Advanced Topics in Communication Networks

Internet Routing and Forwarding

Laurent Vanbever

nsg.ee.ethz.ch

29 Sep 2020

Lecture starts at **14:15**

Materials inspired and/or coming from Olivier Bonaventure, Mike Freedman, Nick Feamster, Alex Snoeren, Jennifer Rexford, and p4.org

Last week on

Advanced Topics in Communication Networks

We *started* to dive in the P4 ecosystem and
started to look at Multiprotocol Label Switching

P4 environment	P4 language	label switching
What is needed to program in P4?	Deeper-dive into the language constructs	the basics

P4 environment

P4 language

label switching

What is needed to program in P4?

P4$_{16}$ introduces the concept of an architecture

P4 Target

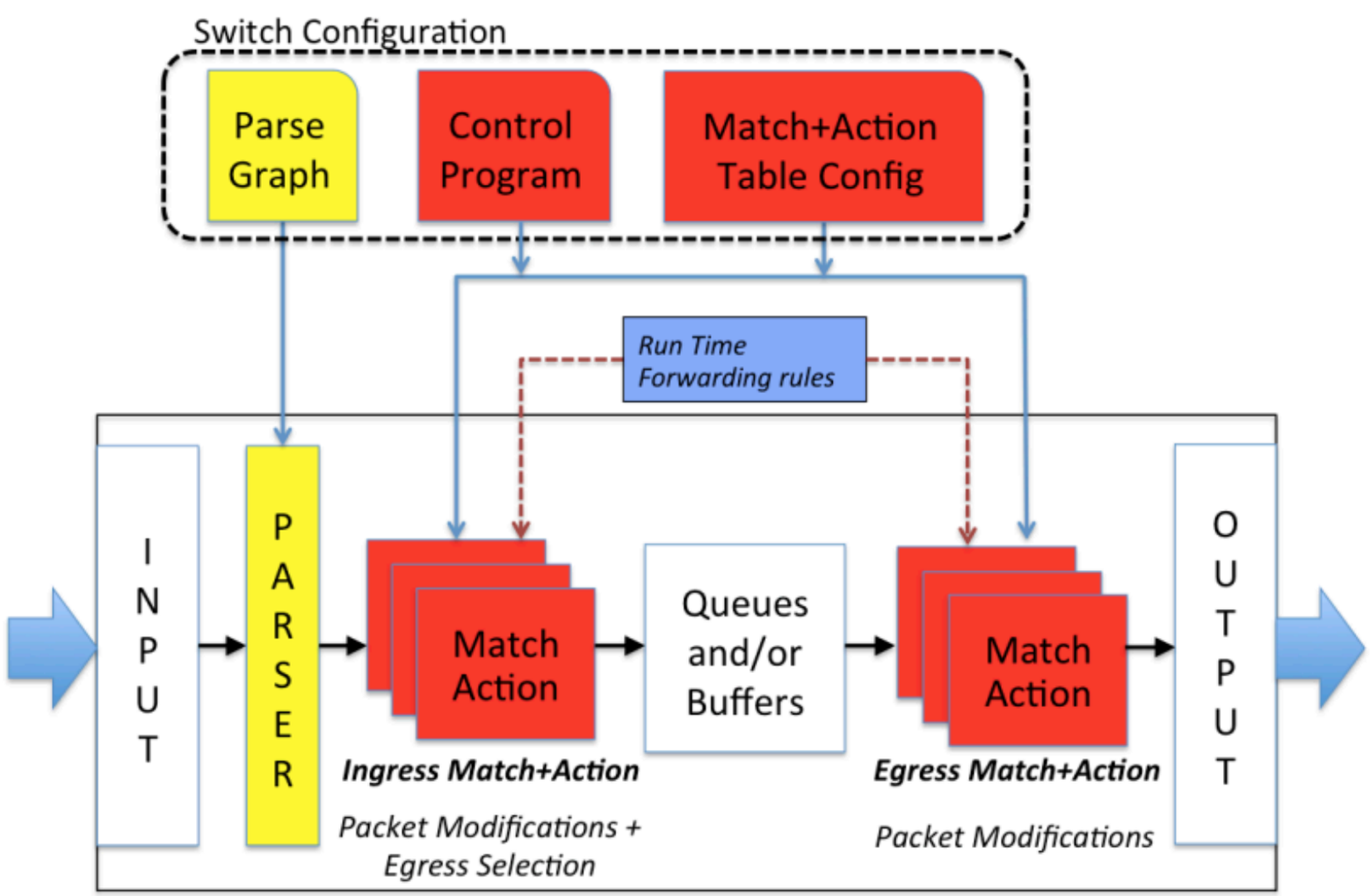
a model of a specific hardware implementation

P4 Architecture

an API to program a target

We'll rely on a simple P4$_{16}$ switch architecture (v1model) which is roughly equivalent to "PISA"

v1model/
simple switch



source https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

P4 environment

P4 language

label switching

Deeper-dive into
the language constructs

Each architecture defines the metadata it supports, including both standard and intrinsic ones

v1model

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    error parser_error;
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
    bit<32> recirculate_flag;
}
```

more info https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4

Each architecture also defines a list of "externs",
i.e. blackbox functions whose interface is known

Most targets contain specialized components

which cannot be expressed in P4 (e.g. complex computations)

At the same time, $P4_{16}$ should be target-independent

In $P4_{14}$ almost 1/3 of the constructs were target-dependent

Think of externs as Java interfaces

only the signature is known, not the implementation

P4$_{16}$ is a statically-typed language with

base types and operators to derive composed ones

`bool`	Boolean value
`bit<W>`	Bit-string of width W
`int<W>`	Signed integer of width W
`varbit<W>`	Bit-string of dynamic length ≤W
`match_kind`	describes ways to match table keys
`error`	used to signal errors
`void`	no values, used in few restricted circumstances
`float`	not supported
`string`	not supported

P4$_{16}$ is a statically-typed language with base types and operators to derive composed ones

Header

```
header Ethernet_h {
 bit<48> dstAddr;
 bit<48> srcAddr;
 bit<16> etherType;
}
```

Header stack

