Network's Adventures in Softwar'land

Damien Saucez

January 2018

Networking technology is at the middle age of CS (1)

Networks are managed by configuration but

- each protocol has its own set of configuration,
- it is impossible to react to sudden unexpected changes.

Networking technology is at the middle age of CS (2)

No abstraction is used so

- one need to know the network details (e.g., link capacity, IP addresses, hw…),
- one need a deep understanding of the deployed protocols and their interactions.

Networking technology is at the middle age of CS (2)

No abstraction is used so

As if we implemented everything in assembly language!

- one need a deep understanding of the deployed protocols and their interactions.

Software Defined Networking concept

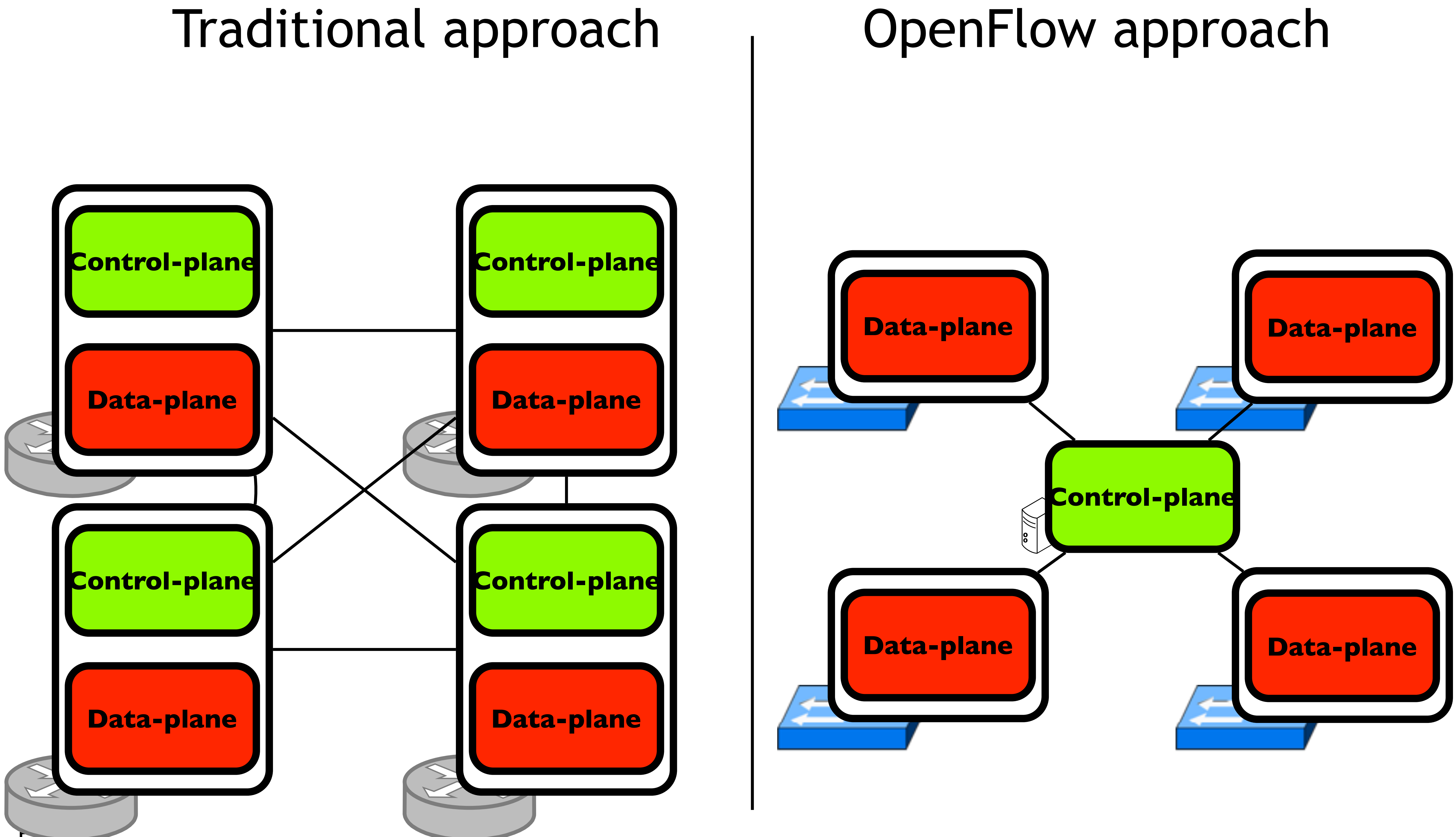
- The traditional approach sees networks as a set of devices to configure.

  Operators are networking experts.

- SDN conceives the network as a program.

  Network logic is implemented by humans but network elements are never touched by humans.

SDN with OpenFlow



Cost reduction with COTS

Data-plane devices only perform forwarding:

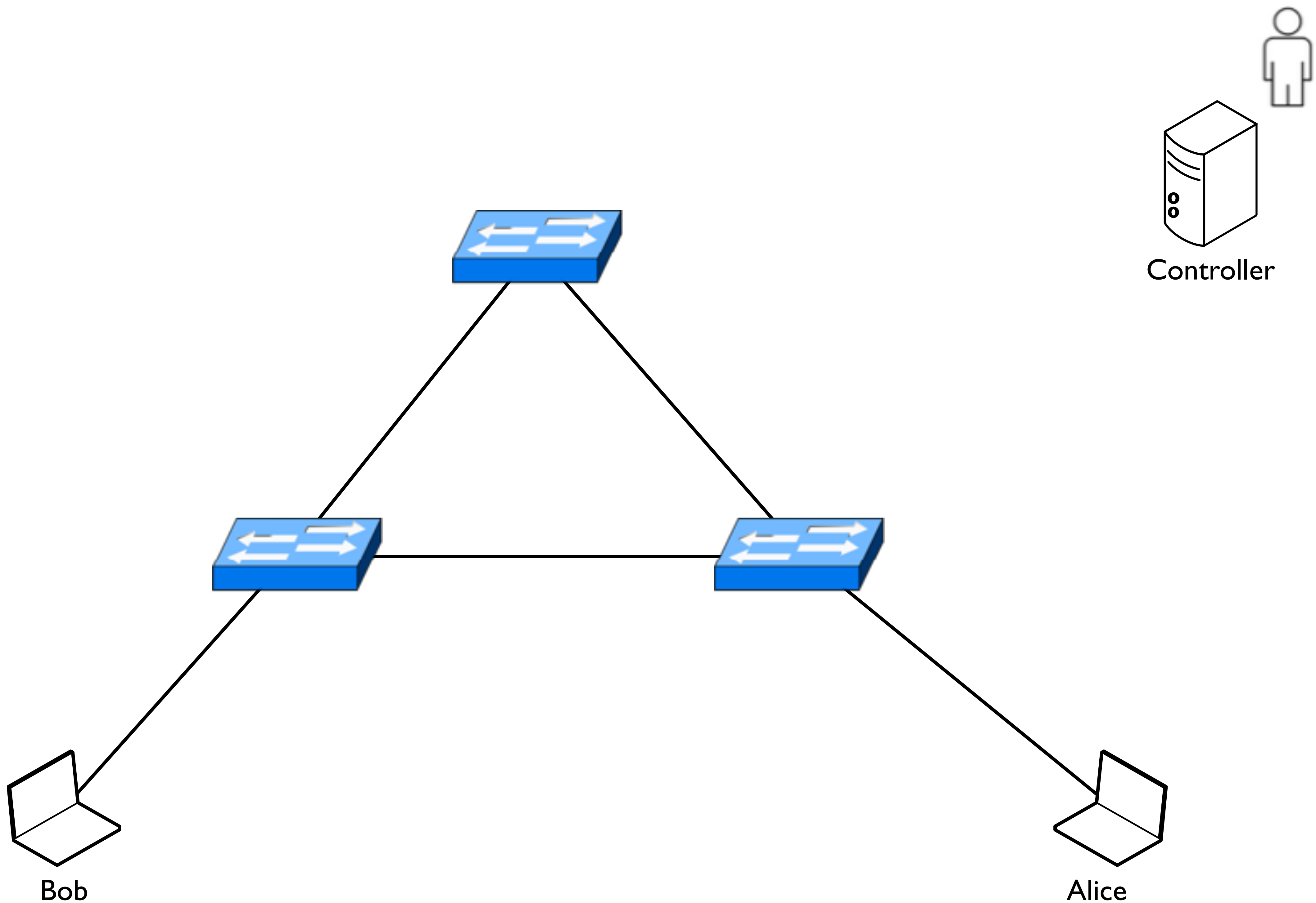
- simple memory structures,
- simple instruction set,

➡ easy virtualisation.

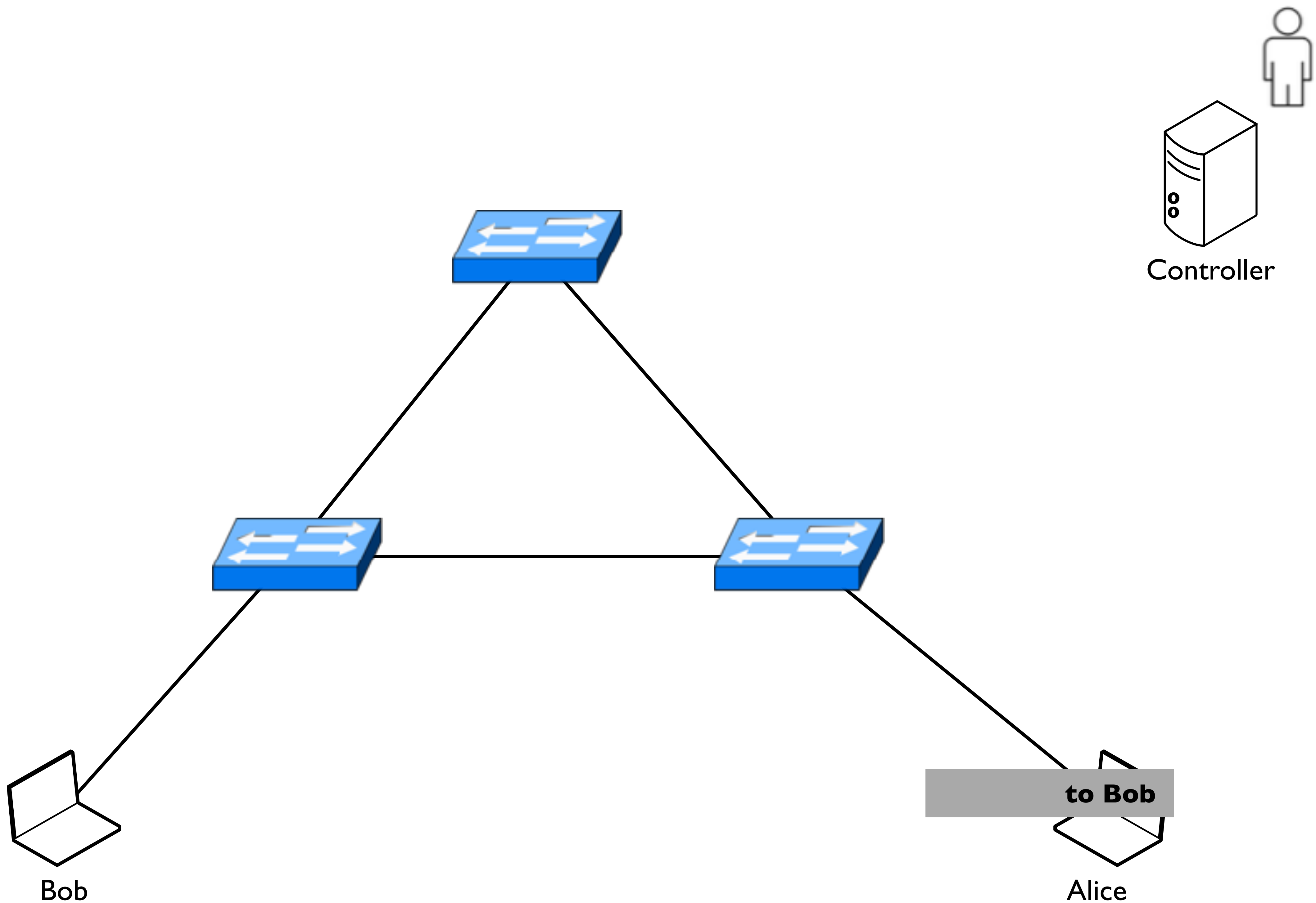
The control plane runs on x86.

