# Communications Network Design

## lecture 21

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

Discipline of Applied Mathematics
School of Mathematical Sciences
University of Adelaide

March 2, 2009

# Input data

One of the key problems in network design is how we get the input data for network design. In particular traffic matrices are one of the key inputs, but are not always easy to measure.

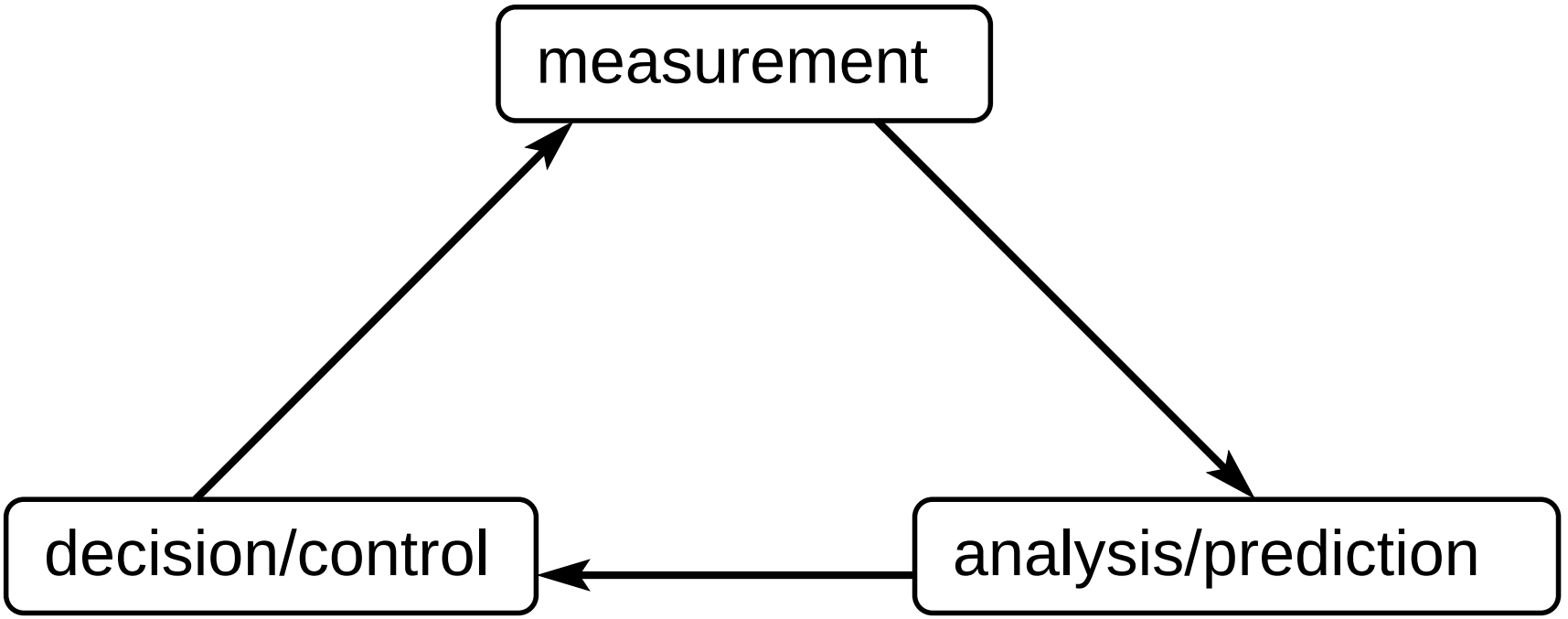
"Measure what is measurable, and make measurable what is not so."

Galilei, Galileo (1564 - 1642)

# Planning Tasks

- optimization (obviously)
- measurements
  - before we can optimize, we need to know the inputs
    - costs
    - traffic matrix
  - some might be given
  - most need to be **measured**
- prediction
  - need to predict inputs out to the planning horizon

# Planning Cycle

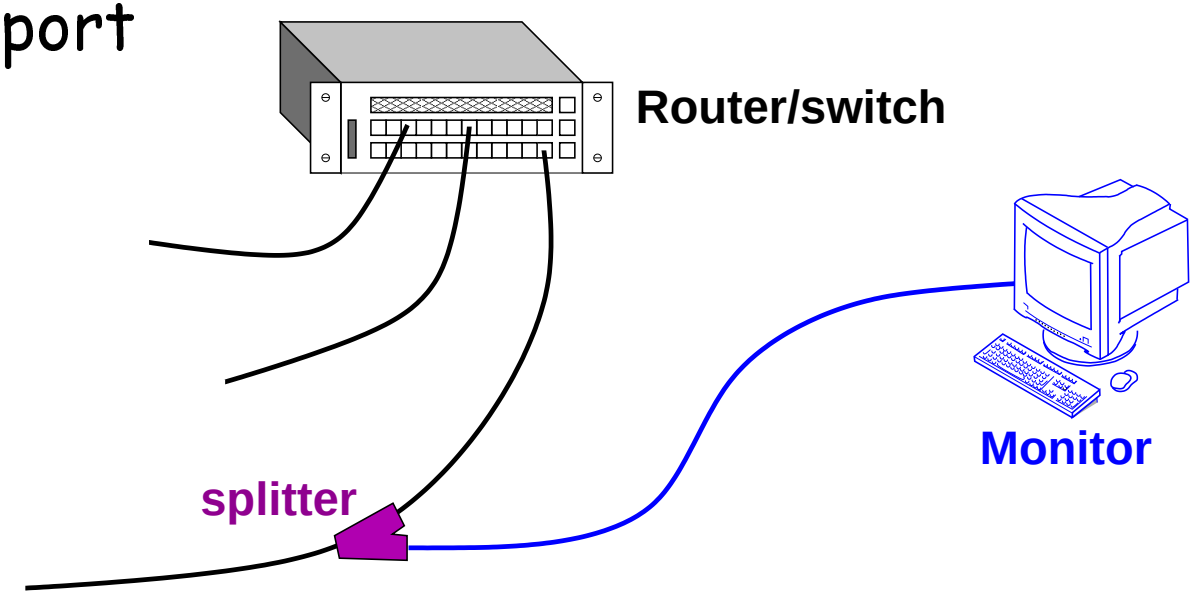


# Traffic Matrices

- key input to design is traffic matrix
  - traffic from node $i$ to $j$.
  - usually averaged over some interval
- notation
  - notation: offered traffic between O-D (Origin-Destination) pair $(p, q)$ is $t_{pq}$
  - can write as a matrix $T = [t_{pq}]$
  - sometimes also write as vector $\mathbf{x}$ by stacking columns of $T$.
- how can we measure it?

# Packet traces

- tap a link, or router
  - optical, or electronic splitter/coupler
  - monitoring port




- record every packet's
  - size
  - time (of first byte)
  - headers (IP, TCP, possibly more)

# Packet traces issues

- timing resolution/accuracy
  - clock resolution (packet transmission time for 1500 byte packet on OC48, 2.5 Gbps, is 4.8 microseconds)
  - clock accuracy: PC clocks have drift, plus interupt latency
  - 2.5 Gbps, min size (40 byte) packets, you have 128 ns to timestamp the packet
- storing the data
  - OC48 2.5 Gbps data rate
  - minimum size packets are basically all header
  - need to get 2.5 Gbps to disk (which is hard)

# Packet trace info

- IP header
  - version, header length, TTL, checksum
  - flags: ToS, ...
  - packet length (size in octets/bytes)
  - source and destination address
  - options
- TCP/UDP header
  - source and destination ports
  - sequence, and ACK numbers, checksum
  - flags: SYN, ACK, ...
  - data offset and pointers
  - options

# tcpdump output

```
1078208222.013538 129.127.5.110.1346 > 229.55.150.208.1345: udp 150
1078208222.754748 129.127.5.117.631 > 129.127.5.255.631: udp 139
1078208222.948664 129.127.5.56.1025 > 129.127.5.255.111: udp 136
1078208222.948673 129.127.5.56.1025 > 224.0.2.2.111: udp 136
1078208223.257521 129.127.5.234.1346 > 229.55.150.208.1345: udp 150
1078208223.516606 129.127.4.9.513 > 129.127.5.255.513: udp 108 (DF) [ttl 1]
1078208223.755331 129.127.5.117.631 > 129.127.5.255.631: udp 137
1078208224.755755 129.127.5.117.631 > 129.127.5.255.631: udp 133
1078208225.756207 129.127.5.117.631 > 129.127.5.255.631: udp 158
1078208228.137869 129.127.5.56.1025 > 129.127.5.255.111: udp 136
1078208228.137881 129.127.5.56.1025 > 224.0.2.2.111: udp 136
1078208231.728471 129.127.4.177.5353 > 224.0.0.251.5353: udp 105
1078208233.257055 129.127.5.56.1025 > 129.127.5.255.111: udp 136
1078208233.257066 129.127.5.56.1025 > 224.0.2.2.111: udp 136
```

