# Is the future of network software?

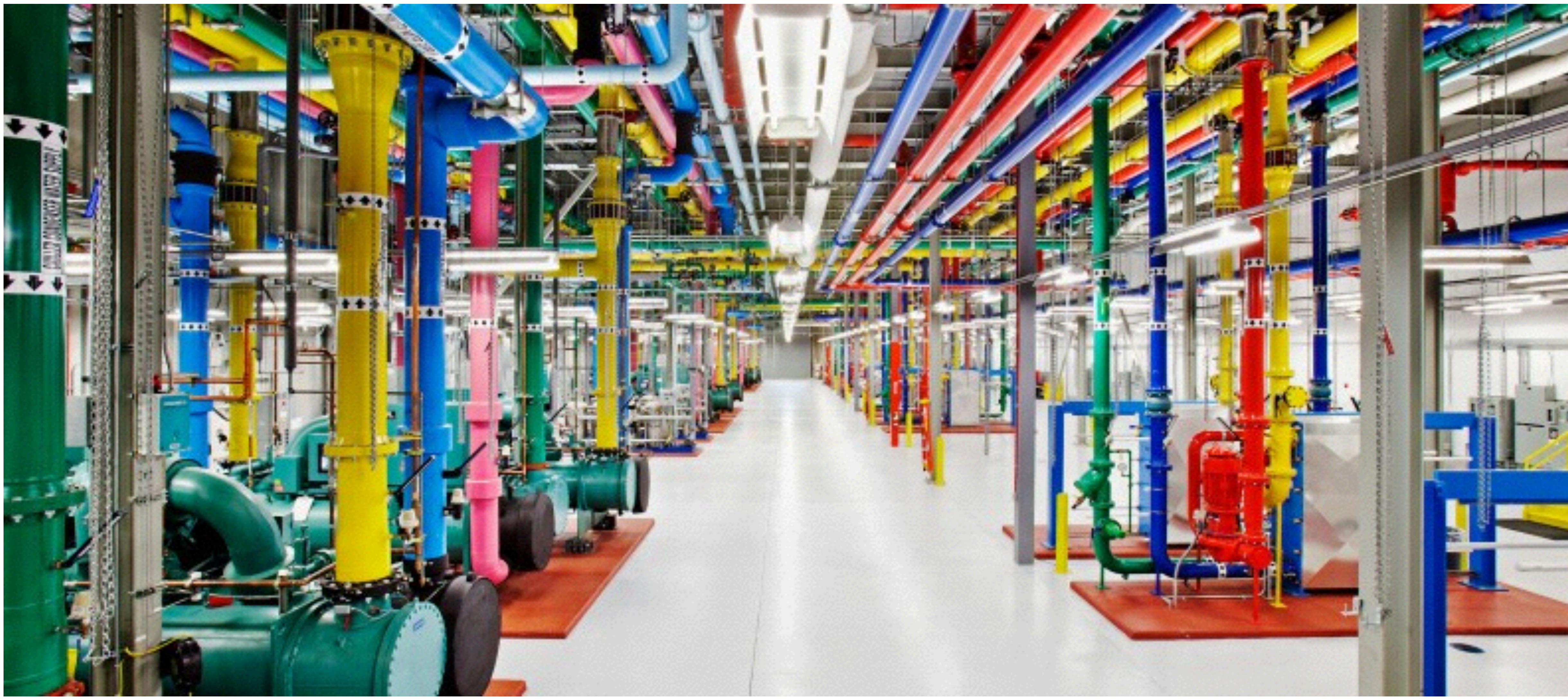
*Damien Saucez*
Inria

March 2015

# Agenda

- Software Defined Networking (SDN)
- The network as a blackbox
- Trade routing for efficiency
- SDN changes the networking ecosystem

# Networking reached an industrial level





[http://wonderfulengineering.com/inside-the-data-center-where-google-stores-all-its-data-pictures/]

# Networks are complex…

- Enterprise and datacenter networks are complex entities because of
  - their scale (tens of thousands of devices, millions of virtual machines, spread around the globe);
  - their feature set (e.g., security, traffic optimisation…);
  - seamless mobility (e.g., smartphones, virtual machines…);
  - management policies (e.g., users must see the same network wherever they are connected, run where electricity is the cheapest).

# Networking technology is at the middle age of CS

- Networks are managed by configuration but
  - each protocol has its own configuration set,
  - each constructor has its own configuration language,
  - it is hard to construct configurations that support all the possible cases.

# Networking technology is at the middle age of CS

- No abstraction is used so the operator needs
  - to know the very details of the topology (e.g., link capacity, IP addresses…),
  - to understand how protocols interact.

# Networking technology is at the middle age of CS

- No abstraction is used so the operator needs

**Yes, as if you implemented everything in assembly language!**

  - to understand how protocols interact.

# Software Defined Networking (SDN)

# Concept of SDN

- SDN conceives the network as a program.
  - Operators do not configure the network, they program it.
  - Operators do not interact directly with devices.
  - Network logic is implemented by humans but network elements are never touched by humans.

# Concept of SDN

- SDN conceives the network as a program.
  - Operators do not configure the network, they

**OpenFlow as an instance of SDN**

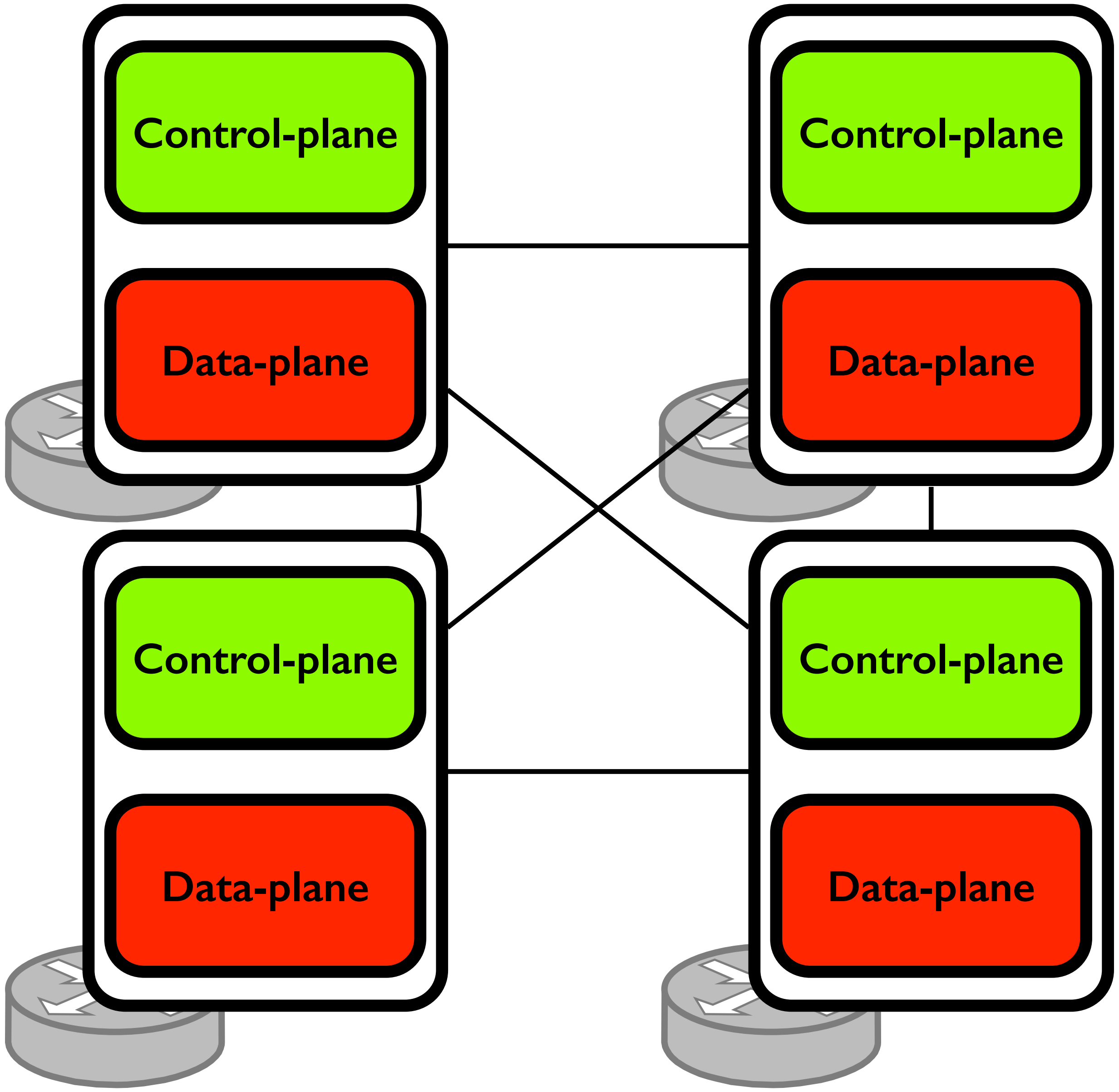
  - Network logic is implemented by human
    network elements are never touched by

# Roles separation

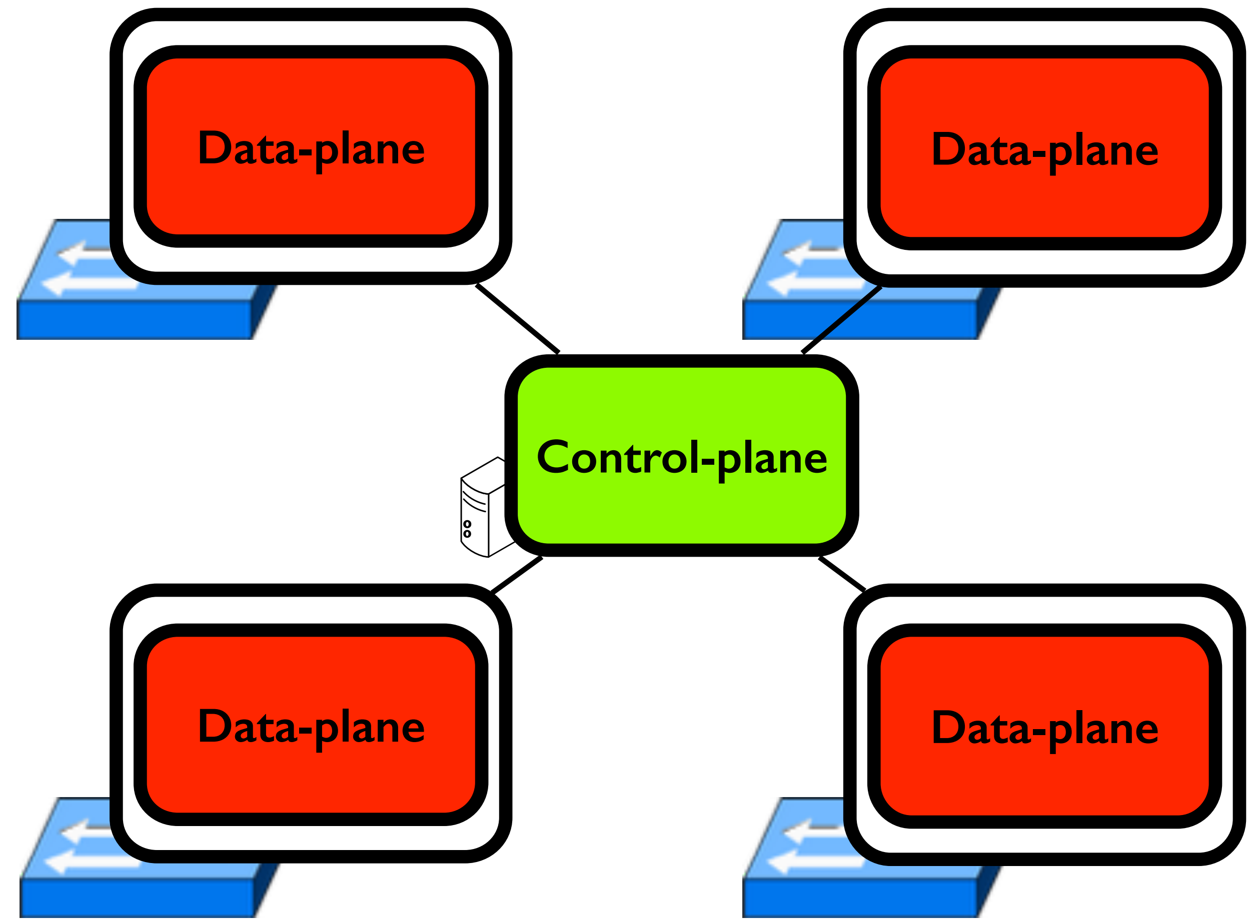
- Programmability of network is reach by decoupling control plane from data plane:
  - network elements are elementary switches,
  - the intelligence is implemented by a logically centralised controller
    - that manages the switches (i.e., install forwarding rules).

