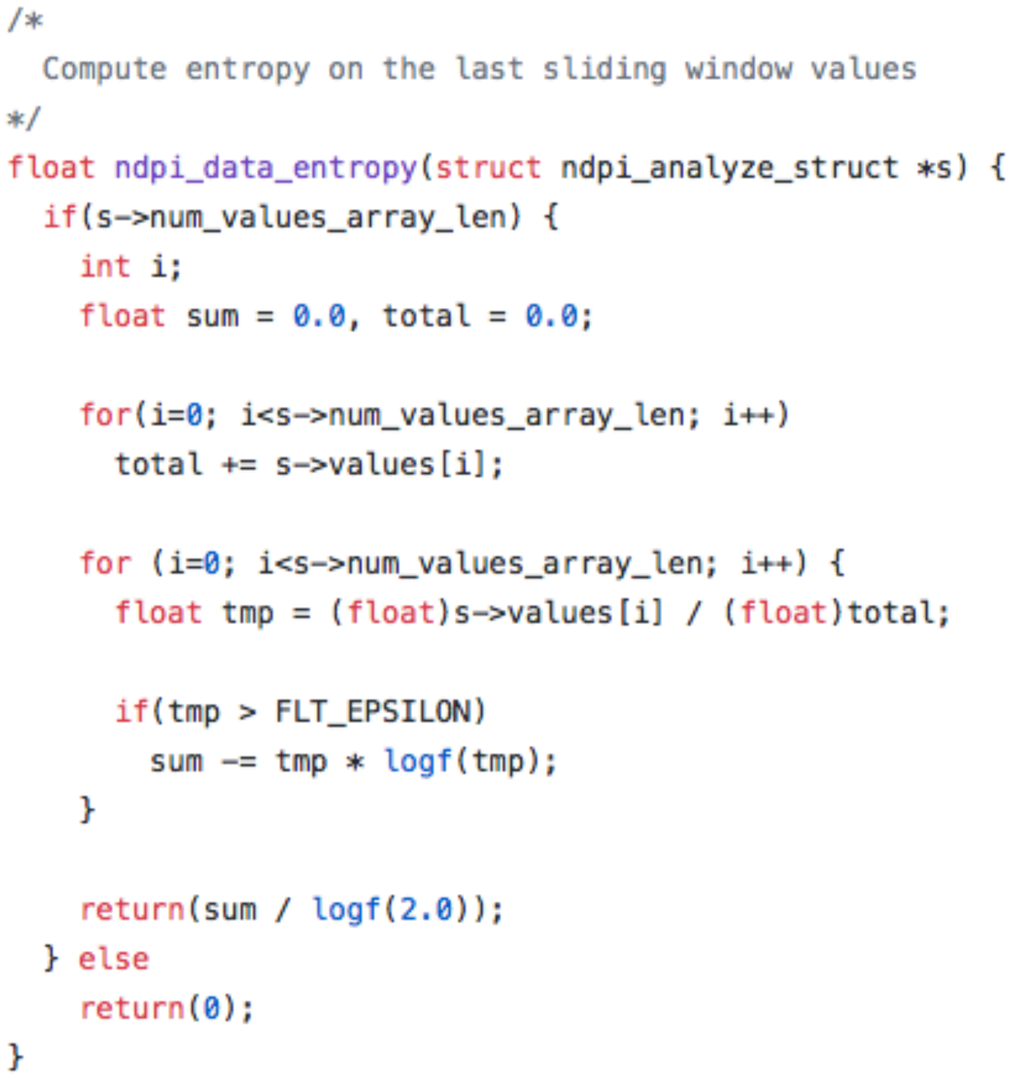
- Every X seconds refill the bucket to max capacity
- At every even the bucket is decreased
- If bucket value is 0 -> alert


Tokens are refilled at max_rate
1 token is consumed by request
Capacity = 5
Bucket size = throttle burst

Entropy [1/2]