# Packet trace example

## Packet trace (snippet)

| | IP header | | | | TCP/UDP header | |
|---|---|---|---|---|---|---|
| timestamp | proto. | src IP | dst IP | size | src port | dst port |
| 1078208222.014 | udp | 129.127.5.110 | 229.55.150.208 | 150 | 1346 | 1345 |
| 1078208222.755 | udp | 129.127.5.117 | 129.127.5.255 | 139 | 631 | 631 |
| 1078208222.949 | udp | 129.127.5.56 | 129.127.5.255 | 136 | 1025 | 111 |
| 1078208222.949 | udp | 129.127.5.56 | 224.0.2.2 | 136 | 1025 | 111 |
| 1078208223.258 | udp | 129.127.5.234 | 229.55.150.208 | 150 | 1346 | 1345 |
| 1078208223.517 | udp | 129.127.4.9 | 129.127.5.255 | 108 | 513 | 513 |
| 1078208223.755 | udp | 129.127.5.117 | 129.127.5.255 | 137 | 631 | 631 |
| 1078208224.756 | udp | 129.127.5.117 | 129.127.5.255 | 133 | 631 | 631 |
| 1078208225.756 | udp | 129.127.5.117 | 129.127.5.255 | 158 | 631 | 631 |
| 1078208228.138 | udp | 129.127.5.56 | 129.127.5.255 | 136 | 1025 | 111 |
| 1078208228.138 | udp | 129.127.5.56 | 224.0.2.2 | 136 | 1025 | 111 |
| 1078208231.728 | udp | 129.127.4.177 | 224.0.0.251 | 105 | 5353 | 5353 |
| 1078208233.257 | udp | 129.127.5.56 | 129.127.5.255 | 136 | 1025 | 111 |
| 1078208233.257 | udp | 129.127.5.56 | 224.0.2.2 | 136 | 1025 | 111 |

# Packet traces pros

- get to see almost everything
  - source
  - destination
  - ports and protocol
  - TCP flags
- to see everything, need to store more than just 40 bytes (e.g. need application headers)
  - but you can!
- very fine grained (timewise)
- suitable for just about any type of modeling

# Packet traces cons

- cost of monitors (1 per link)
  - can put multiple cards/ports on one monitor for low speed monitors
  - have to add installation and maintenance costs
- ginormous datasets
  - at OC48, it takes less than 1 hour to collect a terabyte (min sized packets)
  - even with 1500 byte packet, it only takes 33 hours to collect a terabyte.
  - even simple processing is slow!

# Reducing the data size

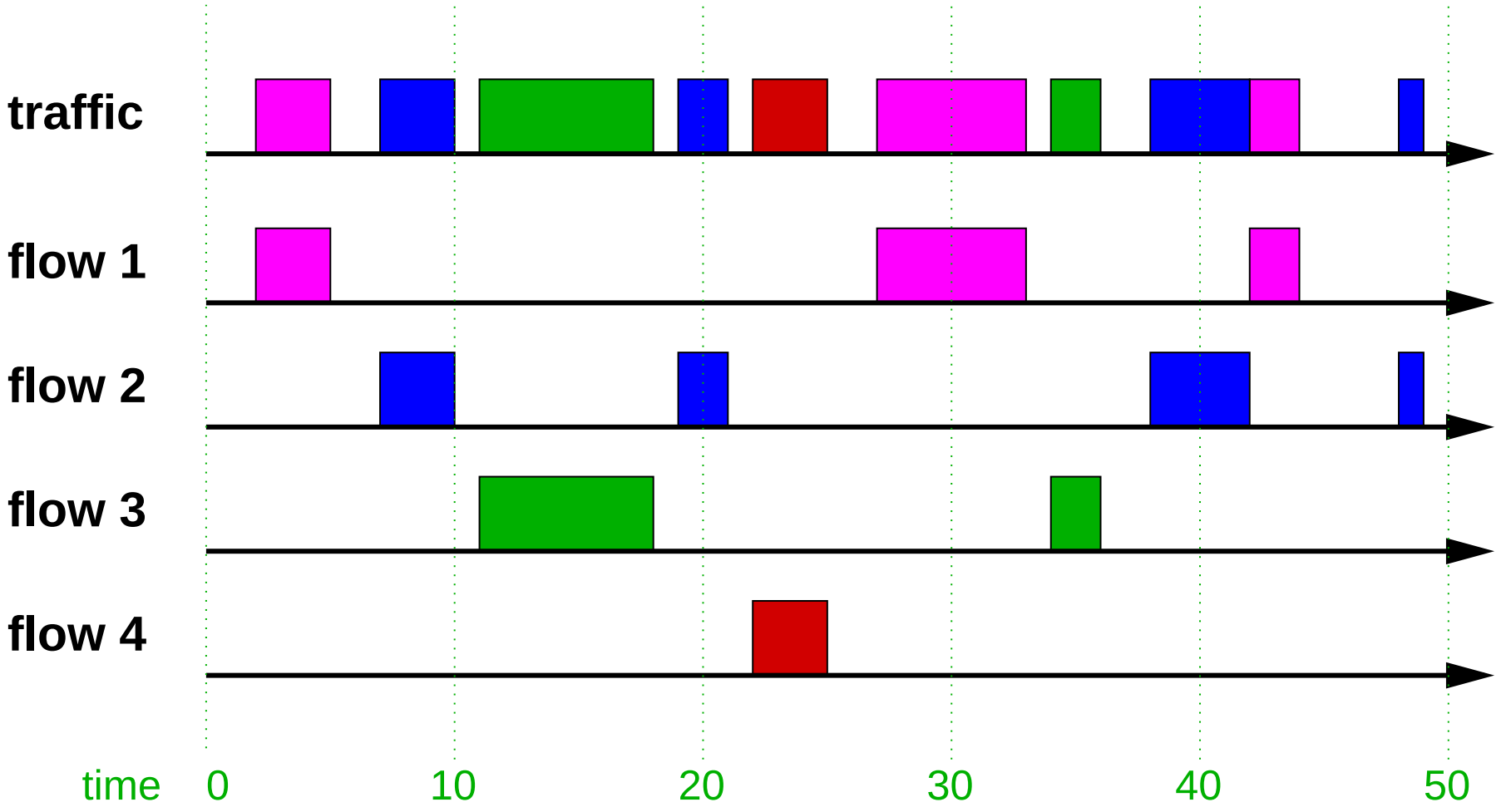
A number of operations can reduce the dataset size

- **sampling:**
  - standard statistical approach
  - simplest case, sample every $N$th packet, or randomly choose $1$ in $N$ packets.
- **filtering:** only look at packets which meet certain requirements, e.g.
  - only TCP packets
  - only packets between two specific IP addresses
- **aggregation:** reduce the granularity of the data somehow.
  - aggregate over time, or **keys**

# Netflow

- idea: aggregate to close to a TCP connection
  - keep one record per flow
  - record key: IP source, dest, protocol and TCP source, dest port
  - record stores: packets, bytes, TCP flags, start and stop time
- practicality: aggregate by key
  - flush records using
    - timeout, O(30 seconds), (to separate similar connections, e.g. DNS)
    - when flow record cache is full
    - every 15 minutes (stop staleness of records)
  - not bi-directional