- No vendor lock-in.

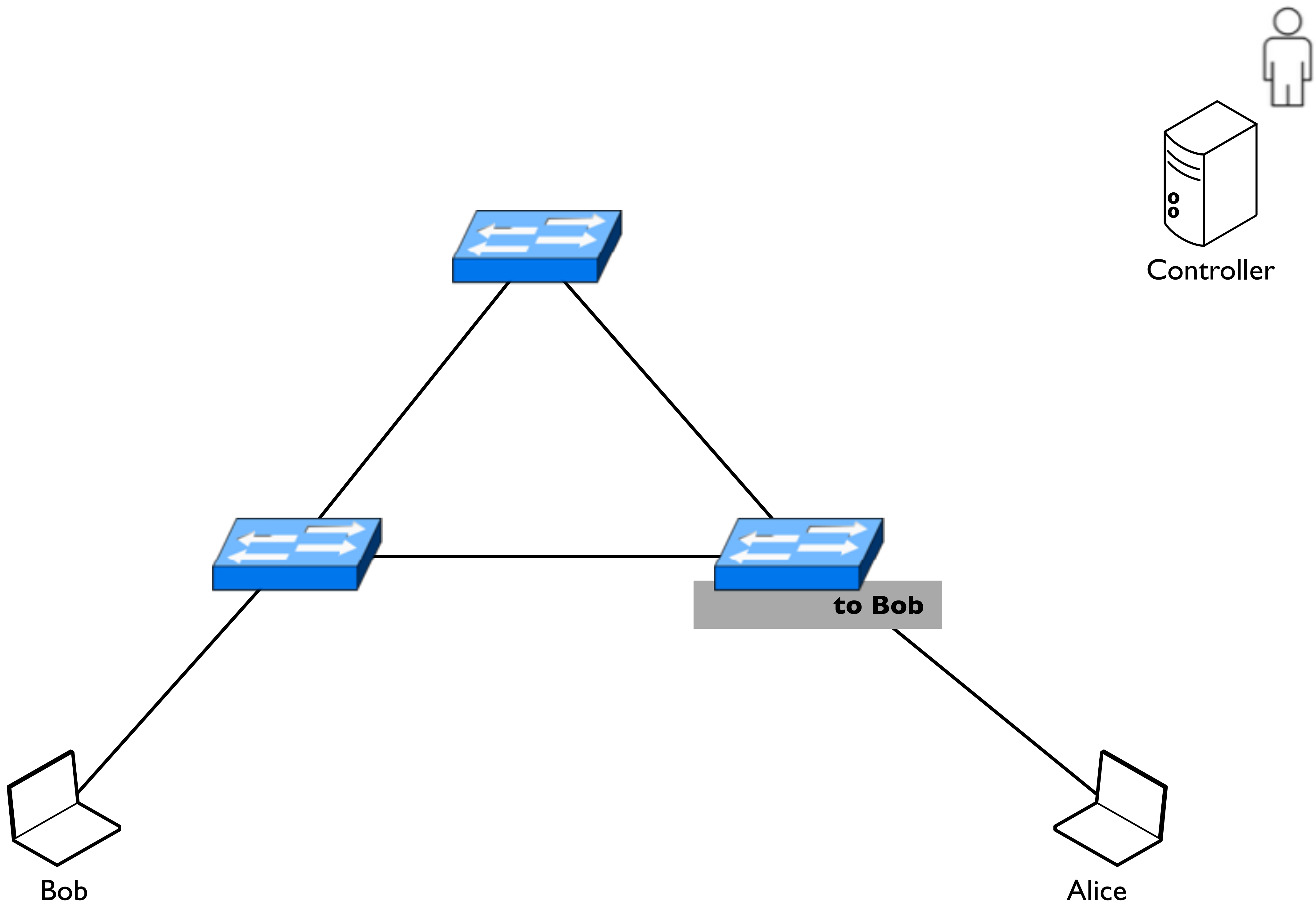
How does it work?



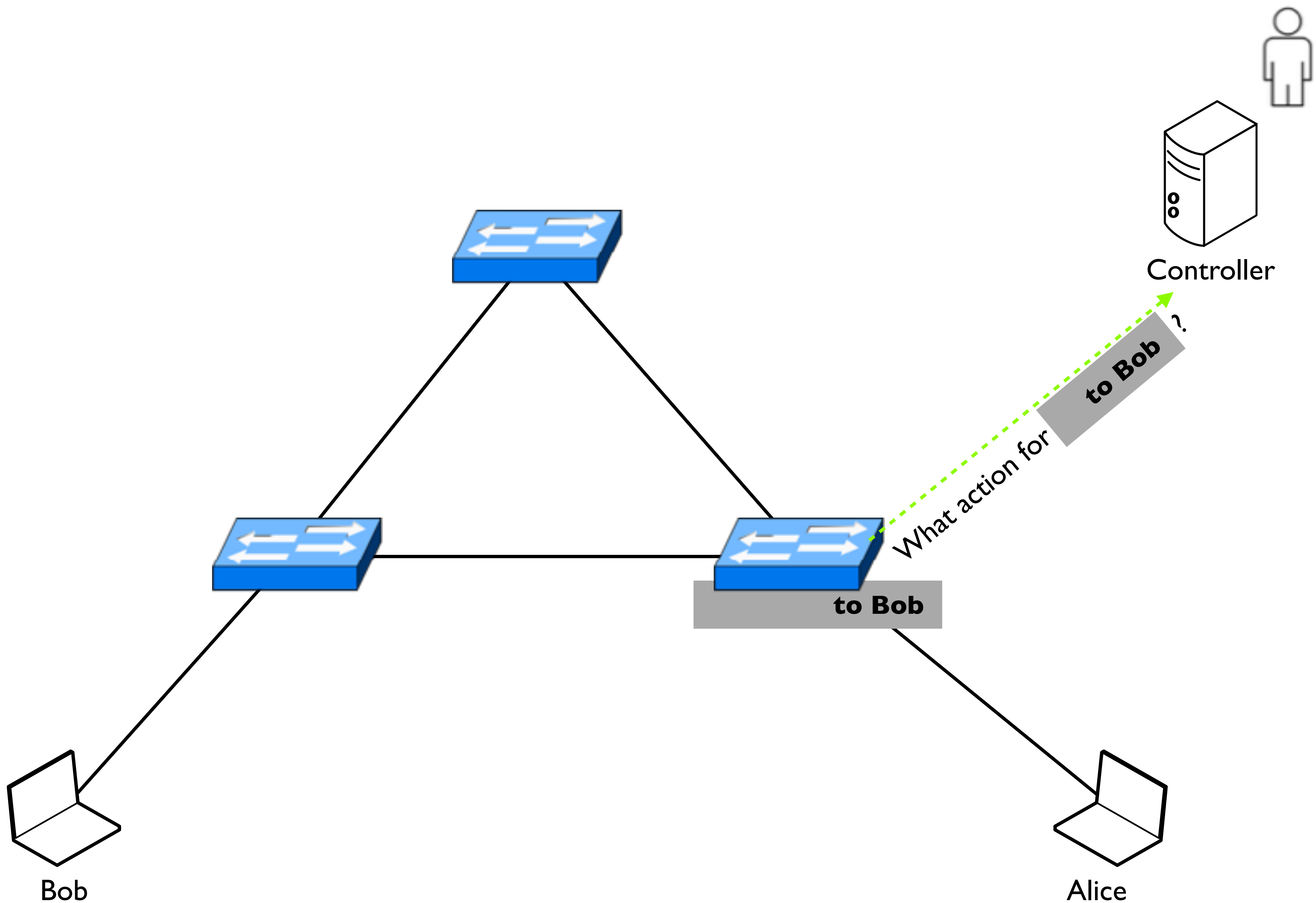
How does it work?



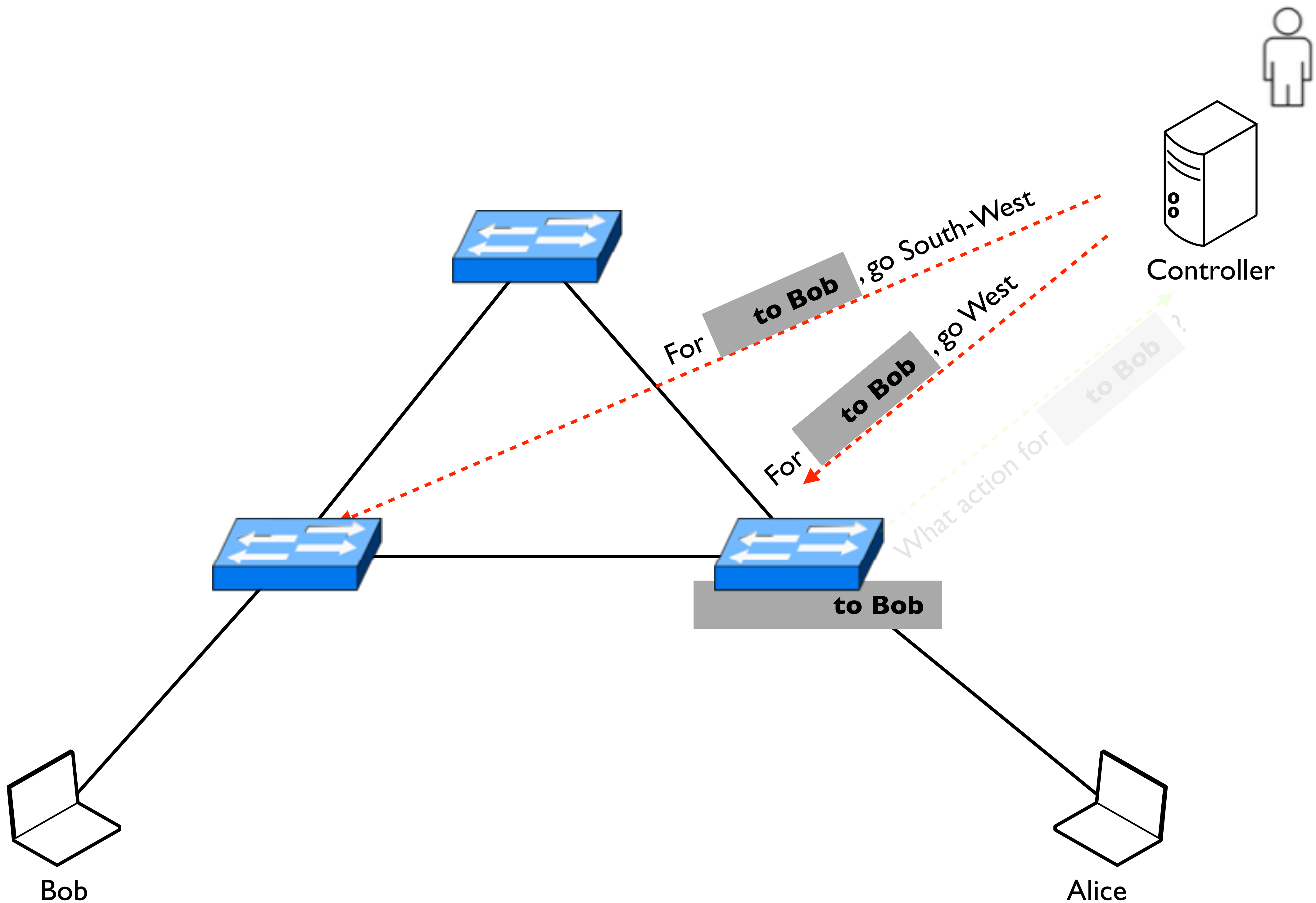
How does it work?