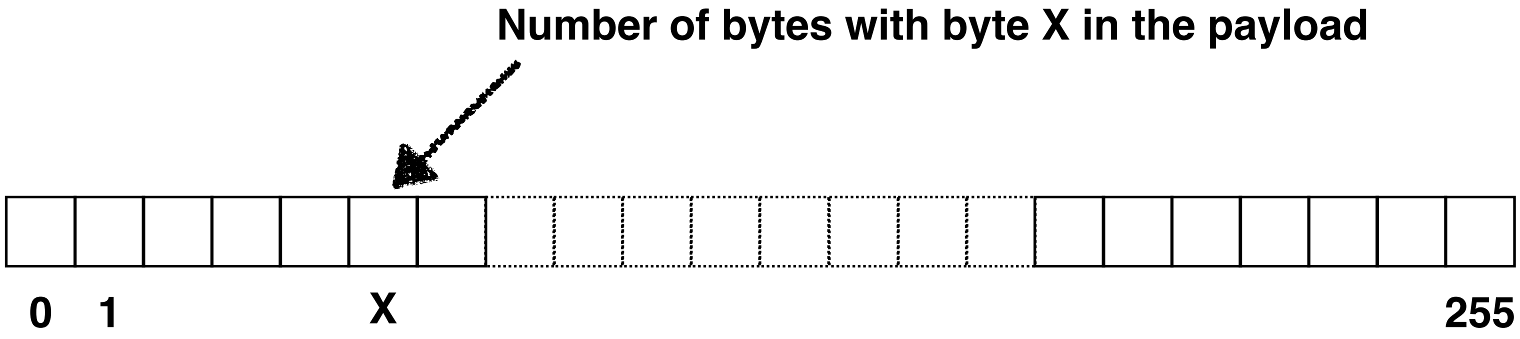
- Metric used to measure how bytes are distributed: the larger the entropy, the greater the uncertainty in predicting the value of an observation.
- Formula: https://csrc.nist.gov/csrc/media/publications/sp/800-90b/draft/documents/draft-sp800-90b.pdf

Entropy [2/2]



```
/*
  Compute entropy on the last sliding window values
*/
float ndpi_data_entropy(struct ndpi_analyze_struct *s) {
  if(s->num_values_array_len) {
    int i;
    float sum = 0.0, total = 0.0;

    for(i=0; i<s->num_values_array_len; i++)
      total += s->values[i];

    for (i=0; i<s->num_values_array_len; i++) {
      float tmp = (float)s->values[i] / (float)total;

      if(tmp > FLT_EPSILON)
        sum -= tmp * logf(tmp);
    }

    return(sum / logf(2.0));
  } else
    return(0);
}
```

Using Entropy: Bytes Entropy



Entropy of raw data before and after encryption (TLS) changes but is it within limited boundaries for homogeneous data.

Server Entropy (SCP)

PDF	PNG	TEXT	
6,418	7,014	7,008	
6,431	7,019	7,007	
6,428	6,994	7,011	
6,426	7,009	7,009	Average
0,007	0,013	0,002	StdDev

Data Binning [1/4]

- Data binning is a technique that allows data to be classified in a small number of “bins”, that in essence is a vector of positive numbers where each bin value contains the number of observations.
- Example: packets lengths can be grouped in 6 bins of size <= 64 bytes, 65-128, 129-256, 257-512, 513-1024, 1025+.

Data Binning [2/4]

- Bins allow data to be classified using a small set of intervals instead of individual values that can lead to observation errors.
- Data is classified by
  - defining the bin number
  - adding data to the individual bins
  - normalising the data so that bins with different number of elements can still be compared.

Data Binning [3/4]

- Bins do not store the data order (i.e. how the individual events happened) but just the data.
- Example: if you want to compare two hosts if they use similar protocols you can create a set of bins (e.g. 256 bins as the number of protocols recognised by nDPI) and for each new flow increase the bin-id that corresponds to the protocol. Then you can compare bins for equality to see what hosts are similar.

Data Binning [4/4]

```
void ndpi_inc_bin(struct ndpi_bin *b,
                  u_int8_t slot_id,
                  u_int32_t val) {
  b->is_empty = 0;

  if(slot_id >= b->num_bins) slot_id = 0;

  switch(b->family) {
  case ndpi_bin_family8:
    b->u.bins8[slot_id] += (u_int8_t)val;
    break;
  case ndpi_bin_family16:
    b->u.bins16[slot_id] += (u_int16_t)val;
    break;
  case ndpi_bin_family32:
    b->u.bins32[slot_id] += (u_int32_t)val;
    break;
  }
}
```

```
/*
  Each bin slot is transformed in a % with respect to the value total
 */
void ndpi_normalize_bin(struct ndpi_bin *b) {
  u_int8_t i;
  u_int32_t tot = 0;

  if(b->is_empty) return;

  switch(b->family) {
  case ndpi_bin_family8:
    for(i=0; i<b->num_bins; i++) tot += b->u.bins8[i];

    if(tot > 0) {
      for(i=0; i<b->num_bins; i++)
        b->u.bins8[i] = (b->u.bins8[i]*100) / tot;
    }
    break;
  case ndpi_bin_family16:
    for(i=0; i<b->num_bins; i++) tot += b->u.bins16[i];

    if(tot > 0) {
      for(i=0; i<b->num_bins; i++)
        b->u.bins16[i] = (b->u.bins16[i]*100) / tot;
    }
    break;
  case ndpi_bin_family32:
    for(i=0; i<b->num_bins; i++) tot += b->u.bins32[i];

    if(tot > 0) {
      for(i=0; i<b->num_bins; i++)
        b->u.bins32[i] = (b->u.bins32[i]*100) / tot;
    }
    break;
  }
}
```

How To Compare Data [1/4]

- Bins are an efficient way of storing observations but we need to find a way to compare them to find similarities (e.g. two hosts with the same behaviour).
- Two values can be compared with <,>,= operators, but how do we compare a vector of data such as a bin?
- Example: 23,45,13,72,32 and 1,45,18,29,43 ?