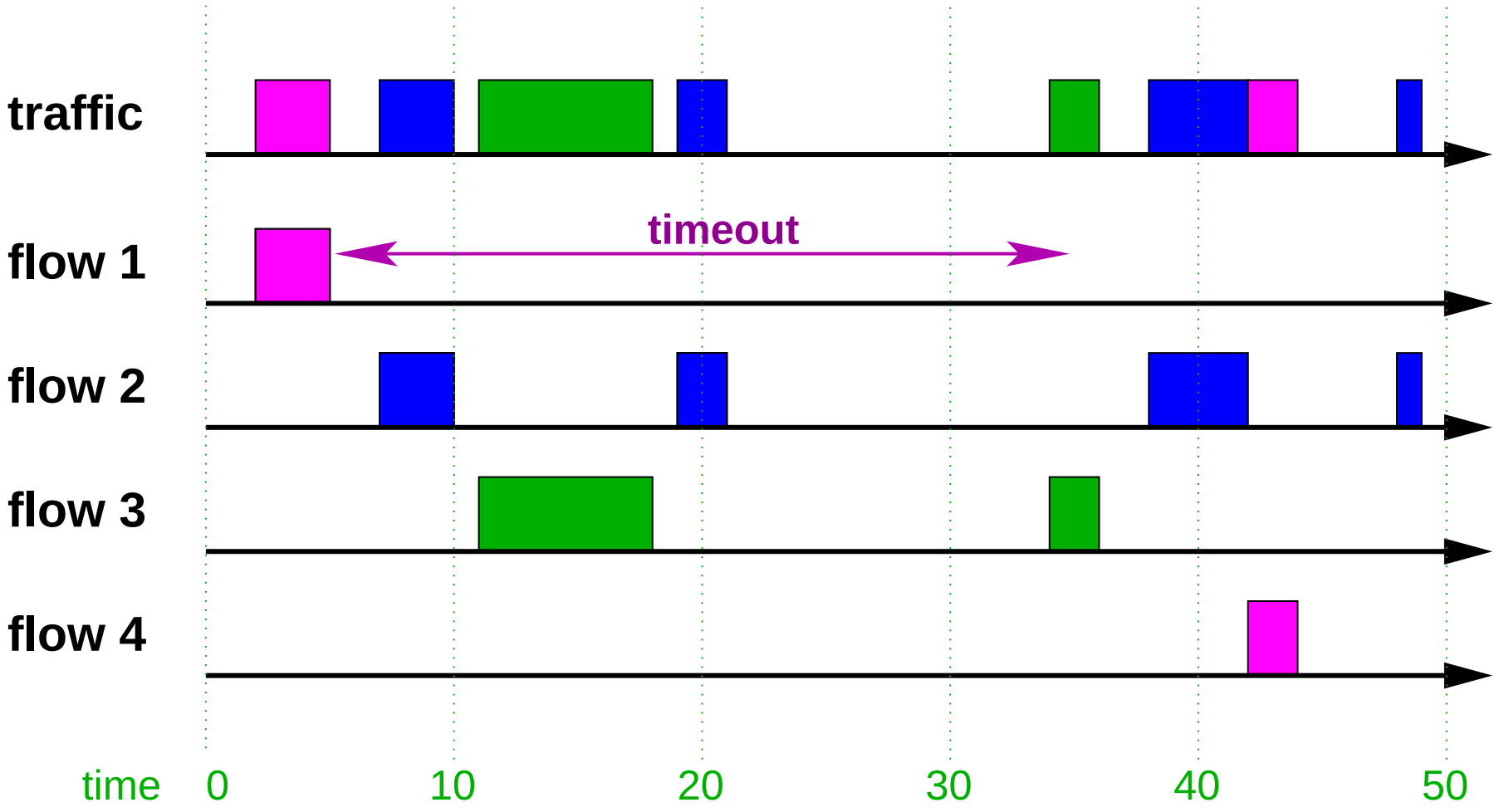
# Netflow



# Netflow records

| key | packets | bytes | start time | stop time |
|---|---|---|---|---|
| pink | 3 | 11 | 2 | 44 |
| blue | 4 | 10 | 7 | 49 |
| green | 2 | 9 | 11 | 6 |
| red | 1 | 3 | 22 | 25 |

# Timeouts



# Netflow records

| key | packets | bytes | start time | stop time |
|---|---|---|---|---|
| pink 1 | 1 | 3 | 2 | 5 |
| blue | 4 | 10 | 7 | 49 |
| green | 2 | 9 | 11 | 6 |
| pink 2 | 1 | 2 | 42 | 44 |

# Netflow example

## Packet trace (snippet)

| timestamp | protocol | src IP | dst IP | src port | dst port | size |
|---|---|---|---|---|---|---|
| 1078208222.014 | udp | 129.127.5.110 | 229.55.150.208 | 1346 | 1345 | 150 |
| 1078208222.755 | udp | 129.127.5.117 | 129.127.5.255 | 631 | 631 | 139 |
| 1078208222.949 | udp | 129.127.5.56 | 129.127.5.255 | 1025 | 111 | 136 |
| 1078208222.949 | udp | 129.127.5.56 | 224.0.2.2 | 1025 | 111 | 136 |
| 1078208223.258 | udp | 129.127.5.234 | 229.55.150.208 | 1346 | 1345 | 150 |
| 1078208223.517 | udp | 129.127.4.9 | 129.127.5.255 | 513 | 513 | 108 |
| 1078208223.755 | udp | 129.127.5.117 | 129.127.5.255 | 631 | 631 | 137 |
| 1078208224.756 | udp | 129.127.5.117 | 129.127.5.255 | 631 | 631 | 133 |
| 1078208225.756 | udp | 129.127.5.117 | 129.127.5.255 | 631 | 631 | 158 |
| 1078208228.138 | udp | 129.127.5.56 | 129.127.5.255 | 1025 | 111 | 136 |
| 1078208228.138 | udp | 129.127.5.56 | 224.0.2.2 | 1025 | 111 | 136 |
| 1078208231.728 | udp | 129.127.4.177 | 224.0.0.251 | 5353 | 5353 | 105 |
| 1078208233.257 | udp | 129.127.5.56 | 129.127.5.255 | 1025 | 111 | 136 |
| 1078208233.257 | udp | 129.127.5.56 | 224.0.2.2 | 1025 | 111 | 136 |

# Netflow example

Netflow records:

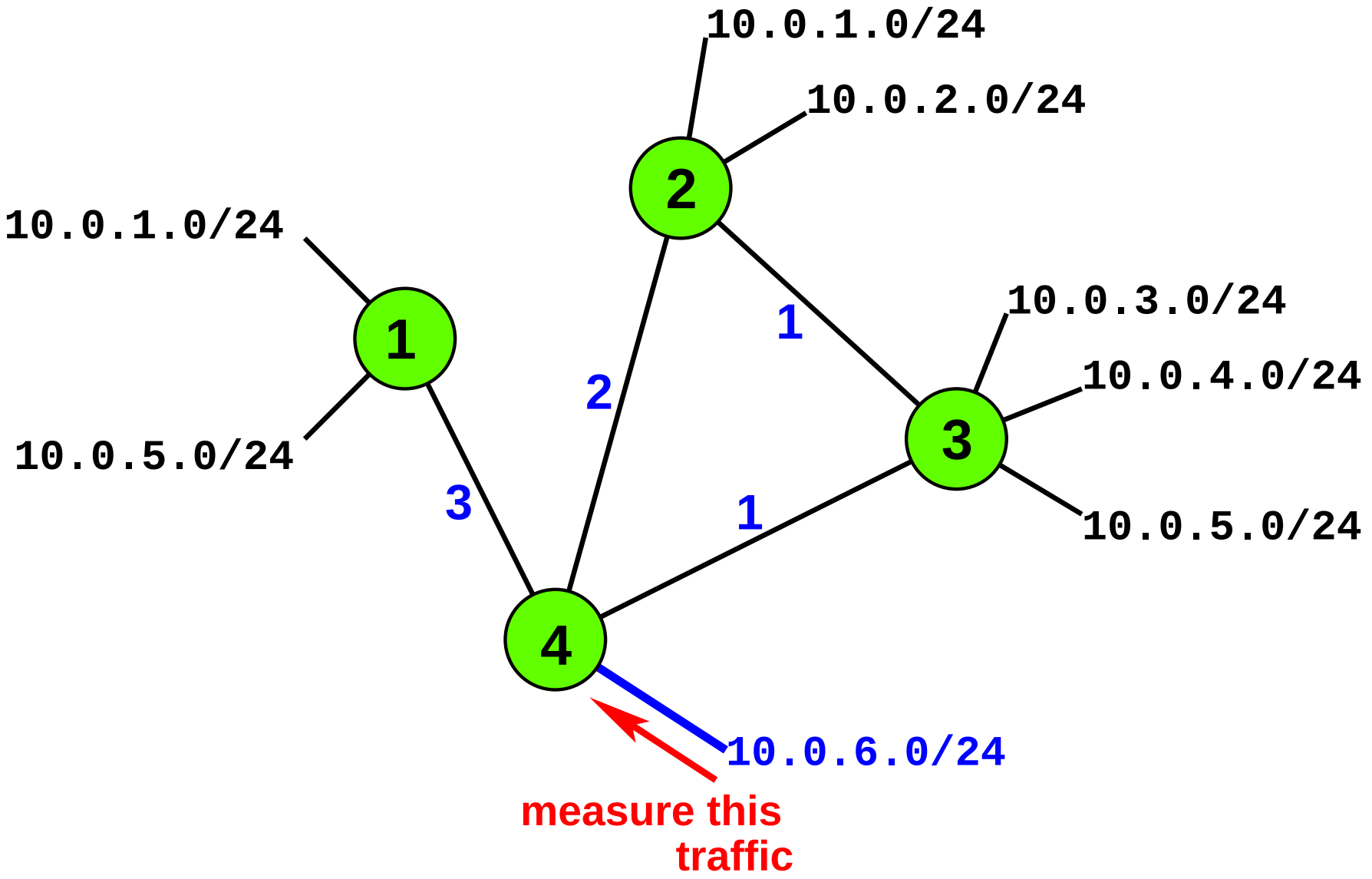
| protocol | src IP | dst IP | src port | dst port | dur | pack. | bytes |
|---|---|---|---|---|---|---|---|
| udp | 129.127.5.234 | 229.55.150.208 | 1346 | 1345 | 0.0 | 1 | 154 |
| udp | 129.127.5.56 | 224.0.2.2 | 1025 | 111 | 10.3 | 3 | 410 |
| udp | 129.127.5.110 | 229.55.150.208 | 1346 | 1345 | 0.0 | 1 | 154 |
| udp | 129.127.4.177 | 224.0.0.251 | 5353 | 5353 | 0.0 | 1 | 102 |
| udp | 129.127.5.117 | 129.127.5.255 | 631 | 631 | 3.0 | 4 | 563 |
| udp | 129.127.5.56 | 129.127.5.255 | 1025 | 111 | 10.3 | 3 | 410 |
| udp | 129.127.4.9 | 129.127.5.255 | 513 | 513 | 0.0 | 1 | 113 |