# Roles separation

Traditional approach



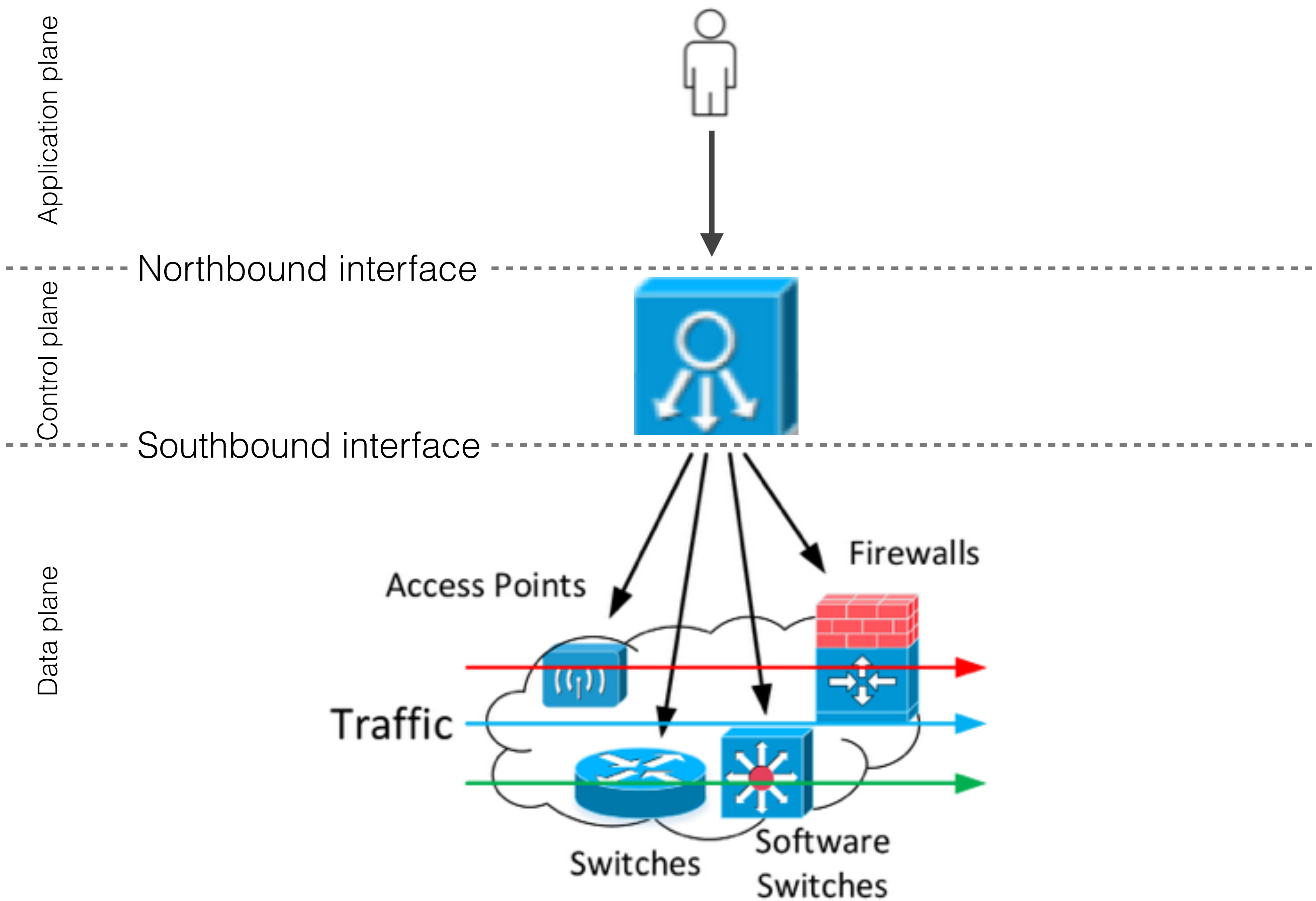

OpenFlow approach



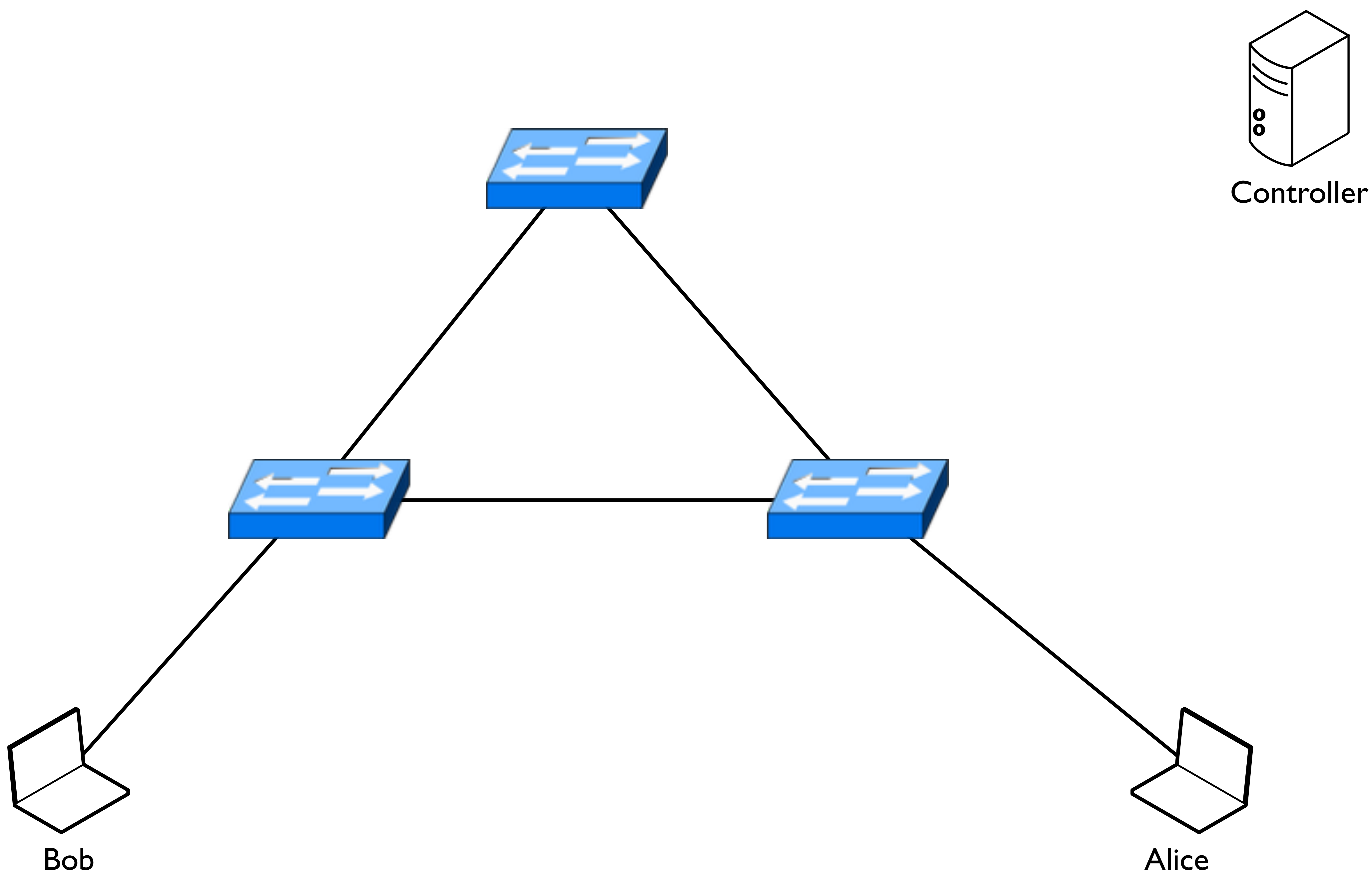
# Cost reduction with COTS

- Data-plane devices only perform forwarding:
  - simple memory structures,
  - simple instruction set,
  - easy virtualisation.
- The control plane runs on x86.
- No vendor lock-in.