How To Compare Data [2/4]

- Supposing that bins have the same number of elements (otherwise comparison is not really possible), we need to
  - normalize data to make sure that the number of bin observations do not affect comparisons
  - compare the individual bin elements with a comparison operator.
- Popular similarity algorithms (see https://en.wikipedia.org/wiki/Similarity_measure)

How To Compare Data [3/4]

- Cosine Similarity

```
/*
   Determines how similar are two bins

   Cosine Similiarity
   0 = Very different
   ... (gray zone)
   1 = Alike
*/

u_int32_t sumxx = 0, sumxy = 0, sumyy = 0;

    for(i=0; i<b1->num_bins; i++) {
      u_int32_t a = ndpi_get_bin_value(b1, i);
      u_int32_t b = ndpi_get_bin_value(b2, i);

      sumxx += a*a, sumyy += b*b, sumxy += a*b;
    }

    if((sumxx == 0) || (sumyy == 0))
      return(0);
    else
      return((float)sumxy / sqrt((float)(sumxx * sumyy)));
```

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

How To Compare Data [4/4]

- Euclidean Distance

```
u_int32_t sum = 0;

for(i=0; i<b1->num_bins; i++) {
  u_int32_t a = ndpi_get_bin_value(b1, i);
  u_int32_t b = ndpi_get_bin_value(b2, i);
  u_int32_t diff = (a > b) ? (a - b) : (b - a);

  if(a != b) sum += pow(diff, 2);
}

/* The lower the more similar */
return(sqrt(sum));
```

- In essence you can compare individual bins by finding which one is more “similar” to another one using a distance algorithm

Jaccard (Similarity) Coefficient [1/3]

Simplest index used to measures the similarity between two sets of data. The higher the number, the more similar the two sets of data.

Jaccard Similarity = (number of observations in both sets) / (number in either set)

J(A, B) = |A∩B| / |A∪B|

Jaccard (Similarity) Coefficient [2/3]

Jaccard can be computed on unsorted data (list) as we do with bins where position matters…

```
#!/usr/bin/env python3

def jaccard_index(l1, l2):
    size_intersect = 0
    size_union     = 0

    for i in range(1, len(l1)):
        if(l1[i] == l2[i]):
            size_intersect = size_intersect+1
        else:
            size_union = size_union+1

    # Calculate the Jaccard index
    return size_intersect / (size_union);

### Main

l1 = [1, 2, 3, 4, 5] # List 1
l2 = [4, 5, 6, 7, 8] # List 2

jaccardIndex = jaccard_index(l1, l2);

# Print the Jaccard index and Jaccard distance
print("Jaccard index    = ", jaccardIndex);
print("Jaccard distance = ", 1-jaccardIndex);
```

```
$ ./jaccard.py
Jaccard index    =  0.0
Jaccard distance =  1.0
```

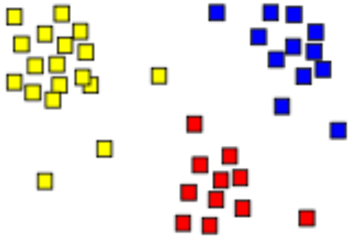
Jaccard (Similarity) Coefficient [3/3]

…or sorted (set) data (position doesn't matter, sort data).

```
#!/usr/bin/env python3

def jaccard_index(s1, s2):
    intersect = s1 & s2; # Compute the intersection {4, 5}
    union     = s1 | s2; # Compute the union        {1, 2, 3, 4, 5, 6, 7, 8}

    # Calculate the Jaccard index
    return len(intersect) / len(union);

### Main
s1 = {1, 2, 3, 4, 5} # Set 1
s2 = {4, 5, 6, 7, 8} # Set 2

jaccardIndex = jaccard_index(s1, s2);

print("Jaccard index    = ", jaccardIndex);
print("Jaccard distance = ", 1-jaccardIndex);
```

```
$ ./jaccard.py
Jaccard index    =  0.25
Jaccard distance =  0.75
```