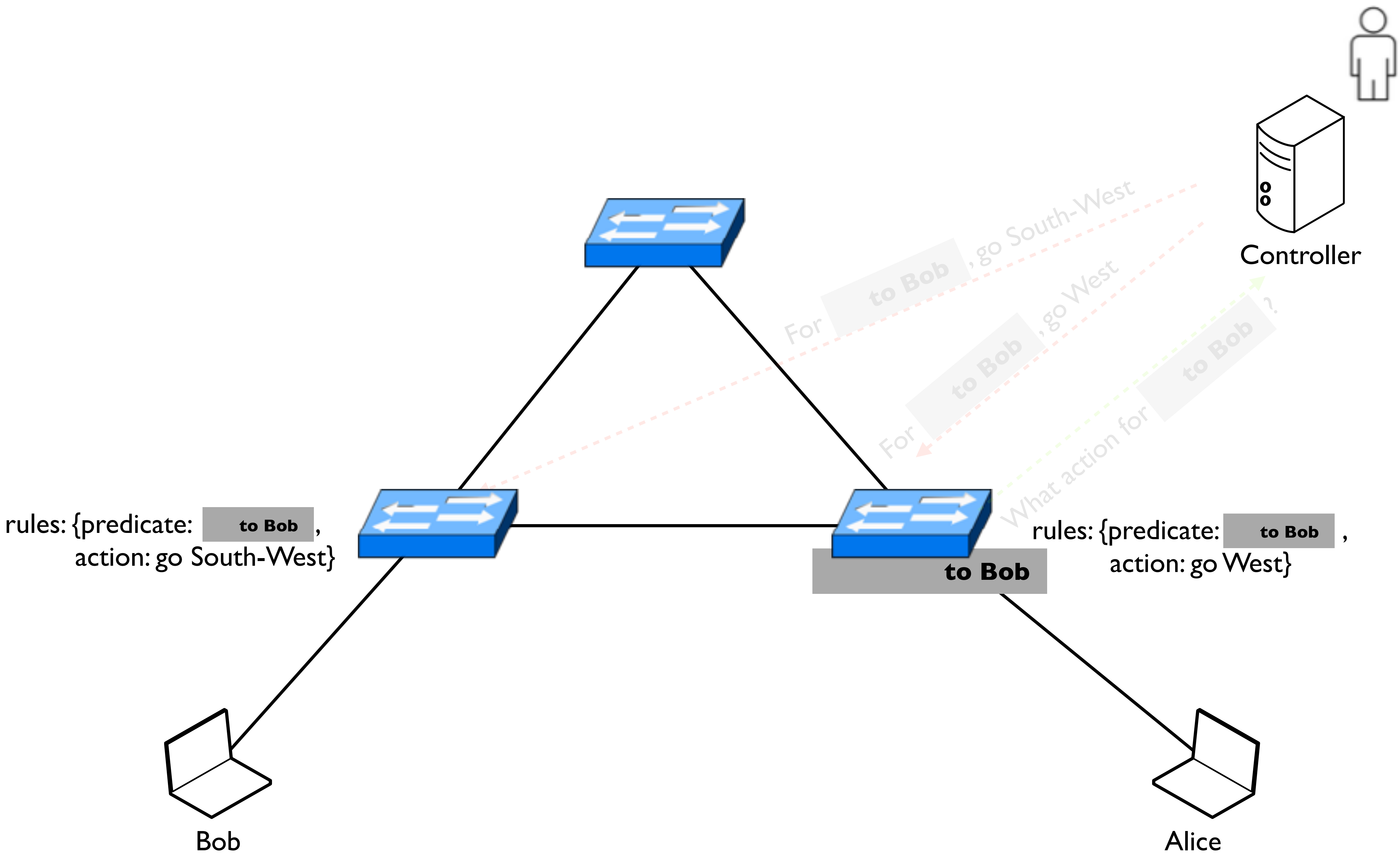
How does it work?



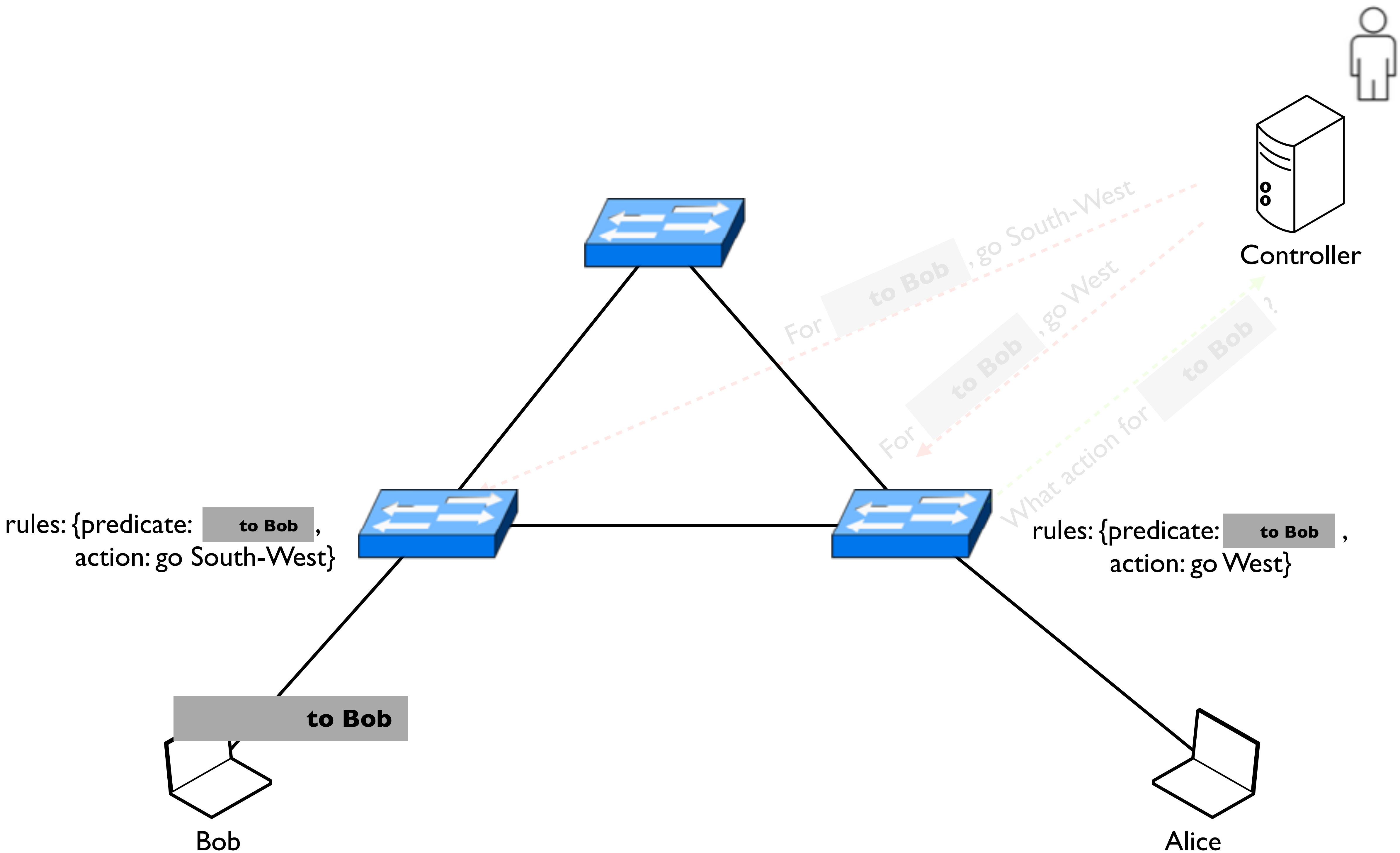
How does it work?



How does it work?



How does it work?



Treat the network as a black box

See the network as a black box [NST+14, NSB+15] so the operator

- follows the declarative programming paradigm to program the network (i.e., what not how),
- sees it as a system with infinite resources (like a computer for an application).

[NST+14] Optimizing rules placement in OpenFlow networks: trading routing for better efficiency, X. N. Nguyen, D. Saucez, T. Turletti, and C. Barakat, in Proc. ACM SIGCOMM HotSDN workshop, August 2014.

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Treat the network as a black box

See the network as a black box [NST+14, NSB+15] so the operator

Networks do not have infinite resources

- sees it as a system with infinite resources (like a computer for an application).

[NST+14] Optimizing rules placement in OpenFlow networks: trading routing for better efficiency, X. N. Nguyen, D. Saucez, T. Turletti, and C. Barakat, in Proc. ACM SIGCOMM HotSDN workshop, August 2014.

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Anatomy of a flow table

A flow table is a partially ordered set of rules

A rule is a tuple composed of

- a predicate to define equivalence classes (i.e., flows)
- an action to be applied on every packet of the same class
- a priority to provide ordering

Predicate	Action	Priority
IP.destination = bob ^ tcp.destination_port = HTTP	forward to West	10
TRUE	drop	0

Flow tables are too small

Rule space is large, $\mathcal{O}(10^9)$,

- because of the flexibility offered by OpenFlow.

Flow table size on COTS is small, $\mathcal{O}(10^4)$,

- because TCAM is expensive and power hungry.

Our objective

Let the network auto(-magically) construct forwarding tables so to maximise network utility under resource constraints.

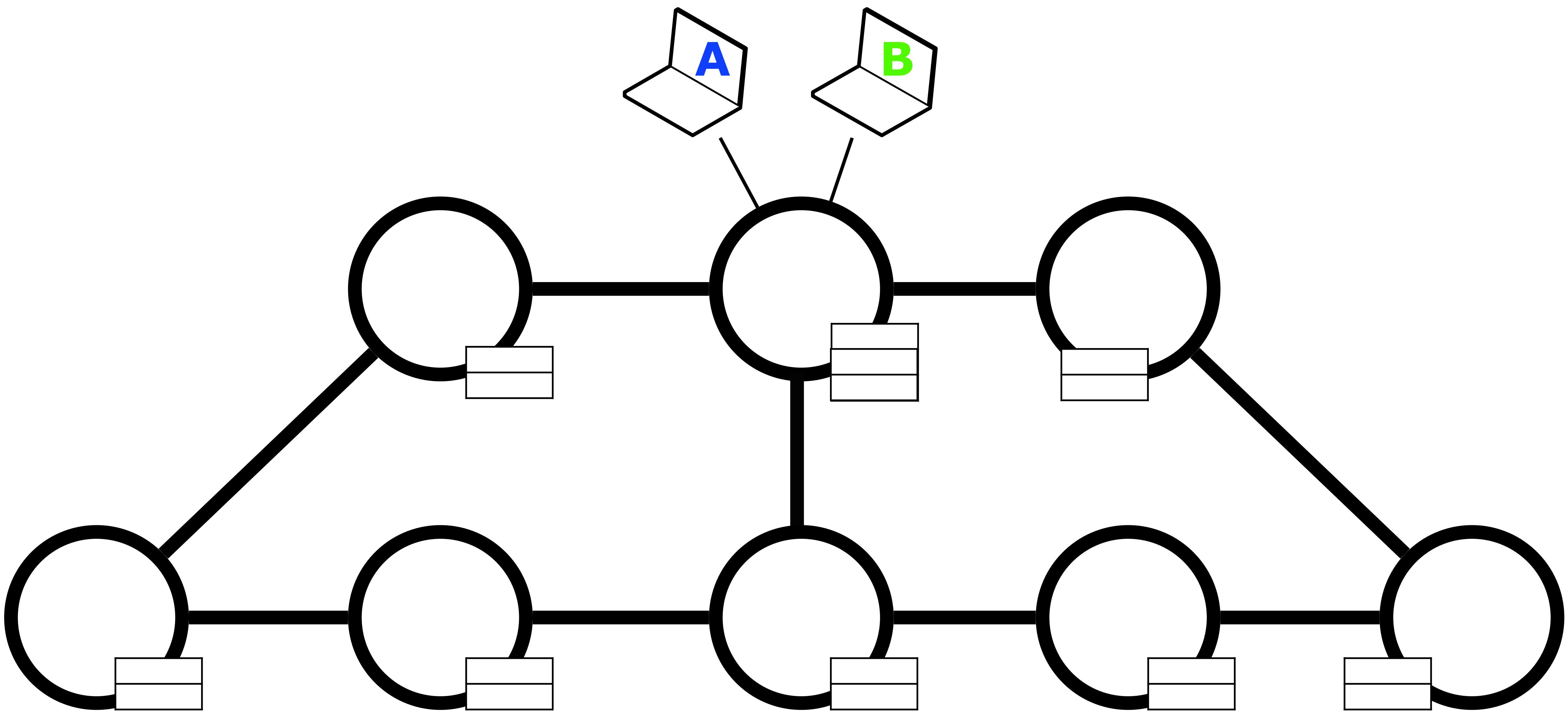
[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Our objective