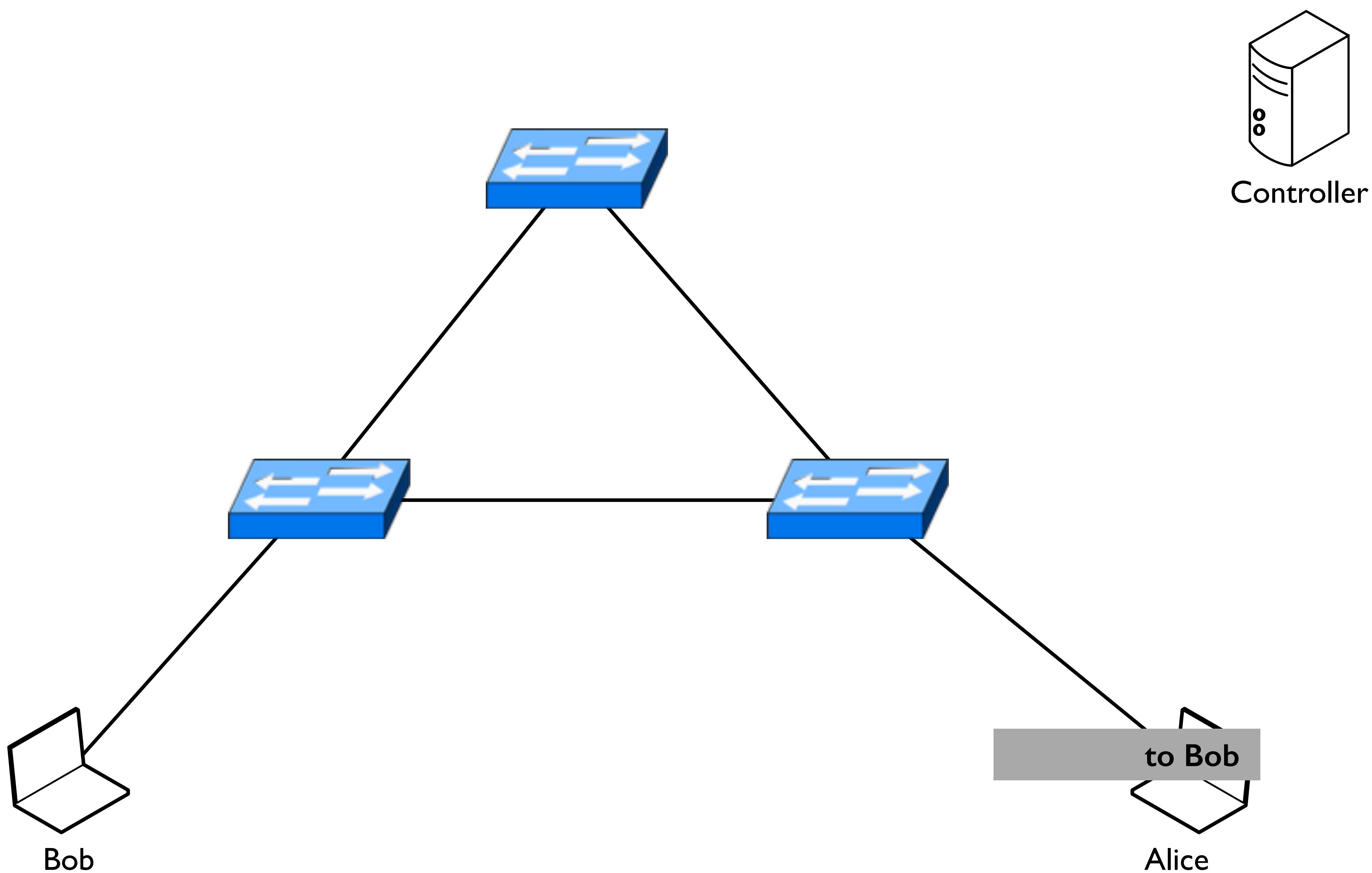
# An API to program the network



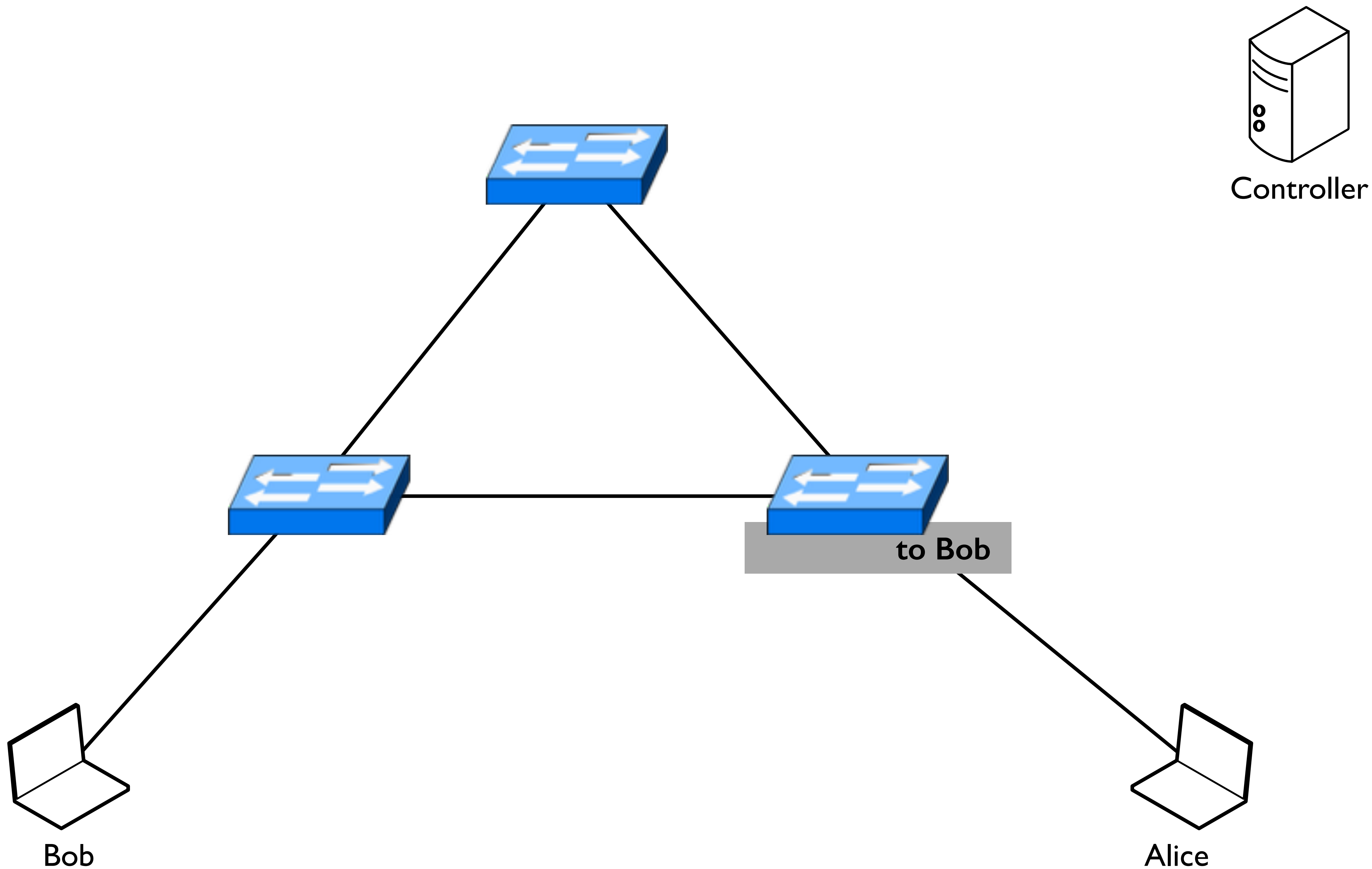
# Southbound interface with OpenFlow



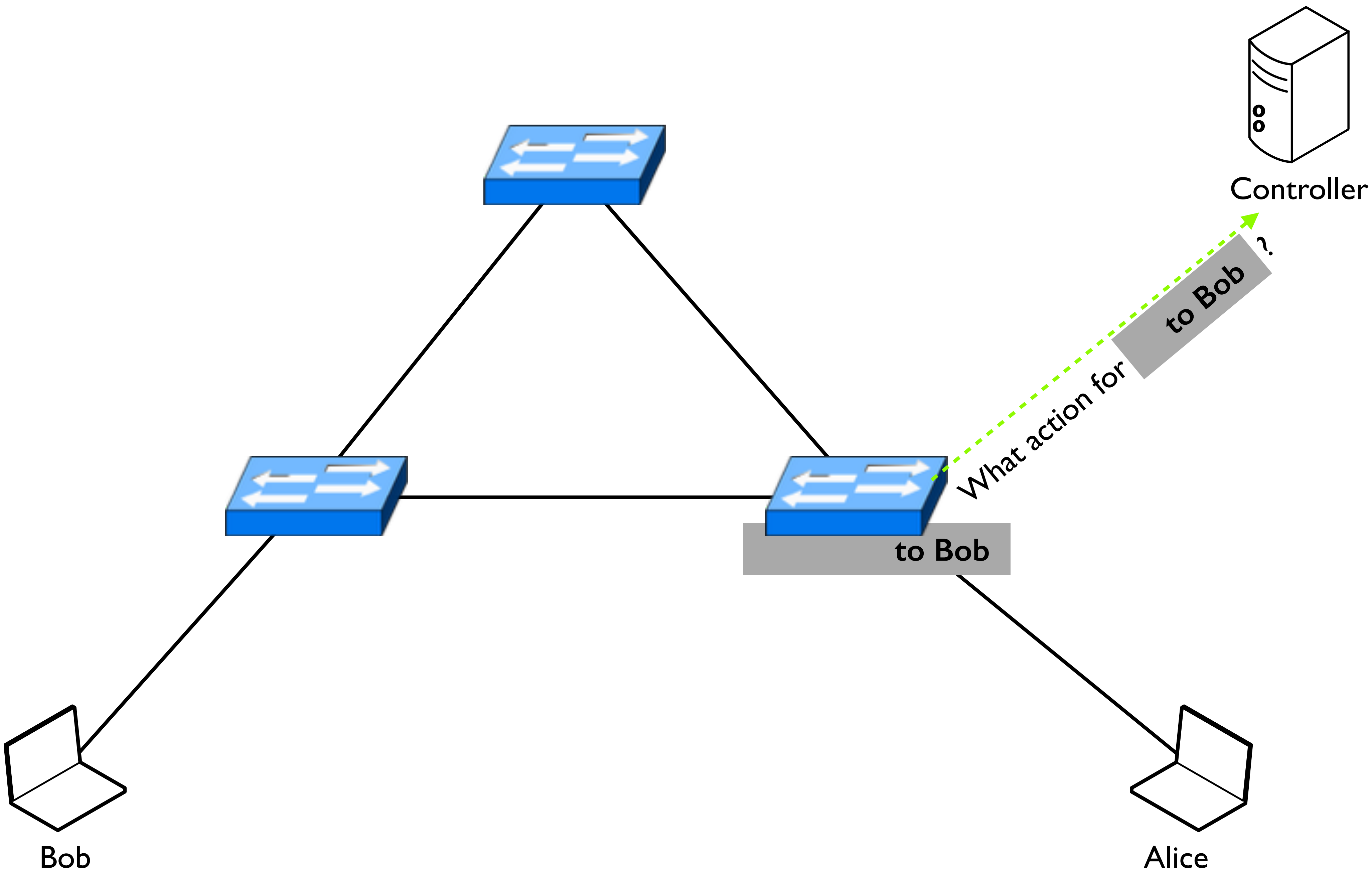
# Southbound interface with OpenFlow



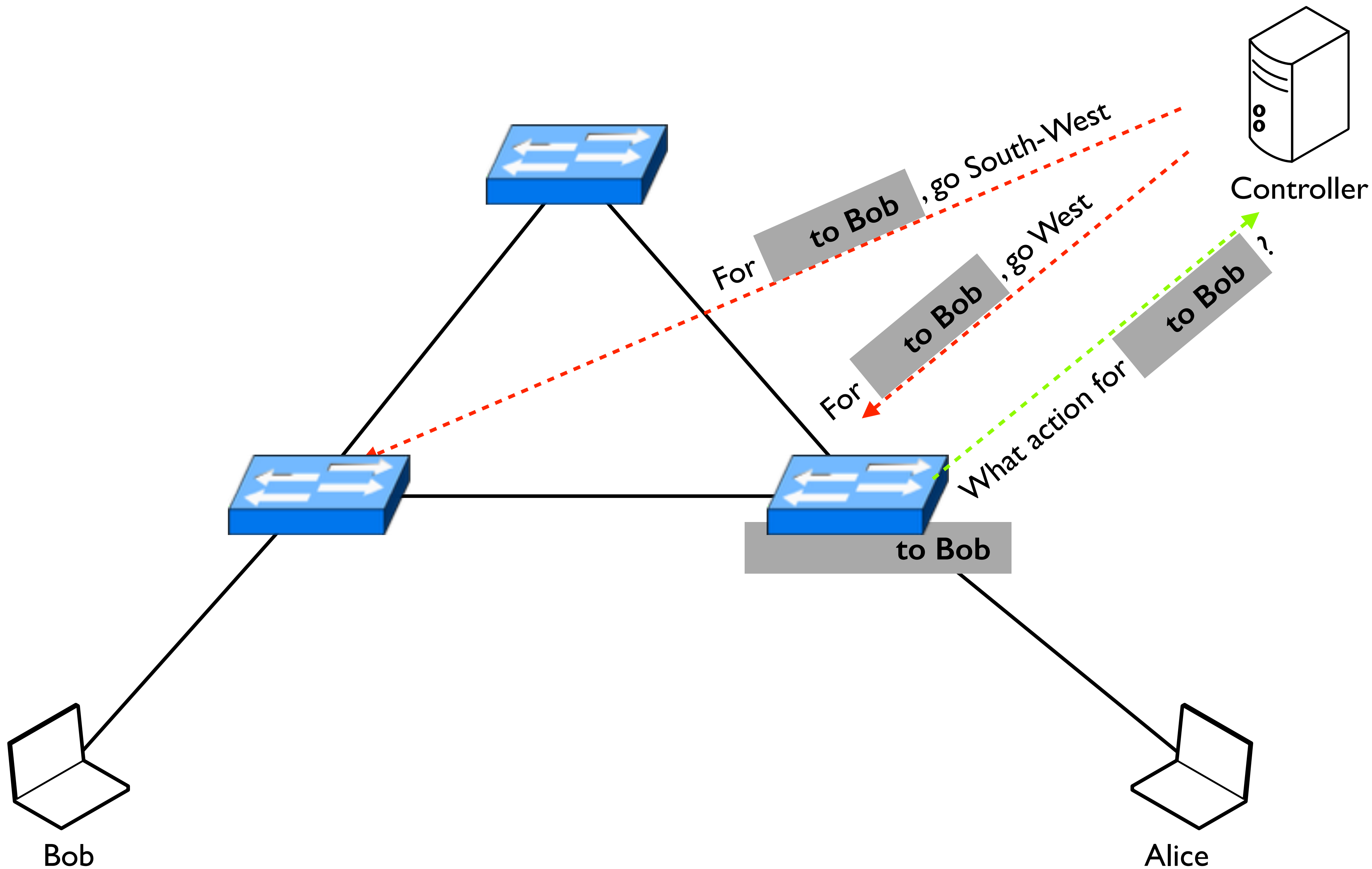
# Southbound interface with OpenFlow



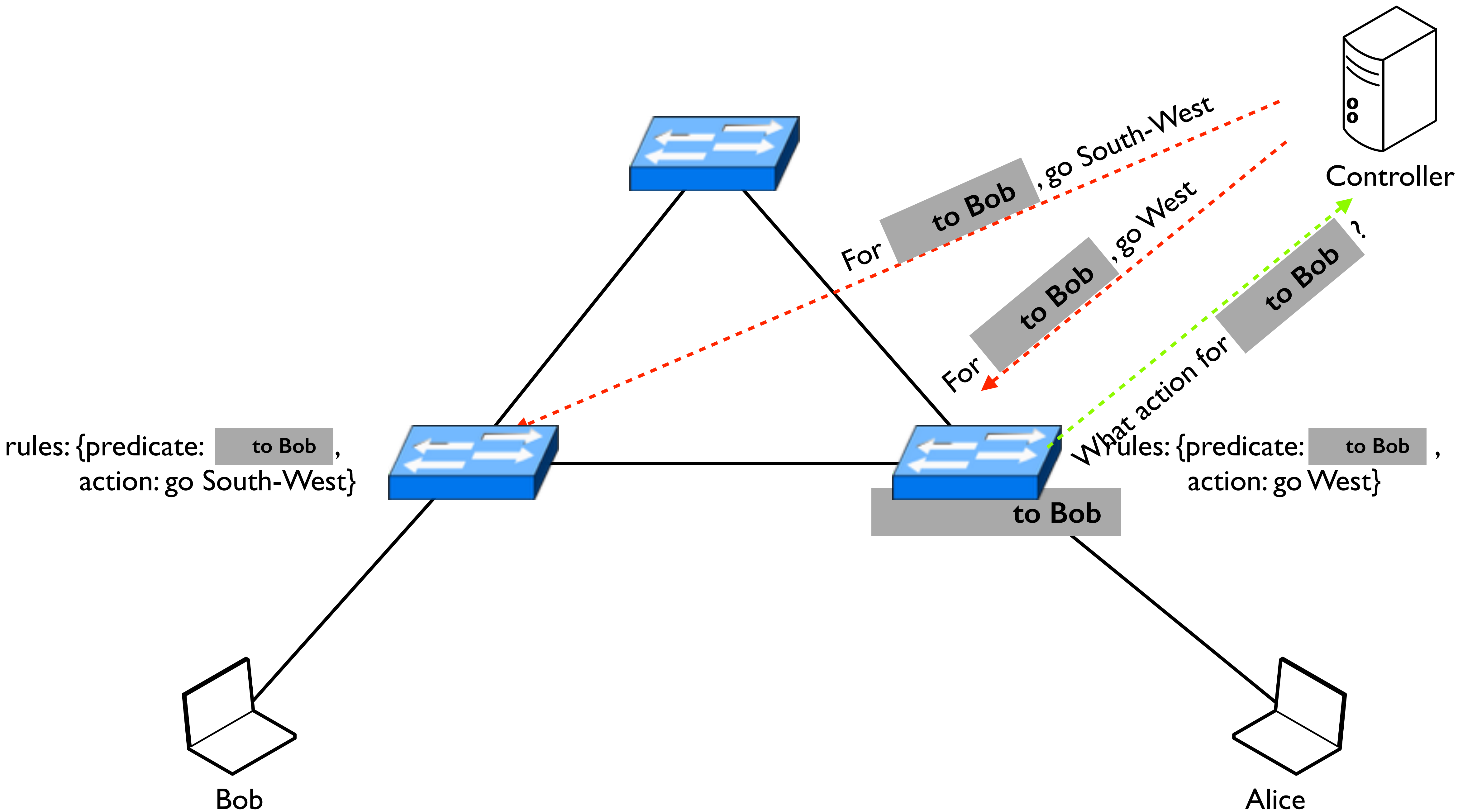
# Southbound interface with OpenFlow



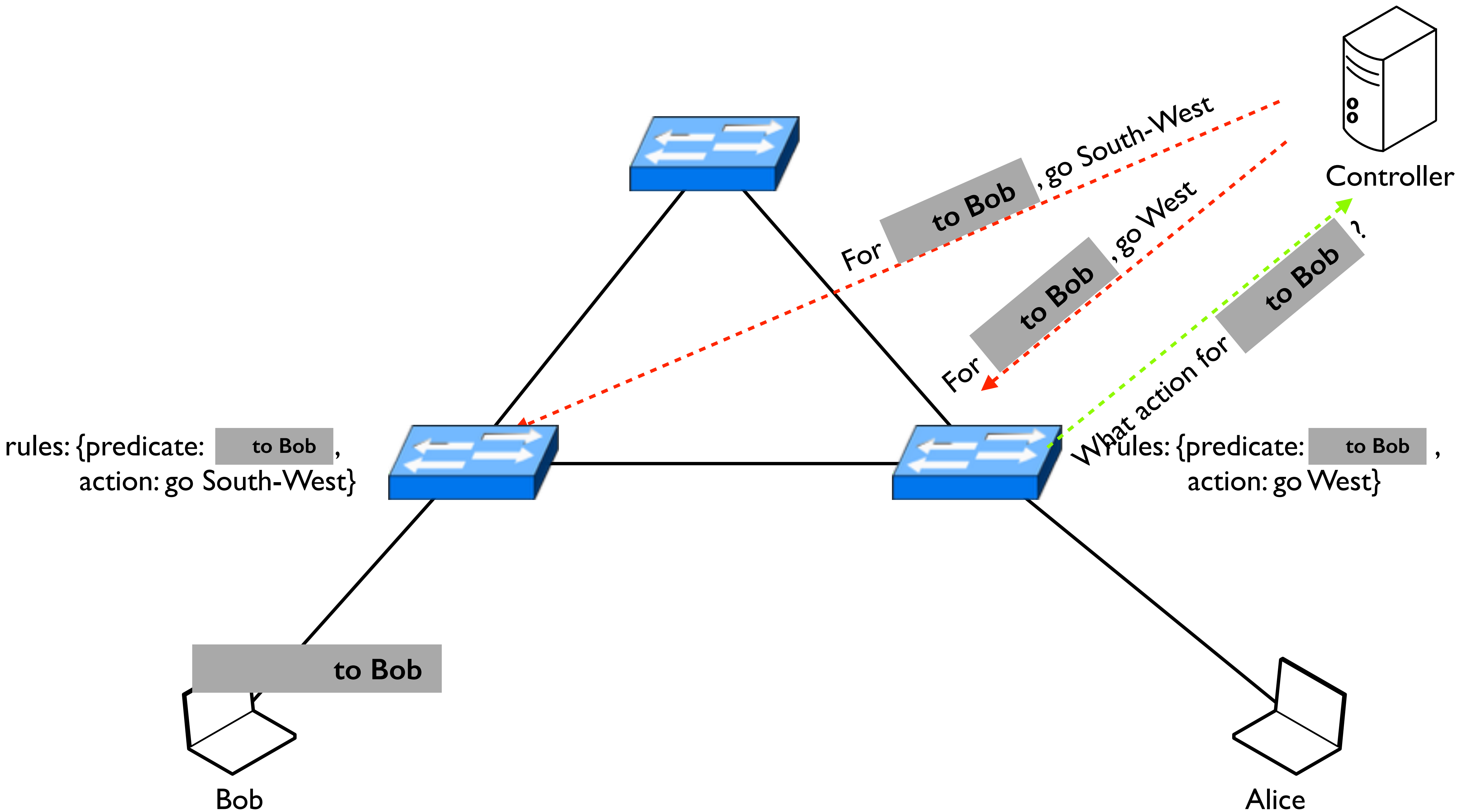
# Southbound interface with OpenFlow



# Southbound interface with OpenFlow



# Southbound interface with OpenFlow



# The network as a blackbox

# SDN brings abstraction

- The network is a black box [NST+14, NSB+15] and the operator
  - only specifies its endpoint policy, no routing policy anymore (i.e., where 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, to appear in IEEE INFOCOM 2015, April 2015.

# SDN brings abstraction

- The network is a black box [NST+14, NSB+15] and 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, to appear in 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

| Predicate | Action | Priority |
| --- | --- | --- |
| IP.destination = bob ^ tcp.destination_port = HTTP | forward to West | 10 |
| TRUE | forward to South | 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.

# How to deal with small flow tables?

- Eviction (e.g., LRU) [VPMB14]
  - remove the least interesting rule when a new rule must be added.
- Compression [CMT+11,IMS13]
  - build rules so to minimise their number.
- Split and distribute [KHK12,NST+14]
  - distribute the rules in network.

# Trade routing for efficiency

# Two policies

- Endpoint policy
  - specifies where packets must be eventually delivered.
- Routing policy
  - specifies the paths that the packets must follow to be eventually delivered.

# Two policies

- Endpoint policy
  - specifies where packets must be eventually

**Routing is an artefact that can be ignored**

  - specifies the paths that the packets must follow to be eventually delivered.

# Our objective

- *Let the network auto(-magically) construct flow tables so to satisfy endpoint policy under resource constraints.*

# Objective

- Find the $|F| \times |L|$ binary allocation matrix $A$ stating whether or not flow $f \in F$ must be transported over link $l \in L$
- that maximises the network utility function $\mathbb{F}(A, \cdots)$,
- according to the endpoint policy $E(f)$ that specifies the set of egress points where a given flow $f$ can be delivered.

# Constraints to respect

- Policies: packets must exit the network at one valid egress point.
- Bandwidth: do not exceed link capacity.
- Memory: do not saturate switches flow table.
- Loops: avoid loops.
- Realism: the solution must be implementable and deployable in real networks.