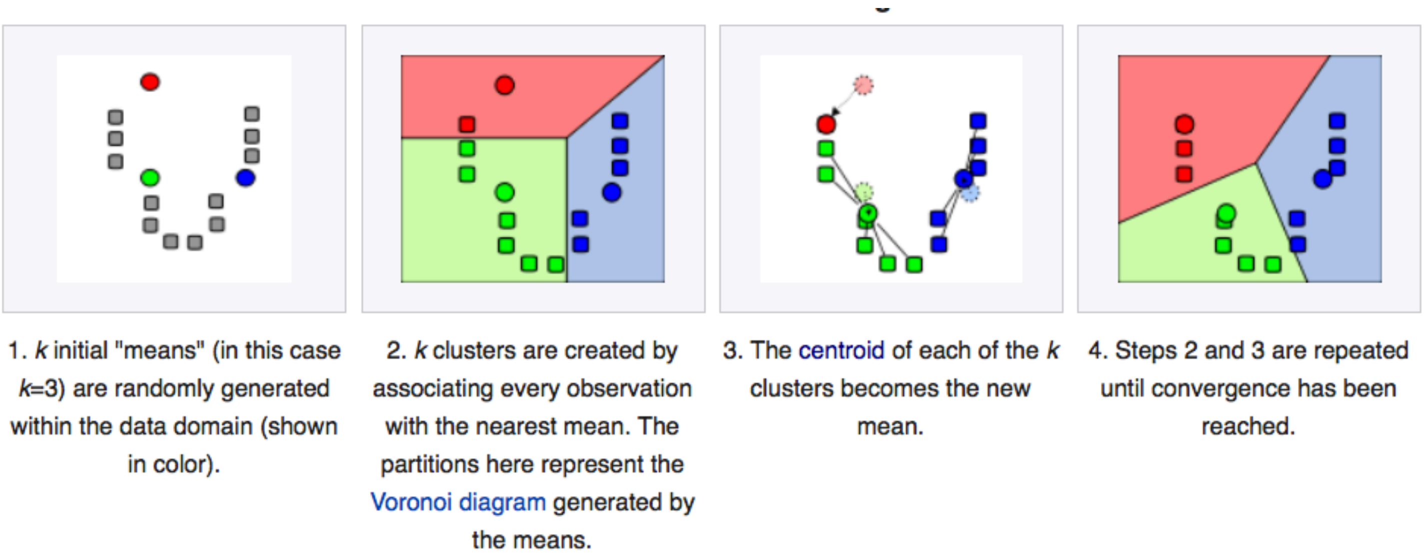
Data Clustering [1/2]



- Clustering means group together values that are similar.
- There are various clustering algorithm families (https://en.wikipedia.org/wiki/Cluster_analysis).
- One of the simplest ones is K-means that has the drawback to take as input ‘K’ that is the number of clusters (other algorithm such as DBSCAN do it automatically but are much more complex).

Data Clustering [2/2]

- K-means (https://en.wikipedia.org/wiki/K-means_clustering)



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

```
int ndpi_cluster_bins(struct ndpi_bin *bins, u_int16_t num_bins,
                      u_int8_t num_clusters, u_int16_t *cluster_ids,
                      struct ndpi_bin *centroids);
```

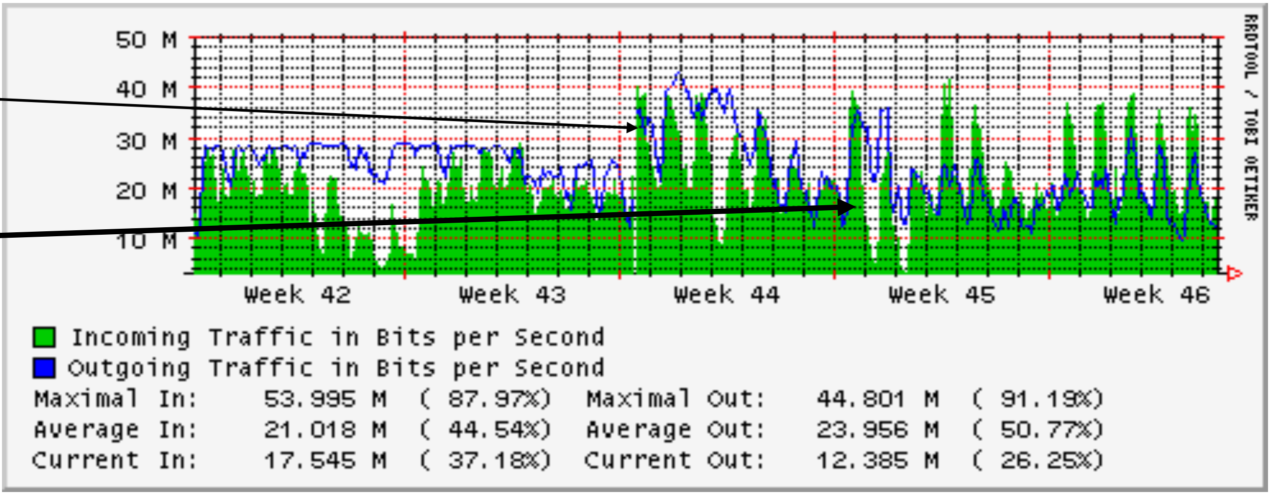
Traffic Measurement: Some Case Studies

Case Study: Bandwidth Management [1/3]

- Lack of bandwidth issues are not tackled purchasing additional bandwidth but managing the existing bandwidth.
  - Lesson learnt: the more bandwidth you have, the more you will use.