# Netflow example application

Traffic matrix

- measure netflow at network entry points (ingress)
  - provides traffic from IP source to dest. address
- aggregate to prefix level (across all ports)
  - get a matrix from IP source prefix to destination prefix
  - matrix is sparse, but large (100k+ prefixes)
  - also, one matrix per ingress point to the network
- to get ingress/egress traffic matrix, need to
  - simulate routing, to compute egress points per ingress, and prefix
  - then aggregate again

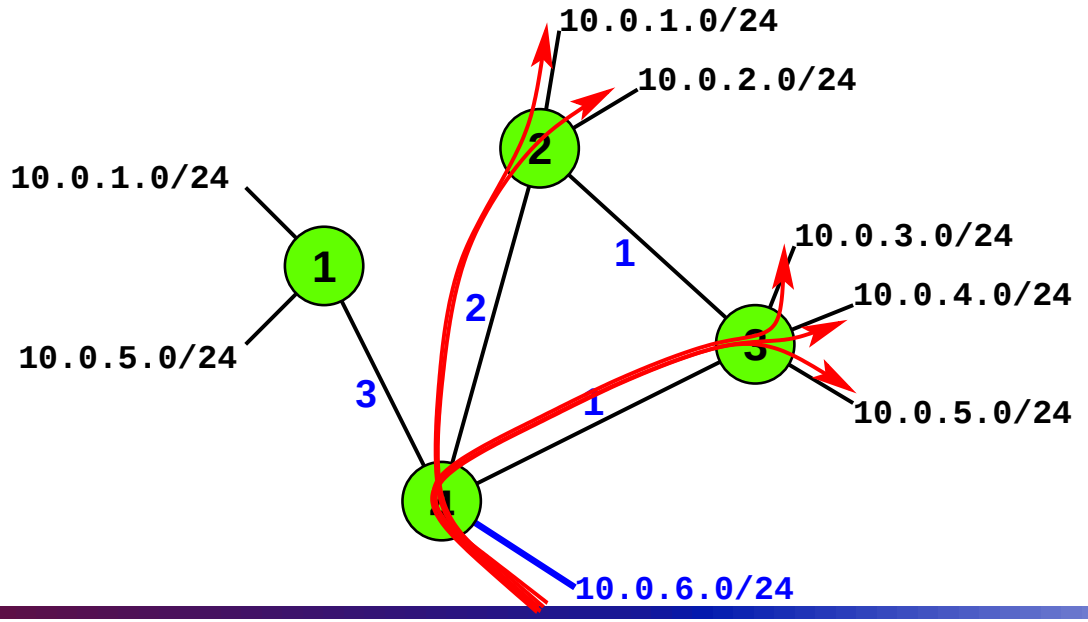
# Netflow example application



# Example traffic matrix computation

Measured incoming traffic at node 4

| ingress node | source prefix | dest prefix | volume | egress node |
|---|---|---|---|---|
| 4 | 10.0.6.0/24 | 10.0.1.0/24 | 10 | 2 |
| 4 | 10.0.6.0/24 | 10.0.2.0/24 | 11 | 2 |
| 4 | 10.0.6.0/24 | 10.0.3.0/24 | 21 | 3 |
| 4 | 10.0.6.0/24 | 10.0.4.0/24 | 6 | 3 |
| 4 | 10.0.6.0/24 | 10.0.5.0/24 | 3 | 3 |



# Example traffic matrix computation

Ingress-egress traffic

| ingress node | egress node | volume |
|---|---|---|
| 4 | 1 | 0 |
| 4 | 2 | 21 |
| 4 | 3 | 30 |
| 4 | 4 | 0 |

Ingress-egress traffic matrix

| | | egress node | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| ingress node | 1 | - | - | - | - |
| | 2 | - | - | - | - |
| | 3 | - | - | - | - |
| | 4 | 0 | 21 | 30 | 0 |

# Netflow pros

- data volume reduction
  - 100:1 reduction on packet traces
  - conservative estimate for real traffic
- collected by router
  - doesn't require special equipment
  - no maintenance cost
- almost standard these days
- keeps much of the useful traffic parameters
  - flows map (somewhat) to connections
  - can see where traffic is going
  - port numbers still visible

# Netflow cons

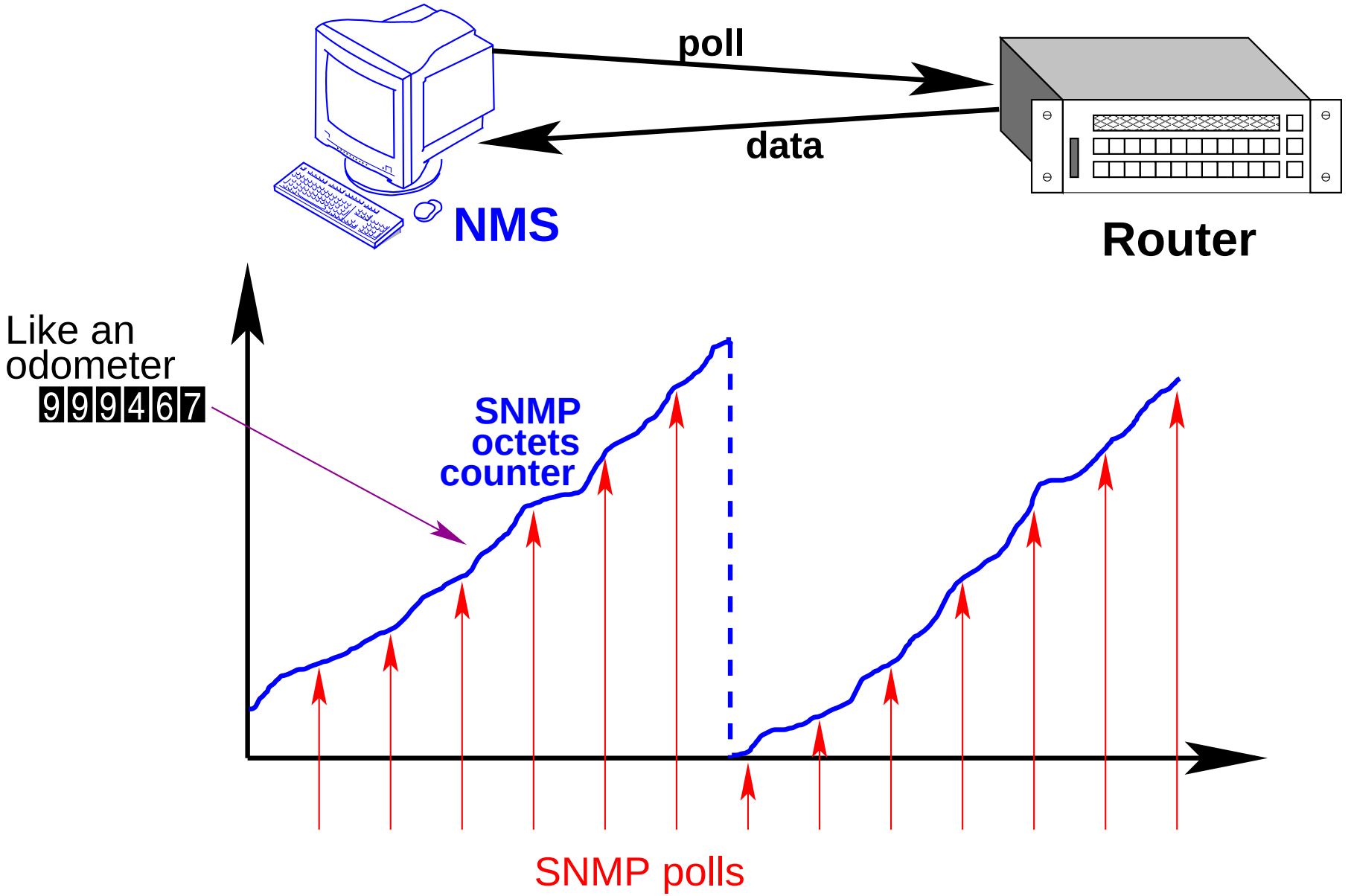
- historically poor vendor support
  - feature interations
  - bugs
  - performance impact on router
- still large volumes
  - can be reduced with sampling
- loose some detail
  - application level headers are now lost forever
  - loose time granularity
    - only have start and stop of flows
    - several minutes (cache flushing cap)
    - don't see traffic per time interval

# SNMP

Simple Network Management Protocol

- not just for measurements
- allows one to collect MIBs (Management Information Bases)
- MIB-II implemented on almost all network equipment
- includes:
  - counts of packets
  - counts of bytes
  - ...