```
header Mpls_h {
 bit<20> label;
 bit<3>  tc;
 bit     bos;
 bit<8>  ttl;
}

Mpls_h[10] mpls;
```

Header union

```
header_union IP_h {
 IPv4_h v4;
 IPv6_h v6;
}
```

P4 operations are similar to C operations and vary depending on the types (unsigned/signed ints, …)

- arithmetic operations `+, -, *`
- logical operations `~, &, |, ^, >>, <<`
- non-standard operations `[m:l]` Bit-slicing

  `++` Bit concatenation

✘ *no* division and modulo (can be approximated)

more info https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

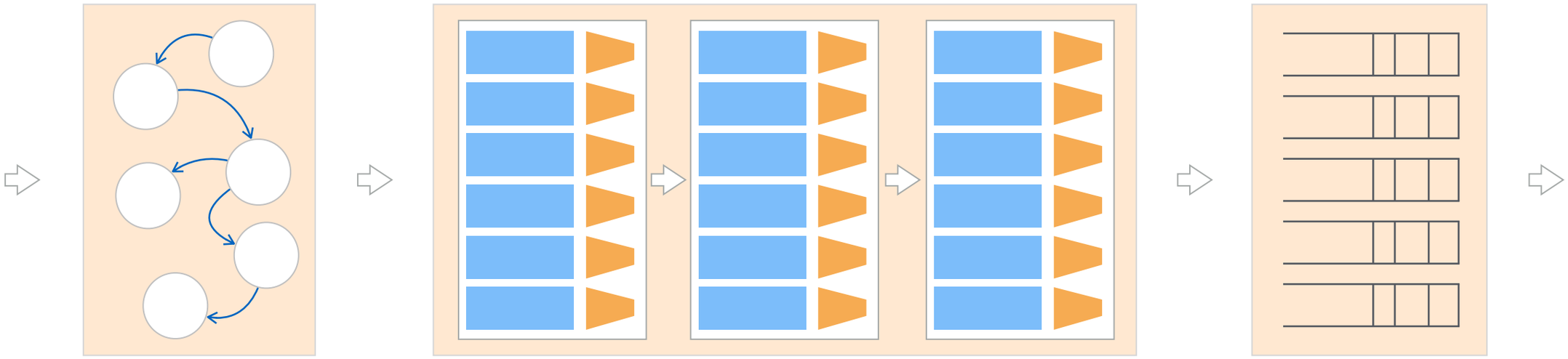
We then continued our journey through the pipeline

(we'll finish the rest today)

Parser

Match-Action Pipeline

Deparser



P4 environment

P4 language

label switching

the basics

Multiprotocol Label Switching

"IP meets virtual circuits"

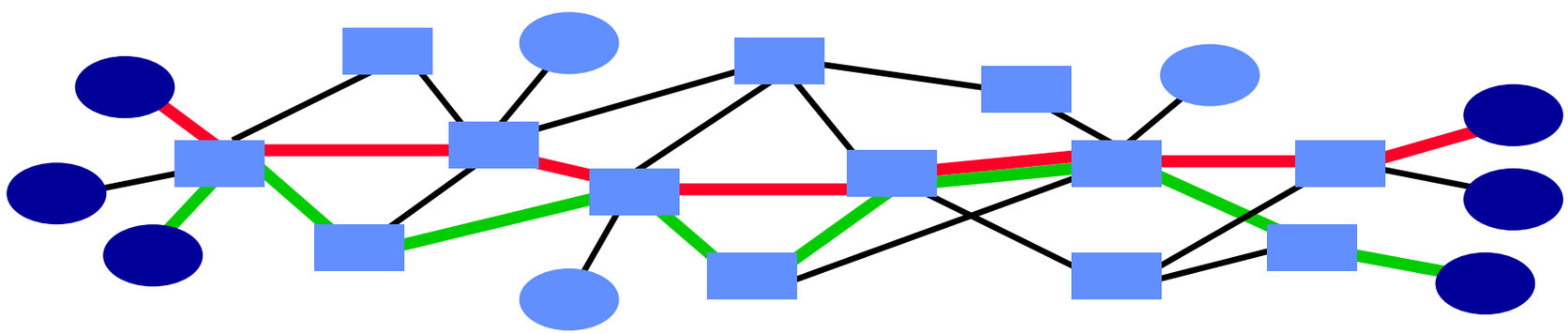
core idea — store the label in-between layer 2 and layer 3

MPLS is often referred to as a layer 2.5 protocol

usages — Traffic engineering, QoS, Fast reroute, VPNs, …

Virtual Circuits

- **Each wire carries many "virtual" circuits.**
  - » **Forwarding based on virtual circuit (VC) identifier**
    - – **IP header: src, dst, etc.**
    - – **Virtual circuit header: just "VC"**
  - » **A path through the network is determined for each VC when the VC is established**
  - » **Use statistical multiplexing for efficiency**
- **Can support wide range of quality of service.**
  - » **No guarantees: best effort service**
  - » **Weak guarantees: delay < 300 msec, …**
  - » **Strong guarantees: e.g. equivalent of physical circuit**



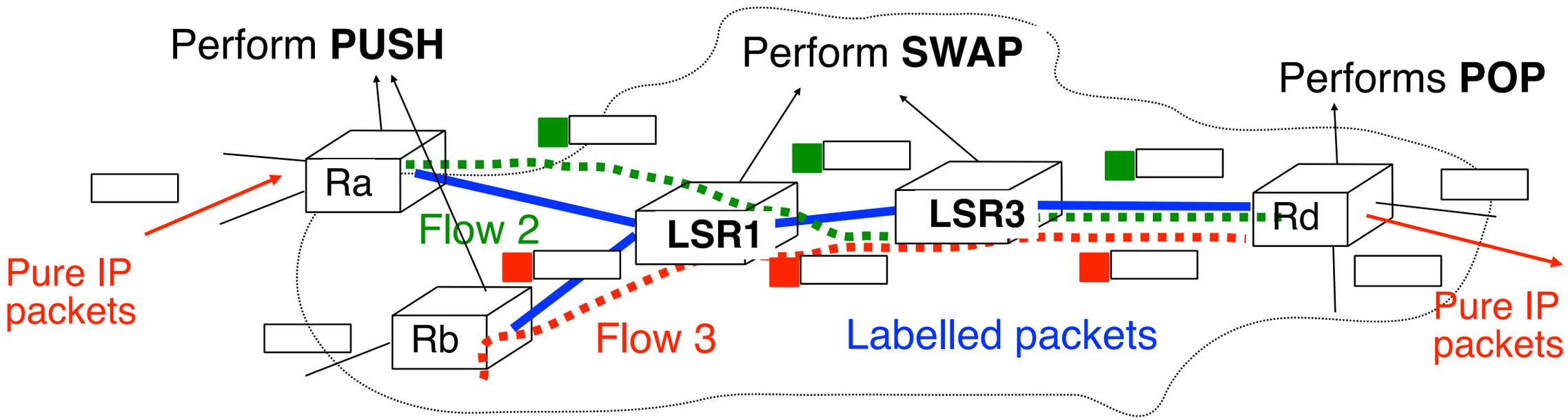
source: David Andersen, CMU

Integrating label swapping and IP (2)

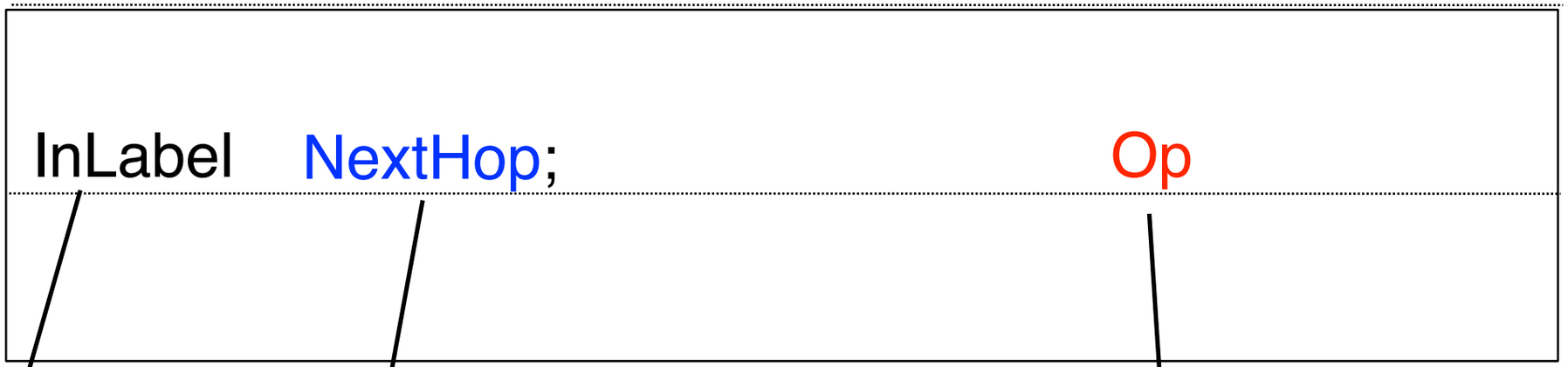
- We need to solve three problems
  1. What do we use as packet label?
  2. What is the behaviour of a core LSR?
  3. What is the behaviour of an edge LSR?

MPLS data plane allows for three operations on a labelled packet

1. PUSH
   - insert a label in front of a received packet
2. SWAP
   - change the value of the label of a received labelled packet
3. POP
   - remove the label in front of a received labelled packet



Content of the Label forwarding table



- Label of the incoming packet
- Next-hop for the packet
  - outgoing interface
    - packet sent to another LSR
  - LSR itself
    - packet destination is LSR
    - packet needs to be routed as a normal IP packet after pop operation
- Operation to be performed on the label
  - swap label
  - push new label (on top)
  - pop label

This week on

Advanced Topics in Communication Networks

We will dive into the P4 ecosystem and look at Multiprotocol Label Switching

P4 language	label switching	traffic engineering
constructs (the end) + stateful applications	the basics (the end)	IP-based / MPLS-based

P4 language

label switching

traffic engineering

constructs (the end) +

stateful applications

Stateful objects in P4

- Table — managed by the control plane
- Register — store arbitrary data
- Counter — count events
- Meter — rate-limiting
- … — …

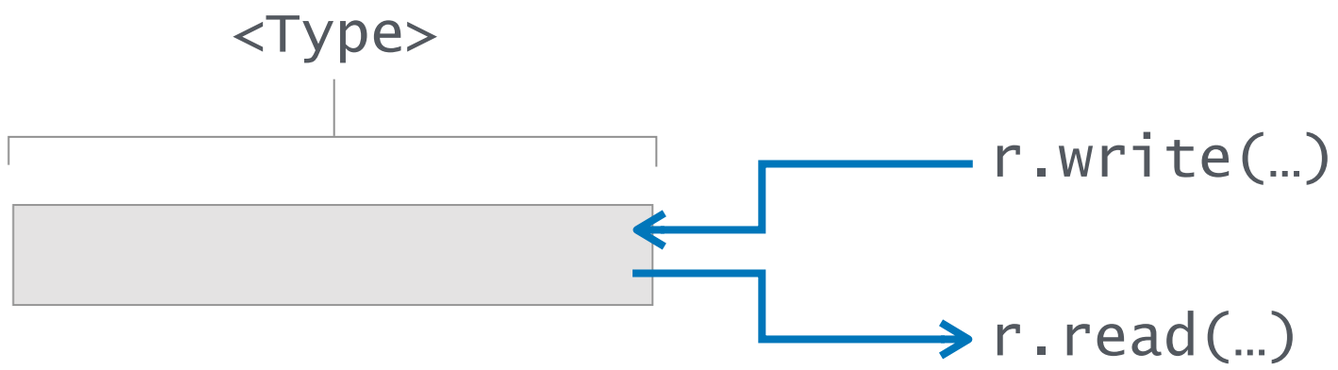
externs in v1model

Stateful objects in P4

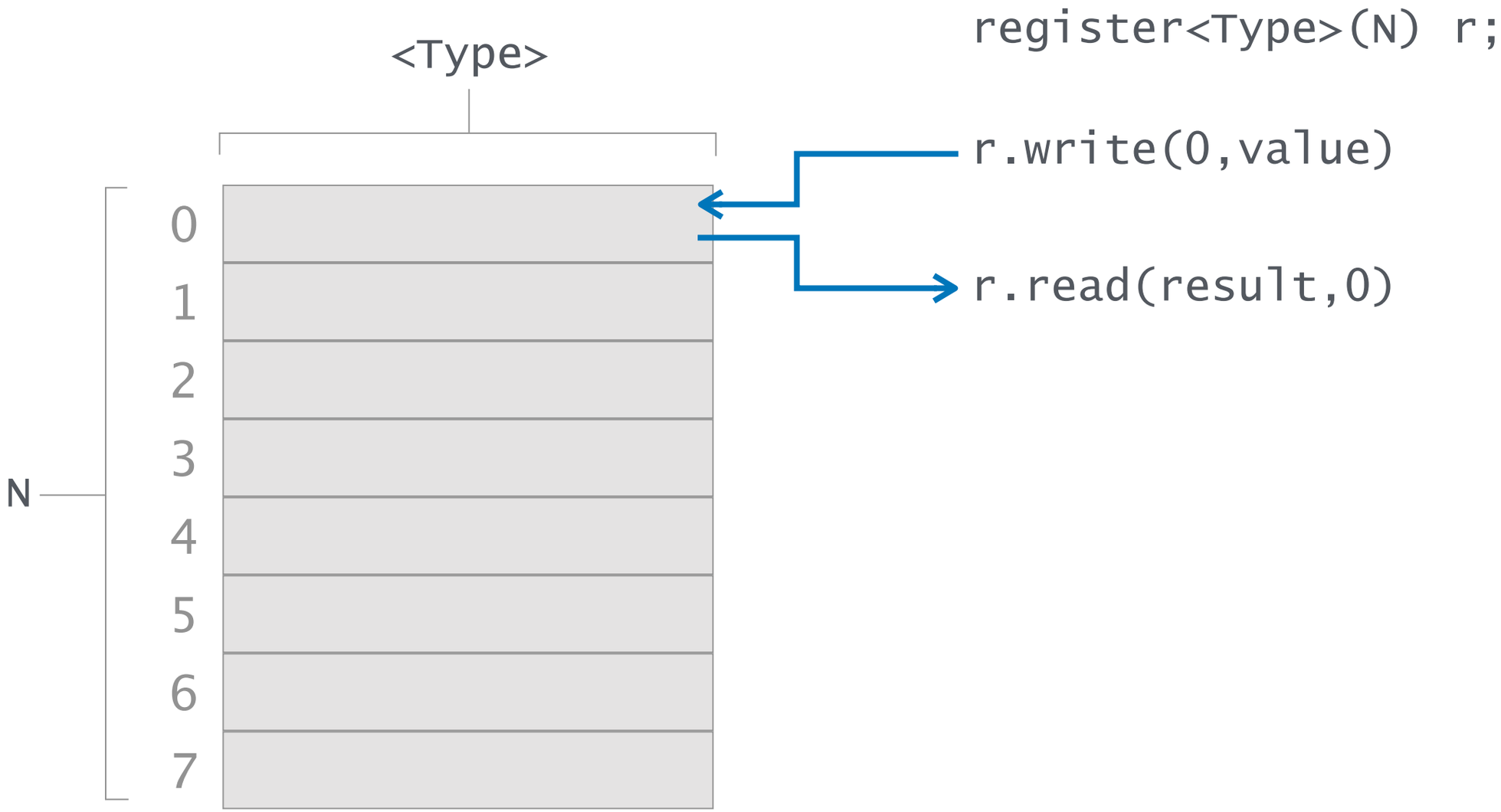
- Table — managed by the control plane
- Register — store arbitrary data
- Counter — count events
- Meter — rate-limiting
- … — …

externs in v1model

Registers are useful for storing
(small amounts of) arbitrary data



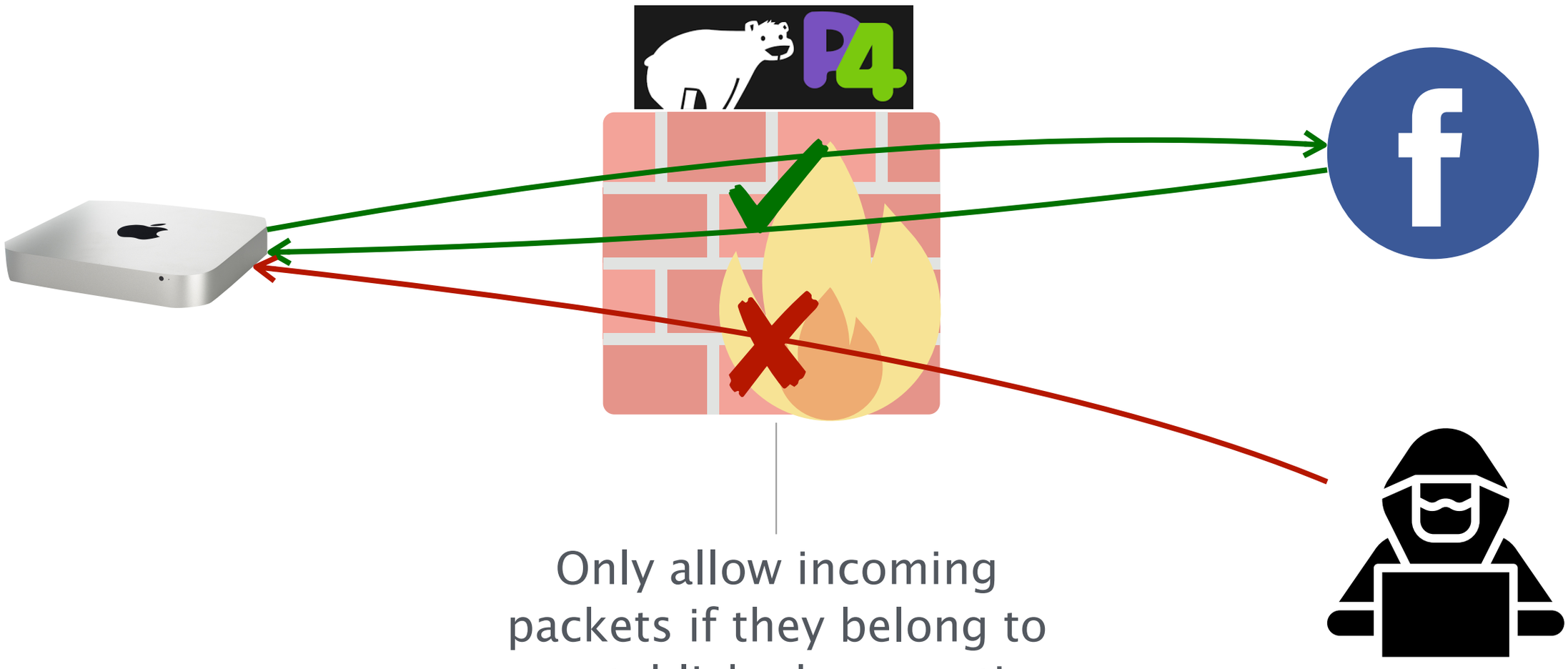
Registers are assigned in arrays



Example: Calculating inter packet gap

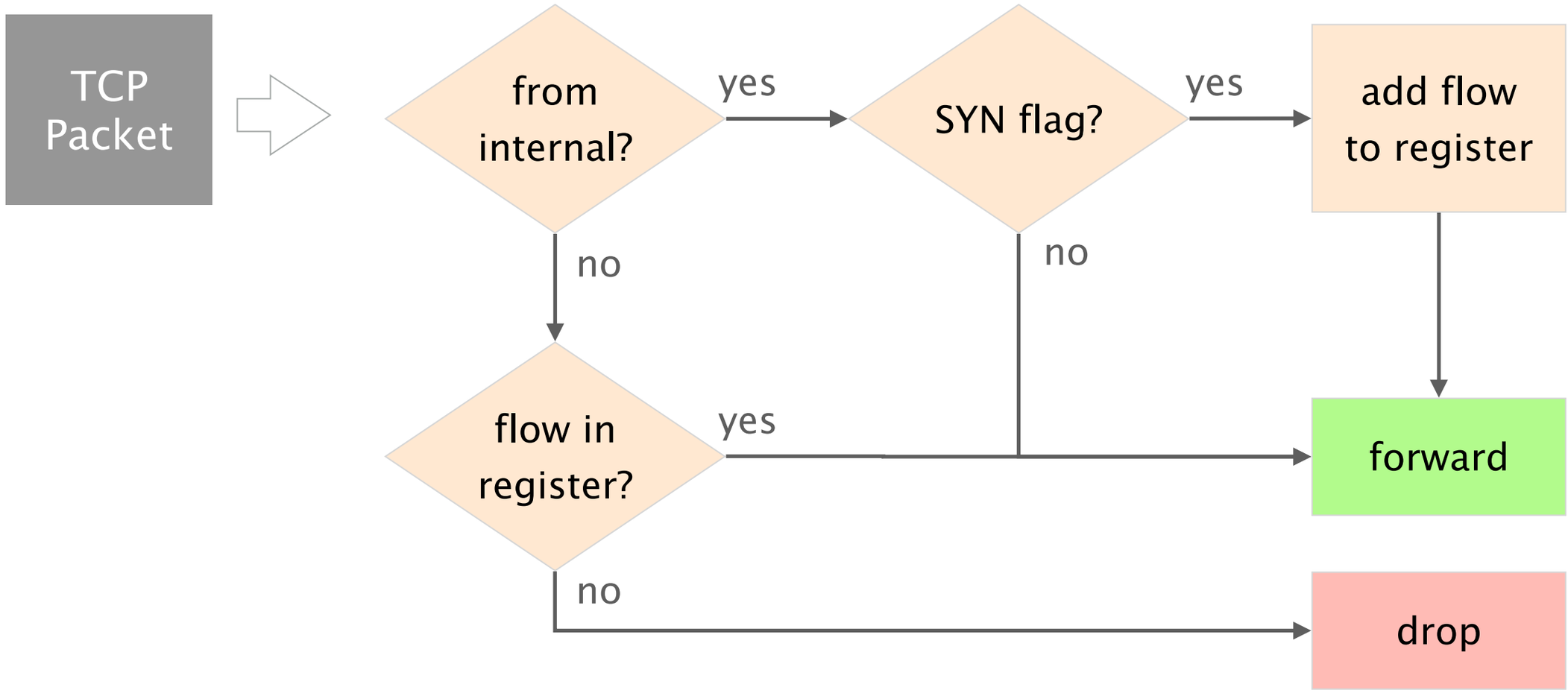
```
register<bit<48>>(16384) last_seen;

action get_inter_packet_gap(out bit<48> interval, bit<32> flow_id)
{
 bit<48> last_pkt_ts;

 /* Get the time the previous packet was seen */
 last_seen.read(last_pkt_ts, flow_id);

 /* Calculate the time interval */
 interval = standard_metadata.ingress_global_timestamp - last_pkt_ts;

 /* Update the register with the new timestamp */
 last_seen.write(flow_id, standard_metadata.ingress_global_timestamp);

 ...
}
```

Example: Stateful firewall



Only allow incoming packets if they belong to an established connection

Example: Stateful firewall



Example: Stateful firewall

```
control MyIngress(...) {
 register<bit<1>>(4096) known_flows;
 ...
 apply {
  meta.flow_id = ... // hash(5-tuple)
  if (hdr.ipv4.isValid()){
   if (hdr.tcp.isValid()){
    if (standard_metadata.ingress_port == 1){
     if (hdr.tcp.syn == 1){
      known_flows.write(meta.flow_id, 1);
     }
    }