Upgrade: 34 -> 45 Mbit

Introduced CoS on P2P Traffic



```
🟩 Incoming Traffic in Bits per Second
🟦 Outgoing Traffic in Bits per Second
Maximal In:     53.995 M  ( 87.97%)  Maximal Out:    44.801 M  ( 91.19%)
Average In:     21.018 M  ( 44.54%)  Average Out:    23.956 M  ( 50.77%)
Current In:     17.545 M  ( 37.18%)  Current Out:    12.385 M  ( 26.25%)
```

Case Study: Bandwidth Management [2/3]

- Solution: Monitor and Find the Answer Yourself for Your Network (there’s no general solution).
  - Analyse how the available bandwidth is used (e.g. why protocol X is used ?).
  - Traffic and Flow matrix: who’s talking to who and what data are they exchanging ?

Case Study: Bandwidth Management [3/3]

Lessons Learnt from Practice:

- Poor performance can be due to use of backup-links because primary ones are unavailable (do you monitor failovers -via SNMP traps- such as STP, port status ?)
- Is your routing suboptimal or very dynamic? (SNMP provides you several MIBS for this purpose).
- Are you shaping too much? CoS (Class of Services) are good but don’t misuse them! (Why don’t you monitor the amount of traffic that is cut by your policers ?)

Case Study: Where is a Host?

- Association between IP address and name

```
deri@tar:~$ nslookup 131.114.21.22
    Server:  localhost
    Address:  127.0.0.1

    Name:    jake.unipi.it
    Address:  131.114.21.22
```

- Association between host and owner
  - namenslookup -type=SOA
  - WAIS [Wide Area Information System] http://www.ai.mit.edu/extra/the-net/wais.html
  - WHOIS [RFC-812]

Whois Example

```
Domain:              unipi.it
Status:              ACTIVE
Created:             1996-01-29 00:00:00
Last Update:         2008-02-14 00:02:47
Expire Date:         2009-01-29

Registrant
  Name:              Universita' degli Studi di Pisa
  ContactID:         UNIV302-ITNIC
  Address:           Centro SERRA
                     Pisa
                     56100
                     PI
                     IT
  Created:           2007-03-01 10:42:01
  Last Update:       2008-01-19 09:46:08

Registrar
  Organization:      Consortium GARR
  Name:              GARR-MNT

Nameservers
  serra.unipi.it
  nameserver.unipi.it
```

IP Geolocation

- Why IP geolocation is important ?
  - Location-based services and advertising/ marketing
  - Fraud detection of Internet transactions
  - Long-distance Internet traffic analysis (traceroute, AS latency study)
  - Web page language selection based on remote user location

Where in the World is host X ?

- RFC 1876: A Means for Expressing Location Information in the Domain Name System

- http://www.caida.org/tools/utilities/netgeo
- http://www.maxmind.com/
- http://www.geobytes.com/

Wi-Fi Positioning System (WPS)

Geolocating with Wi-Fi can exploit extra techniques for geolocations including:

- Wi-Fi Signal strength
- SSID (Service Set Identifier) and Mac Address Fingerprinting

Operating Systems (e.g. iOS/Android WPS) and Web browsers (e.g. Mozilla Location Services) natively offer APIs for geolocating hosts through

TCP/IP Stack Fingerprinting [1/3]

- Active
  Send probe packets in order to guess the host OS (http://nmap.org/)
- Passive
  Look at 3-way handshake and compare it with a database of known signatures in order to guess the host OS (http://ettercap.sf.net/, https://github.com/p0f/p0f).

TCP/IP Fingerprinting: Ettercap [2/3]

```
WWWW:MSS:TTL:WS:S:N:D:T:F:LEN:OS

WWWW: 4 digit hex field indicating the TCP Window Size
MSS : 4 digit hex field indicating the TCP Option Maximum Segment Size
      if omitted in the packet or unknown it is "_MSS"
TTL : 2 digit hex field indicating the IP Time To Live
WS  : 2 digit hex field indicating the TCP Option Window Scale
      if omitted in the packet or unknown it is "WS"
S   : 1 digit field indicating if the TCP Option SACK permitted is true
N   : 1 digit field indicating if the TCP Options contain a NOP
D   : 1 digit field indicating if the IP Don't Fragment flag is set
T   : 1 digit field indicating if the TCP Timestamp is present
F   : 1 digit ascii field indicating the flag of the packet
      S = SYN
      A = SYN + ACK
LEN : 2 digit hex field indicating the length of the packet
```