Finding the optimal is *unrealistic (NP-hard)*

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

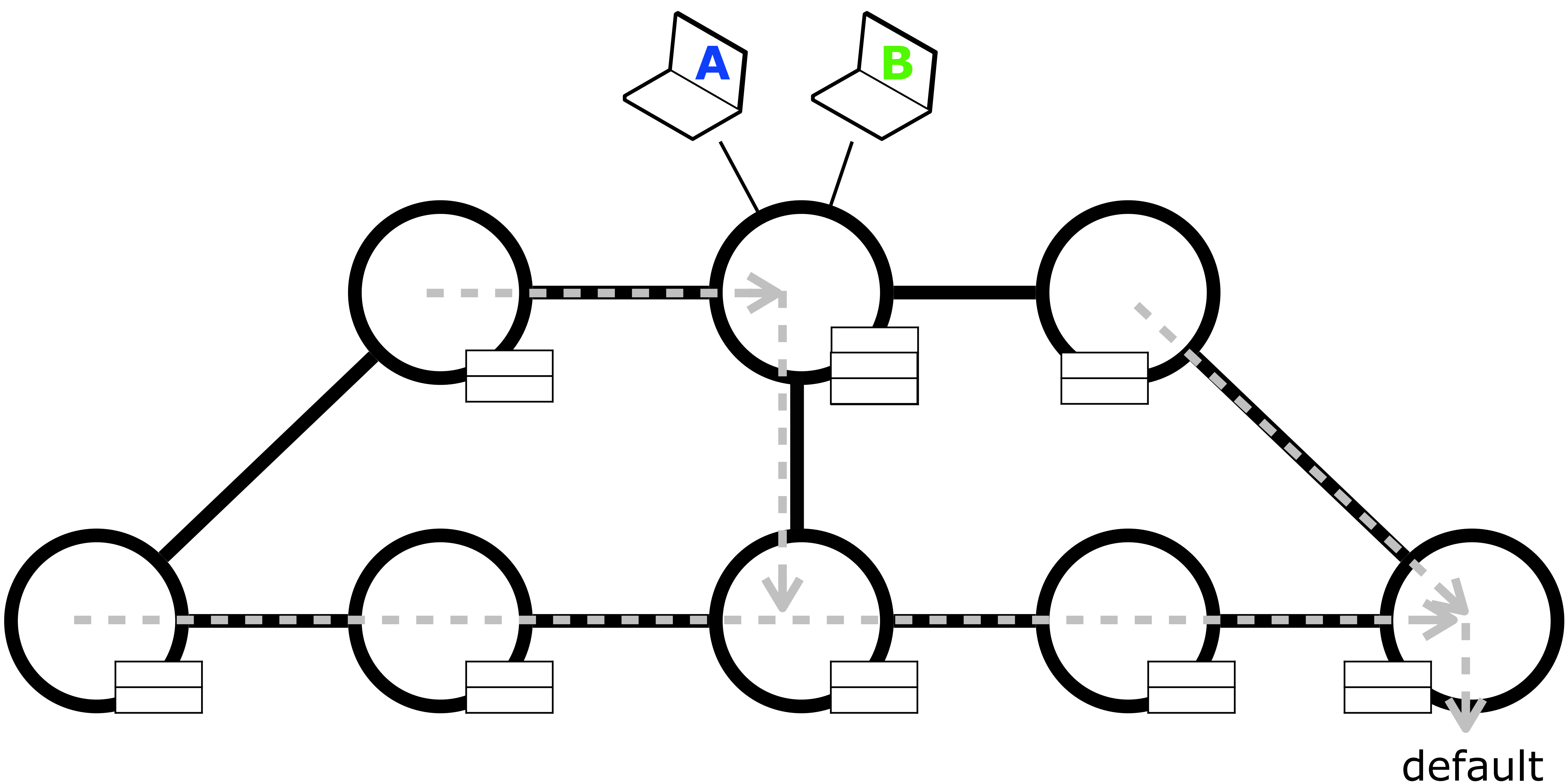
Leverage default operations



default path

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

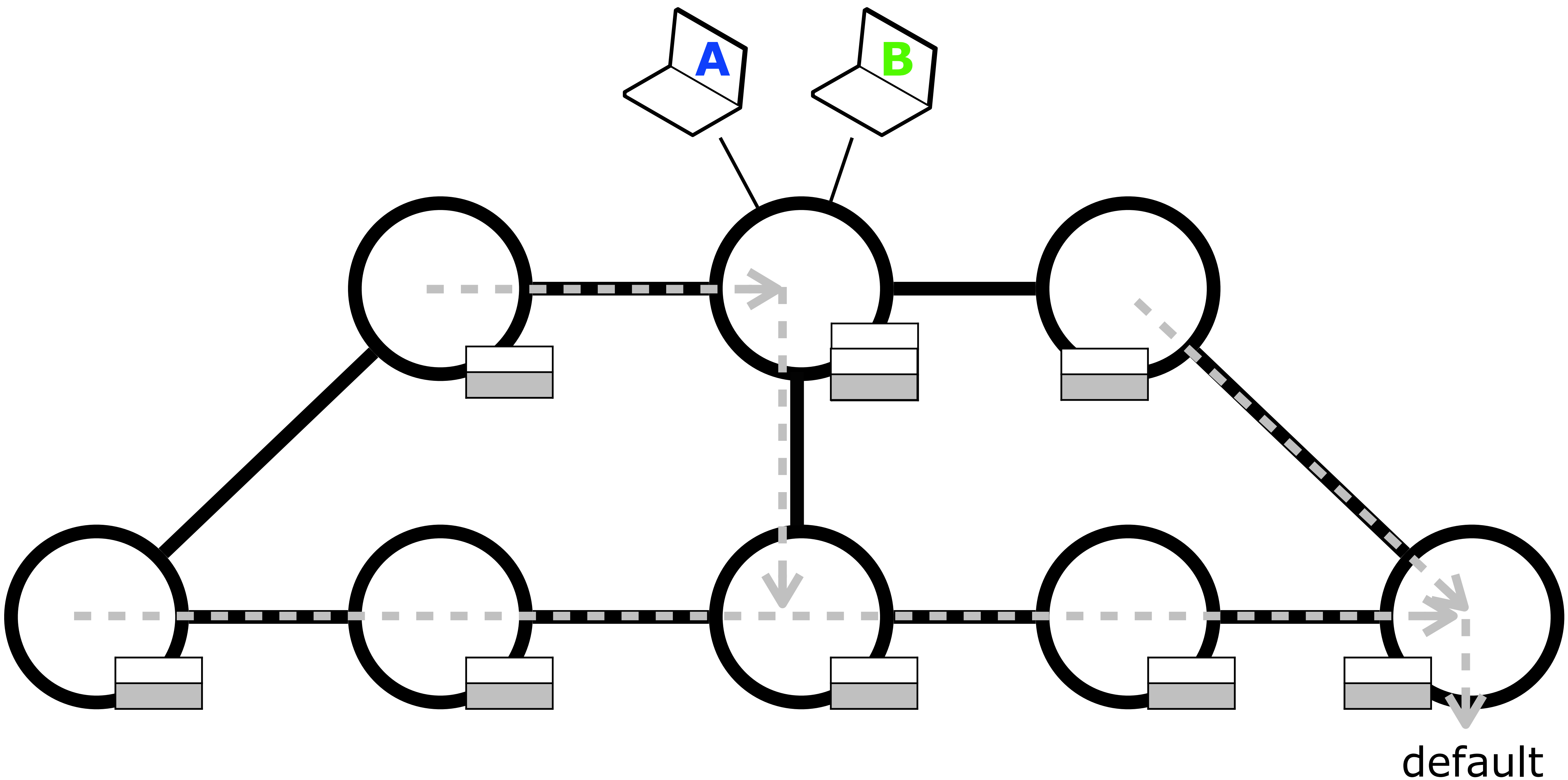
Leverage default operations



default path

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

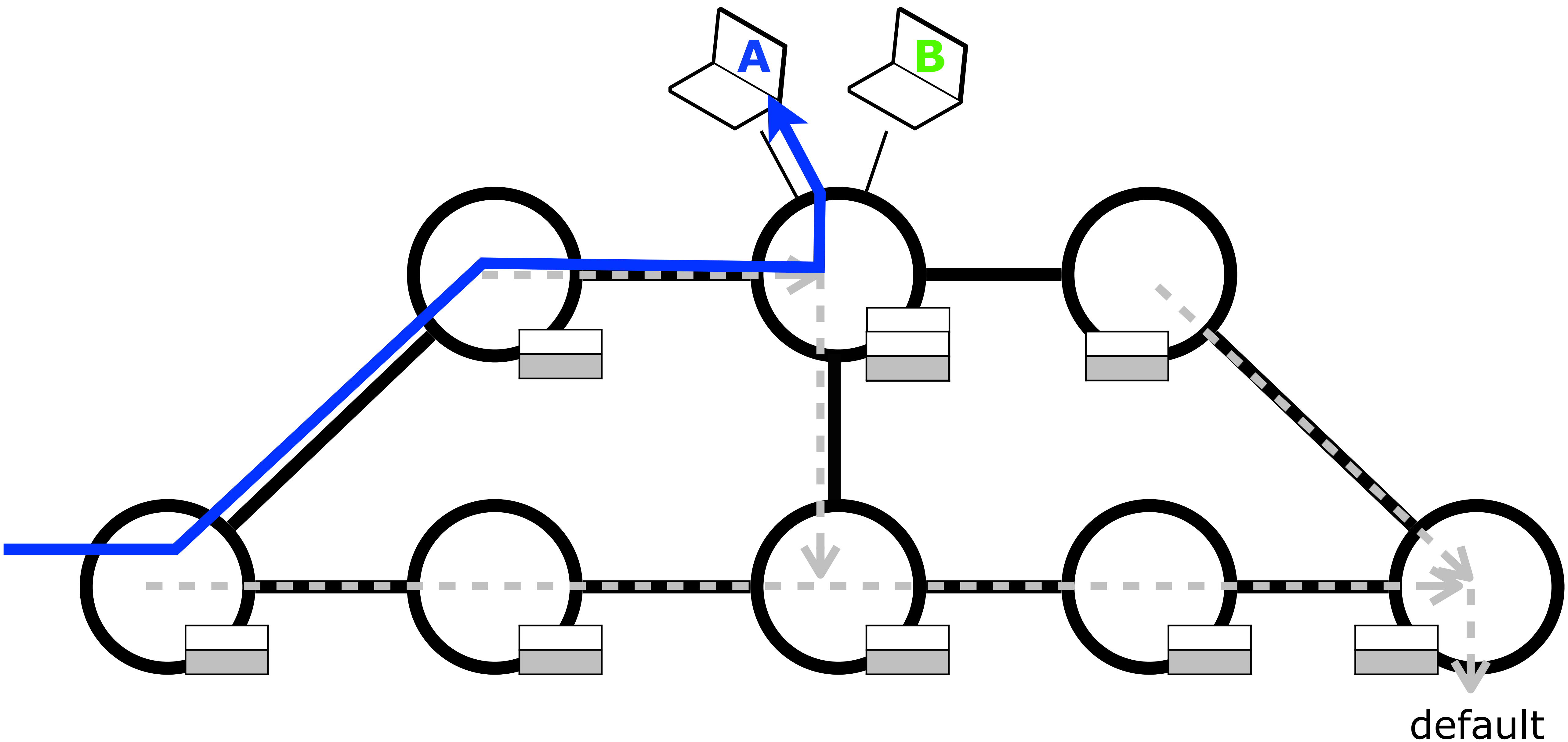
Leverage default operations



default path

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

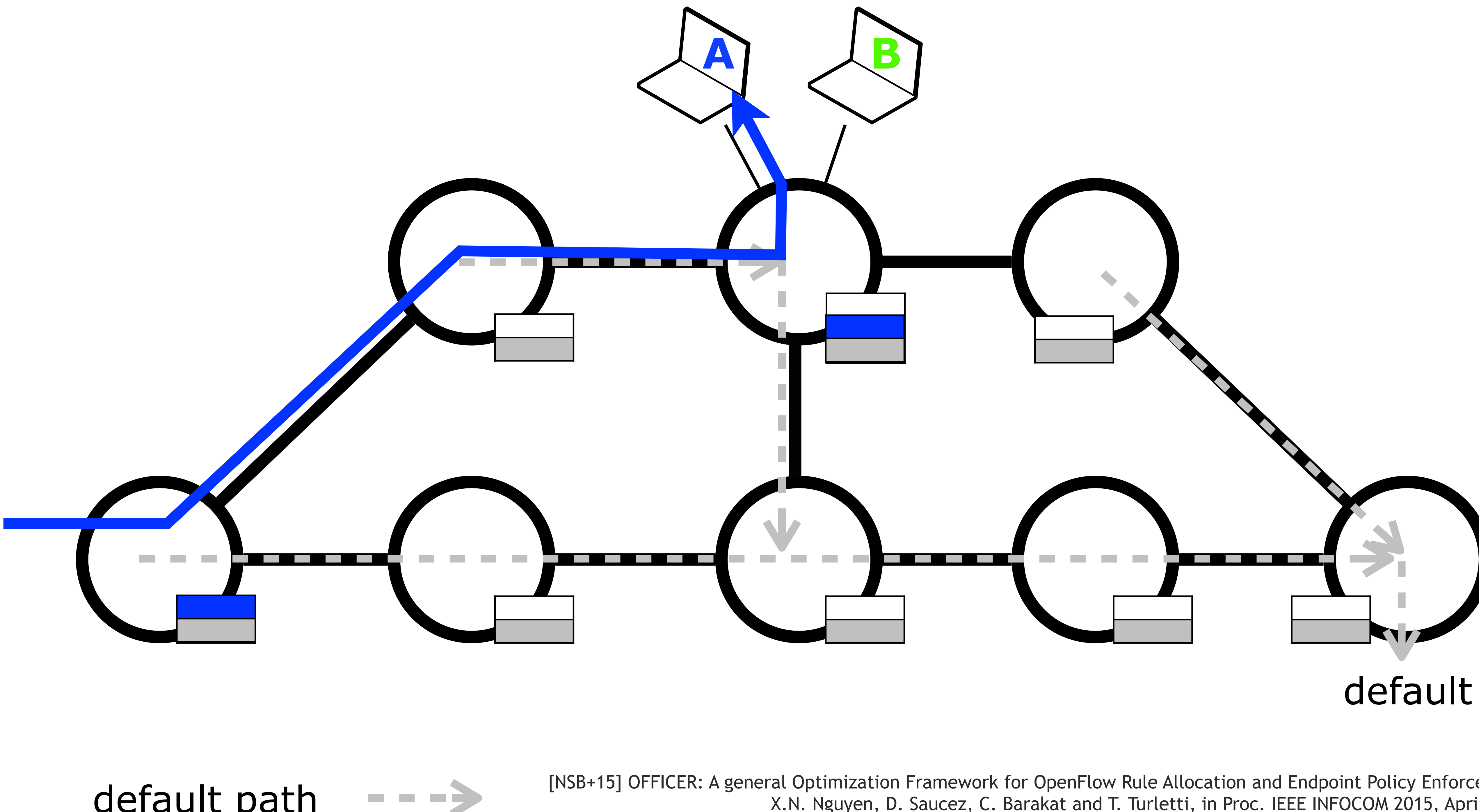