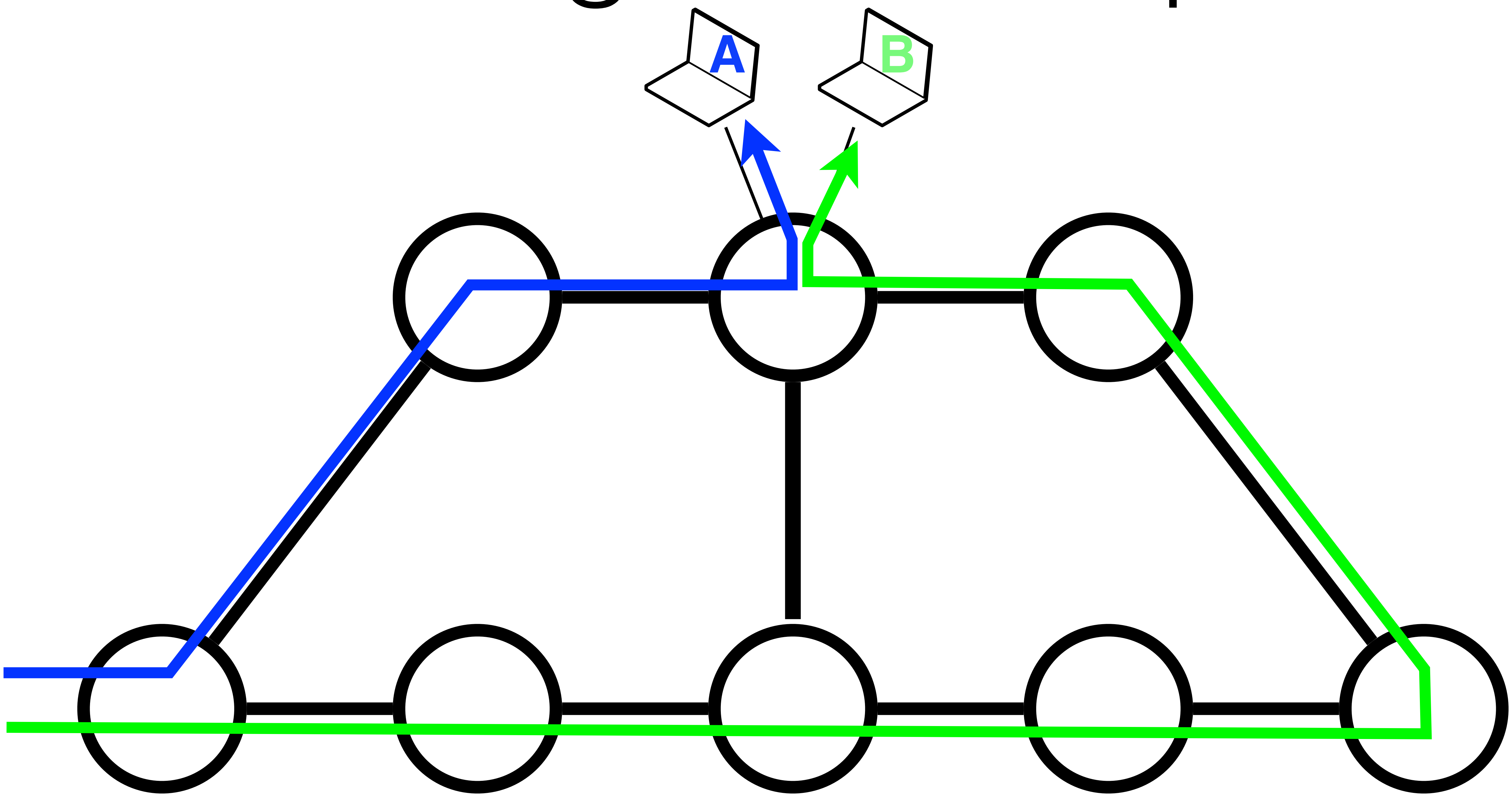
# NP-hardness

- *The rule allocation problem defined to maximise network utility satisfaction is NP-hard [NSB+15].*
    - 0-1 Knapsack problem

# NP-hardness

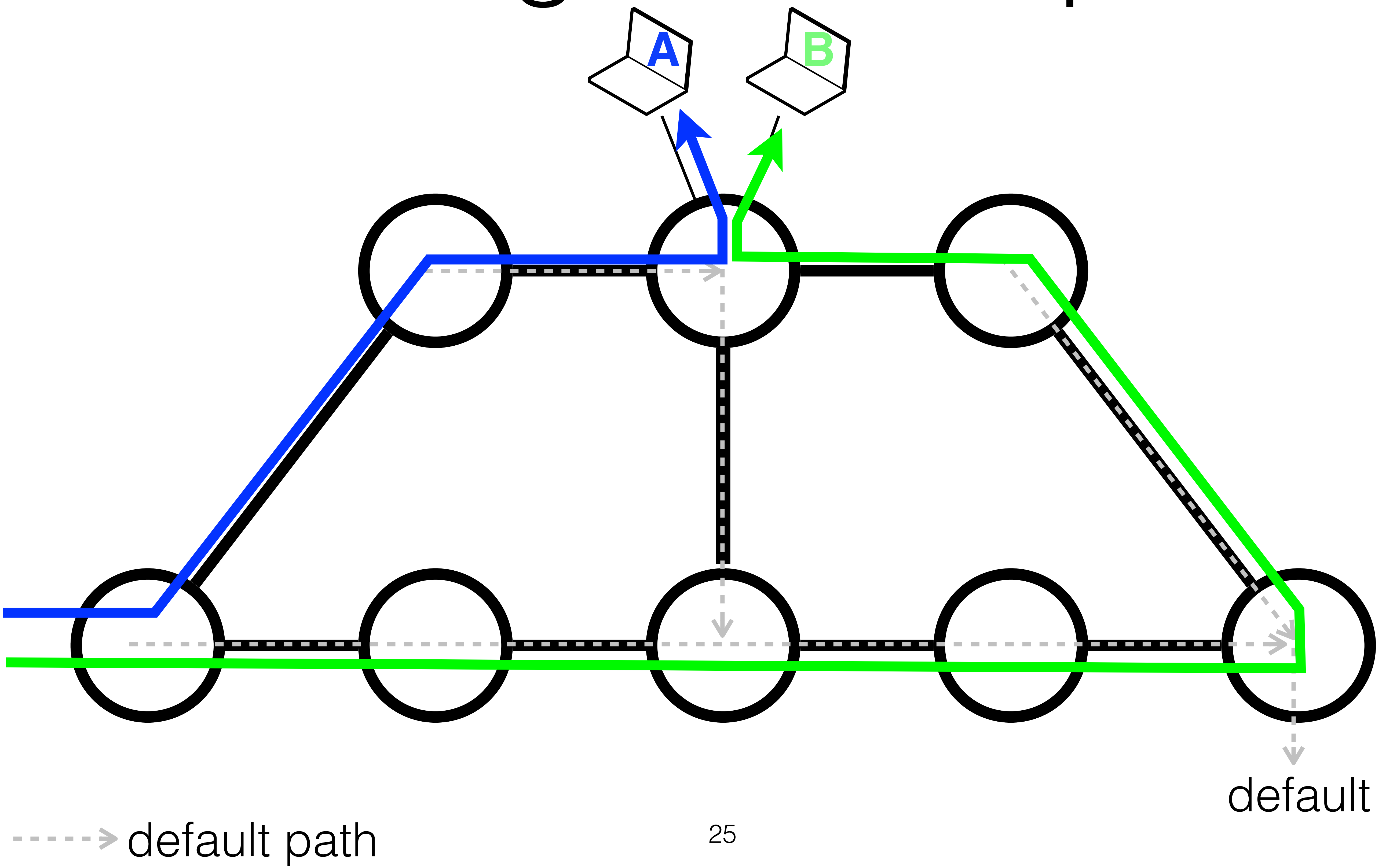
**Trying to find the optimal does invalidate the *realism* constraint**