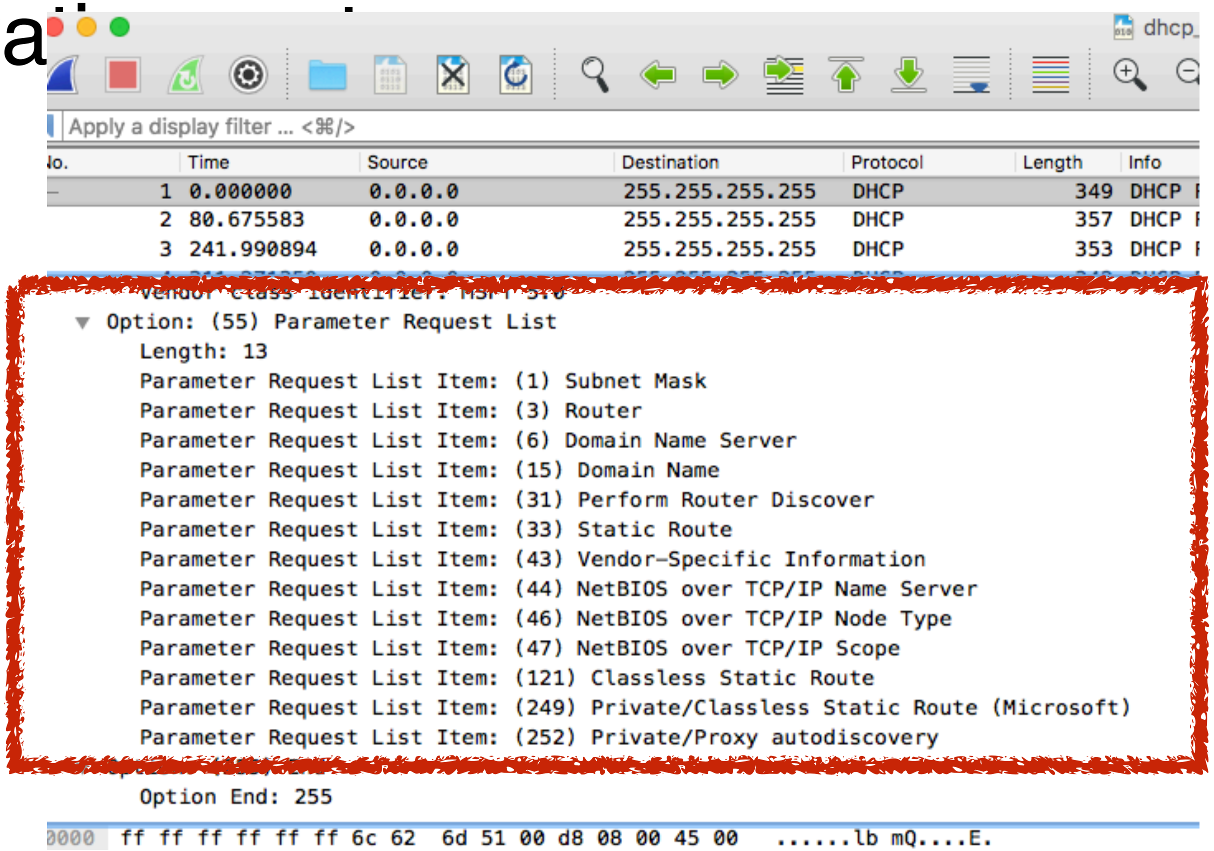
TCP/IP Fingerprinting: Limitations [3/3]

These techniques are less popular than some years ago because

- Operating systems randomize TCP/IP information making them less reliable than they used to be.
- There are several papers that describe how to defeat OS fingerprinting tools.
- OS signatures are often outdated making the tool usable only for detecting old OS versions.

DHCP Fingerprinting [1/2]

- DHCP fingerprinting is a technique that exploits DHCP field 55 to guess the opera