Leverage default operations



default path

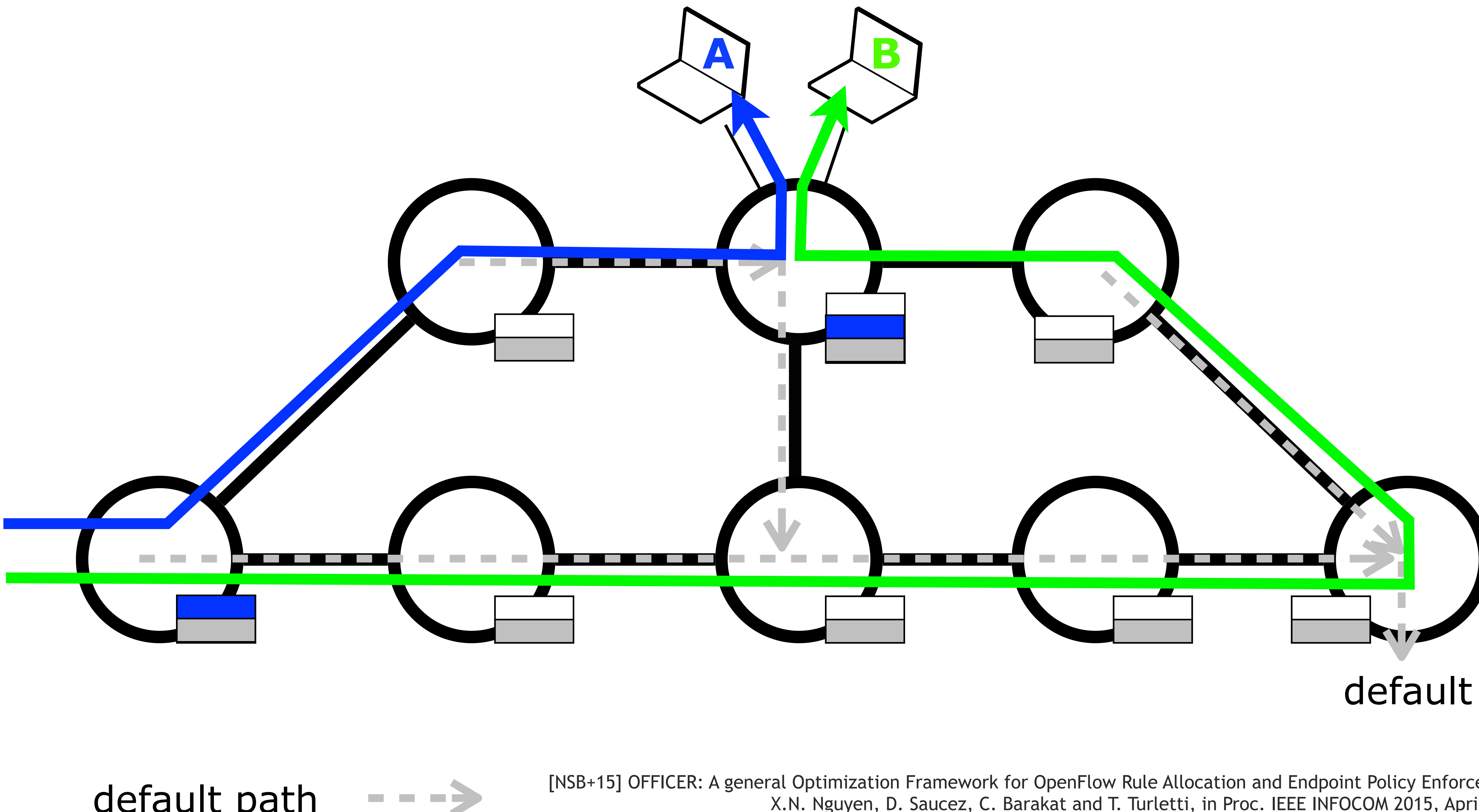
[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Leverage default operations



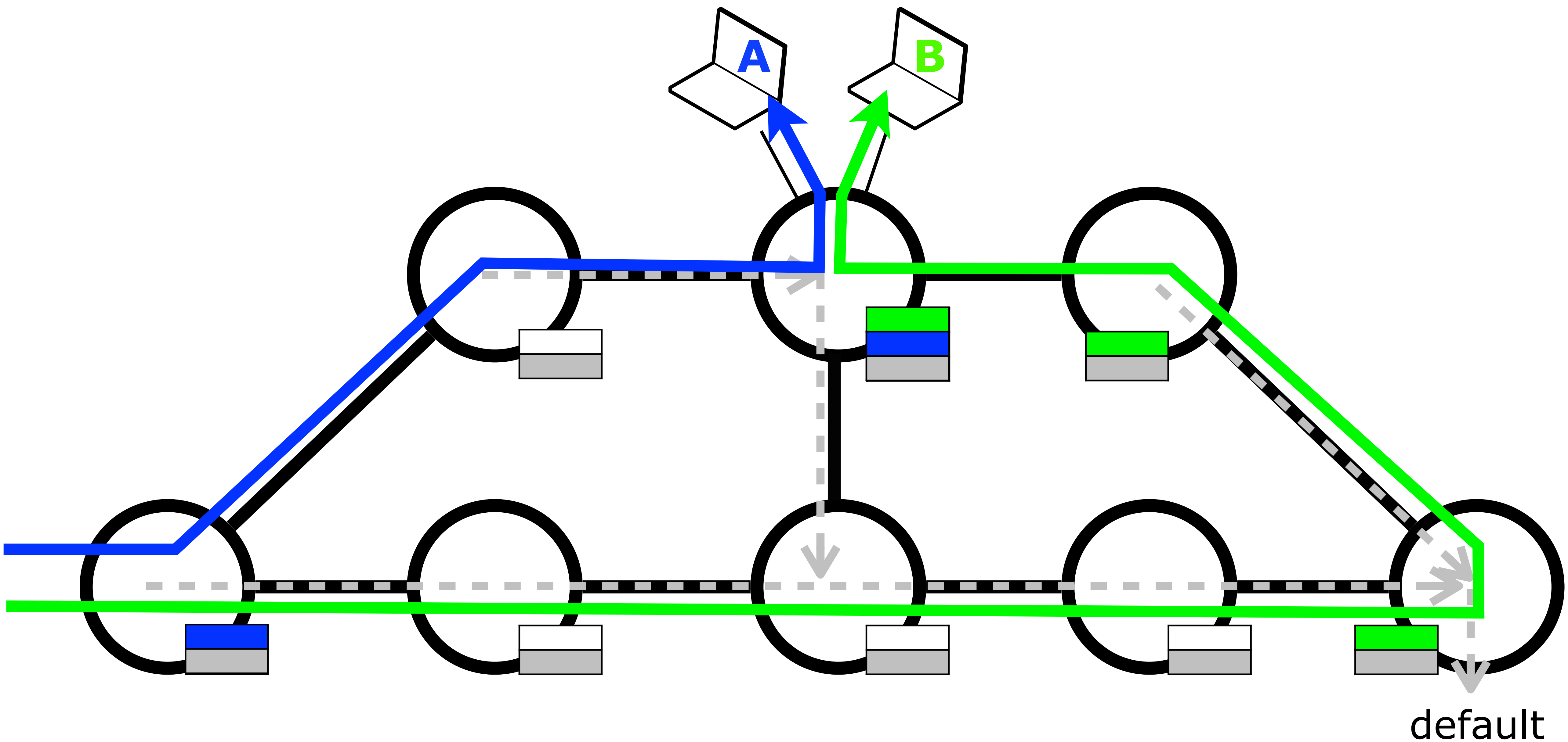
[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Leverage default operations



[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Leverage default operations



default path

[NSB+15] OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement, X.N. Nguyen, D. Saucez, C. Barakat and T. Turletti, in Proc. IEEE INFOCOM 2015, April 2015.