# SNMP data collection



# Irregular sampling

- Why?
  - missing data (transport on UDP, in-band)
  - delays in polls
  - poler sync (mulitple pollers)
  - staggered polls
- Why care?
  - time series analysis
  - comparisons/totals between links
  - correlation to other data sources

# SNMP pros

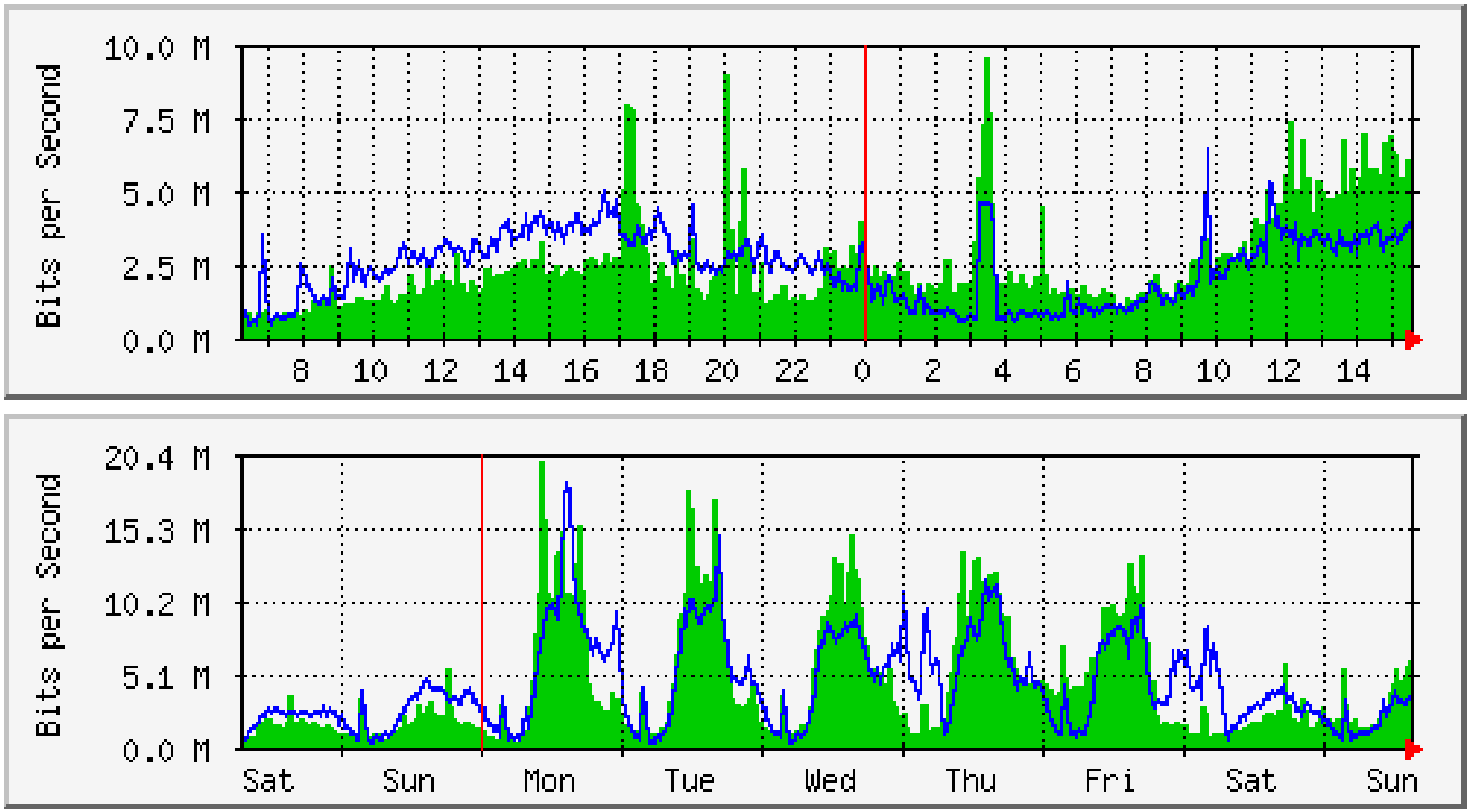
- simple, and easy
- low overhead
- ubiquitous
- lots of practice in use
- lots of historical data

# SNMP cons

- data quality
  - missing data
  - ambiguous data
  - irregular time sampling
- coarse time scale (> 1 minutes, typically 5)
- octet counters don't tell you
  - what type of traffic (applications)
  - where traffic is going (source and destination)
  - hard to detect DoS, or other such attacks
- coarse time scale (> 1 minutes, typically 5)

# Example SNMP data

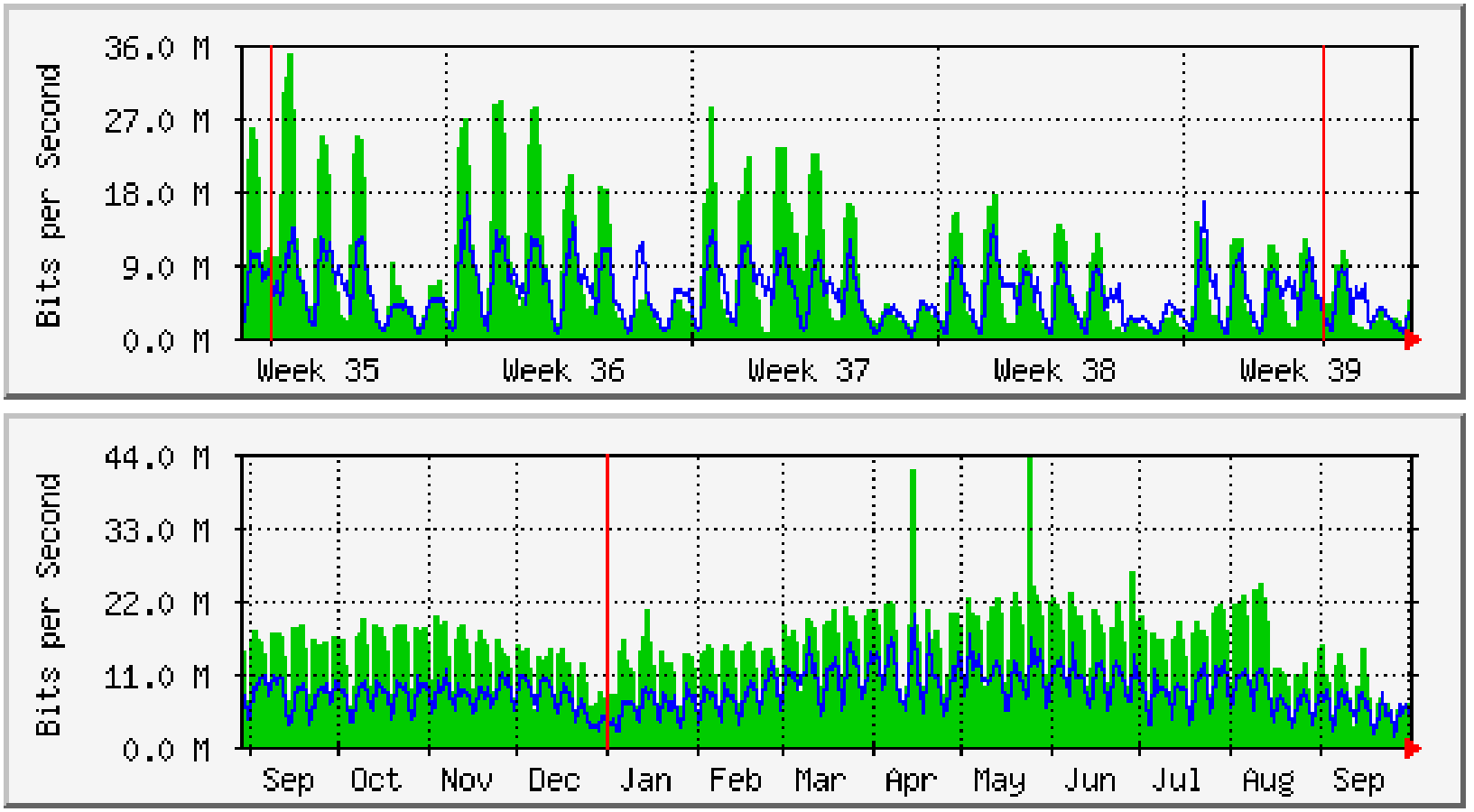
RRD Tool (or MRTG) are the commonest form of available SNMP data