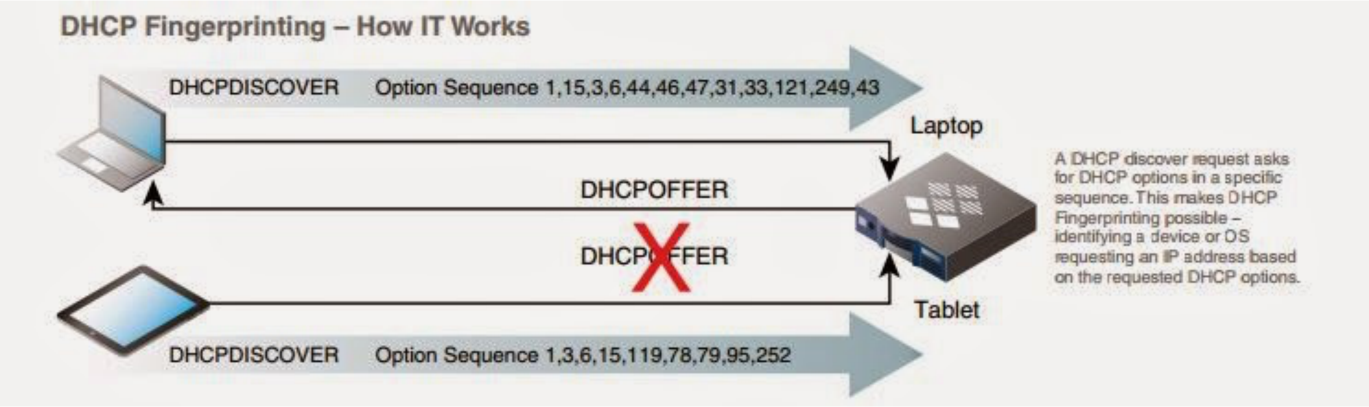


```
No.   Time         Source     Destination       Protocol   Length  Info
   1  0.000000     0.0.0.0    255.255.255.255   DHCP          349  DHCP
   2  80.675583    0.0.0.0    255.255.255.255   DHCP          357  DHCP
   3  241.990894   0.0.0.0    255.255.255.255   DHCP          353  DHCP
```

```
    Vendor class identifier: MSFT 5.0
  ▼ Option: (55) Parameter Request List
      Length: 13
      Parameter Request List Item: (1) Subnet Mask
      Parameter Request List Item: (3) Router
      Parameter Request List Item: (6) Domain Name Server
      Parameter Request List Item: (15) Domain Name
      Parameter Request List Item: (31) Perform Router Discover
      Parameter Request List Item: (33) Static Route
      Parameter Request List Item: (43) Vendor-Specific Information
      Parameter Request List Item: (44) NetBIOS over TCP/IP Name Server
      Parameter Request List Item: (46) NetBIOS over TCP/IP Node Type
      Parameter Request List Item: (47) NetBIOS over TCP/IP Scope
      Parameter Request List Item: (121) Classless Static Route
      Parameter Request List Item: (249) Private/Classless Static Route (Microsoft)
      Parameter Request List Item: (252) Private/Proxy autodiscovery

      Option End: 255
```

```
0000  ff ff ff ff ff ff 6c 62  6d 51 00 d8 08 00 45 00   ......lb mQ....E.
```

DHCP Fingerprinting [2/2]


DHCP Fingerprinting – How IT Works
DHCPDISCOVER
Option Sequence 1,15,3,6,44,46,47,31,33,121,249,43
Laptop
DHCPOFFER
DHCPOFFER
Tablet
DHCPDISCOVER
Option Sequence 1,3,6,15,119,78,79,95,252

A DHCP discover request asks for DHCP options in a specific sequence. This makes DHCP Fingerprinting possible – identifying a device or OS requesting an IP address based on the requested DHCP options.

- Many security vendors have created fingerprint databases embedded in devices, to prevent specific/outdated operating systems from connecting to a LAN and thus potentially creating security issues.

Use Case: Malicious Hosts Detection

- Monitoring tools need to know what IPs belong to compromised hosts in order to trigger alerts whenever such IPs are observed.
- Even though list of malicious IPs are not always a great solution as they can become quickly outdated, there are several companies that provide daily updates/digests.
- One of the (free) most popular ones is https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt

DGA Domains [1/3]

- DGAs (Domain Generation Algorithm) are used i various families of malware to generate rendezvous points for command & control (see previous slide).
- Crypto-locker apps often use DGAs for this purpose.
- Usually DGAs take as input a seed that is used to generate many pseudo-random domain names.
- Tor HTTPS certificates use DGA-generated host names that can be used to detect this protocol by

DGA Domains [2/3]

- The malware keep generating domain names up until there is one registered that is used to connect to the “malware network”.