Let’s be real…

Flow tables are large enough but…

the workload is unknown:

- unknown distributions (size, inter-arrival…),
- non-stationary processes.

Let’s be real…

Flow tables are large enough but…

the workload is unknown:

Offline optimisation is impossible

- non-stationary processes.

Where is the problem?

Switches are good only at switching.

Control-plane is the real bottleneck:

- installation time >>> packets inter arrival time,
- controller treatment rate is bounded.

Where is the problem?

Switches are good only at switching.

Limit the number of requests to the controller

- controller treatment rate is bounded.

1st approach

1st approach

Maximum load on controller: $c \in [0; 1]$

1st approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow at epoch t ? : $u^t \in \{0,1\}$

1st approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow at epoch t ? : $u^t \in \{0,1\}$

Model controller load with a queue:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

1st approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow at epoch t ? : $u^t \in \{0,1\}$

Model controller load with a queue:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

Reward for optimising flow at epoch t: r^t

1st approach

Maximum load on controller: $c \in [0; 1]$

Use the controller for flow at epoch t ? : $u^t \in \{0, 1\}$

Model controller load with a queue:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

Reward for optimising flow at epoch t: r^t

Use controller if

$$Q(t) \leq V \cdot r^t$$

Math to networking: the wrong way

Remember:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$
$$Q(t) \leq V \cdot r^t$$

Easy:

- two sums,
- one comparison.

Math to networking: the wrong way

Remember:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

A switch can't do that

- two sums,
- one comparison.

Math to networking: the right way

Remember:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

$$Q(t) \leq V \cdot r^t$$

Easy:

- Looks like a leaky bucket.

$$B(k+1) = \min[B(k) - a(k) + \bar{a}, MAX]$$

$$a(k) = d(k) \leq B(k)$$

Math to networking: the right way

Remember:

$$Q(t+1) = \max\left[Q(t) + u^t - c, 0\right]$$

A switch should be able to do that

$$B(k+1) = \min[B(k) - a(k) + \bar{a}, MAX]$$

$$a(k) = d(k) \leq B(k)$$

Switches are just pipelines of match-action tables

Frame parsing

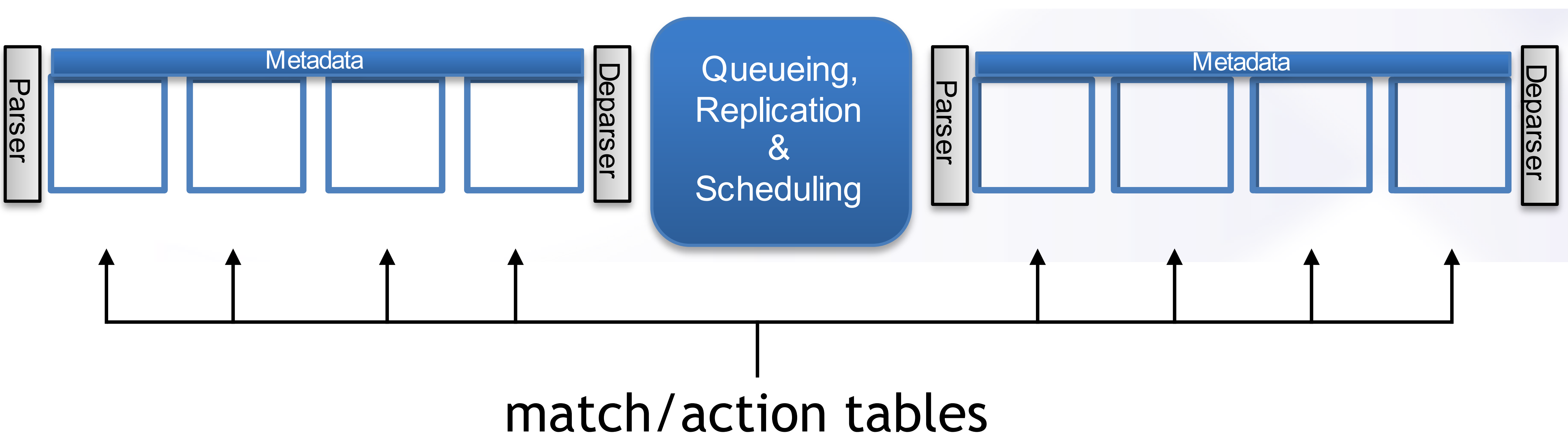
Match-action pipelines

[BDG+14] P4: Programming Protocol-Independent Packet Processors, P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, ACM Sigcomm Computer Communications Review (CCR). Volume 44, Issue #3, July 2014.

Switches are just pipelines of match-action tables

Frame parsing

Match-action pipelines

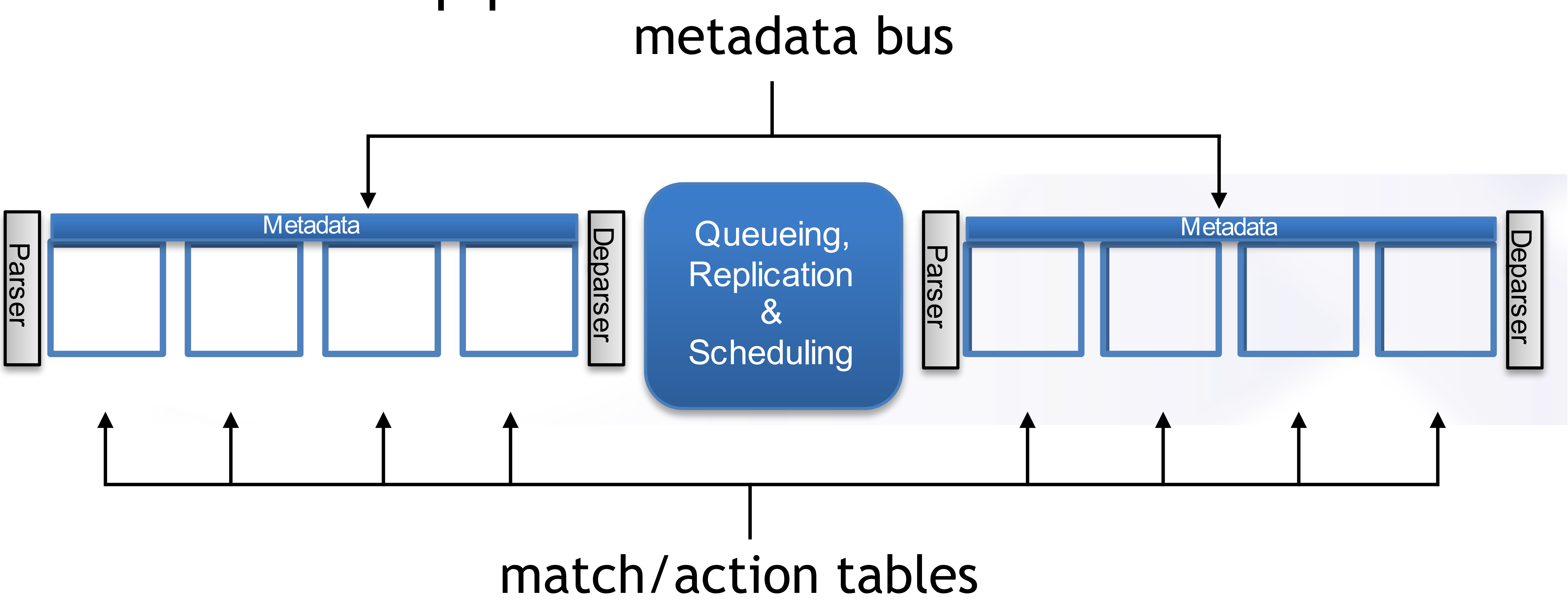


[BDG+14] P4: Programming Protocol-Independent Packet Processors, P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, ACM Sigcomm Computer Communications Review (CCR). Volume 44, Issue #3, July 2014.

Switches are just pipelines of match-action tables

Frame parsing

Match-action pipelines



[BDG+14] P4: Programming Protocol-Independent Packet Processors, P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, ACM Sigcomm Computer Communications Review (CCR). Volume 44, Issue #3, July 2014.