http://www.aarnet.edu.au/network/mrtg.html

# Example SNMP data

RRD Tool (or MRTG) are the commonest form of available SNMP data

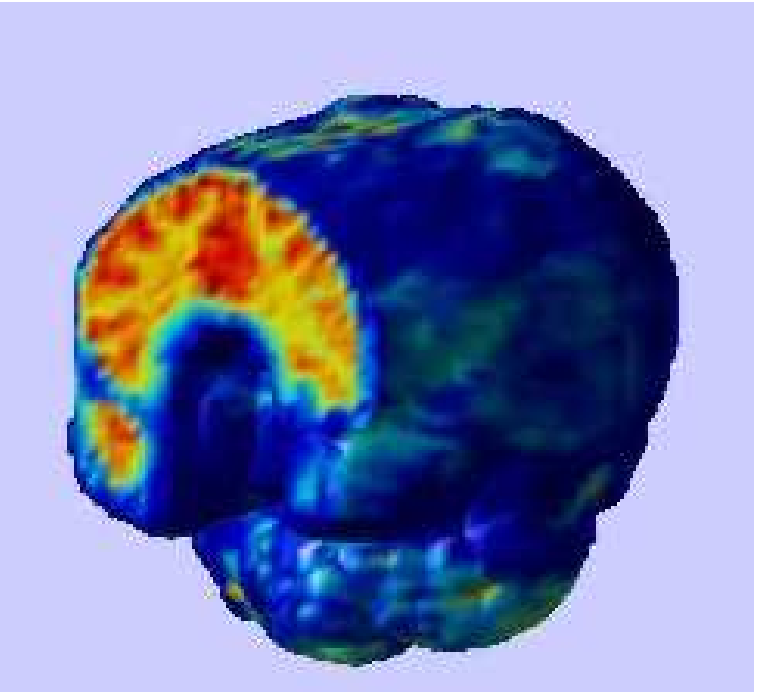



http://www.aarnet.edu.au/network/mrtg.html

# The Problem

- SNMP is easy to get
  - doesn give traffic matrix
- Traffic matrix is not easy
- A solution: network tomography
  - get TM data from links

# Who put the CAT in CATscan?



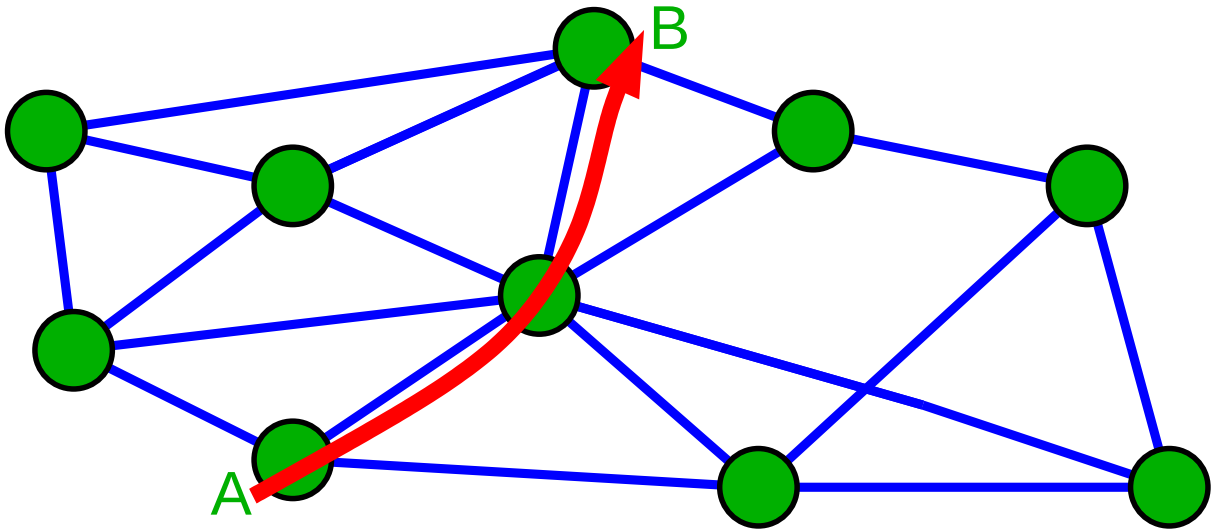
- people don't like you cutting their head open!
- so indirect methods are used to peer inside
- Computer Axial Tomography (CAT)
  - Tomo- from the Greek **tomos** meaning "section"

# Performance Tomography

- we need ways to see inside the Internet
- similar to CATscan
  - we often only have indirect measurements
    - ISPs don't give each other access to their networks
    - practical problems in measurements
  - but the measurements are different
    - we have end-to-end performance measurements
    - want to get link performance measurements
- result is an inverse problem
  - more about these later

# History

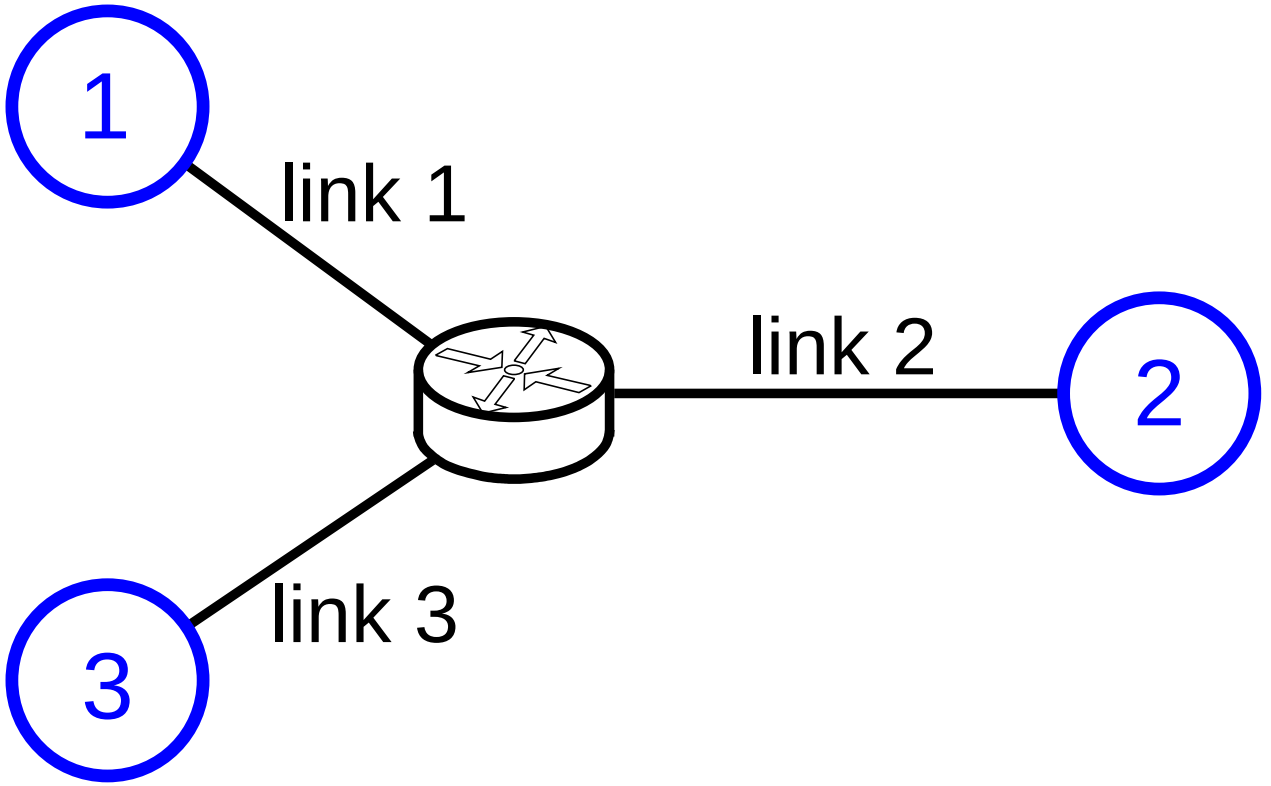
- Vardi [?] coined the term Network Tomography
- term referred to a different problem
  - Traffic matrix estimation




$$T = \begin{pmatrix} x_{AA} & x_{AB} & x_{AC} & \cdots \\ x_{BA} & x_{BB} & x_{BC} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