    if (standard_metadata.ingress_port == 2){
     known_flows.read(meta.flow_is_known, meta.flow_id);
     if (meta.flow_is_known != 1){
      drop(); return;
     }
    }
   }
   ipv4_lpm.apply();
  }
 }
}
```

register to memorize established connections

add to register if it it is a SYN packet from internal

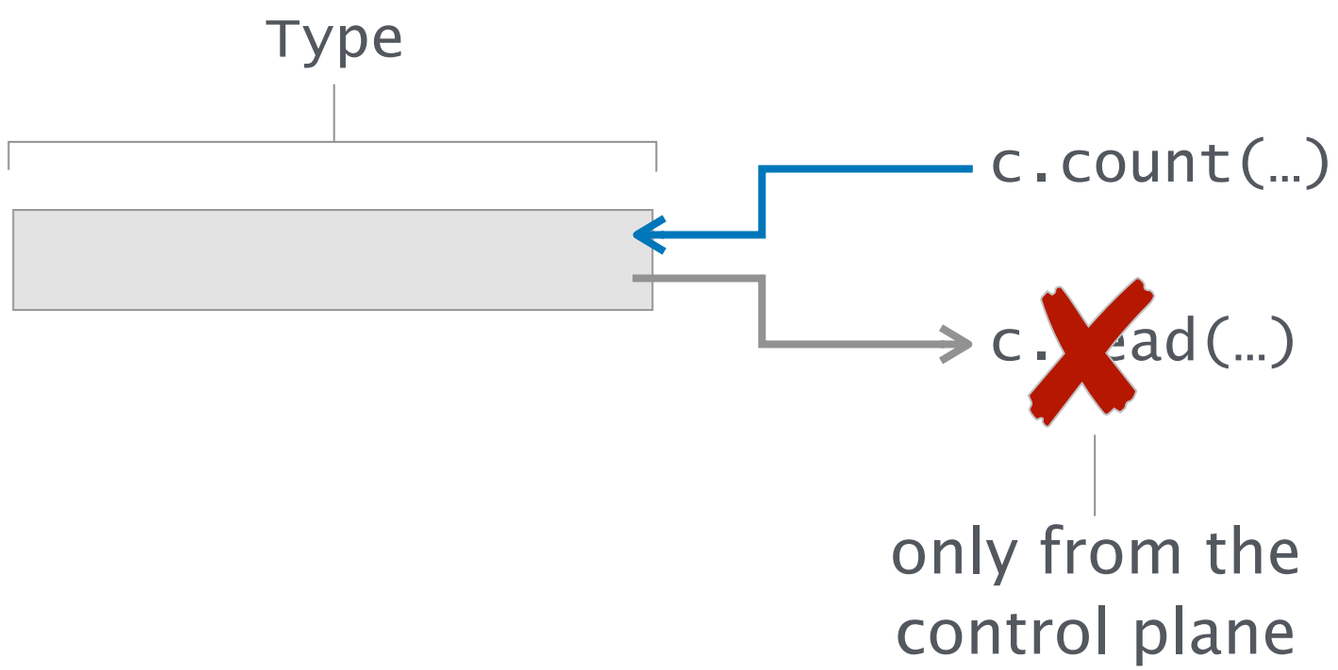
drop if the packet does not belong to a known flow and comes from outside

Stateful objects in P4

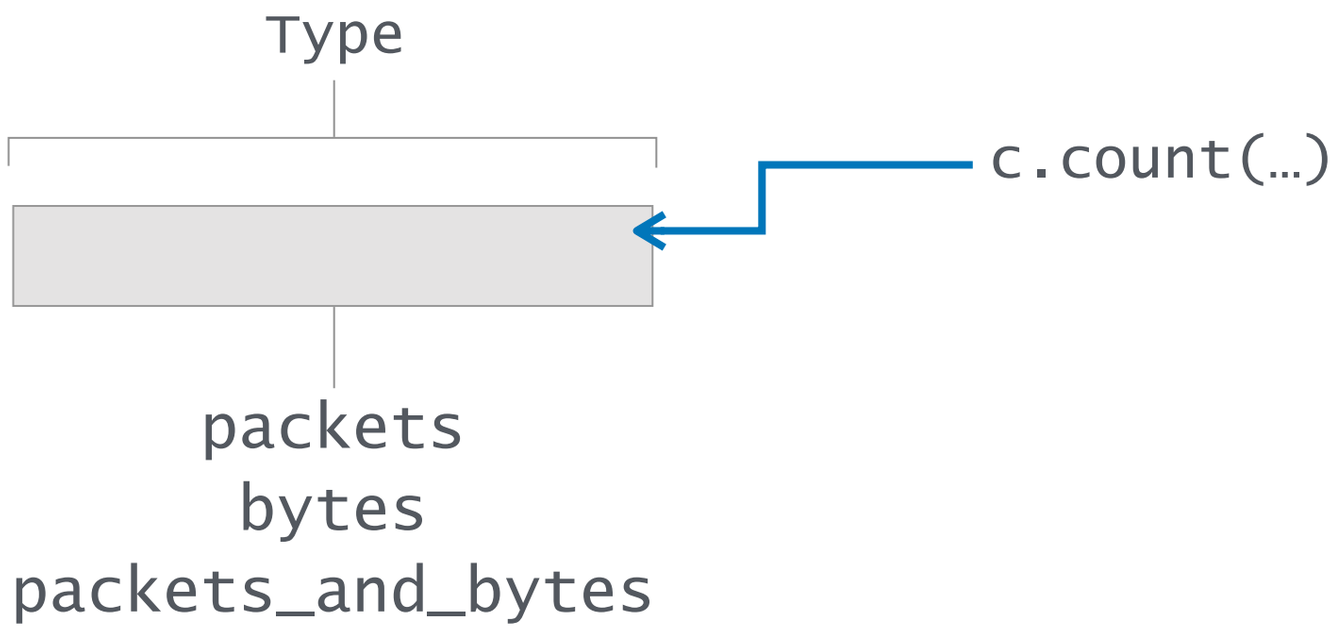
- Table — managed by the control plane
- Register — store arbitrary data
- Counter — count events
- Meter — rate-limiting
- … — …

externs in v1model

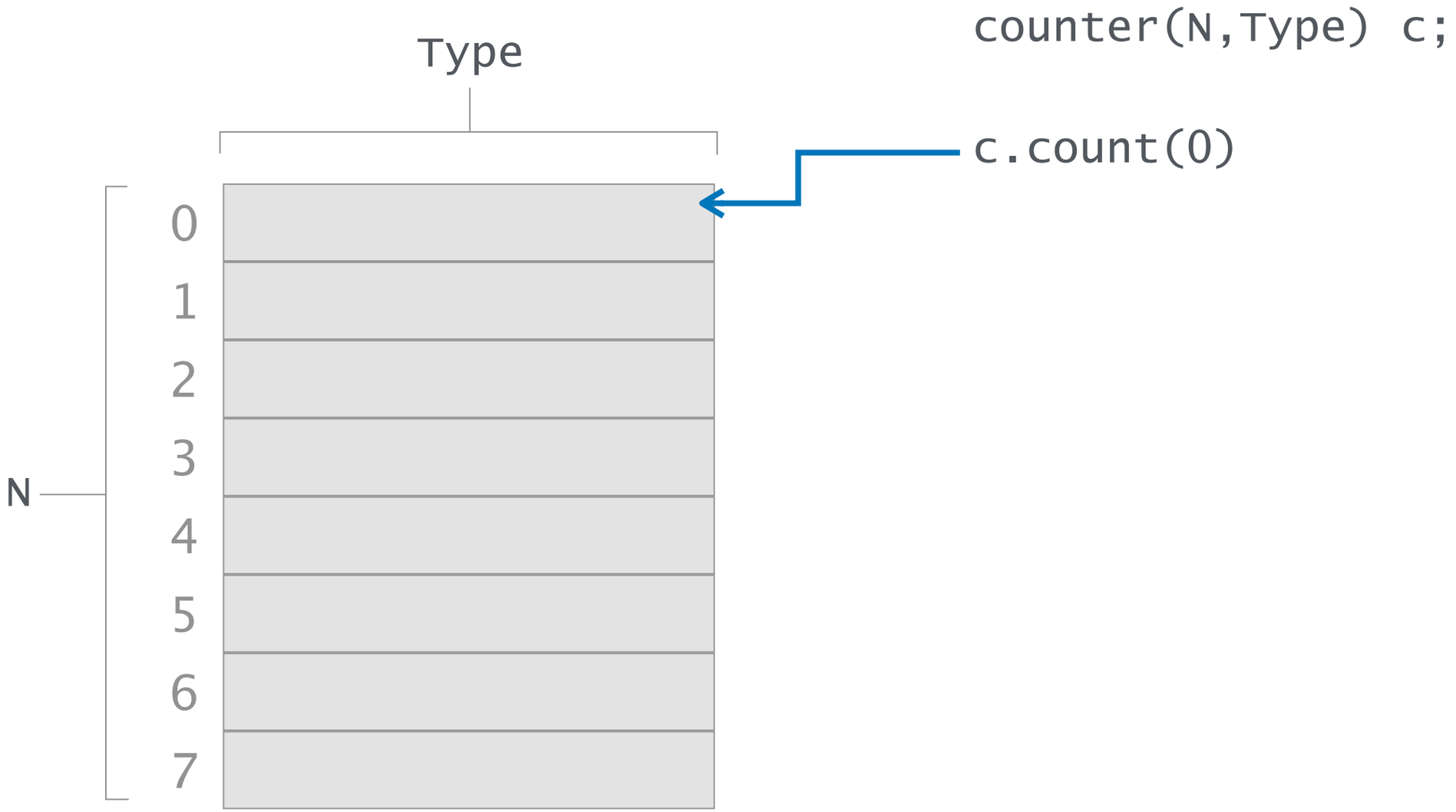
Counters are useful for… counting



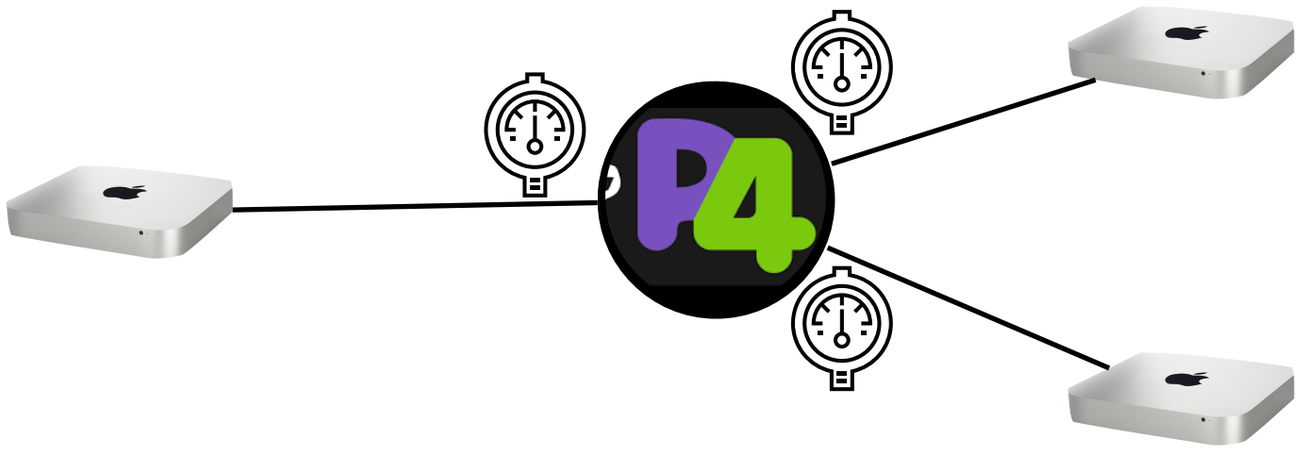
Counters can be of three different types



Like registers, counters are assigned in arrays



Example: Counting packets and bytes arriving at each port




```
control MyIngress(...) {

    counter(512, CounterType.packets_and_bytes) port_counter;

    apply {
        port_counter.count((bit<32>)standard_metadata.ingress_port);
    }
}
```

use the ingress port as counter index

Example: Reading the counter values from the control plane

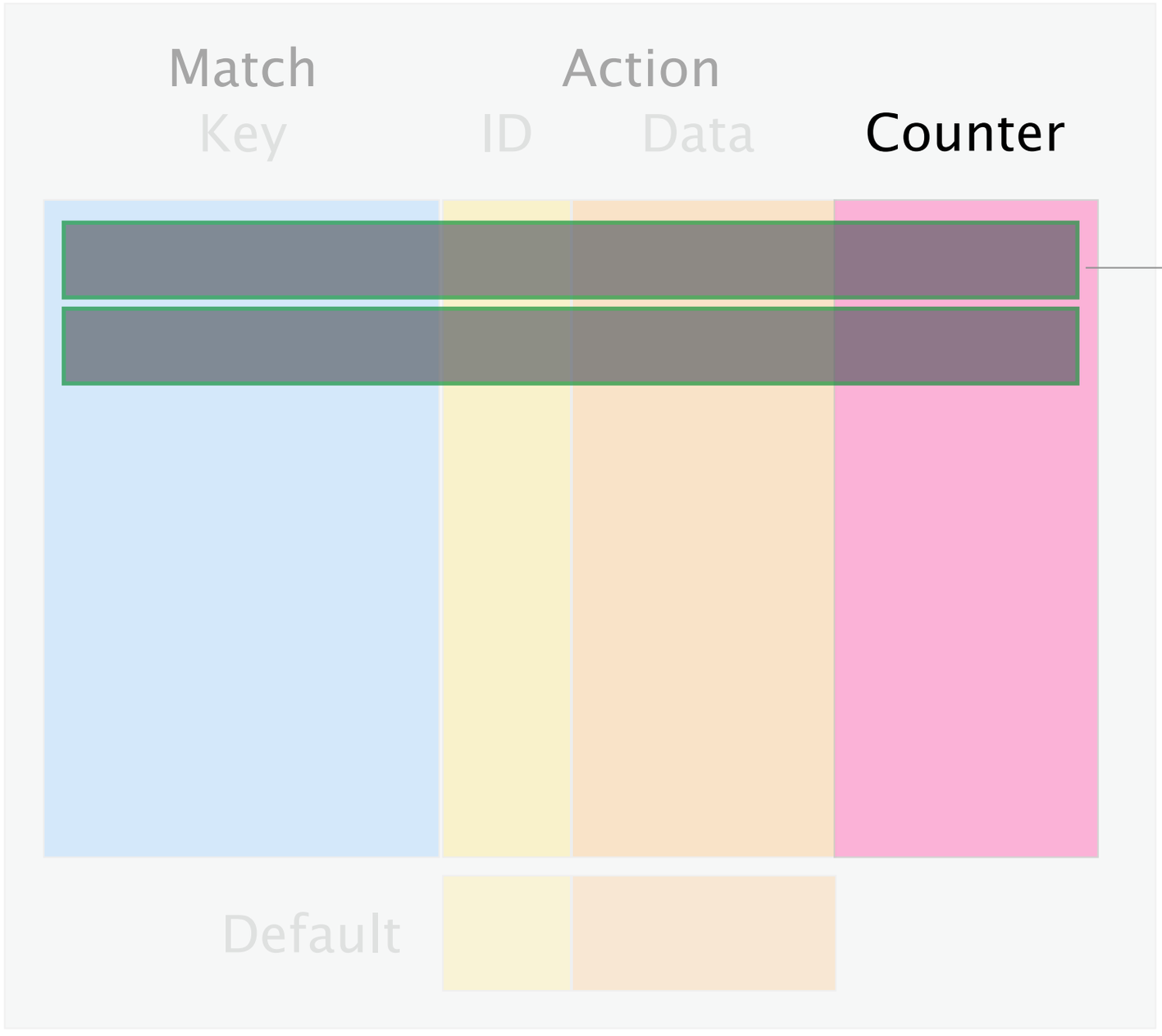
Control Plane

```
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]=  BmCounterValue(packets=13, bytes=1150)
```

```
control MyIngress(...) {

    counter(512, CounterType.packets_and_bytes) port_counter;

    apply {
        port_counter.count((bit<32>)standard_metadata.ingress_port);
    }
}
```

use the ingress port as counter index

Direct counters are a special kind of counters that are attached to tables



Each entry has a counter cell that counts when the entry matches

Example: Counting packets and bytes arriving at each port *using a direct counter*