# Leverage default path

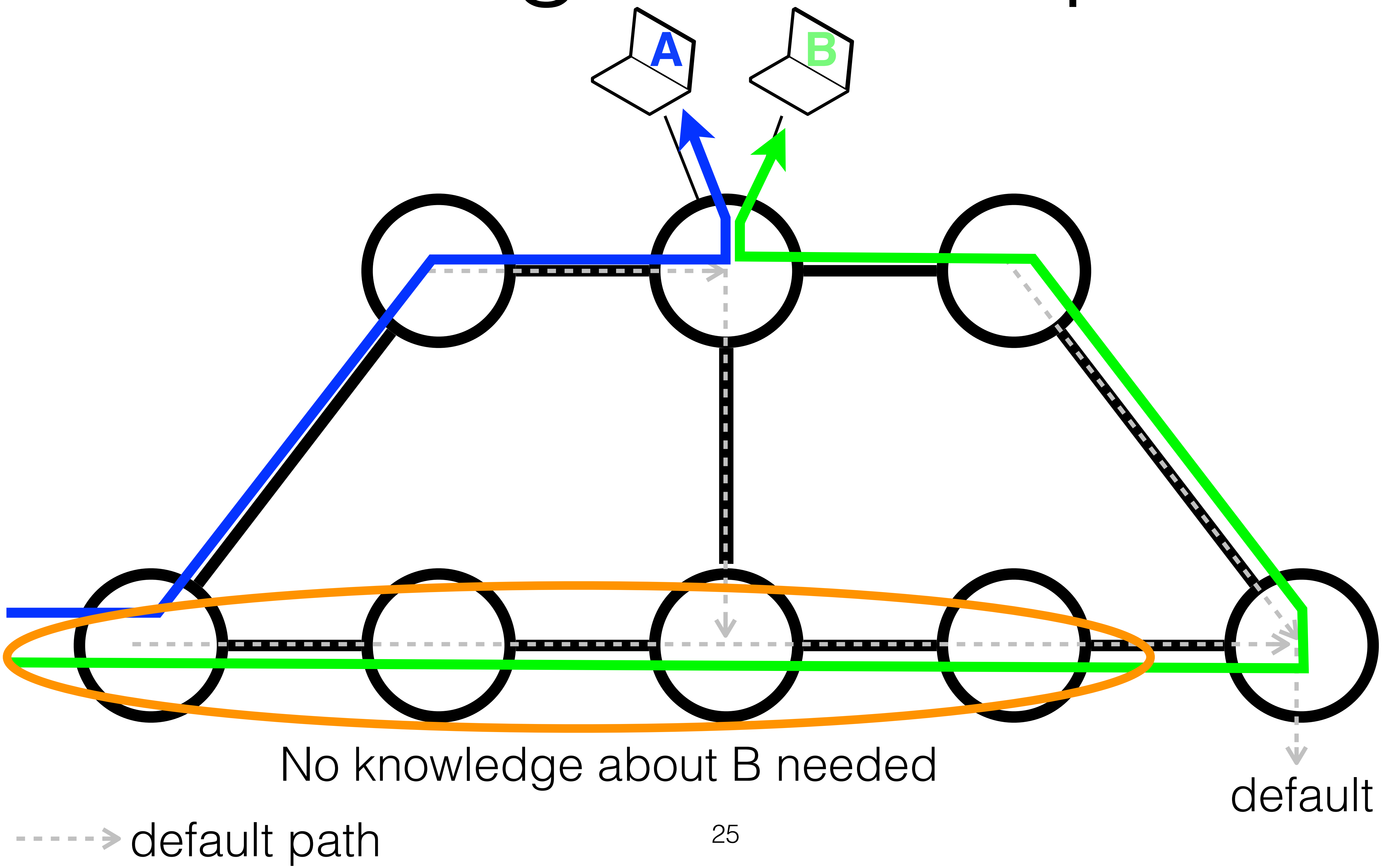



default path

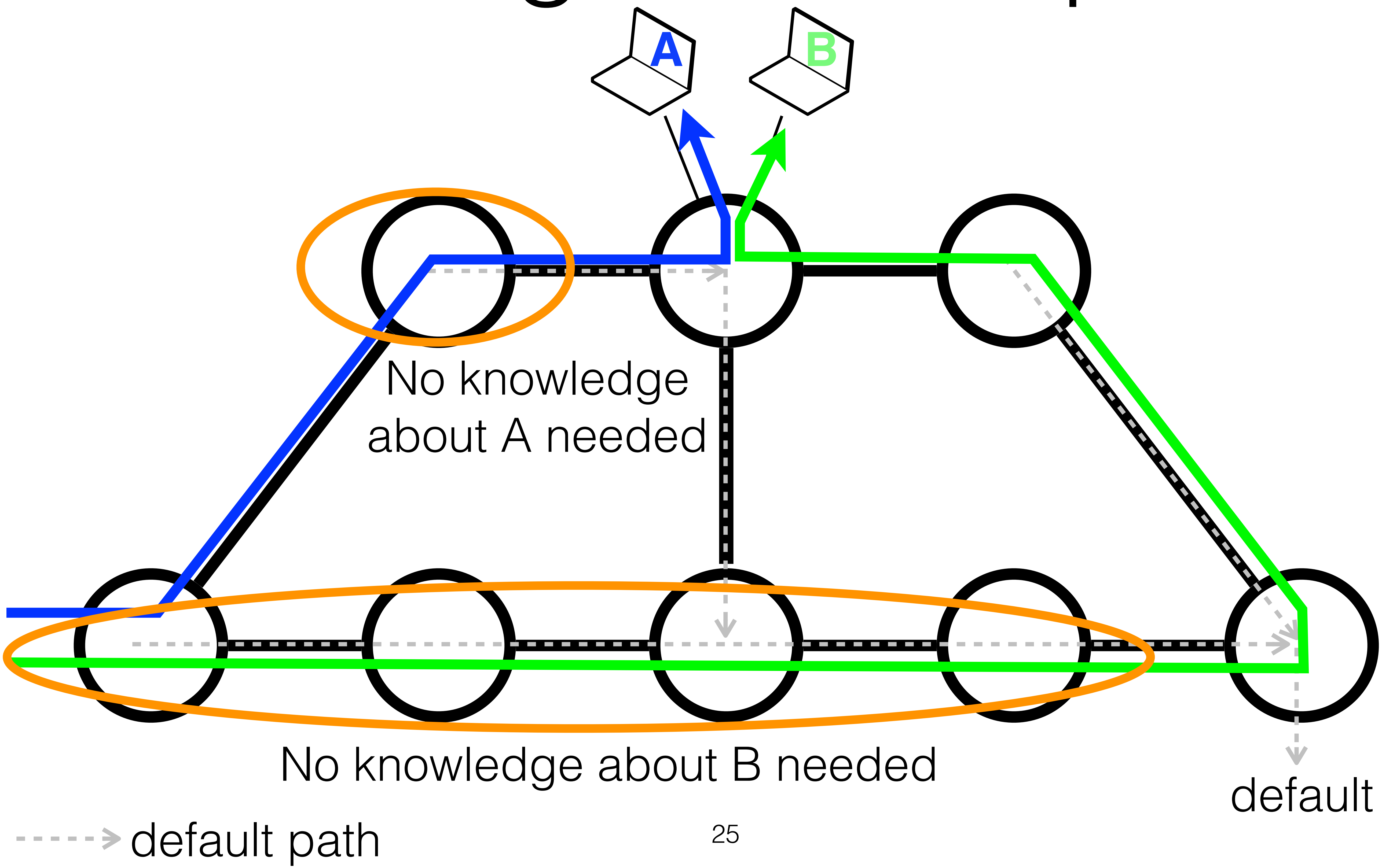
# Leverage default path



# Leverage default path



# Leverage default path

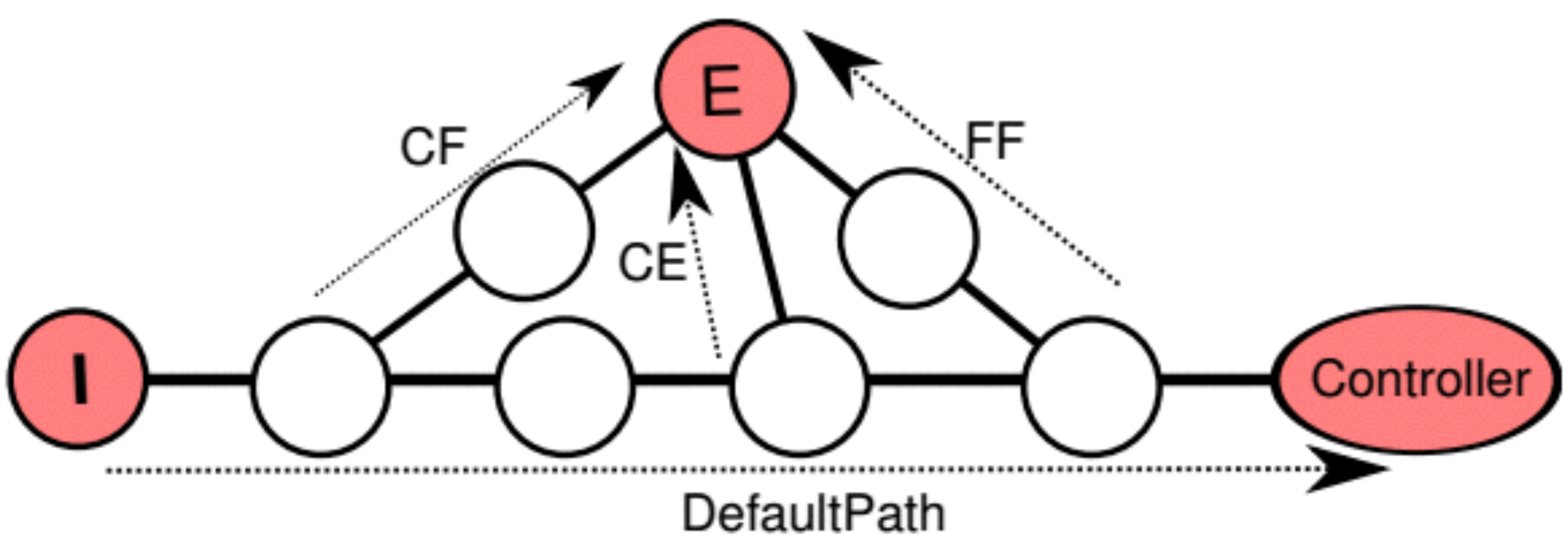


# OFFICER $\mathcal{O}\left(n \log n\right)$ greedy heuristic

- Following the default path induces no signalling/ memory cost.
- Follow as much as possible the default path but eventually deflect packets to one of their egress points [NSB+15].

# Deflection point strategies

- CF: closest first.
- CE: close to egress.
- FF: farthest first.



# OFFICER $\mathcal{O}\left(n \log n\right)$ greedy heuristic

**INPUT:** flow weights collection $W : F \times E \rightarrow \mathbb{R}_{+}$, set of network switches $S$, set of links $L^{+}$, set of default path per flow $DefaultPath$, a default path is a set of switches, annotated with a rank, on the path towards the controller.
**OUTPUT:** $A$, a $|F|$-by-$|L^{+}|$ binary matrix

$A \leftarrow [0]_{F.L^{+}}$
$M \leftarrow \texttt{sort}(W, descending)$
**for all** $(f, e) \in M$ **do**
  $sequence \leftarrow \texttt{sort}(DefaultPath(f), ascending)$
  **for all** $s \in sequence$ **do**
    **if** $canAllocate(A, f, e, s)$ **then**
      $allocate(A, f, e, s)$
      **break**

# OFFICER $\mathcal{O}(n \log n)$ greedy heuristic

**INPUT:** flow weights collection $W : F \times E \to \mathbb{R}_{+}$, set of network switches $S$, set of links $L^{+}$, set of default path per flow $DefaultPath$, a default path is a set of switches, annotated with a rank, on the path towards the controller.
**OUTPUT:** $A$, a $|F|$-by-$|L^{+}|$ binary matrix

```
A ← [0]_{F.L+}
M ← sort(W, descending)
for all (f, e) ∈ M do
    sequence ← sort(DefaultPath(f), ascending)
    for all s ∈ sequence do
        if canAllocate(A, f, e, s) then
            allocate(A, f, e, s)
            break
```

Try most promising flows first.

# OFFICER $\mathcal{O}(n \log n)$ greedy heuristic

**INPUT:** flow weights collection $W : F \times E \to \mathbb{R}_+$, set of network switches $S$, set of links $L^+$, set of default path per flow $DefaultPath$, a default path is a set of switches, annotated with a rank, on the path towards the controller.
**OUTPUT:** $A$, a $|F|$-by-$|L^+|$ binary matrix

```
A ← [0]_{F.L+}
M ← sort(W, descending)
for all (f, e) ∈ M do
    sequence ← sort(DefaultPath(f), ascending)
    for all s ∈ sequence do
        if canAllocate(A, f, e, s) then
            allocate(A, f, e, s)
            break
```

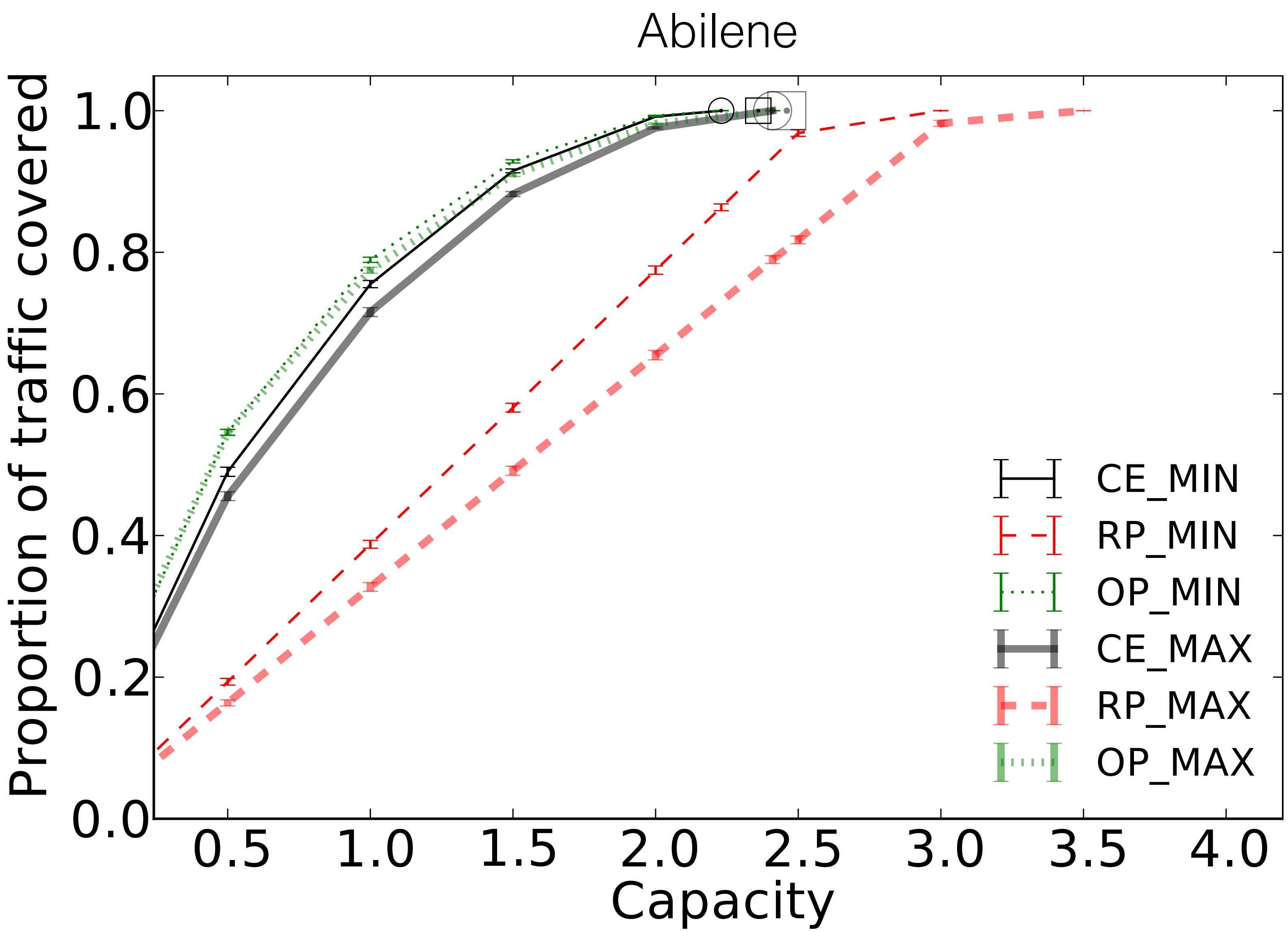
Try most promising flows first.