Drift-plus-penalty Workflow

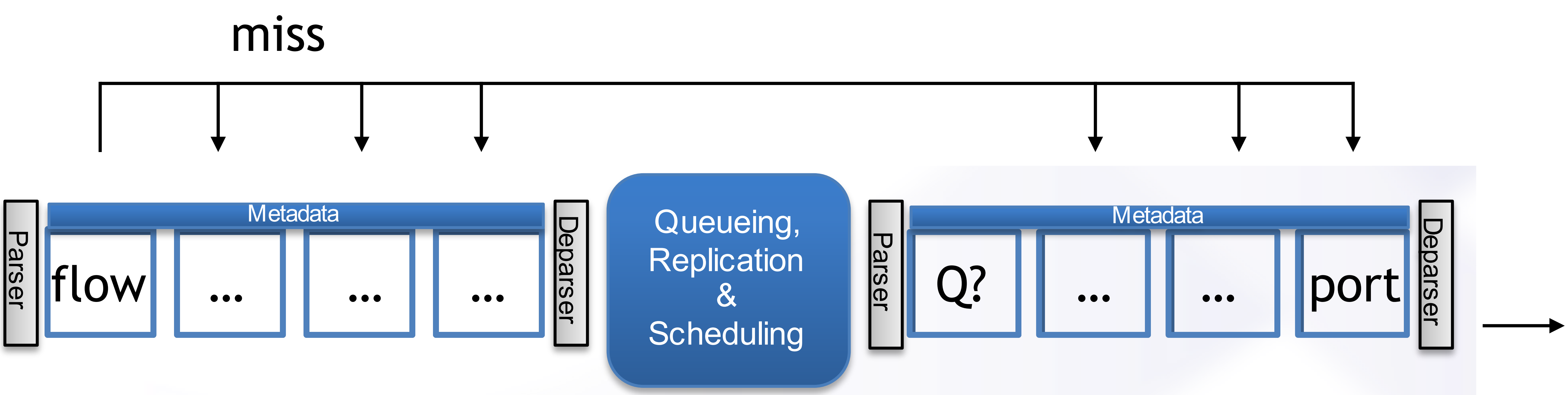
slow path

fast path

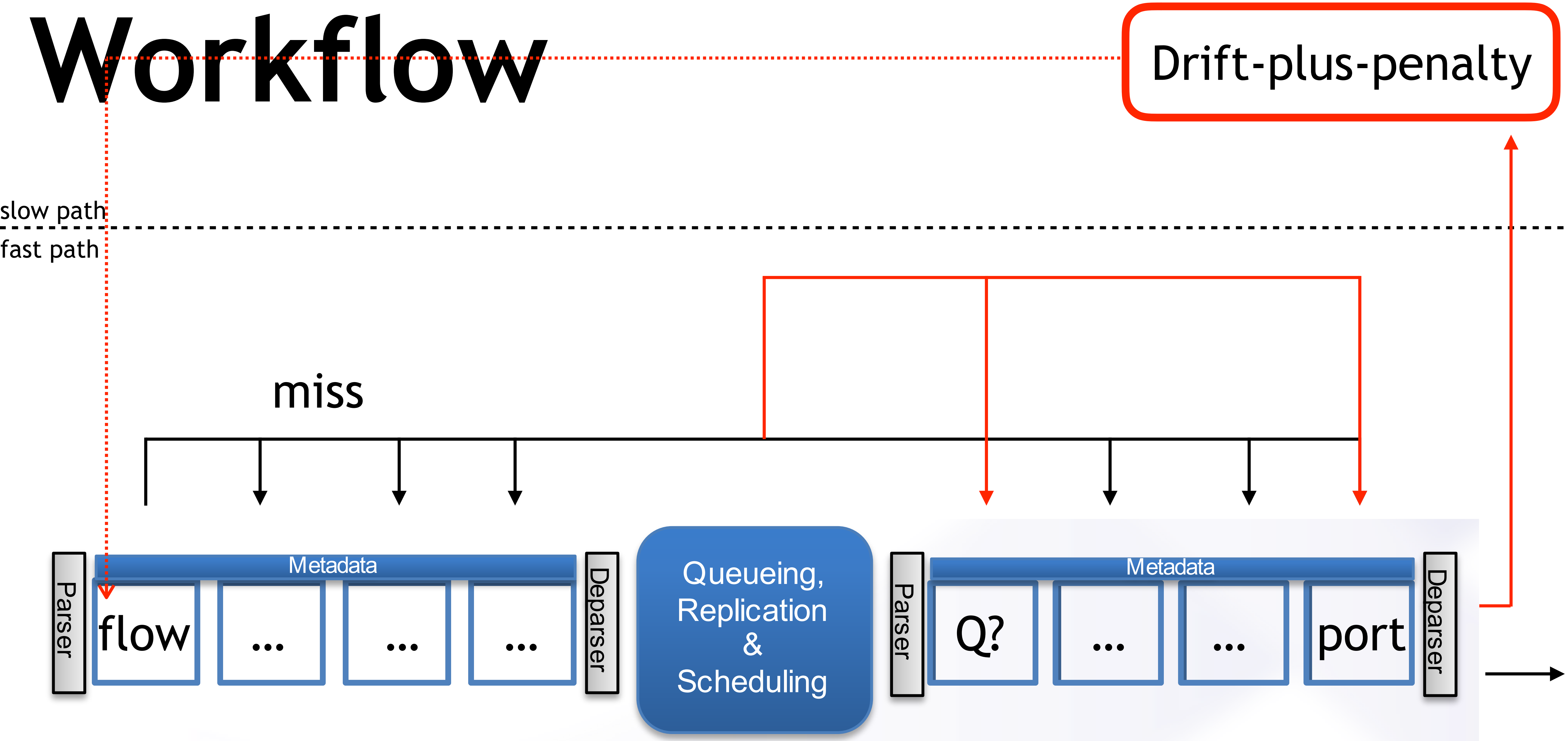
Drift-plus-penalty Workflow

slow path

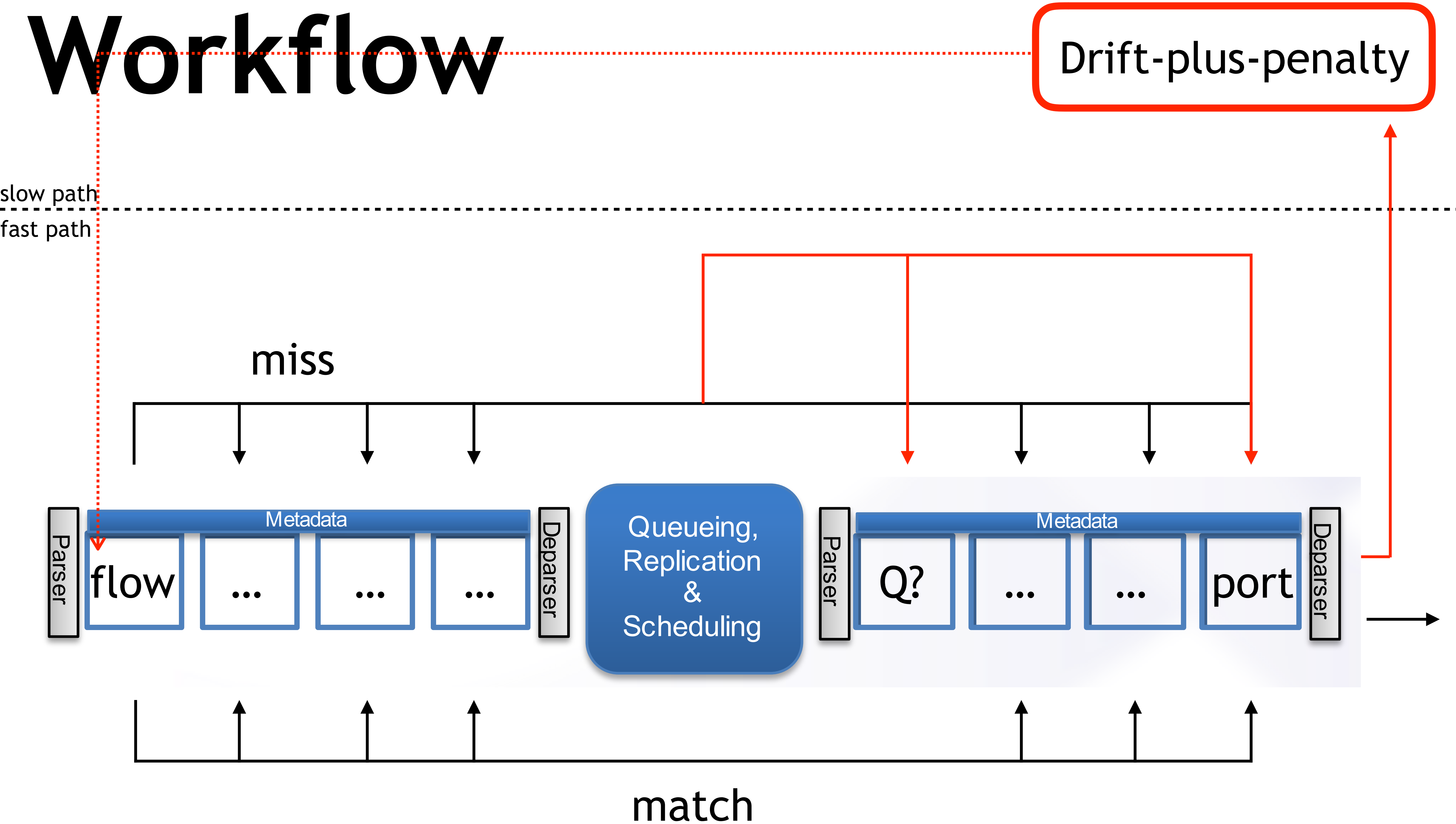
fast path



Drift-plus-penalty Workflow



Drift-plus-penalty Workflow



Implementation with the P4 DSL

```
table flow_table {
    reads {
        ...
    }
    actions {
        add_flow;
        ...
    }
    ...
}
```

Implementation with the P4 DSL

```
table flow_table {
    reads {
        ...
    }
    actions {
        add_flow;
        ...
    }
    ...
}

action add_flow() {
        clone_ingress_pkt_to_egress(250, copy_to_cpu_fields);
}
```

Not 100% implementable on the fast path

To implement our drift-plus-penalty we need:

- to compute Q (with a leaky bucket),
- to translate epoch in rate (no distribution knowledge),
- to remember rejected flows (update tables on the fly).

Not 100% implementable on the fast path

To implement our drift-plus-penalty we need:

- to compute Q (with a leaky bucket)

Need to find another way

- to remember rejected flows (update tables on the fly).

2nd approach

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

2nd approach

Maximum load on controller: $c \in [0; 1]$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

2nd approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow k? : $u^k \in \{0,1\}$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

2nd approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow k? : $u^k \in \{0,1\}$

Limit controller load

$$\limsup_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[u^k\right] \leq c$$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

2nd approach

Maximum load on controller: $c \in [0;1]$

Use the controller for flow k? : $u^k \in \{0,1\}$

Limit controller load

$$\limsup_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[u^k\right] \leq c$$

Reward for optimising flow k: r^k

$$\max_{u} \limsup_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[u^k r^k\right]$$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Optimal strategy

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Optimal strategy

Group flows in ranked classes

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Optimal strategy

Group flows in ranked classes

Use the controller for class k ? : $u_k \in [0; 1]$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Optimal strategy

Group flows in ranked classes

Use the controller for class k ? : $u_k \in [0;1]$

Threshold-based optimal:

$$u_j(\alpha^*) = \begin{cases} 1 & j \leq \lfloor \alpha^* \rfloor \\ \alpha^* - \lfloor \alpha^* \rfloor & j = \lfloor \alpha^* \rfloor + 1 \\ 0 & j \geq \lfloor \alpha^* \rfloor + 2 \end{cases}$$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Optimal strategy

Group flows in ranked classes

Use the controller for class k ? : $u_k \in [0;1]$

Threshold-based optimal:

$$u_j(\alpha^*) = \begin{cases} 1 & j \leq \lfloor \alpha^* \rfloor \\ \alpha^* - \lfloor \alpha^* \rfloor & j = \lfloor \alpha^* \rfloor + 1 \\ 0 & j \geq \lfloor \alpha^* \rfloor + 2 \end{cases}$$

For α^* a solution of

$$\sum_{j=1}^{\lfloor \alpha \rfloor} p_j + (\alpha - \lfloor \alpha \rfloor) \cdot p_{\lfloor \alpha \rfloor + 1} = c$$

[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Math to networking: the right way

Easy:

- Estimate class probabilities
- Install 4 OpenFlow rules with priority 0

  One for classes $\leq \lfloor \alpha^* \rfloor$

  One for classes $\geq \lfloor \alpha^* \rfloor + 2$

  Two for class $\lfloor \alpha^* \rfloor + 1$ with complementary weights

Math to networking: the right way

Easy:

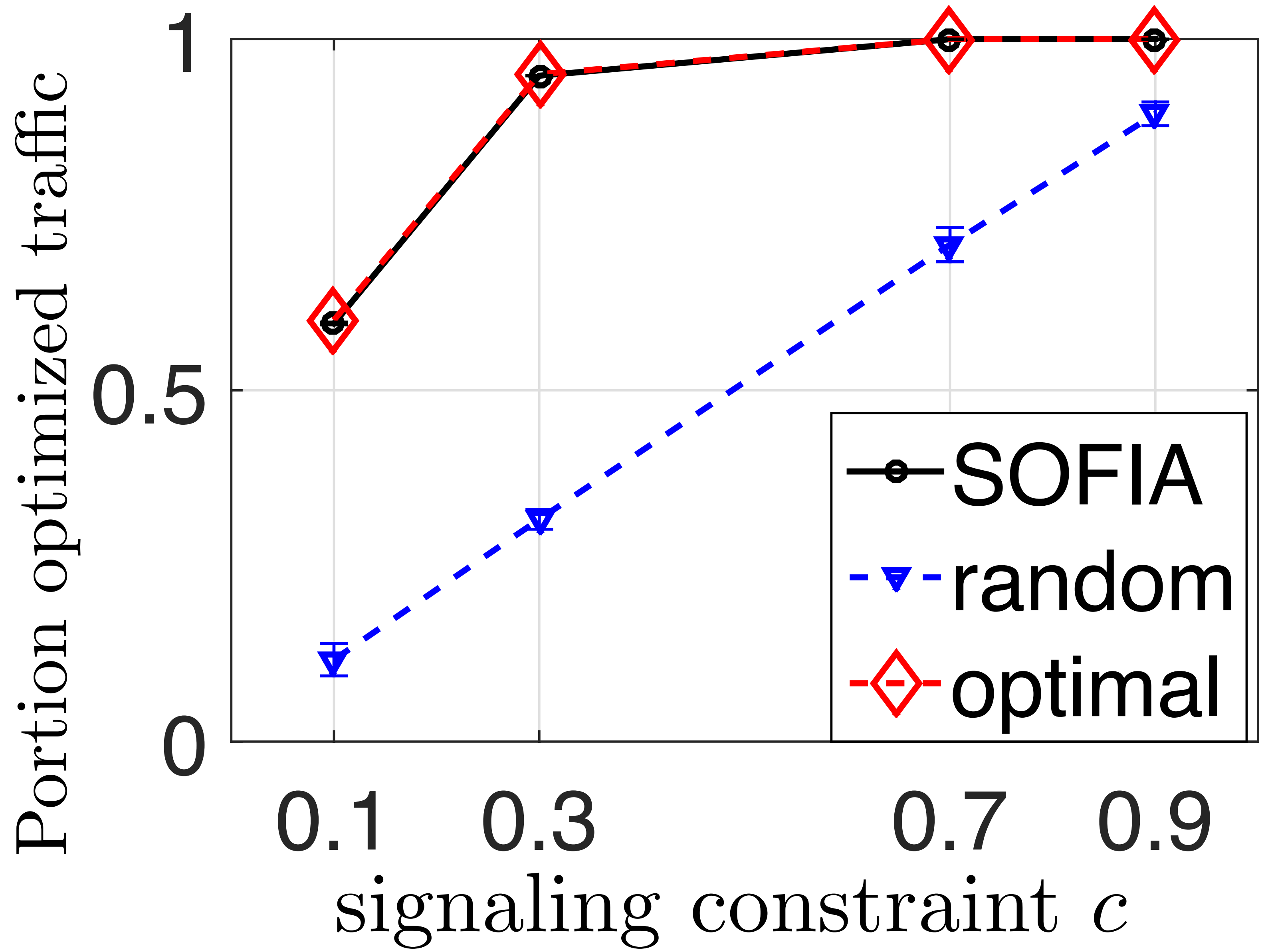
- Estimate class probabilities

It works!