```
control MyIngress(...) {

 direct_counter(CounterType.packets_and_bytes) direct_port_counter;

 table count_table {
  key = {
   standard_metadata.ingress_port: exact;
  }
  actions = {
   NoAction;
  }
  default_action = NoAction;
  counters = direct_port_counter;
  size = 512;
 }

 apply {
  count_table.apply();
 }
}
```

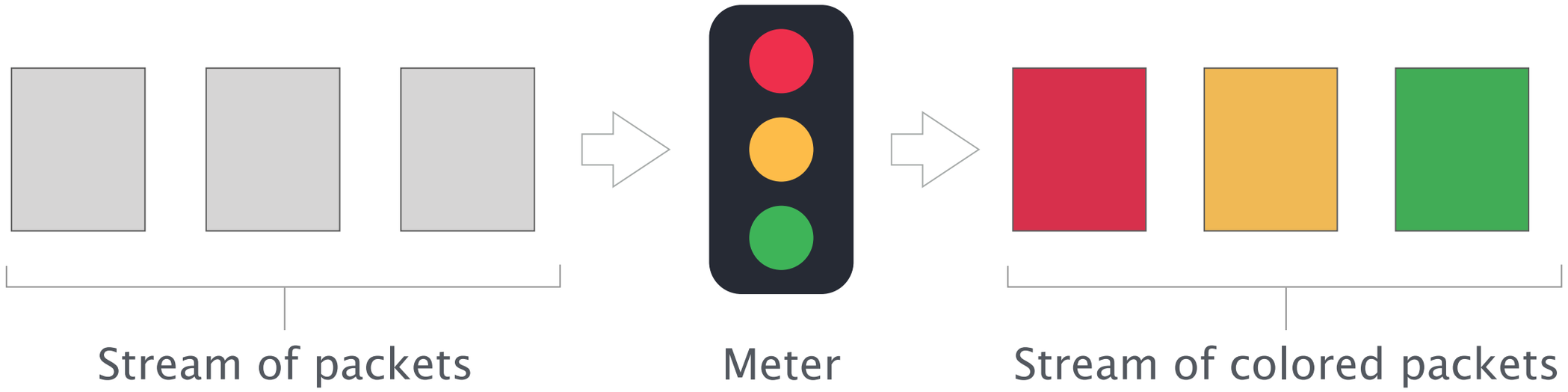
attach counter to table

Stateful objects in P4

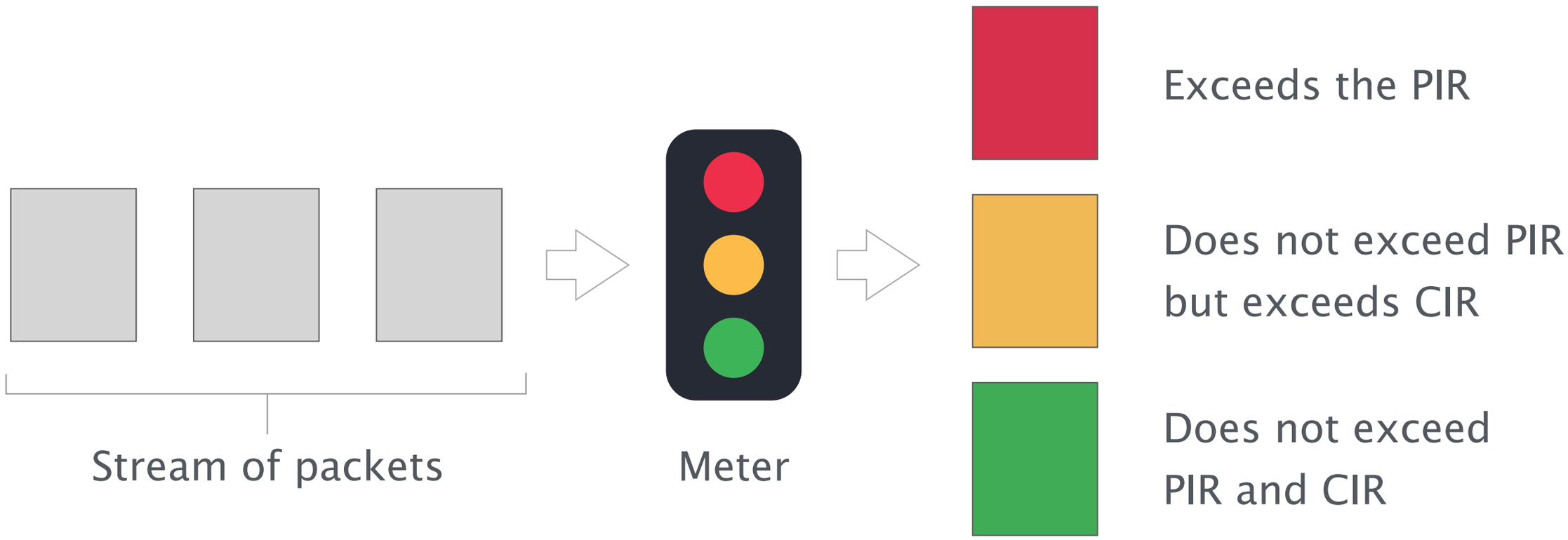
- Table — managed by the control plane
- Register — store arbitrary data
- Counter — count events
- Meter — rate-limiting
- … — …

externs in v1model

Meters



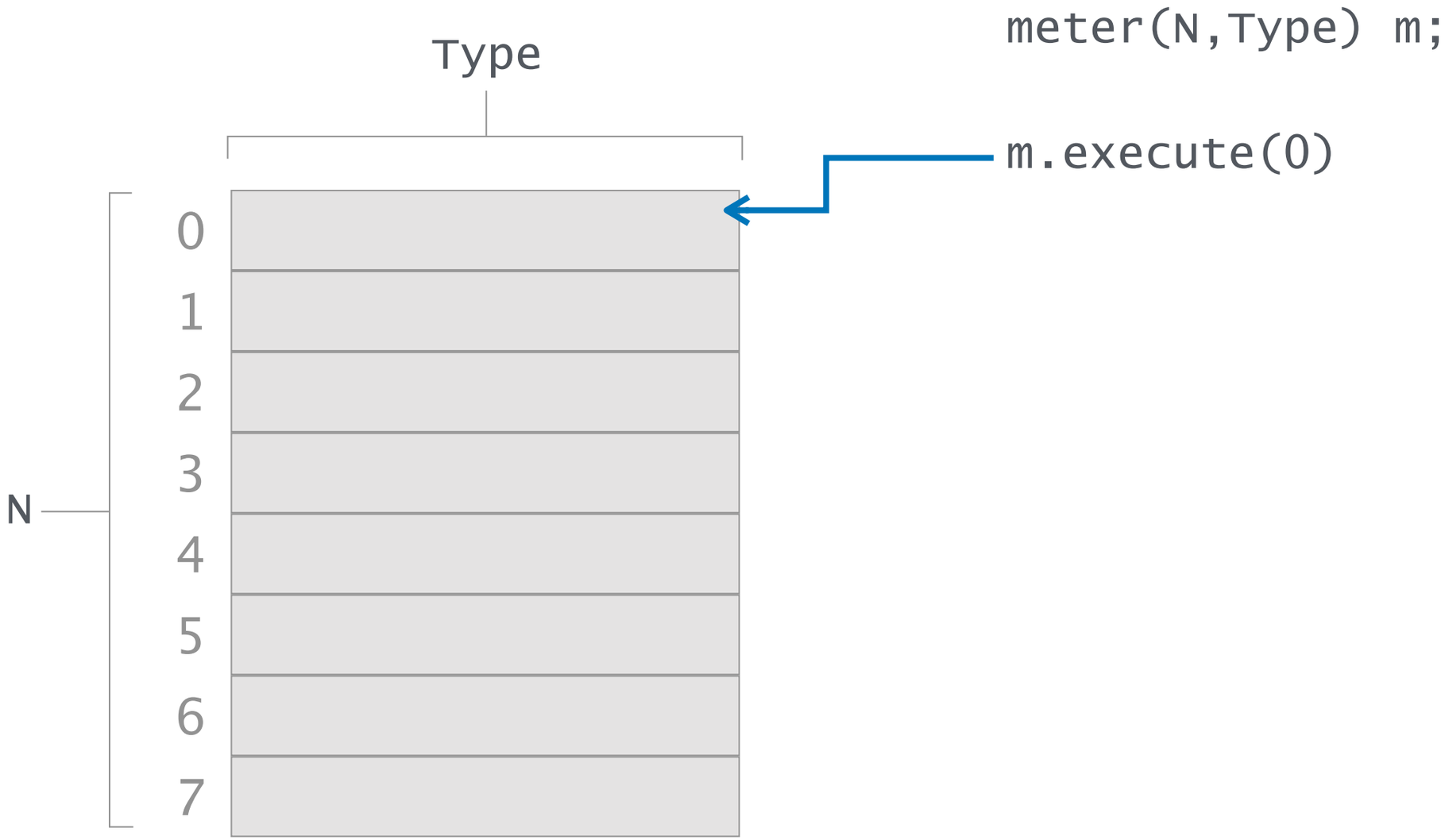
Meters



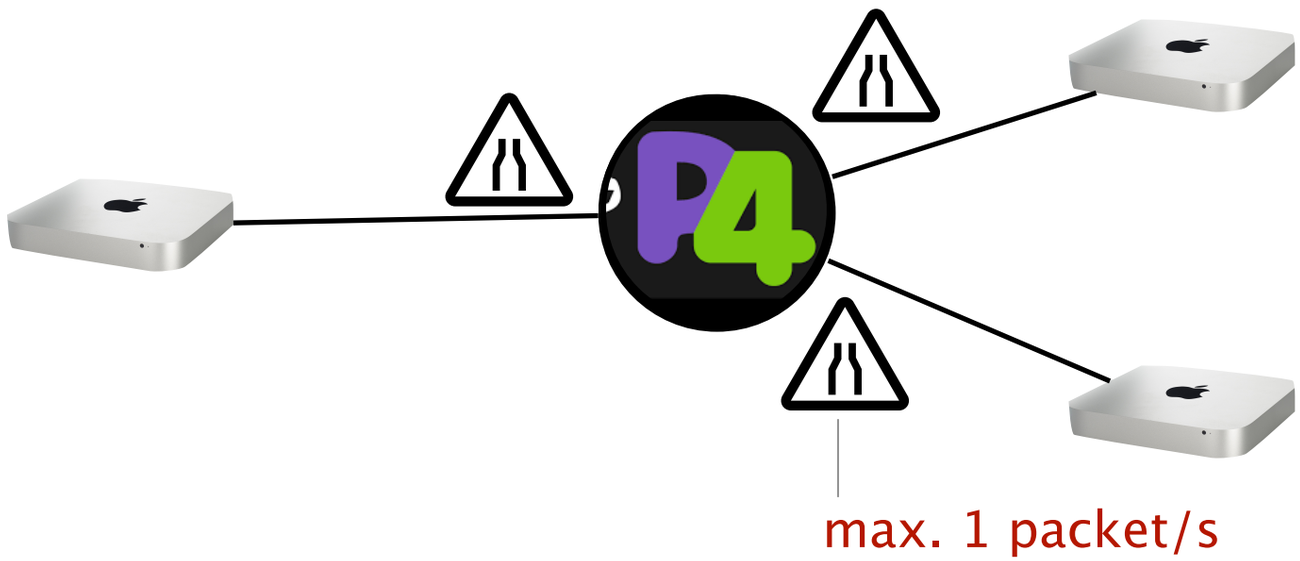
Parameters:			
	PIR	Peak Information Rate	[bytes/s] or [packets/s]
	CIR	Committed Information Rate	[bytes/s] or [packets/s]

more info https://tools.ietf.org/html/rfc2698

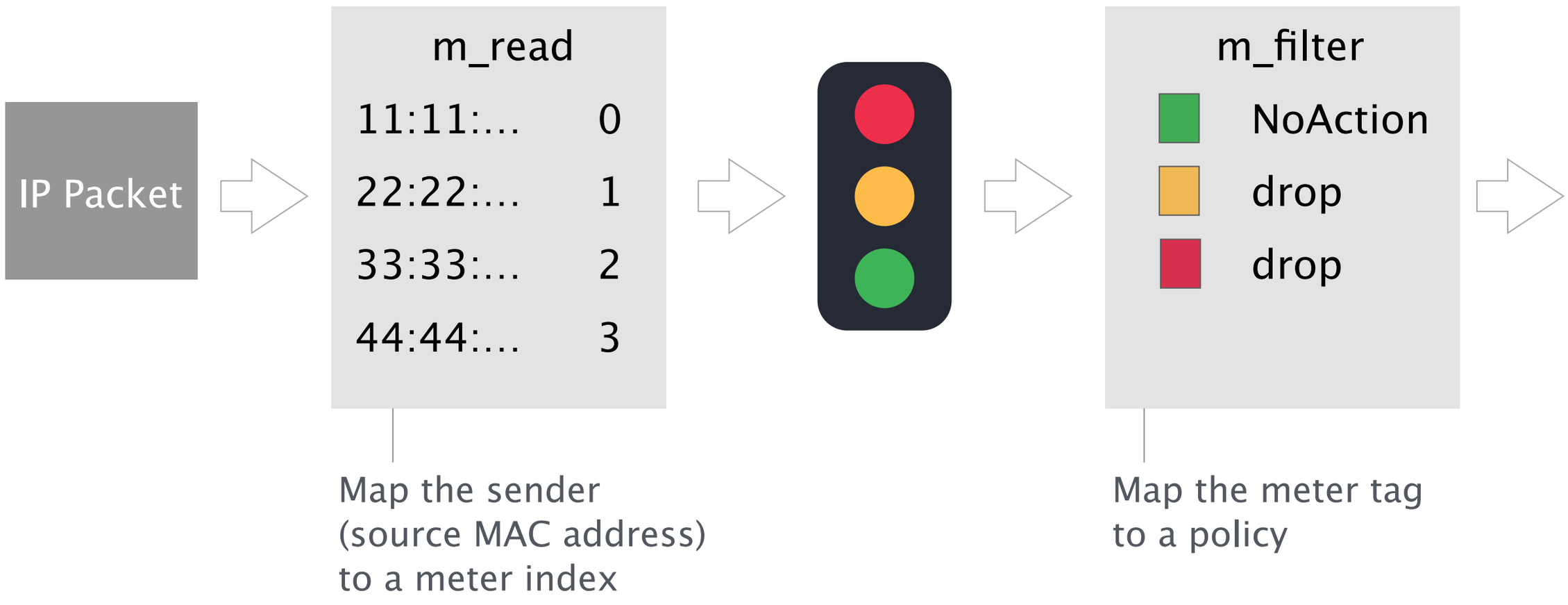
Like registers and counters,
meters are assigned in arrays



Example: Using a meter for rate-limiting



Example: Using a meter for rate-limiting



Example: Using a meter for rate-limiting

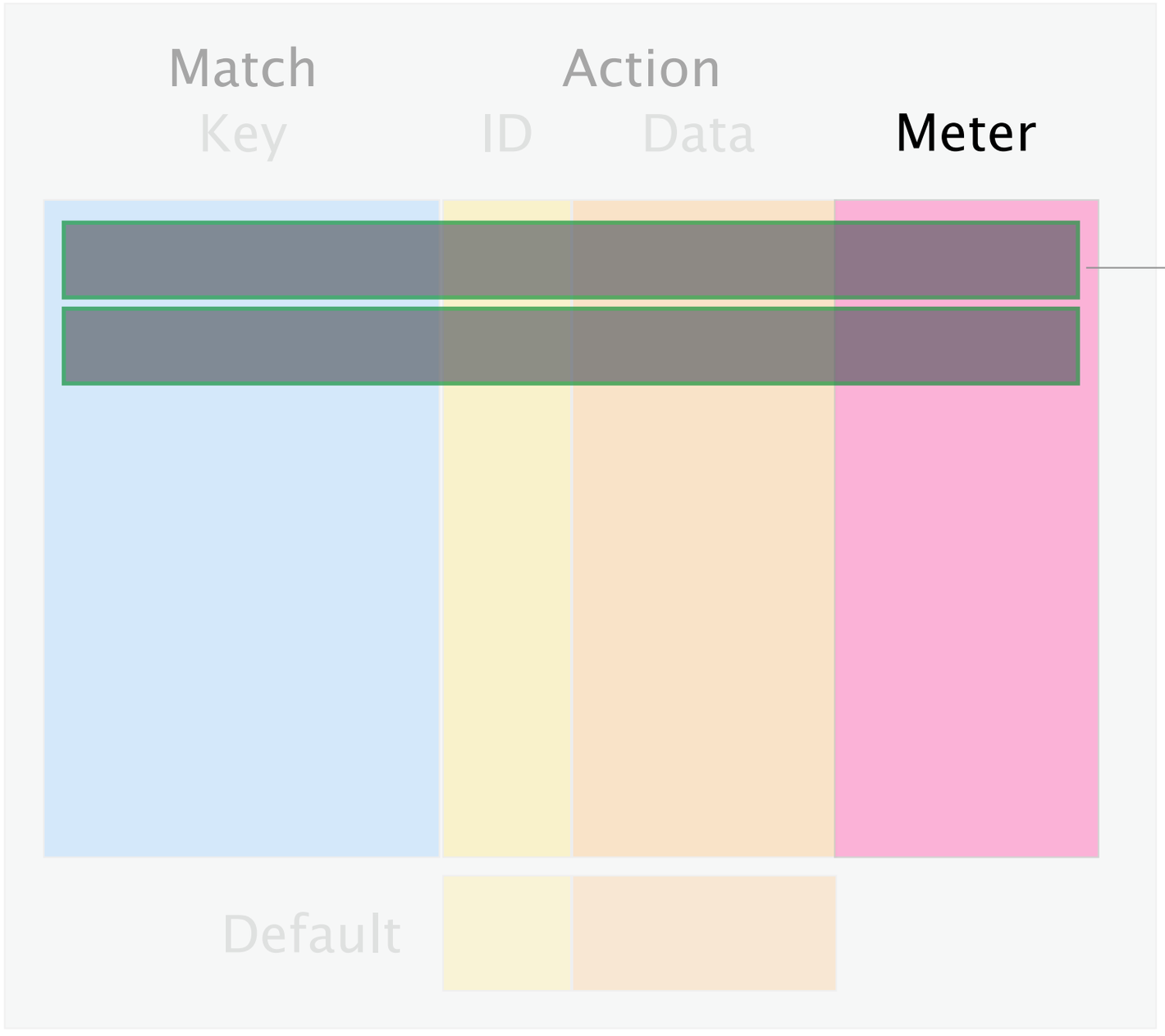
```
control MyIngress(...) {
 meter(32w16384, MeterType.packets) my_meter;

 action m_action(bit<32> meter_index) {
  my_meter.execute_meter<bit<32>>(meter_index, meta.meter_tag);
 }

 table m_read {
  key = { hdr.ethernet.srcAddr: exact; }
  actions = { m_action; NoAction; }
  ...
 }
 table m_filter {
  key = { meta.meter_tag: exact; }
  actions = { drop; NoAction; }
  ...
 }

 apply {
  m_read.apply();
  m_filter.apply();
 }
}
```

packet meter

execute meter

handle packets depending on meter tag

Direct meters are a special kind of meters that are attached to tables



Each entry has a meter cell that is executed when the entry matches

Example: Using a meter for rate-limiting