Try most promising deflection point first.

# Trading routing for better efficiency

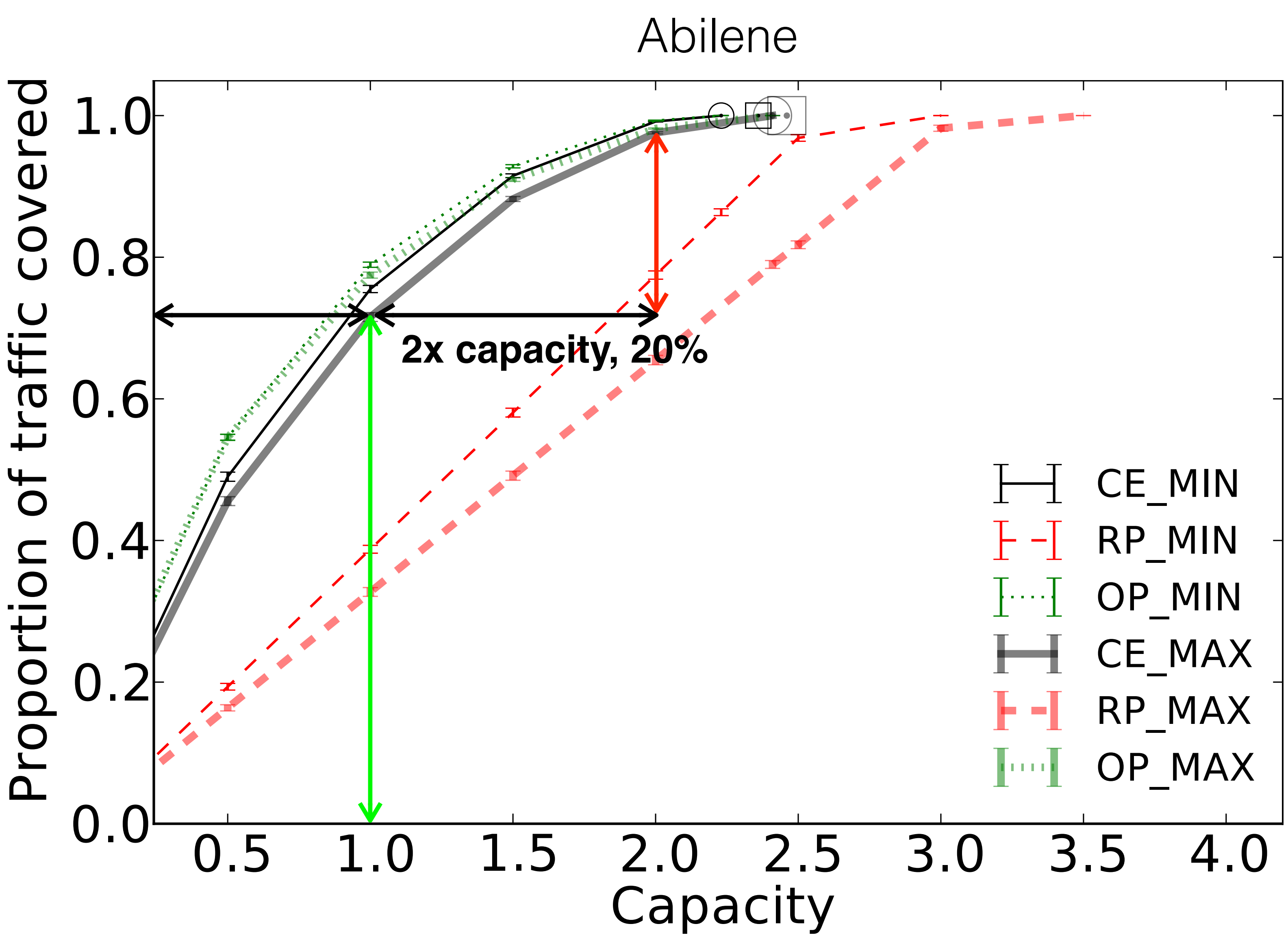
- Trace based simulations on ISP and data-center topologies show that the black box approach:
  - improves network resource utilisation
  - without severely altering performance (i.e., negligible path stretch).
- Reaching optimality is expensive (i.e., small marginal gain while increasing network resources).

# Marginal gain of increasing memory decreases with the total memory




Capacity = # of entries / # of flows

# Marginal gain of increasing memory decreases with the total memory




Capacity = # of entries / # of flows

# Pre-Conclusion

- Software Defined Networking to conceive networks as programs instead of set of devices to manually configure.
- We propose to make the network a black box.
- Hiding the network to operators gives flexibility but stresses the physical infrastructure.
  - Need to define algorithms to map the objective to a realisation.

# Pre-Conclusion

- Software Defined Networking to conceive networks as programs instead of set of devices to manually configure.

**Techniques never decided anything in networking…**

stresses the physical infrastructure.

  - Need to define algorithms to map the objective to a realisation.

# SDN changes the networking ecosystem

[http://blogs.cisco.com/news/open-standards-open-source-open-loop]

# Standardisation vs Softwarisation

- Standards Development Organization (SDO) (e.g., IETF, ITU-T) drive networking industry since 40 years.
  - Well established gouvernance.
- Open Source Software (OSS) projects produce softwares.
  - No gouvernance.

# Time scales

- 2+ year to draft paper specifications in SDOs.
  - Consensus is hard to get,
  - validation is tedious.
- 1 year to think, design and implement a software in OSS.
  - Focus on one technical objective.

# The risks with SDOs

- SDOs gouvernance provides
  - efficient integrated development and maintenance processes,
  - broad and long term vision of the problem
  - concentration of efforts.
- SDOs are old gigantic institutions
  - averse to changes,
  - slow to react,
  - hard to enter for new actors.

# The risks with OSS

- OSS are agile and quickly respond to needs.
- OSS lack of gouvernance causes
  - security flaws,
  - small fragmented communities (little funding, dogmatic vision),
  - uncertainty of maintenance.

# SDN pushes towards OSS

- Without SDN:
  - network algorithm implementations are bound to the device supporting them,
  - hardware and software producers are the same companies.
    - Hard for new actors to enter the market.
- With SDN:
  - network algorithm implementation are independent of the hardware,
  - hardware and software producers are different companies.
    - Any innovative actor can enter the market easily.

➡ Costs reduction.

# SDN pushes towards OSS

- Without SDN:
  - network algorithm implementations are bound to the device supporting them,
  - hardware and software producers are the same companies.

**SDOs and OSS must form a collaborative loop**

- hardware and software producers are different companies.
  - Any innovative actor can enter the market easily.

➡ Costs reduction.

# Is the future of network software?

*Damien Saucez*
Inria

March 2015

# References


[1] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'One Big Switch' Abstraction in Software-Defined Networks," in *ACM CoNEXT*, Dec. 2013.
[2] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.
[3] M. Thorup and U. Zwick, "Compact routing schemes," in *SPAA*, 2001.
[4] S. e. a. Jain, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM*, 2013.
[5] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized Zero-Queue Datacenter Network," in *ACM SIGCOMM*, August 2014.
[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM CCR*, vol. 38, no. 2, Mar. 2008.
[7] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency," in *ACM HotSDN*, Apr. 2014.
[8] H. Ballani, P. Francis, T. Cao, and J. Wang, "Making routers last longer with viaggre," in *USENIX NSDI*, Berkeley, CA, USA, 2009, pp. 453–466.
[9] X. Jin, H. Liu, R. Gandhi, and S. Kandula, "Dynamic Scheduling of Network Updates," in *ACM SIGCOMM*, 2014.
[10] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM*, Apr. 2013, pp. 545–549.
[11] T. Benson, A. Akella, and D. a. Maltz, "Network traffic characteristics of data centers in the wild," *IMC*, 2010.
[12] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *SIGCOMM CCR*, vol. 32, no. 4, pp. 133–145, 2002.
[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM*, 2008.
[14] "Abilene," http://http://sndlib.zib.de.
[15] L. Saino, C. Cocora, and G. Pavlou, "A Toolchain for Simplifying Network Simulation Setup," in *SIMUTOOLS*, 2013.
[16] P. Wette, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "MaxiNet: Distributed Emulation of Software-Defined Networks," in *IFIP Networking Conference*, 2014.
[17] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *ACM HotSDN*, 2012, pp. 7–12.
[18] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," *SIGCOMM CCR*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
[19] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu, "Domainflow: Practical flow management method using multiple flow tables in commodity switches," in *ACM CoNEXT*, 2013.
[20] A. Iyer, V. Mann, and N. Samineni, "Switchreduce: Reducing switch state and controller involvement in openflow networks," in *IFIP Networking Conference*, 2013.
[21] "OpenFlow Switch Specification," http://www.opennetworking.org/.
[22] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *SIGCOMM CCR*, 2010.
[23] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Vcrib: Virtualized rule management in the cloud," in *USENIX HotCloud*, 2012.
[24] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing," in *IEEE GLOBECOM*, 2014.