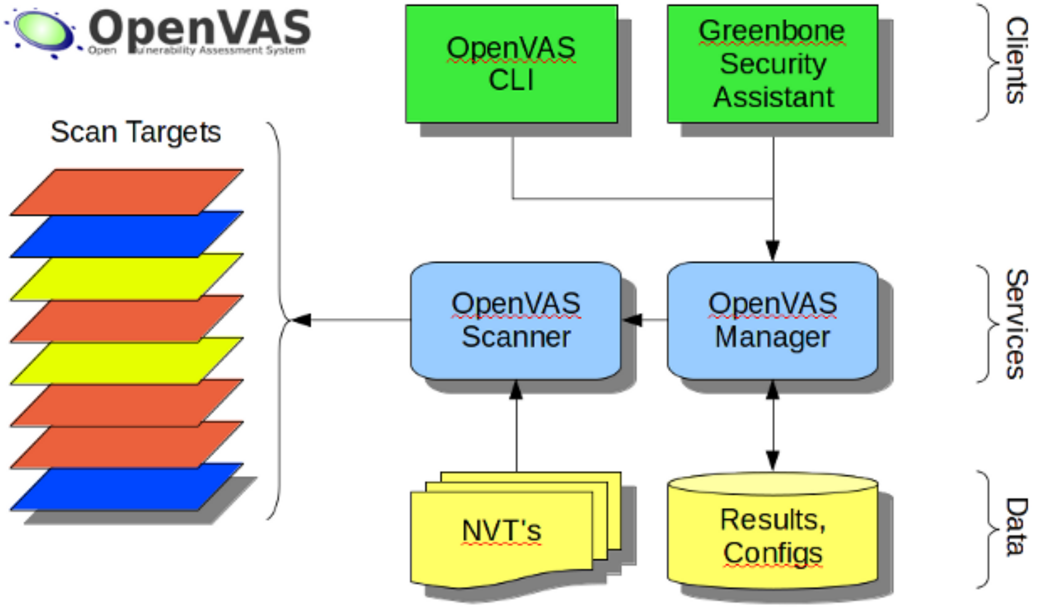
```
def generate_domain(year, month, day):
    """Generates a domain name for the given date."""
    domain = ""

    for i in range(16):
        year = ((year ^ 8 * year) >> 11) ^ ((year & 0xFFFFFFF0) << 17)
        month = ((month ^ 4 * month) >> 25) ^ 16 * (month & 0xFFFFFFF8)
        day = ((day ^ (day << 13)) >> 19) ^ ((day & 0xFFFFFFFE) << 12)
        domain += chr(((year ^ month ^ day) % 25) + 97)

    return domain
```

DGA Domains [3/3]

- DGA domains can be detected using impossible/good bi-grams (i.e. two letter combinations that do not exist in languages) for dissecting the domain name.
- Example:
  - www.fgd2iwya7vinfutj5wq5we.com
  - www.qbtxzhetq4s2f.com
  - www.fgd2iwya7vinfutj5wq5we.net

Case Study: Security Scanner

- Nessus [http://www.tenable.com/]
- OpenVAS [http://www.openvas.org/]



Case Study: Network Security

Security is a process not a product (BS7799).

- Are you able to detect network anomalies ?
- Are you sure you know what to monitor ? Most of issues are produced by traffic we never expect to see in your network: monitor everything, filter things you expect to see, look at the rest and explain this happened.
- Do you have an automatic fault recovery ? Supposing you detect the problem (e.g. SNMP trap) is your system reacting automatically or waiting for you to come back

Case Study: P2P Detection

- P2P is hard to detect with standard methods:
  - It cannot be detected using fingerprints (e.g. port-2-protocol association).
- However it can be detected…
  - It can be detected in terms of deviation for a standard behaviour (e.g. a workstation cannot open more than X connections/minute nor keep more than Y connections open).
  - Analysis of initial payload bytes in order to detect the protocol.
  - High percentage of unsuccessful TCP connection establishment.
  - Packets/Bytes ratio above the average (P2P sources send many packets, mostly for talking with peers).
  - Identification of client-to-client (> 1024) communications with no FTP command channel open.

Case Study: SPAM Detection

- Large, open networks (e.g. universities, ISPs) are the best places for sending spam (unsolicited email).
- How to identify SPAM sources:
  - Problem similar to P2P but simpler (SMTP-only, 1 connection = 1 email) .
  - Select the set of top N SMTP senders.
  - Remove from the set all the known SMTP servers.
  - Studies shown that in average an host does not send more that 8-10 emails/minute.
  - Very simple problem to tackle using flow-based protocols

Case Study: Virus/Trojan Detection

- Problem similar to SPAM detection but more complex as the protocol/ports used are not fixed.
- Attacks do not have a precise target: they somehow behave as network scanners.
- Detection:
  - If the problem is known (e.g. traffic on UDP port 135) focus only on these selected traffic patterns.
  - Keep an eye on ICMP messages (e.g. port/destination unreachable) as they are the best way to detect network scanners.

Final Remarks

So What Can we Basically Expect from Network Monitoring ?

- Ability to automatically detect those issues that are permanently monitored (e.g. no traffic on the backbone link: network down ?).
- Receive alarms about potential (e.g. CPU Utilisation is too high) and real (e.g. disk is full) problems.
- Automatic notification and restore for known problems with known solutions (e.g. main link down, the backup link is used).
- Report to humans for all those problems that need attention and that cannot be restored (e.g. host X is unreachable).

Monitoring Caveats

- If a monitoring application needs human assistance for a problem that could be solved automatically, then the monitoring applications is not completely useful.
- Alarm (100% sure that there is something wrong) != Warning (maybe this is an issue): don’t pretend to be precise/catastrophic if this is not the case.
- Alarms are useless if there’s nobody who looks at them.
- Too many (false) alarms = no alarm: humans tend to ignore facts if some of them are proven to be false.