```
control MyIngress(...) {
 direct_meter<bit<32>>(MeterType.packets) my_meter;

 action m_action(bit<32> meter_index) {
  my_meter.read(meta.meter_tag);
 }

 table m_read {
  key = { hdr.ethernet.srcAddr: exact; }
  actions = { m_action; NoAction; }
  meters = my_meter;
  ...
 }
 table m_filter { ... }

 apply {
  m_read.apply();
  m_filter.apply();
 }
}
```

direct meter

read meter

attach meter to table

Summary

	Data plane interface		Control plane interface	
Object	read	modify/write	read	modify/write
Table	`apply()`	—	yes	yes
Register	`read()`	`write()`	yes	yes
Counter	—	`count()`	yes	reset
Meter	`execute()`		configuration only	

P4 language

label switching

traffic engineering

the basics (the end)

P4 language

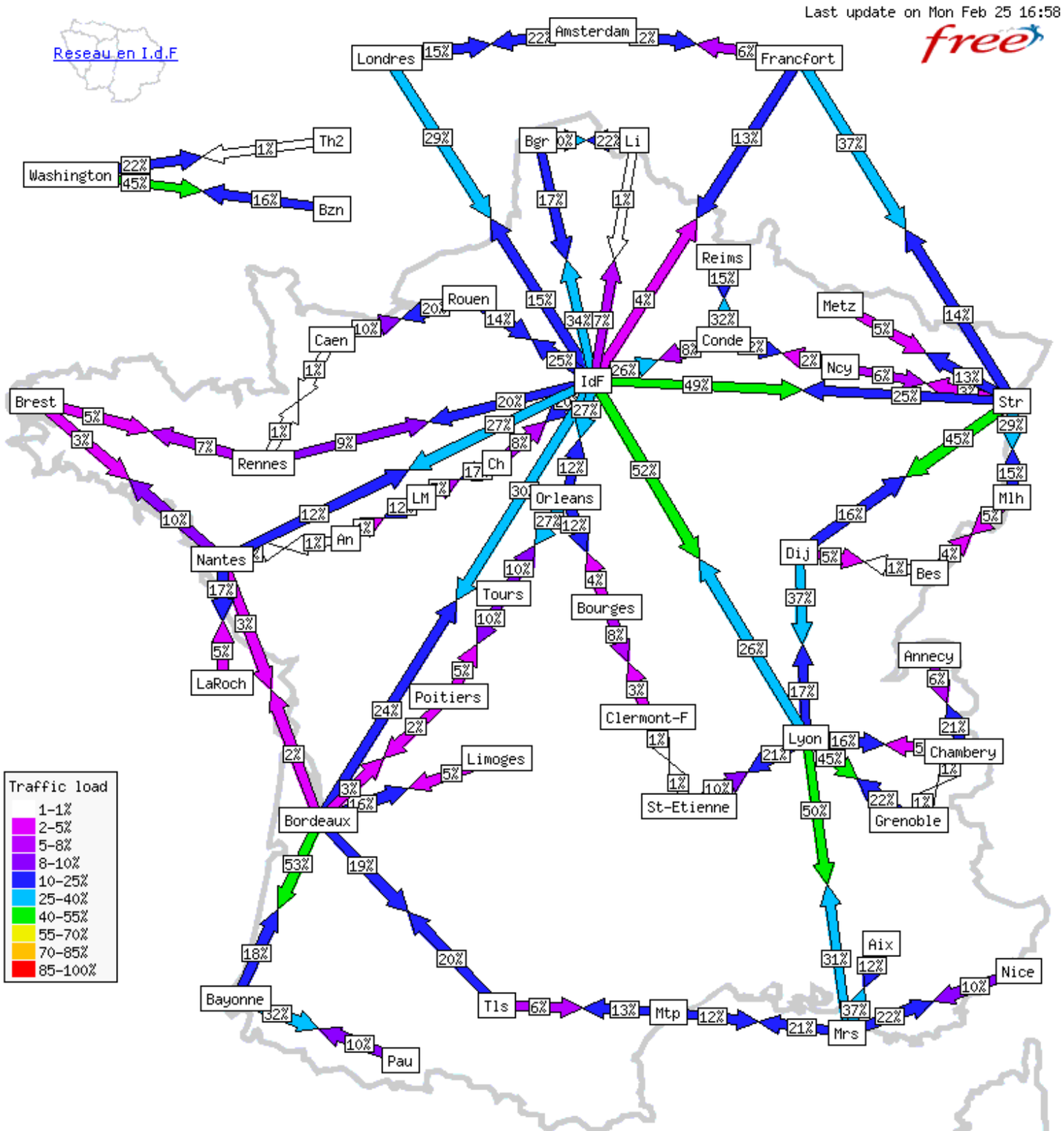
label switching

traffic engineering

IP-based / MPLS-based

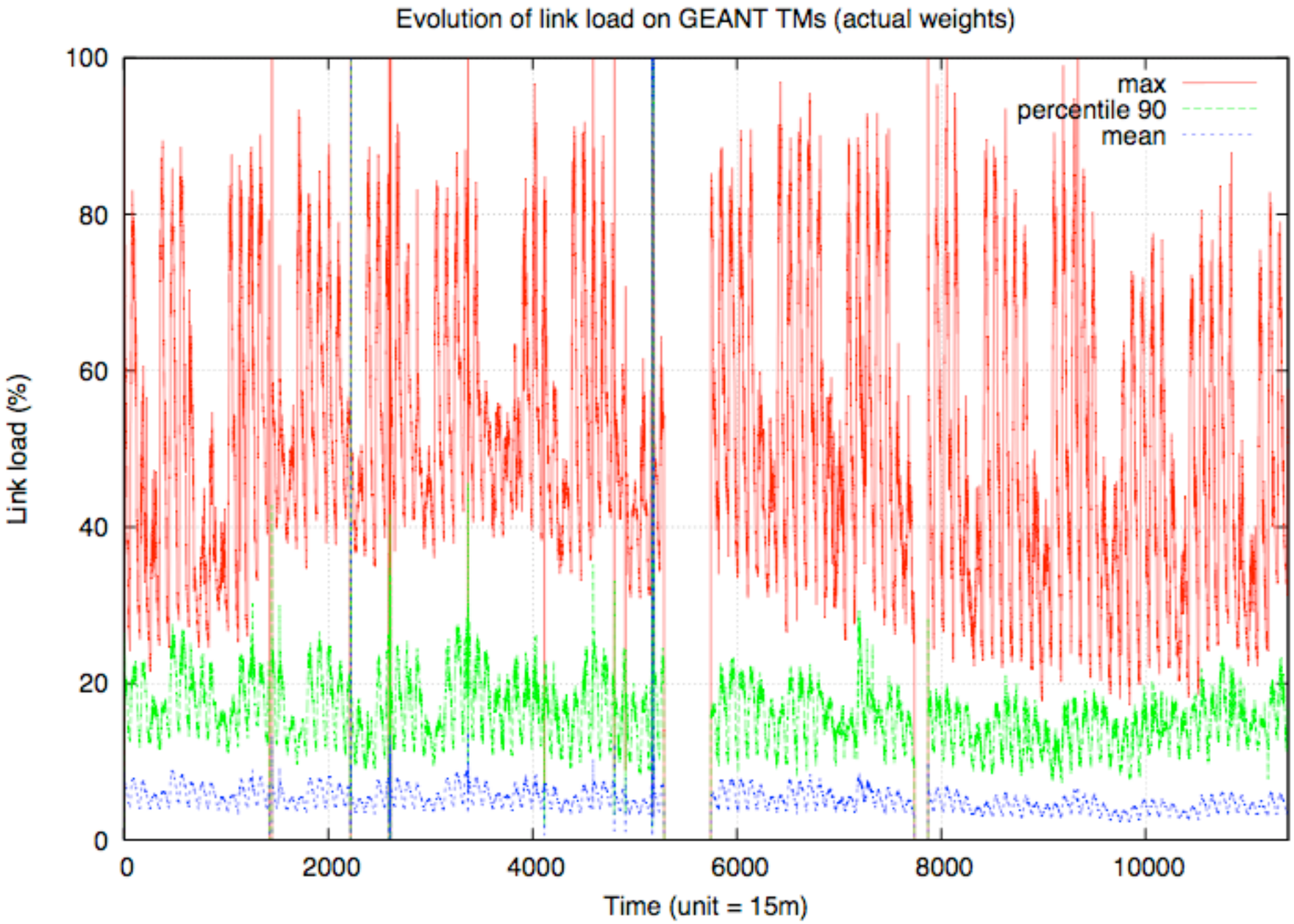
source: Olivier Bonaventure, UCLouvain

Traffic load in large IP networks



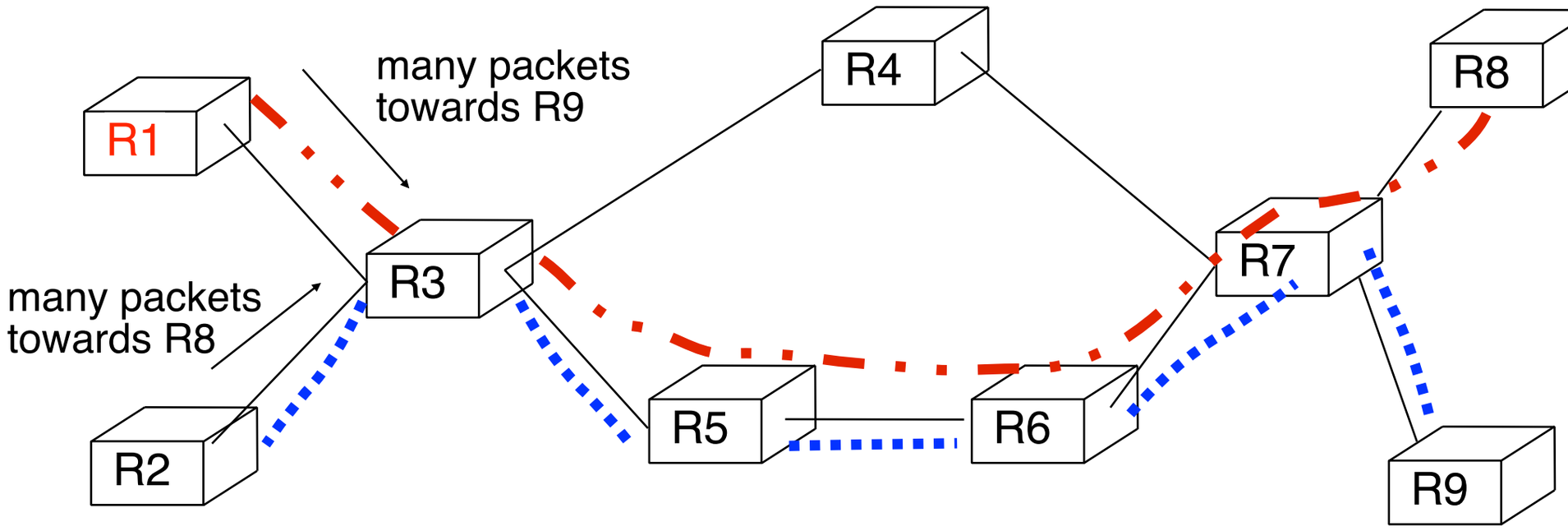
Evolution of link load

- Example with GEANT



Traffic engineering

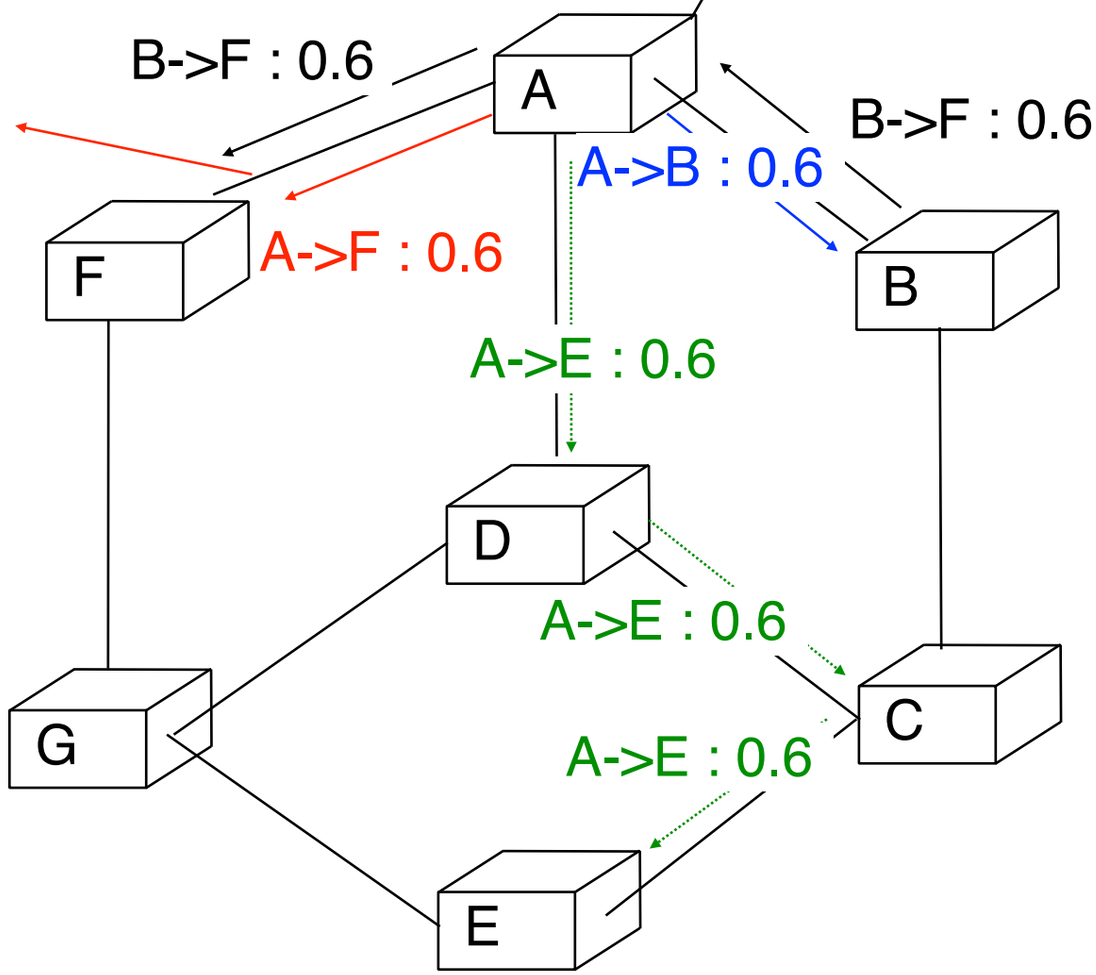
- Problem
  - Shortest path chosen by IP routing does not always lead to a good network utilisation
    - fish problem



- How to better optimise the network utilisation ?
- How to react to changes in traffic conditions

Traffic engineering simple case study

A's routing table
F: d=1 : West
D: d=1 : South
B: d=1 : East
G: d=2 : West
C: d=2 : South
E: d=3 : South



Traffic matrix
A->B : 0.6
A->F : 0.6
B->F : 0.6
A->E : 0.6

B's routing table
A: d=1 : North
C: d=1 : South
D: d=2 : South
E: d=2 : South
F: d=2 : North
G: d=3 : South

IP based traffic engineering

- How to solve the traffic engineering problem in a pure IP network ?
  - Two types of solutions
  - Router-level traffic engineering
    - Allow a router to use several paths instead of a single one for a given route
    - Possible with most router implementations
  - Network-level traffic engineering
    - Force aggregate traffic flows to follow some paths inside the network
    - Possible in some cases by playing with link costs

Equal Cost Multipath

- Simple network-level load balancing mechanism supported in OSPFv2
  - Principle
    - OSPF distributes the complete network topology to all routers inside network
    - based on this topology, each router computes the routes towards all destinations
    - if a router finds several equal cost paths reaching one destination, it may balance its traffic over these paths
    - load balancing is done at the discretion of this router without coordination with other routers
      - since routes are equal cost routes, loops will not occur provided that the routing table is stable

Equal Cost Multipath (2)

- Example

F's routing table

...

E: d=2 : South

D's routing table

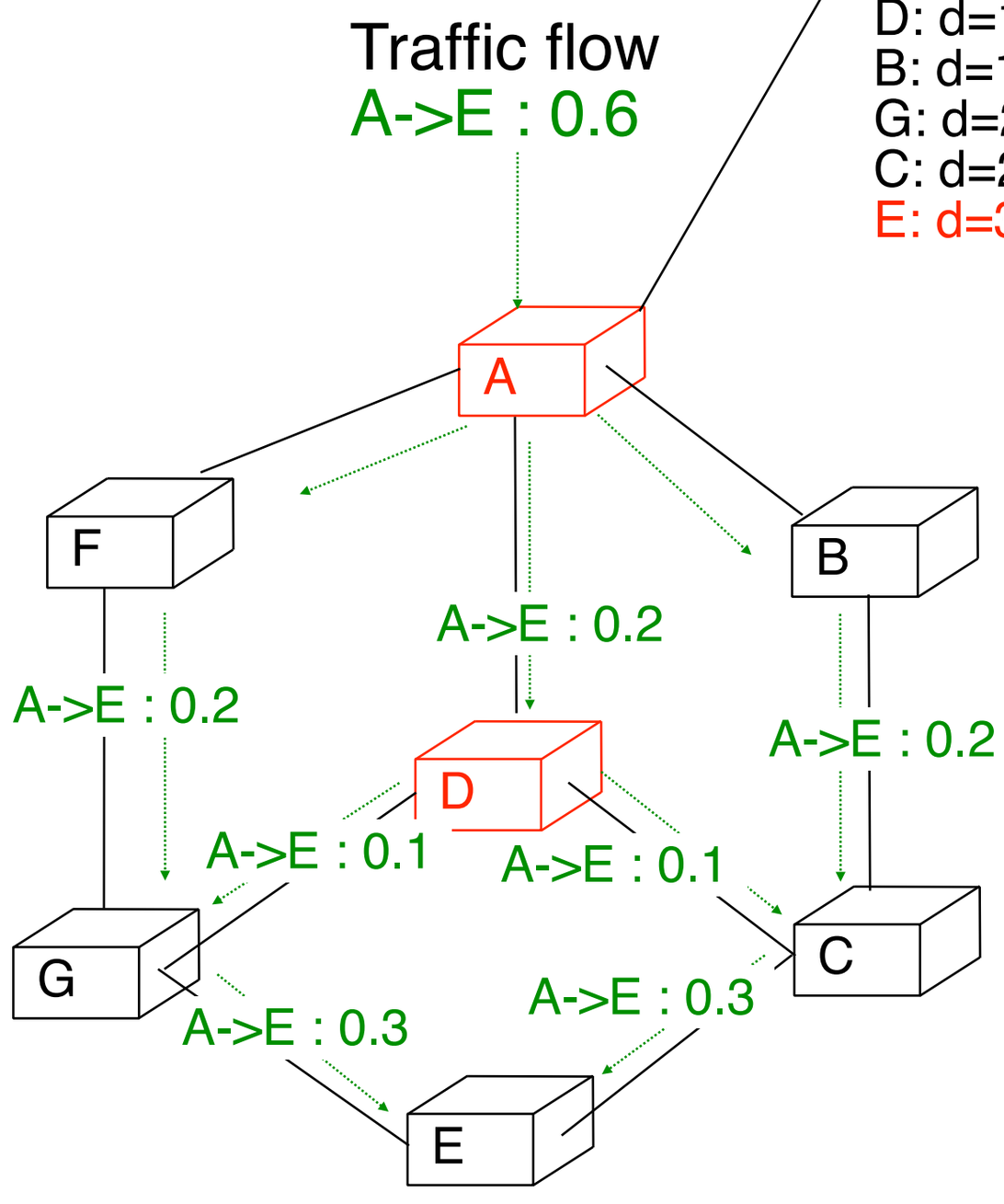
...

E: d=2 : South-West
South-East

G's routing table

...

E: d=2 : South-East



A's routing table

F: d=1 : West
D: d=1 : South
B: d=1 : East
G: d=2 : West, South
C: d=2 : South, East
E: d=3 : East, South, West

B's routing table

...

E: d=2 : South

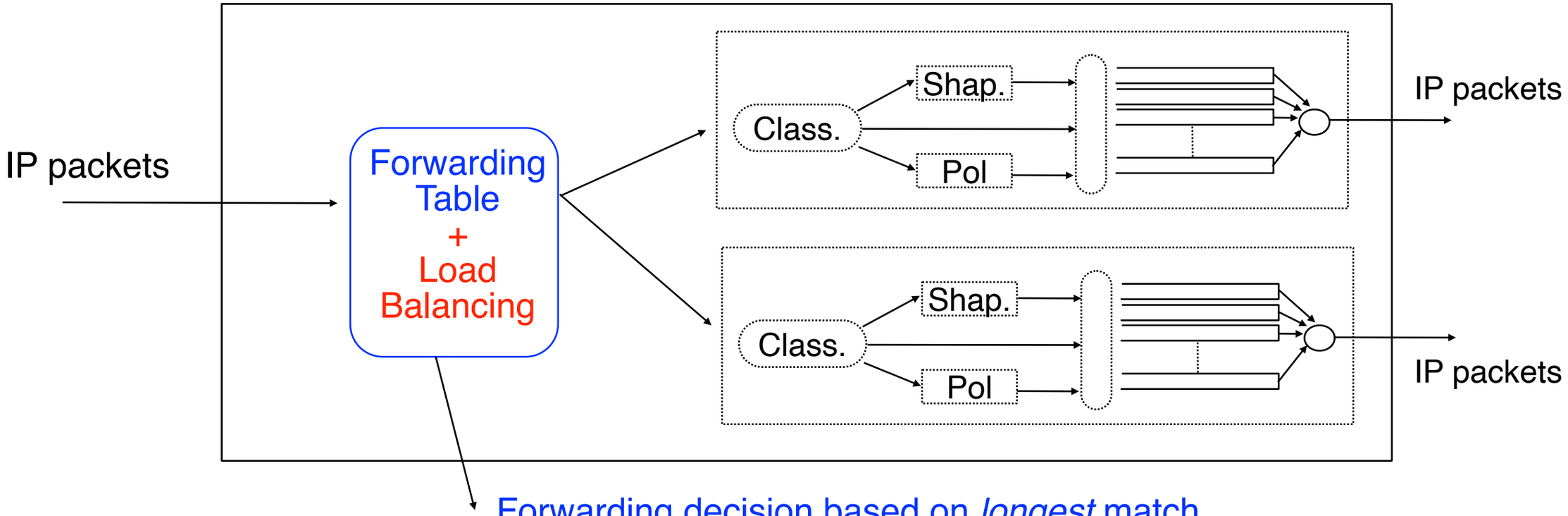
C's routing table

...

E: d=1 : South-West

How to dispatch IP packets ?

- Principle
  - For each destination, remember the P equal paths instead of a single one
    - place those paths in forwarding table
  - when a packet arrives, load balancing algorithm selects one path among the P available paths



Forwarding decision based on *longest* match
Load balancing implies that forwarding table may contain several entries for each destination

Load balancing algorithms

- Simple solution
  - (Deficit) Round-Robin or variants to dispatch packets on a per packet basis

  - Advantages
    - easy to implement since number of paths is small
    - traffic will be divided over the equal cost paths on a per packet basis
      - each path will carry the same amount of packets/traffic

- Drawbacks
  - two packets from the same TCP connection may be sent on different paths and thus be reordered
    - TCP performance can be affected by reordering

Per TCP connection load balancing

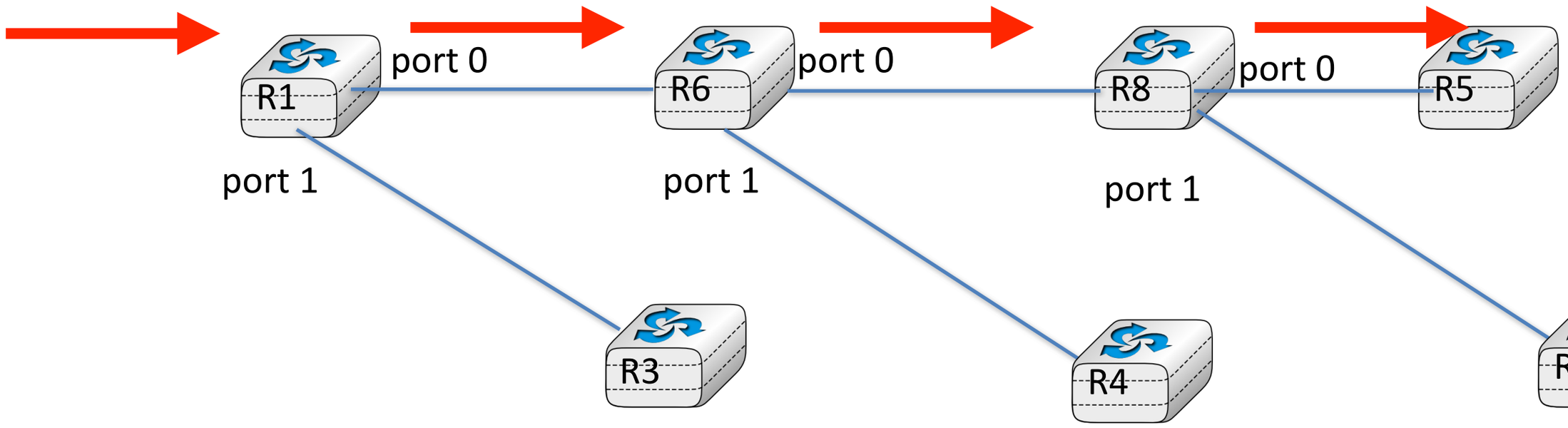
- Principle
  - Perform load balancing on a per TCP connection basis
  - Router identifies the connection to which each packet belongs and all packets from same connection are sent on same path
    - no reordering inside TCP connections
- Issues to address
  - How to efficiently select the path for each TCP conn.
    - Router should not need to maintain a table containing