One for classes $\geq \lfloor \alpha^* \rfloor + 2$

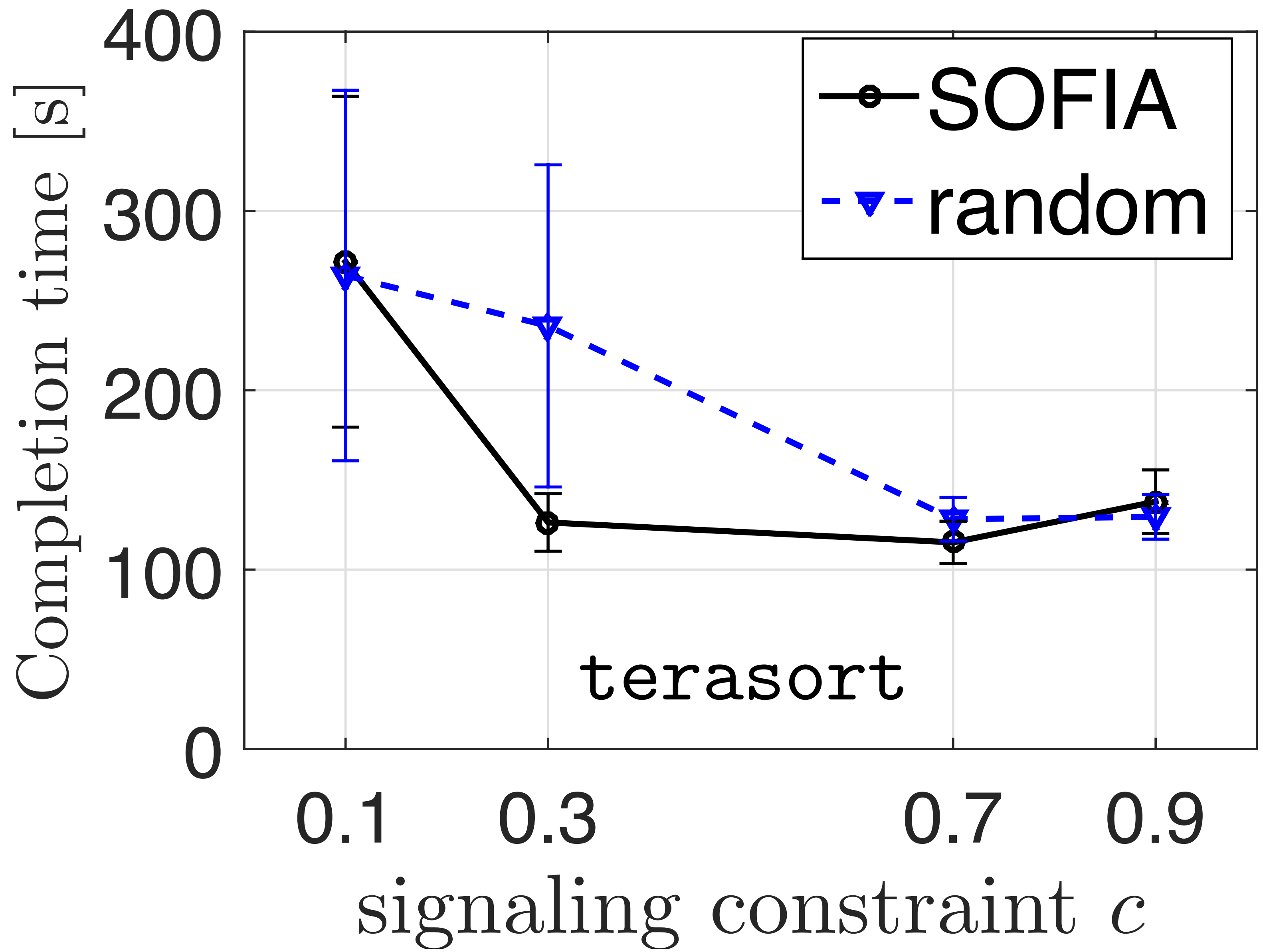
Two for class $\lfloor \alpha^* \rfloor + 1$ with complementary weights

In a 4 nodes Hadoop cluster



[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

In a 4 nodes Hadoop cluster



[DMM+18] Blind, Adaptive and Robust Flow Segmentation in Datacenters, F. De Pellegrini, L. Maggi, A. Massaro, D. Saucez, J. Leguay, E. Altman, in Proc. IEEE INFOCOM 2018, April 2018.

Take away message

“Theoretical” and “practical” knowledges need each other

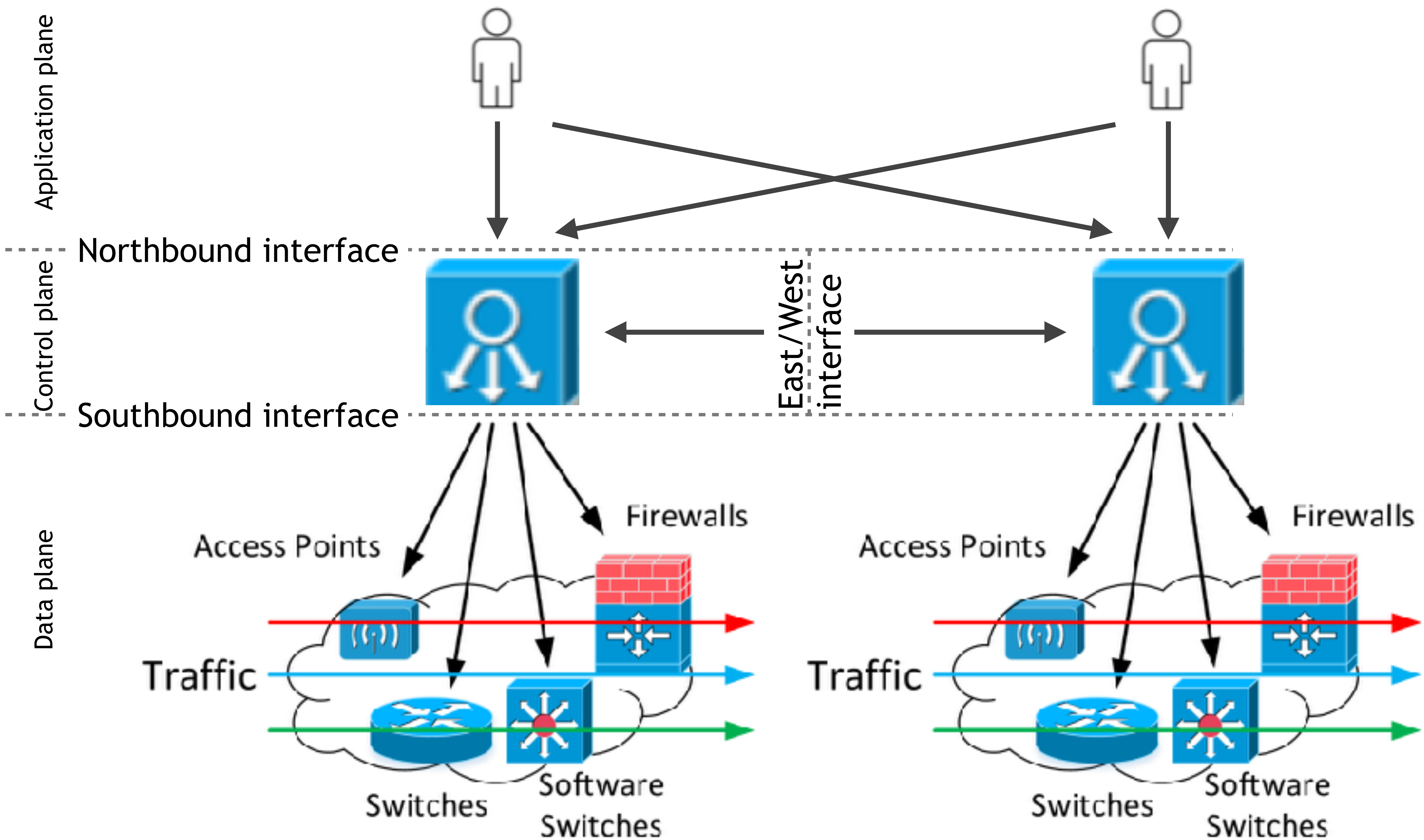
Work and exchange your thoughts with the specialists.

Network's Adventures in Softwar'land

Damien Saucez

January 2018

APIs to program the network



OpenFlow to separate roles

Programmability of network is reached by decoupling control plane from data plane in OpenFlow:

- network elements are elementary switches,
- the intelligence is implemented by a logically centralised controller

  that manages the switches (i.e., install forwarding rules).

Fast-path workflow

```
control ingress {
    apply(flow_table);

    if (super_meta.fast != 1){
       // L2 switch
       apply(mac_table);        // figure out the next port to forward the packet to
    }
}

control egress {
    if (standard_metadata.instance_type == 0){
         apply(no_arp_table);  // do not forward ARP's
    }
    else {
         apply(redirect);
    }
}
```

Flow-table definition

```
table flow_table {
    reads {
        ipv4.srcAddr : exact;
        ipv4.dstAddr : exact;
        ipv4.protocol: exact;
        super_meta.srcPort  : exact;
        super_meta.dstPort  : exact;
    }
    actions {
        _nop;
        _drop;
        add_flow;
        set_fast_forward;
    }
    size: 65535;
}
```

Flow-table actions

```
action add_flow() {
        modify_field(super_meta.fast, 0);
        clone_ingress_pkt_to_egress(250, copy_to_cpu_fields);
}

field_list copy_to_cpu_fields {
        super_meta;
        standard_metadata;
}

action set_fast_forward(iface) {
   modify_field(standard_metadata.egress_spec, iface);
   modify_field(super_meta.fast, 1);
}
```

Redirect table definition

```
table redirect {
    reads {
        standard_metadata.instance_type : exact;
    }
    actions {
        _drop;
        _nop;
        do_cpu_encap;
    }
    size : 16;
}
```

Redirect actions

```
// == Headers for CPU
header cpu_header_t cpu_header;

field_list copy_to_cpu_fields {
        super_meta;
        standard_metadata;
}


action do_cpu_encap() {
        // CPU
        add_header(cpu_header);
        modify_field(cpu_header.etherType, ethernet.etherType);
        modify_field(ethernet.etherType, ETHERTYPE_CPU);
        modify_field(cpu_header.preamble, 0);
        modify_field(cpu_header.if_index, super_meta.ingress_port);
}
```

Implement a new protocol

```
#define ETHERTYPE_CPU 0xDEAD
header_type cpu_header_t {
    fields {
        preamble : 64;
        if_index : 16;
        etherType: 16;
    }
}
```

```
parser parse_cpu_header {
    extract(cpu_header);
    return select(latest.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
```

```
parser parse_ethernet {
    extract(ethernet);
    set_metadata(super_meta.etherType, ethernet.etherType);
    return select(latest.etherType) {
        ETHERTYPE_CPU  : parse_cpu_header;
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
}
```