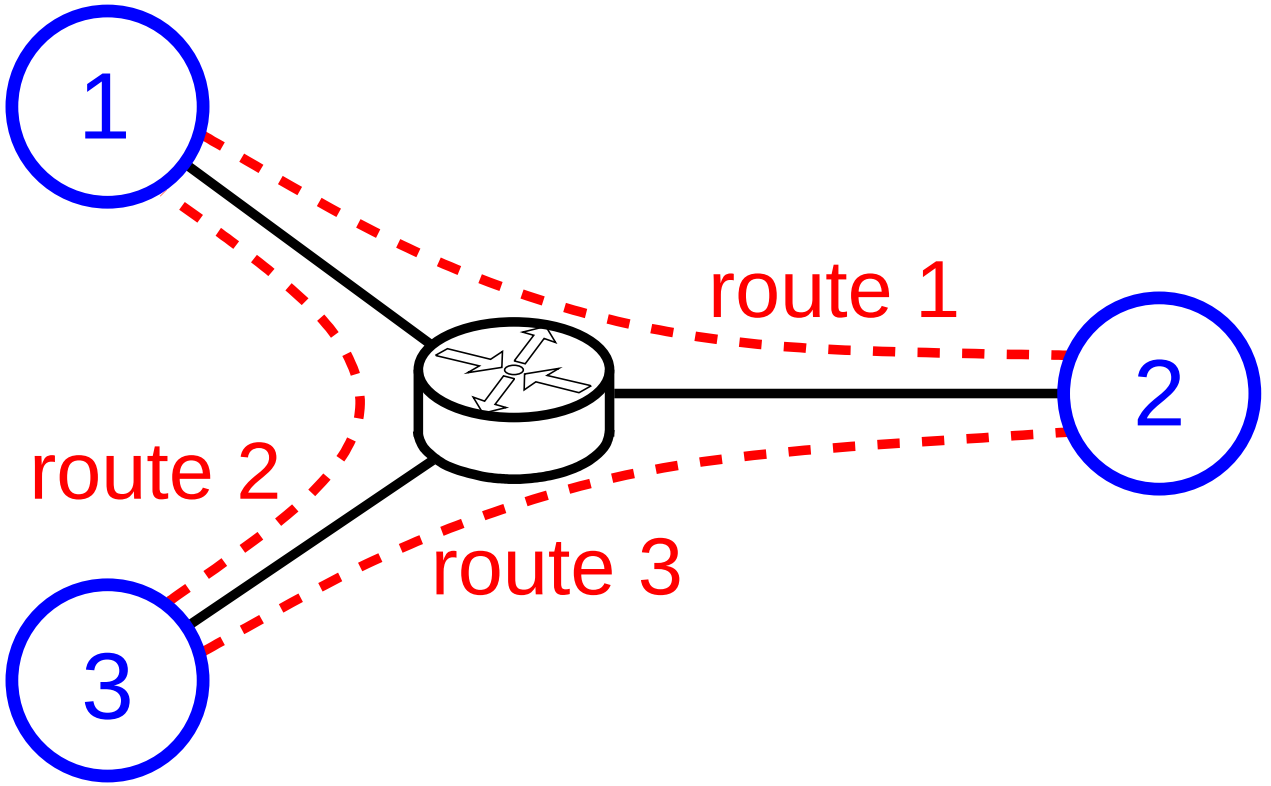
# Problem formulation

- measured traffic on link $i$ is $y_i$



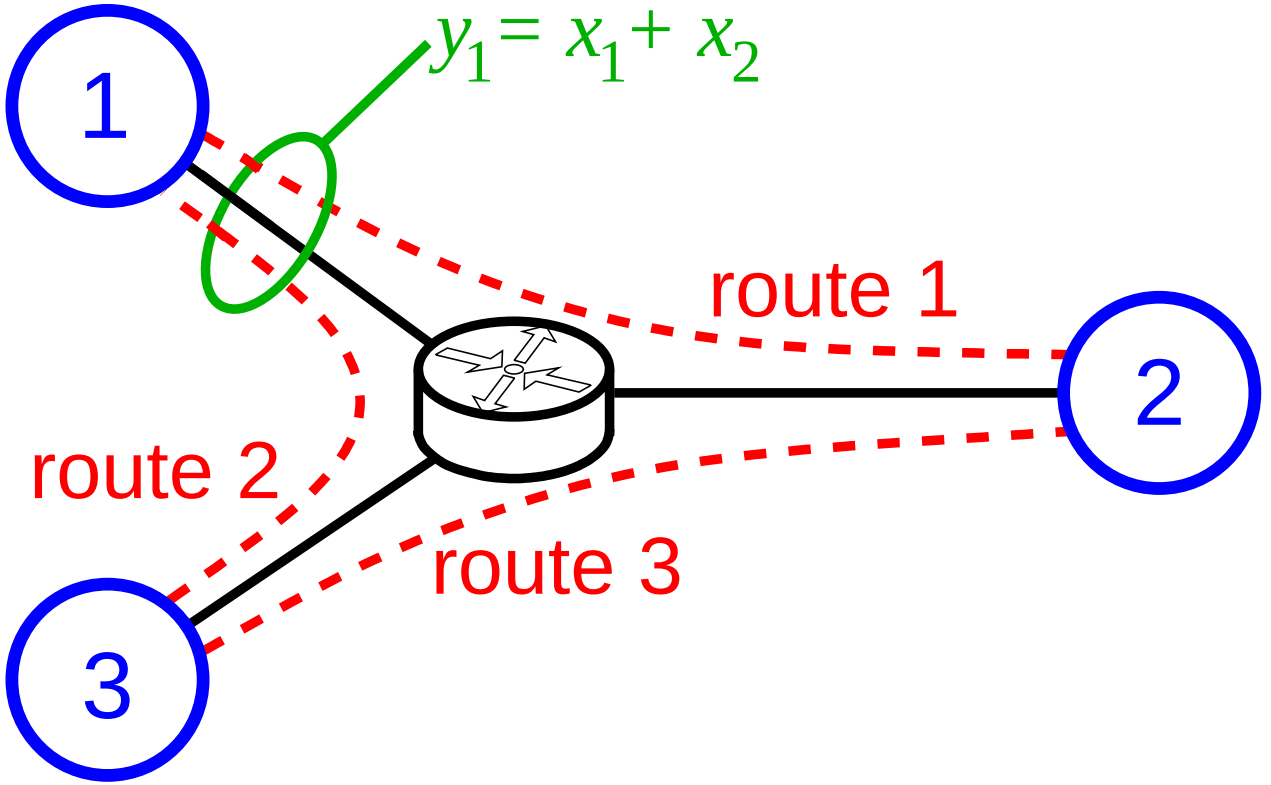
# Problem formulation

- want to estimate traffic on route $j$ is $x_j$



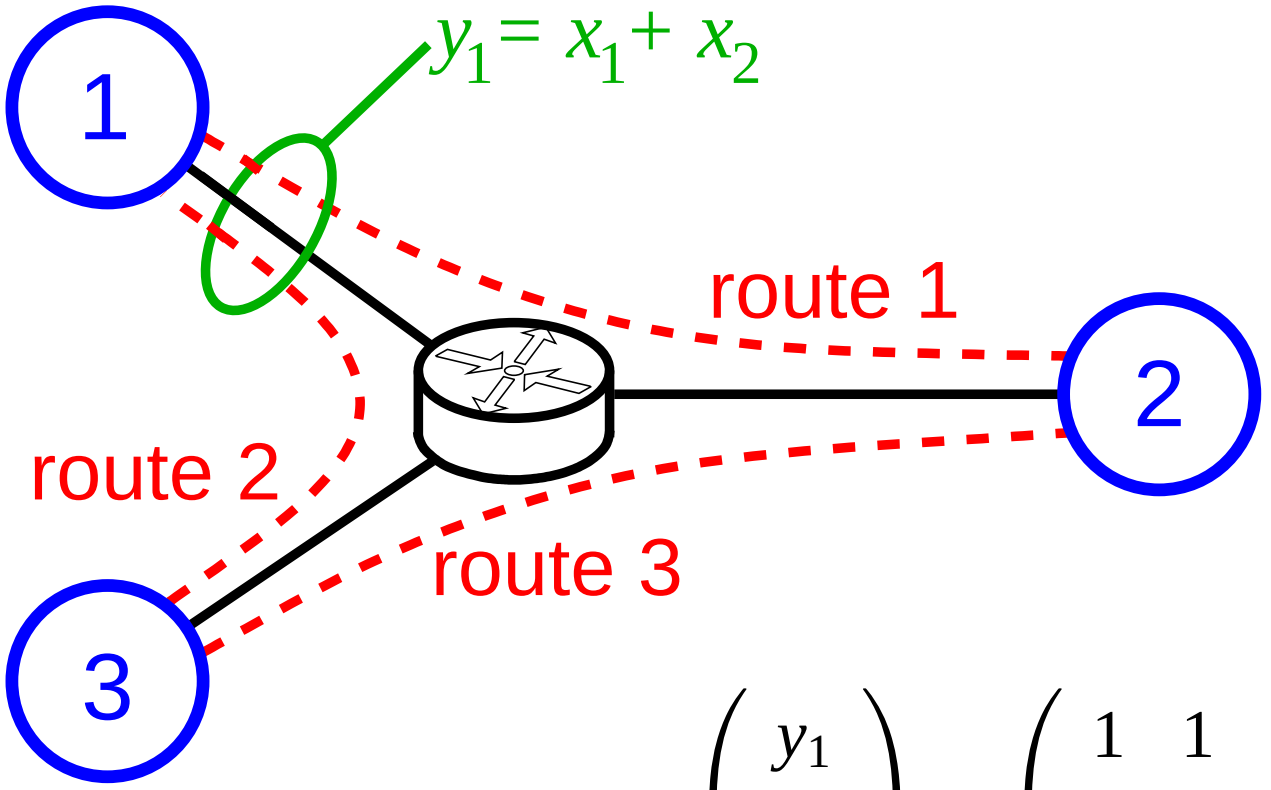
# Problem formulation

- there is a simple linear relationship



# Problem formulation

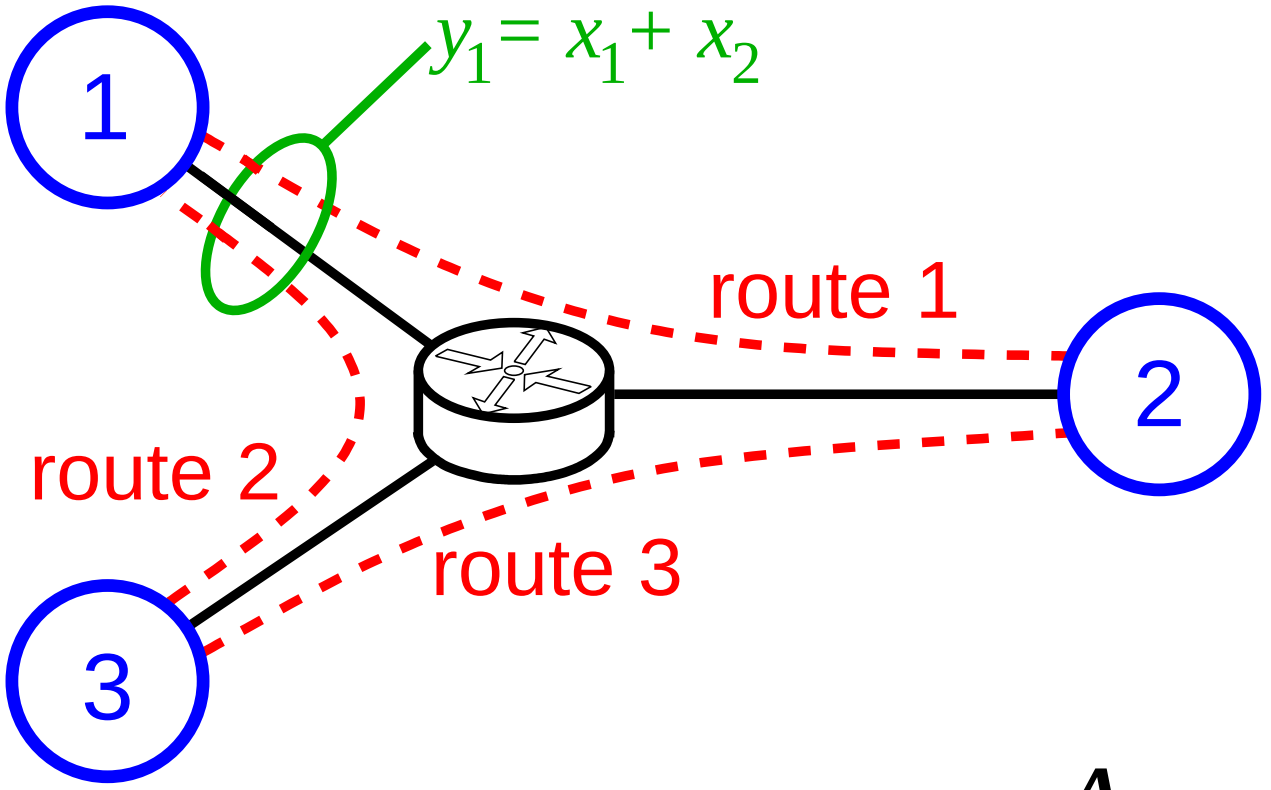
- so we can build a set of equations




$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

# Problem formulation

- so we can build a set of equations




$$\mathbf{y} = A\mathbf{x}$$

# Solution

We know $\mathbf{y}$ and want to get $\mathbf{x}$, so solve

$$\mathbf{y} = A\mathbf{x}$$

Three cases:

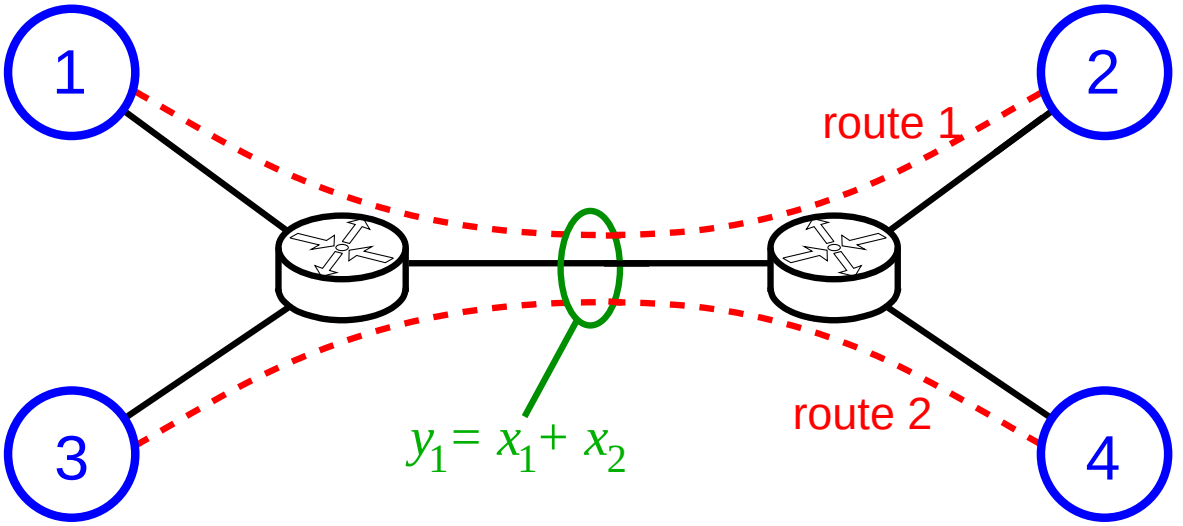
1. $A$ is square and invertible

$$x = A^{-1}y$$

2. $A$ has more rows than columns
   - becomes a least squares problem
   - solve using regression
3. $A$ has more columns than rows
   - this is a tricky case
   - many possible solutions (underdetermined)

# The catch

- in realistic networks
  - often more unknowns than data
  - case (3) — underdetermined!




$$y_1 = (1\ 1)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

- problems like this occur often
  - called underconstrained linear inverse problems

# Inverse problems

Tomographic techniques are used in many areas:

- Ocean Acoustic Tomography