      - IP src, IP dest, Src port, Dest port : path to utilise
    - TCP connections towards to busy server should not be all sent sent on the same path

Per TCP connection load balancing

- How to perform load balancing without maintaining state for each TCP connection ?
  - Principle
    - concatenate IP src, IP dest, IP protocol, Src port, and Dest port from the IP packet inside a bit string
      - bitstring = [IP src:IP dest:IP protocol:Src port:Dest port]
    - compute path = Hash(bitstring) mod P
      - hash function should be easy to implement and should produce very different numbers for close bitstring values
      - candidate hash functions are CRC, checksum, ...
  - Advantages
    - all packets from TCP connection sent on same path
    - traffic to a server will be divided over the links
  - Drawback
    - does not work well if a few TCP connections carry a large fraction of the total traffic
    - Polarisation issues if all routers use same hash

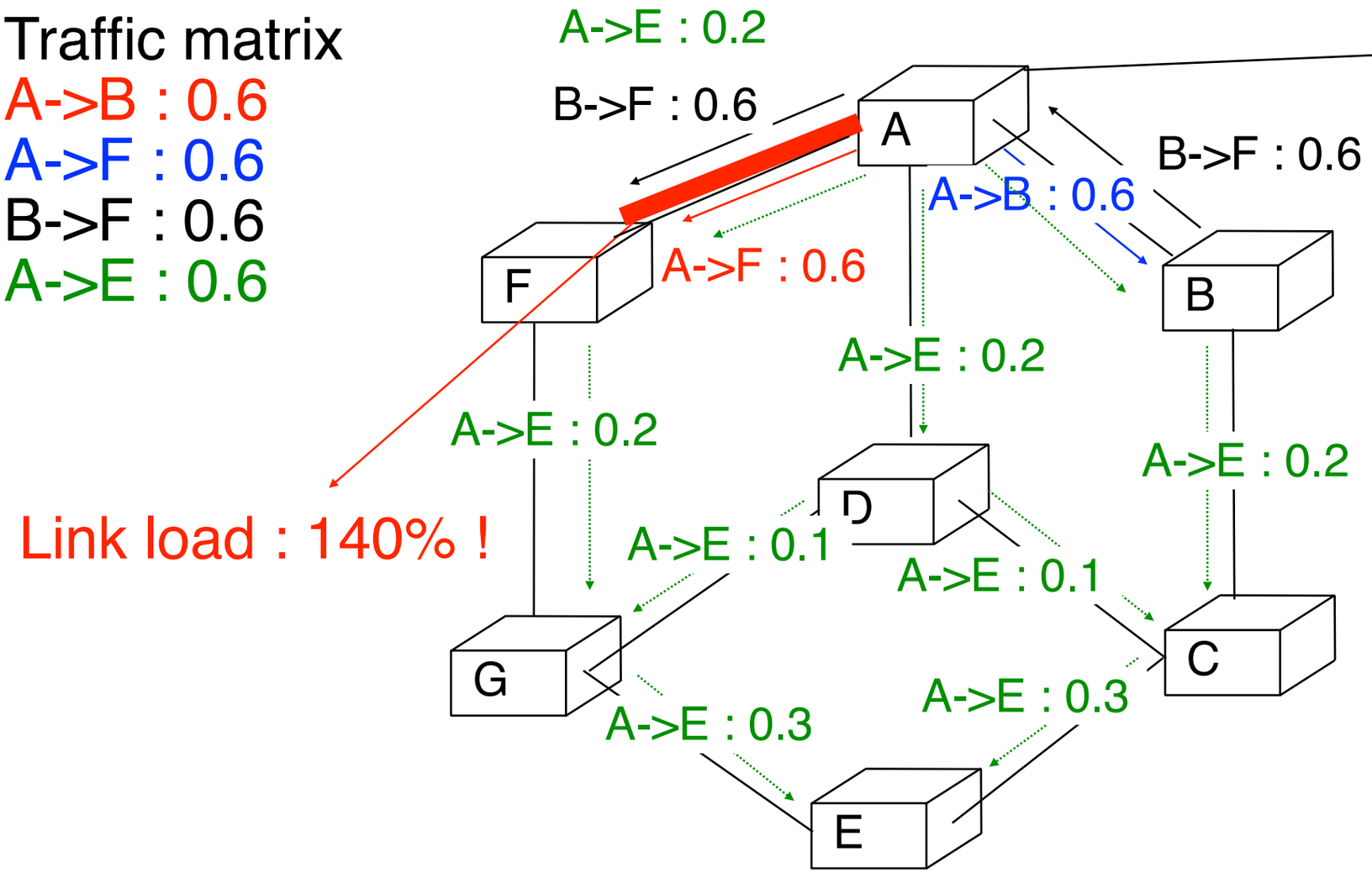
The polarisation problem

- How to prevent this polarisation problem ?



Limitations of Equal Cost Multipath

Traffic matrix
A->B : 0.6
A->F : 0.6
B->F : 0.6
A->E : 0.6



A's routing table
F: d=1 : West
D: d=1 : South
B: d=1 : East
G: d=2 : West, South
C: d=2 : South, East
E: d=3 : East, South, West

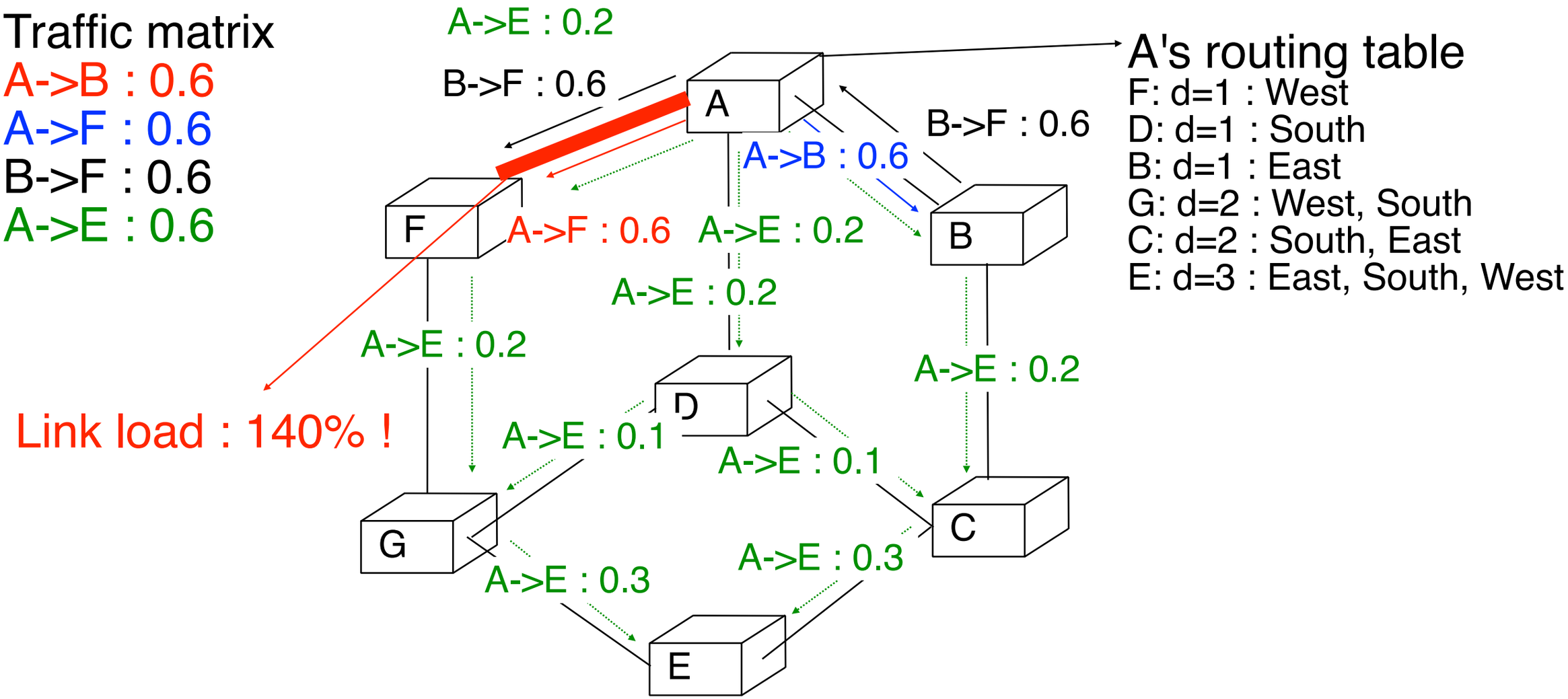
- Drawbacks of ECMP
  - load balancing only works for *exactly equal* costs paths and few paths are exactly equal
  - local decision taken by each individual router

IP-based network level traffic engineering

- How to improve the traffic distribution throughout the entire network ?
  - Principle
    - IGP link cost influences the utilisation of this link
      - Typical IGP link cost settings include
        - 1 for each link to select shortest path measured in hops
        - =link delay to select shortest path measured in seconds
        - f(bandwidth) to select shortest-high bandwidth path
        - example : $\frac{M}{link\ bandwidth}$
    - Careful selection of the IGP link costs to balance traffic
      - rerouting traffic outside a busy link by manually tweaking costs
      - optimising the flow of traffic instead a network for a given traffic matrix can considering it as a classical optimisation problem
        - Can be difficult if routers do not support ECM
        - possible with some restrictions when routers support ECM

IP-based network level traffic engineering (2)

Traffic matrix
A->B : 0.6
A->F : 0.6
B->F : 0.6
A->E : 0.6



A's routing table
F: d=1 : West
D: d=1 : South
B: d=1 : East
G: d=2 : West, South
C: d=2 : South, East
E: d=3 : East, South, West

Link load : 140% !

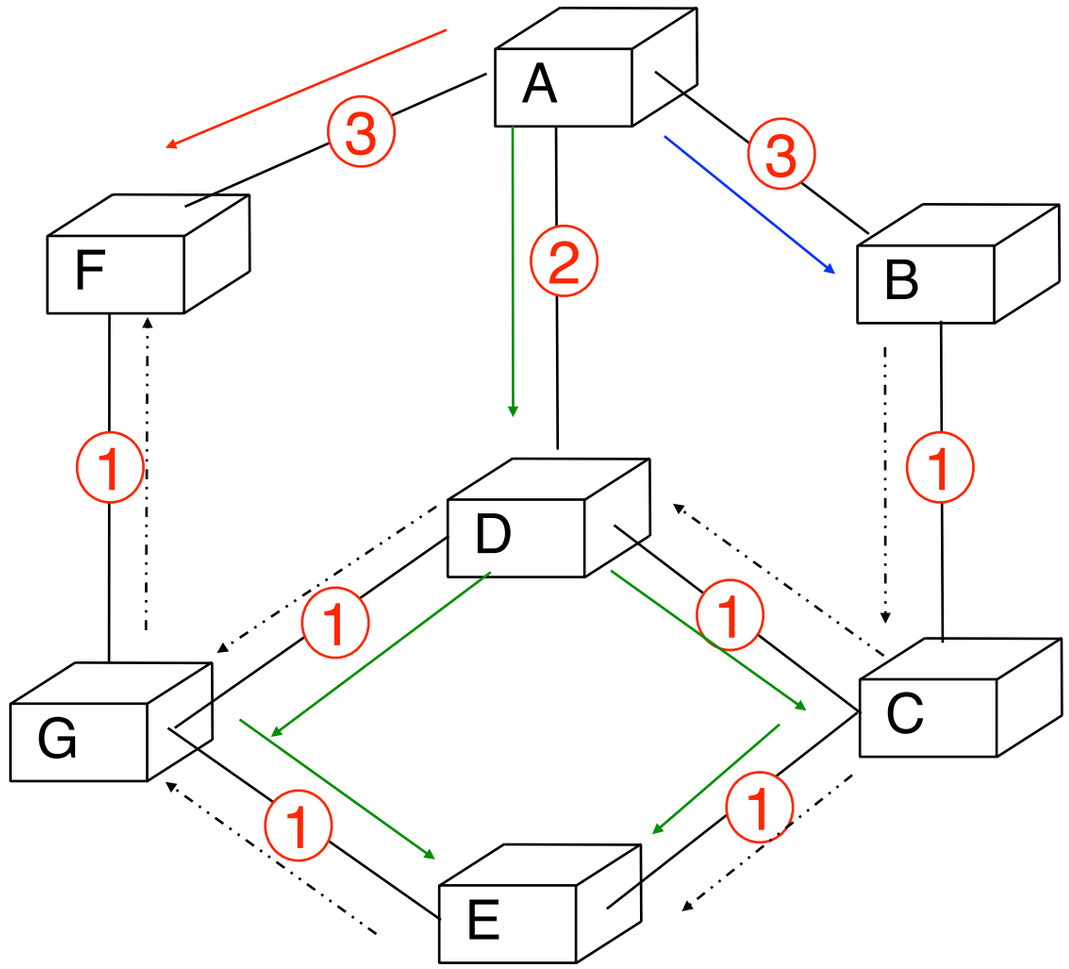
- How to improve the traffic distribution ?
  - A should send traffic towards E only via its South port
  - B should send traffic towards F only via its South port
  - Possible by changing the IGP link costs

IP-based network level traffic engineering (3)

- Possible setting of the IGP link costs

D's routing table
A: d=2: North
B: d=2: South-East
C: d=1: South-East
E: d=2: S-E, S-W
F: d=2: S-W
G: d=1: S-W

A's routing table
F: d=3: West
D: d=2: South
B: d=3: East
E: d=4: South
G: d=3: South
C: d=3: South



B's routing table
A: d=3: North
C: d=1: South
D: d=2: South
E: d=E: South
F: d=4: South
G: d=3: South

Traffic matrix
A->B : 0.6
A->F : 0.6
B->F : 0.6
A->E : 0.6

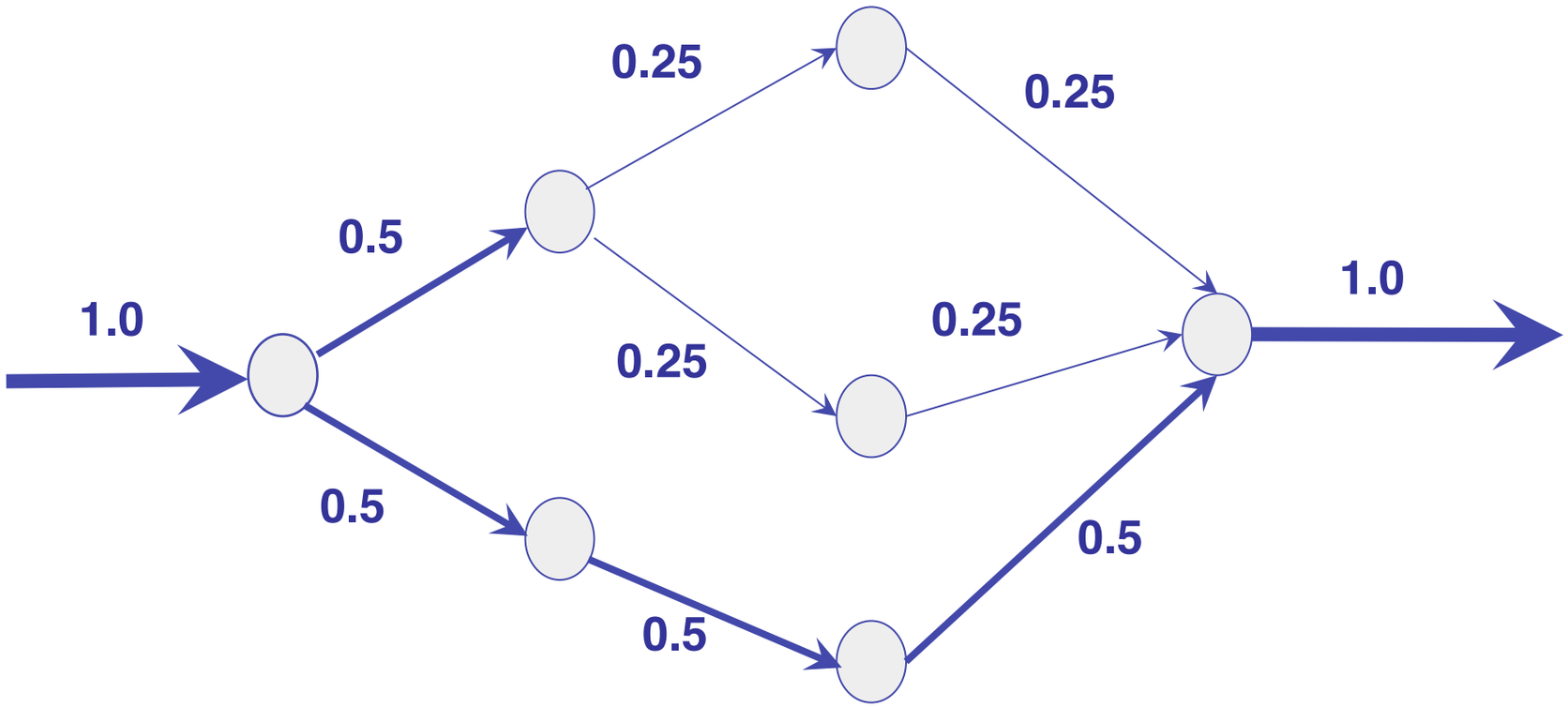
C's routing table
B: d=1: North
A: d=3: North-West
D: d=1: North-West
E: d=1: South-West
F: d=3: N-W, S-W
G: d=2:N-W, S-W

Optimization Problem

- Input: graph *G(R,L)*
  - *R* is the set of routers
  - *L* is the set of unidirectional links
  - c_l is the capacity of link *l*
- Input: traffic matrix
  - $M_{i,j}$ is traffic load from router *i* to *j*
- Output: setting of the link weights
  - w_l is weight on unidirectional link *l*
  - $P_{i,j,l}$ is fraction of traffic from *i* to *j* traversing link *l*

source: Alex Snoeren, UCSD

Equal-Cost Multipath (ECMP)

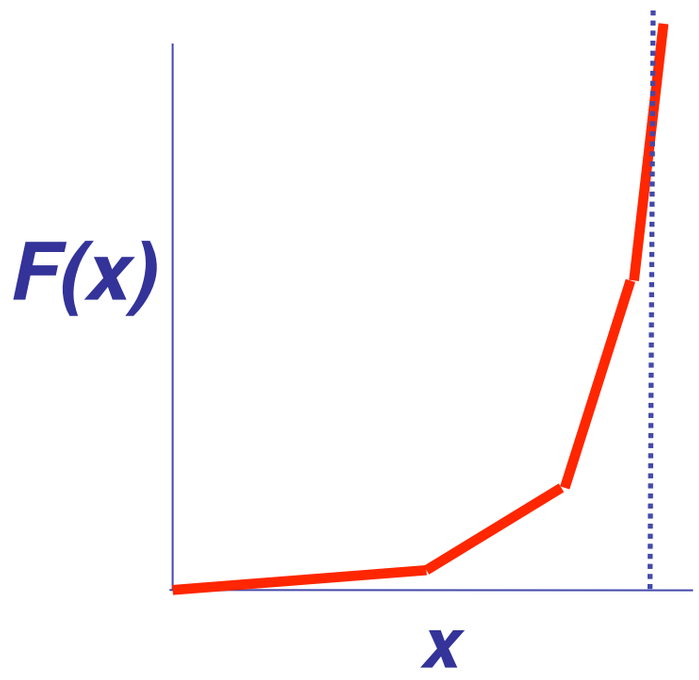


Values of $P_{i,j,l}$

source: Alex Snoeren, UCSD

Objective Function

- Computing the link utilization
  - Link load: $u_l = \sum_{i,j} M_{i,j} P_{i,j,l}$
  - Utilization: u_l / c_l
- Objective functions
  - $\min(\max_l (u_l/c_l))$
  - $\min(\sum F(u_l/c_l))$



source: Alex Snoeren, UCSD

Complexity of Optimization

- NP-complete optimization problem
  - No efficient algorithm to find the link weights
  - Even for simple objective functions
  - Have to resort to *searching* through weight settings

- Clearly suboptimal, but effective in practice
  - Fast computation of the link weights
  - Good performance, compared to “optimal” solution

source: Alex Snoeren, UCSD

Operational Realities

- Minimize number of changes to the network
  - Changing just 1 or 2 link weights is often enough
- Tolerate failure of network equipment
  - Weights settings usually remain good after failure
  - … or can be fixed by changing one or two weights
- Limit dependence on measurement accuracy
  - Good weights remain good, despite random noise
- Limit frequency of changes to the weights
  - Joint optimization for day & night traffic matrices

source: Alex Snoeren, UCSD

Check out this paper if you want to learn more about it

URL: https://dinamico2.unibg.it/martignon/documenti/architetture/OSPF_TE.pdf

Internet Traffic Engineering by Optimizing OSPF Weights

Bernard Fortz
Service de Mathématiques de la Gestion
Institut de Statistique et de Recherche Opérationnelle
Université Libre de Bruxelles
Brussels, Belgium
bfortz@smg.ulb.ac.be

Mikkel Thorup
AT&T Labs-Research
Shannon Laboratory
Florham Park, NJ 07932
USA
mthorup@research.att.com

Abstract—**Open Shortest Path First (OSPF) is the most commonly used intra-domain internet routing protocol. Traffic flow is routed along shortest paths, splitting flow at nodes where several outgoing links are on shortest paths to the destination. The weights of the links, and thereby the shortest path routes, can be changed by the network operator. The weights could be set proportional to their physical distances, but often the main goal is to avoid congestion, i.e. overloading of links, and the standard heuristic recommended by Cisco is to make the weight of a link inversely proportional to its capacity.**

Our starting point was a proposed AT&T WorldNet backbone with demands projected from previous measurements. The desire was to optimize the weight setting based on the projected demands. We showed that optimizing the weight settings for a given set of demands is NP-hard, so we resorted to a local search heuristic. Surprisingly it turned out that for the proposed AT&T WorldNet backbone, we found weight settings that performed within a few percent from that of the *optimal general routing* where the flow for each demand is optimally distributed over all paths between source and destination. This contrasts the common belief that OSPF routing leads to congestion and it shows that for the network and demand matrix studied

nodes and arcs represent routers and the links between them. Each arc a has a capacity $c(a)$ which is a measure for the amount of traffic flow it can take. In addition to the capacitated network, we are given a demand matrix D that for each pair (s,t) of nodes tells us how much traffic flow we will need to send from s to t. We shall refer to s and t as the source and the destination of the demand. Many of the entries of D may be zero, and in particular, $D(s,t)$ should be zero if there is no path from s to t in G. The routing problem is now, for each non-zero demand $D(s,t)$, to distribute the demanded flow over paths from s to t. Here, in the general routing problem, we assume there are no limitations to how we can distribute the flow between the paths from s to t.

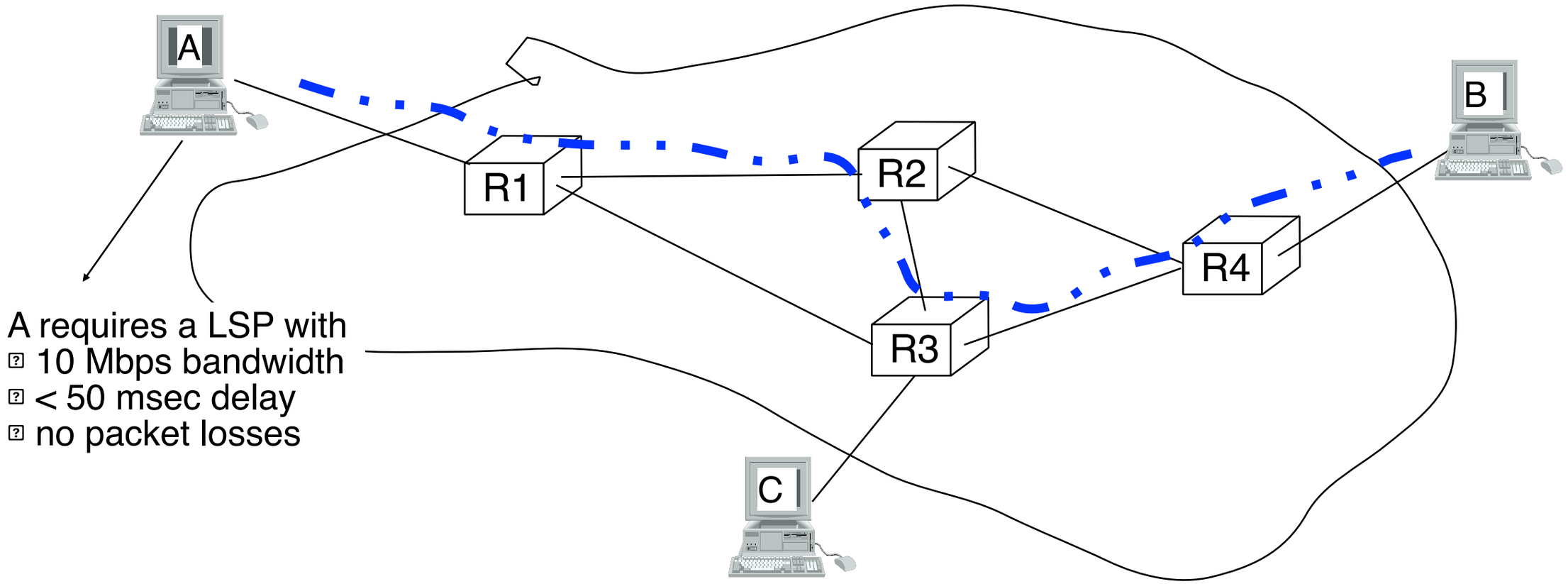
The above definition of the general routing problem is equivalent to the one used e.g. in Awduche et al. [1]. Its most controversial feature is the assumption that we have an estimate of

MPLS-based traffic engineering

- Principle of the solution
  - Build a normal IP or IP+MPLS network
    - packet forwarding on shortest path towards destination
  - Collect traffic statistics at edge routers and information about link load
    - identify the most congested parts of the network
  - Ingress routers establish LSPs along a well chosen path to divert large traffic flows away from heavily loaded links

- Issues to solve
  - How can an ingress router determine an acceptable path for a given traffic flow ?
  - How to establish a LSP along a chosen path ?

Selecting a path with constraints



- How can we establish a LSP with QoS constraints through the network ?
  - Need information about capabilities of each link
  - Need an algorithm to select the best path according to specific constraints

Constrained routing

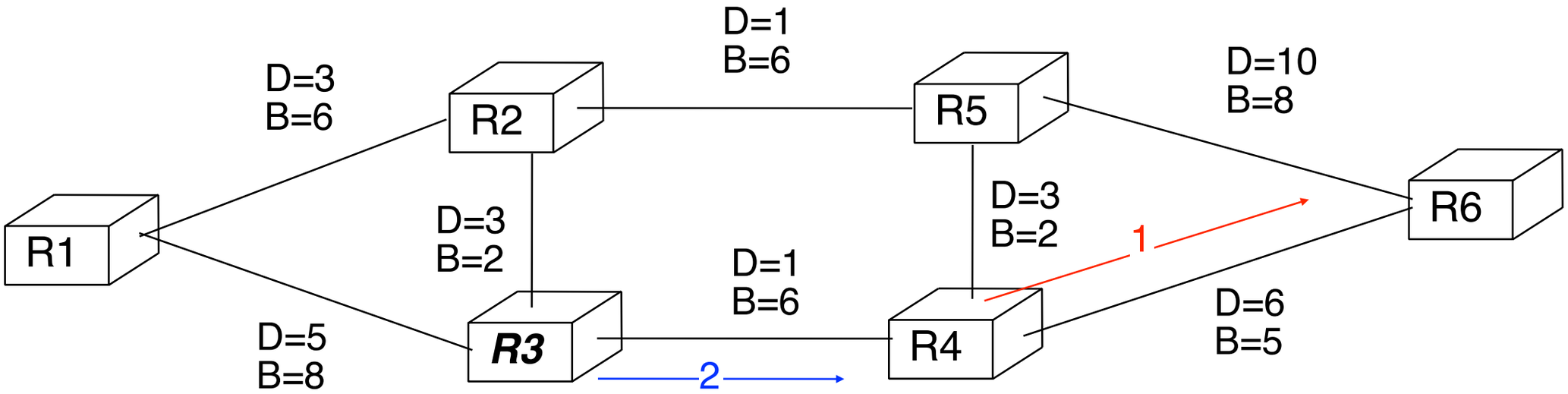
- What should be added to traditional routing algorithms ?
  - a way to distribute information about current network state