  http://www.oal.whoi.edu/tomo2.html
- Archaeology
  http://archaeology.huji.ac.il/ct/
- Medical Imaging
  http://www.triumf.ca/welcome/petscan.html
- Manufacturing
  http://www.tomography.umist.ac.uk/intro.shtml
- Seismology
  http://www.itso.ru/GEOTOMO/paper_moscow2003/index.html

There are many solution techniques.

# Tomogravity [?, ?]

- solution of underconstrained problems requires side information
- side-information for traffic matrices is to use a gravity model as a prior distribution for the matrix
- solve using regularization
  - frame as an optimisation

$$\begin{aligned} &\text{minimise} && d(\mathbf{g},\mathbf{x}) \\ &\text{subject to} && \\ & && \mathbf{y} = A\mathbf{x} \\ & && \mathbf{x} \geq 0 \end{aligned}$$

where $d(\mathbf{g},\mathbf{x})$ is the distance of the solution $\mathbf{x}$ from the simple gravity model $\mathbf{g}$.

# Discussion

- Notice that the solution is found by optimization
  - optimization is good for lots of things
- Solutions can't be guaranteed
  - they contain errors
  - our results: errors around 12%
- should we worry about errors?
  - have to do prediction anyway
    - go to a course on time-series to find out how
  - should ensure that our optimization is robust
    - sensitivity analysis against possible errors
    - robust optimization
      - see next lecture

# References