    - routers must know load of remote links to choose paths meeting constraints for flows with QoS guarantees
  - a way to compute a path subject to constraints
    - current routing algorithms find shortest path
    - how can we find a path with
      - minimum hop count
      - at least 10 Mbps
      - at most 10 msec of delay

Distributing load information

- Distance vector routing protocols [RIP,BGP]
  - routers conspire to distribute routing table
  - difficult to inform routers of load on remote links
  - difficult to support constrained routing
- Link state routing protocols [OSPF, IS-IS]
  - routers conspire to distribute network map
    - simple to add information about network load
  - routers distribute link state packets with load info
    - delay is already distributed as the IGP metric
    - bandwidth/link load is main information to distribute
    - tradeoff between frequent distribution (accurate information) and rare distribution (avoid network overload)
  - each router knows topology and load of each link and can find constrained paths

Distributing load information (2)

- Potential problem
  - Link load information is not distributed immediately
  - routers must establish flows based on partial information about the current load in the network



1. new flow [B=4] is created between R4 and R6

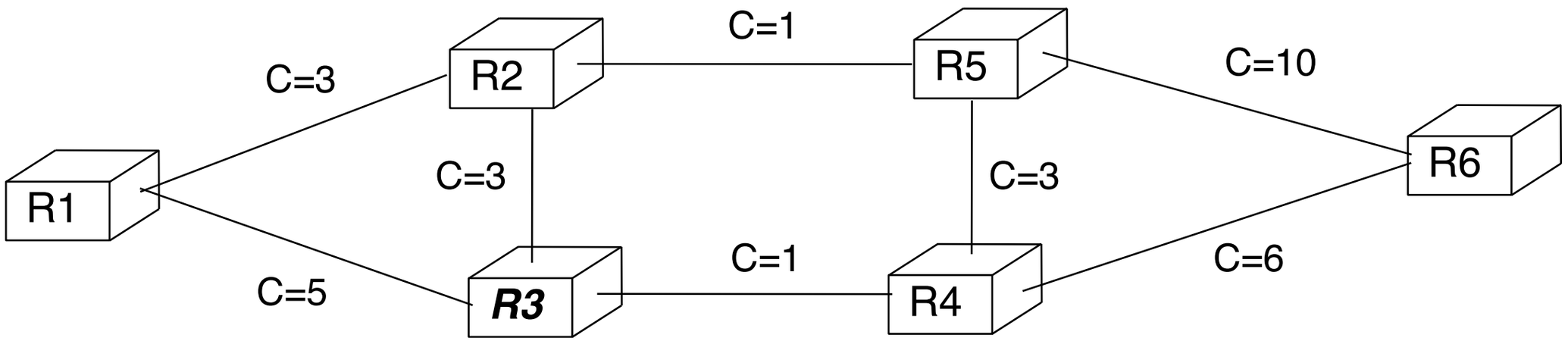
1. before information about load changes, R3 wants to create a new flow [B=2] towards R6
   - R3 believes that R3-R4-R6 is the best path

Constraints

- Three types of constraints on path selection
  - Additive constraint
    - find path minimising sum of link characteristics
      - example
        - hop count
        - link delay or cost
  - Multiplicative constraint
    - find path minimising product of link characteristics
      - example
        - loss rate
  - Concave constraint
    - find path containing links whose characteristic is always above a given constraint
      - example
        - bandwidth
        - resource class or color

Finding a constrained path

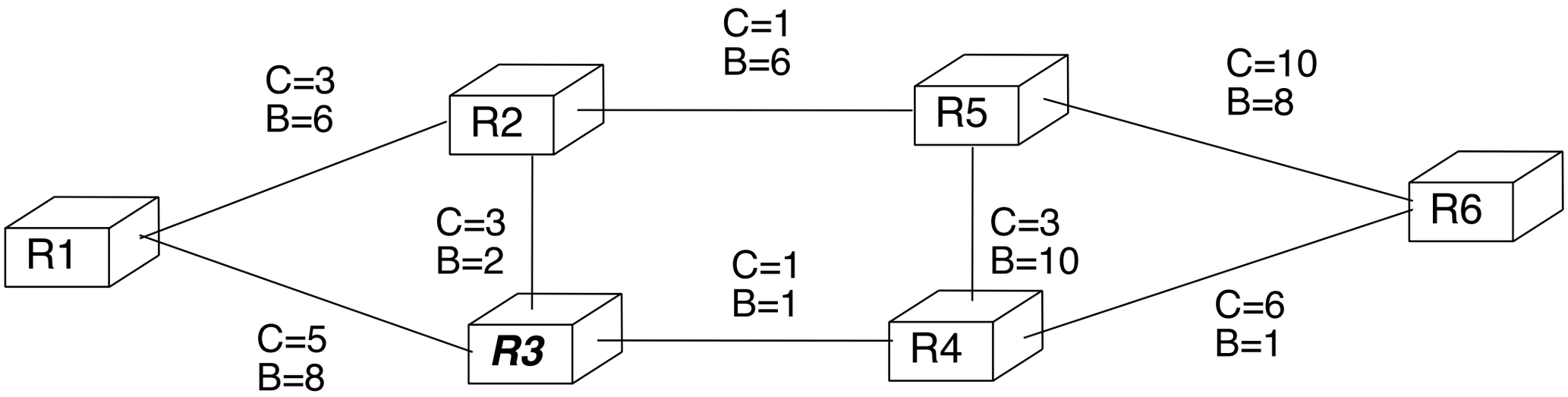
- Single additive or multiplicative constraints
  - apply Dijkstra's algorithm
    - example



- 2 or more additive/multiplicative constraints
  - unfortunately problem is NP-complete
    - need to evaluate all possible paths to find exact solution
    - several heuristics have been proposed in literature to find acceptable solutions

Finding a constrained path (2)

- Concave constraints
  - fortunately easy to handle
    - remove from the network map all links that do not satisfy the constraint
    - use Dijkstra's algorithm on the reduced map
    - example
      - find shortest 3 Mbps from R3 to R6



Constrained routing in IP networks

- Several solutions proposed by researchers
  - Lessons learned
    - Constrained routing should be applied to flows and not on a per packet basis
      - Currently, constrained routing in IP networks is only used with MPLS to establish LSPs
    - Bandwidth and delay are key constraints
      - delay jitter is less important and difficult to efficiently support
    - Path selection should be performed by the source
      - the source of a flow selects an explicit route
      - intermediate nodes perform connection admission control but do not perform any constrained routing decision
        - path selection algorithm does not need to be standardised
        - if the new flow is acceptable, establishment continues otherwise the source will have to compute another path
- Existing constrained routing protocols
  - OSPF-TE, ISIS-TE, PNNI (ATM)

OSPF-TE
An example constrained routing protocol

- Extension to OSPF designed to aid in the establishment of traffic engineered LSPs
  - OSPF-TE distributes new info about each link
    - link type and link Id
    - local and remote IP addresses
    - traffic engineering metric
      - additional metric to specify the cost of this link
    - maximum bandwidth
      - maximum amount of bandwidth usable on this link
    - maximum reservable bandwidth
      - maximum amount of bandwidth that can be reserved by LSPs
    - unreserved bandwidth
      - amount of bandwidth that is not yet reserved by LSPs
    - resource class/color
      - can be used to specify the type of link (e.g. Expensive link would be colored in red and cheap links in green)

Advanced Topics in Communication Networks

Internet Routing and Forwarding

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

29 Sep 2020