# Backup

# Resource constraints

- Bandwidth: do not exceed link capacity

$$\forall l \in L^+ : \sum_{f \in F} p_f a_{f,l} \leq B_l$$

- Memory: do not saturate switches flow table
  - naive compression: no cost when the action is the same as the default rule

$$\forall s \in S : \sum_{v \in N^{\leftarrow}(s) \setminus \{def(s)\}} \sum_{f \in F} a_{f,(s,v)} \leq C_s$$

$$\sum_{s \in S} \sum_{v \in N^{\leftarrow}(s) \setminus \{def(s)\}} \sum_{f \in F} a_{f,(s,v)} \leq M$$

# Endpoint policy constraints

- Packets must exit the network at one valid egress point.
- If it is not possible, they have to be taken care of by the controller.

$$\forall f \in F, \forall l \in E \setminus E^*(f) : a_{f,l} = 0$$
$$\forall f \in F : \sum_{l \in E^*(f)} a_{f,l} = 1$$

# Path length constraint

- One can limit the maximum length of the path to the egress if needed (then it is not really a black box…)

$$\forall f \in F : \sum_{l \in L^+} a_{f,l} \leq \alpha(f)$$

| Notation | Description |
|---|---|
| $F$ | Set of flows. |
| $S$ | Set of OpenFlow switches composing the network. |
| $S_e$ | Set of external nodes directly connected to the network but not part of the network to be optimized (e.g., hosts, provider or customer switches, controllers, blackholes). |
| $S^+$ | Set of all nodes ($S^+ = S \cup S_e$). |
| $L$ | Set of directed links, defined by $(s, d) \in S \times S$, where $s$ is the origin of the link and $d$ is its termination. |
| $I$ | Set of directed ingress links that connect external nodes to OpenFlow switches, defined by $(s, d) \in S_e \times S$. The particular ingress link of a flow $f \in F$ is written $l_f$ by abuse of notation. |
| $E$ | Set of directed egress links that connect the OpenFlow switches to external nodes, defined by $(s, d) \in S \times S_e$. |
| $L^+$ | Set of all directed links (i.e., $L^+ = L \cup I \cup E$). |
| $N^{\rightarrow}(s) \subseteq S^+$ | set of incoming neighboring nodes of switch $s \in S$ (i.e., neighbors from which $s$ can receive packets). |
| $N^{\leftarrow}(s) \subseteq S^+$ | Set of outgoing neighboring nodes of switch $s \in S$ (i.e., neighbors towards which $s$ can send packets). |
| $E(f) \subseteq E$ | Set of valid egress links for flow $f \in F$ according to the endpoint policy. |
| $E^*(f) \subseteq E$ | $E^*(f) = E(f) \cup *$, where $*$ denotes the set of links attached to the controller. |
| $def(s) \in S^+$ | Next hop toward the controller from switch $s \in S$. |
| $M$ | Total switch memory limitation. |
| $C_s$ | Memory limitation of switch $s \in S$. |
| $B_l$ | Capacity of link $l \in L^+$. |
| $p_f$ | Packet rate of flow $f \in F$. |

# Network constraints

- Avoid loop with the flow conservation constraint

$$\forall f \in F, \forall s \in S : \sum_{v \in N^{\rightarrow}(s)} a_{f,(v,s)} = \sum_{v \in N^{\leftarrow}(s)} a_{f,(s,v)}$$

- Sanity checks

$$\forall f \in F, \forall l \in L^{+} : a_{f,l} \in \{0,1\}$$

$$\forall f \in F : a_{f,l} = \begin{cases} 0 & \text{if } l \in I \setminus \{l_f\} \\ 1 & \text{if } l = l_f \end{cases}$$

# Resource constraints

- Bandwidth: do not exceed link capacity

$$\forall l \in L^+ : \sum_{f \in F} p_f a_{f,l} \leq B_l$$

- Memory: do not saturate switches flow table
  - naive compression: no cost when the action is the same as the default rule

$$\forall s \in S : \sum_{v \in N^{\leftarrow}(s) \setminus \{def(s)\}} \sum_{f \in F} a_{f,(s,v)} \leq C_s$$

$$\sum_{s \in S} \sum_{v \in N^{\leftarrow}(s) \setminus \{def(s)\}} \sum_{f \in F} a_{f,(s,v)} \leq M$$

# Endpoint policy constraints

- Packets must exit the network at one valid egress point.
- If it is not possible, they have to be taken care of by the controller.

$$\forall f \in F, \forall l \in E \setminus E^*(f) : a_{f,l} = 0$$
$$\forall f \in F : \sum_{l \in E^*(f)} a_{f,l} = 1$$

# Path length constraint

- One can limit the maximum length of the path to the egress if needed (then it is not really a black box…)

$$\forall f \in F : \sum_{l \in L^+} a_{f,l} \leq \alpha(f)$$

# Two policies

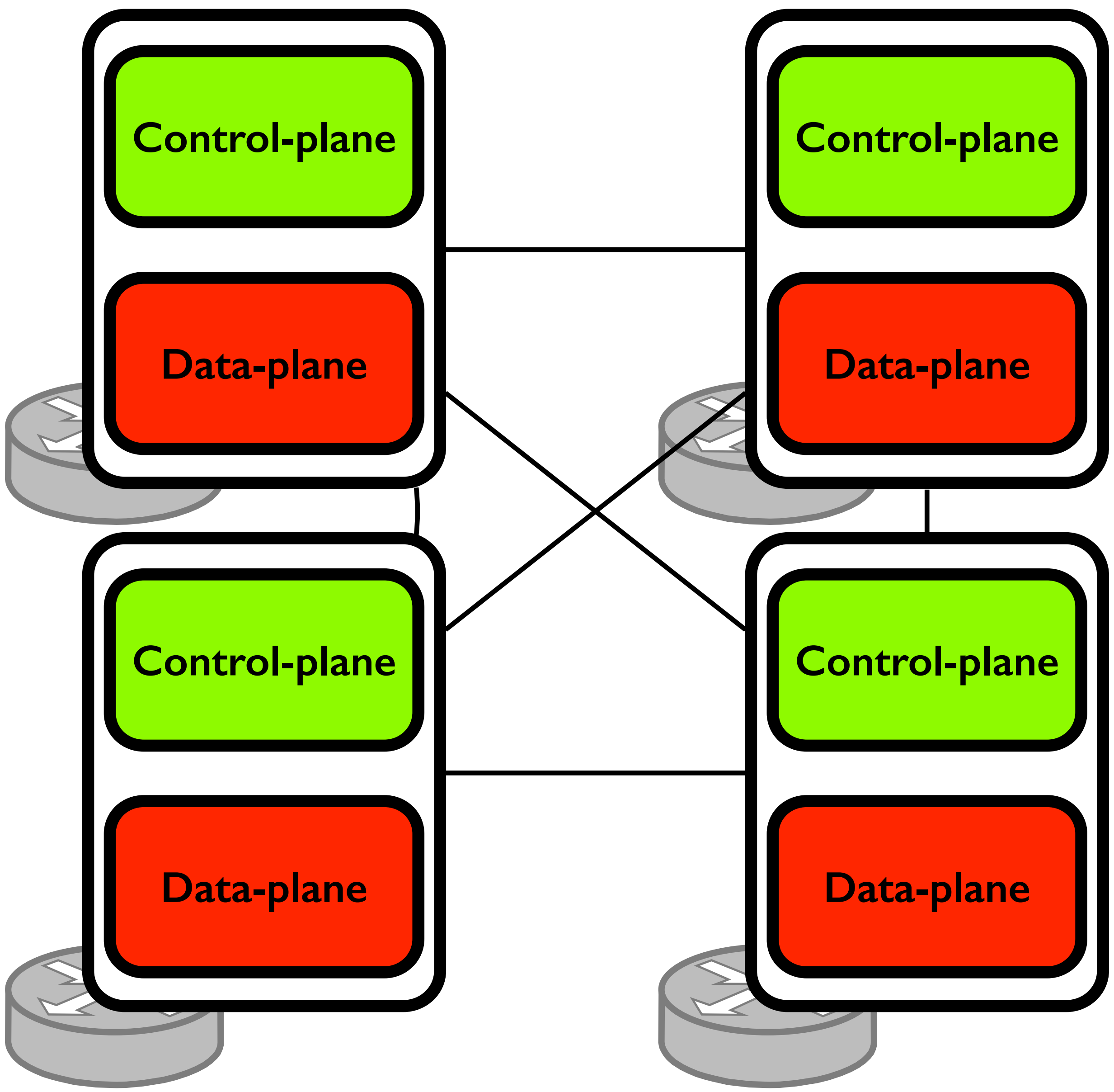
- Endpoint policy
  - specifies where packets must be eventually delivered.
- Routing policy
  - specifies the paths that the packets must follow to be eventually delivered.

# OpenFlow to separate roles

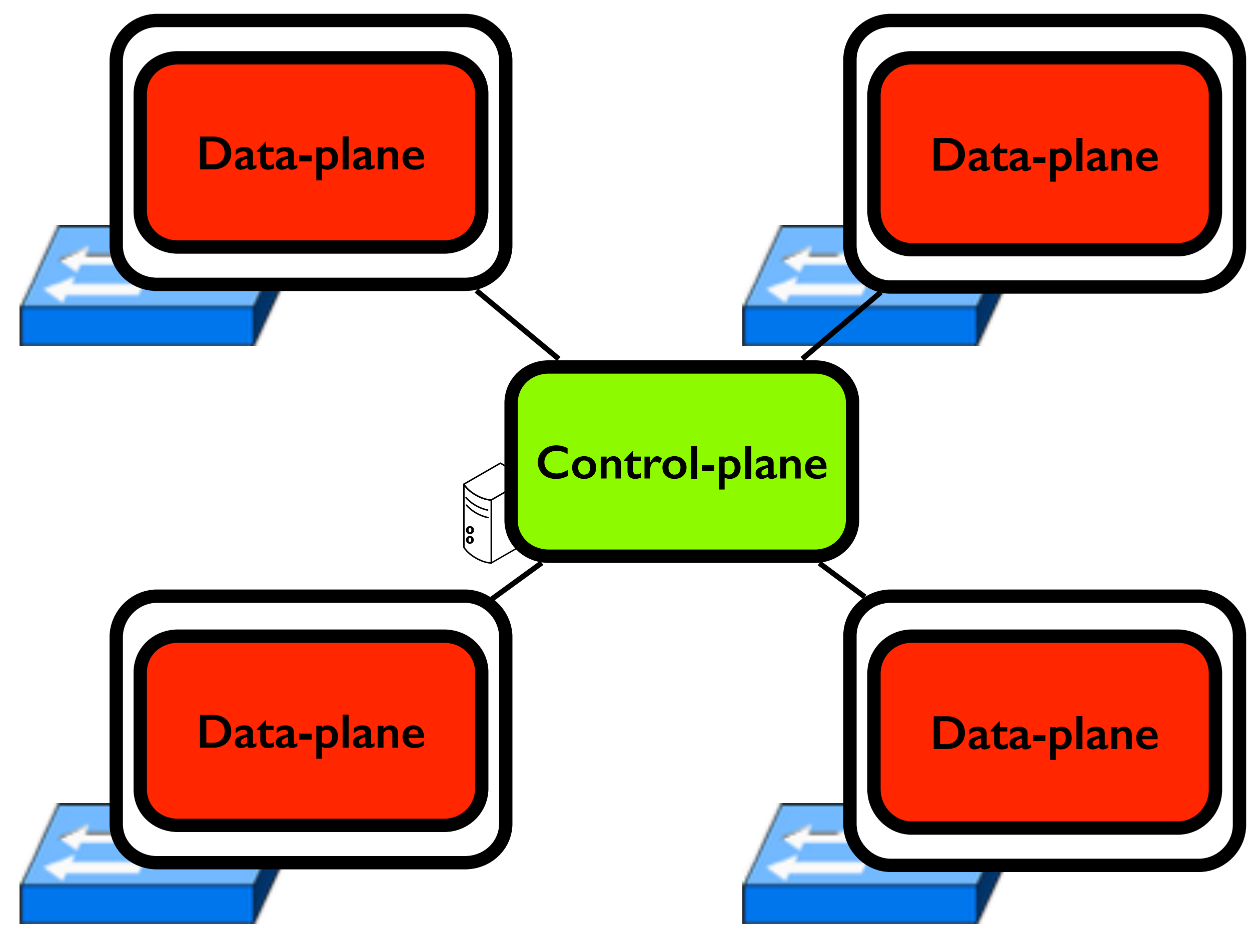
- Programmability of network is reached by decoupling control plane from data plane:
  - network elements are elementary switches,
  - the intelligence is implemented by a logically centralised controller
    - that manages the switches (i.e., install/remove forwarding rules).

# OpenFlow with a picture

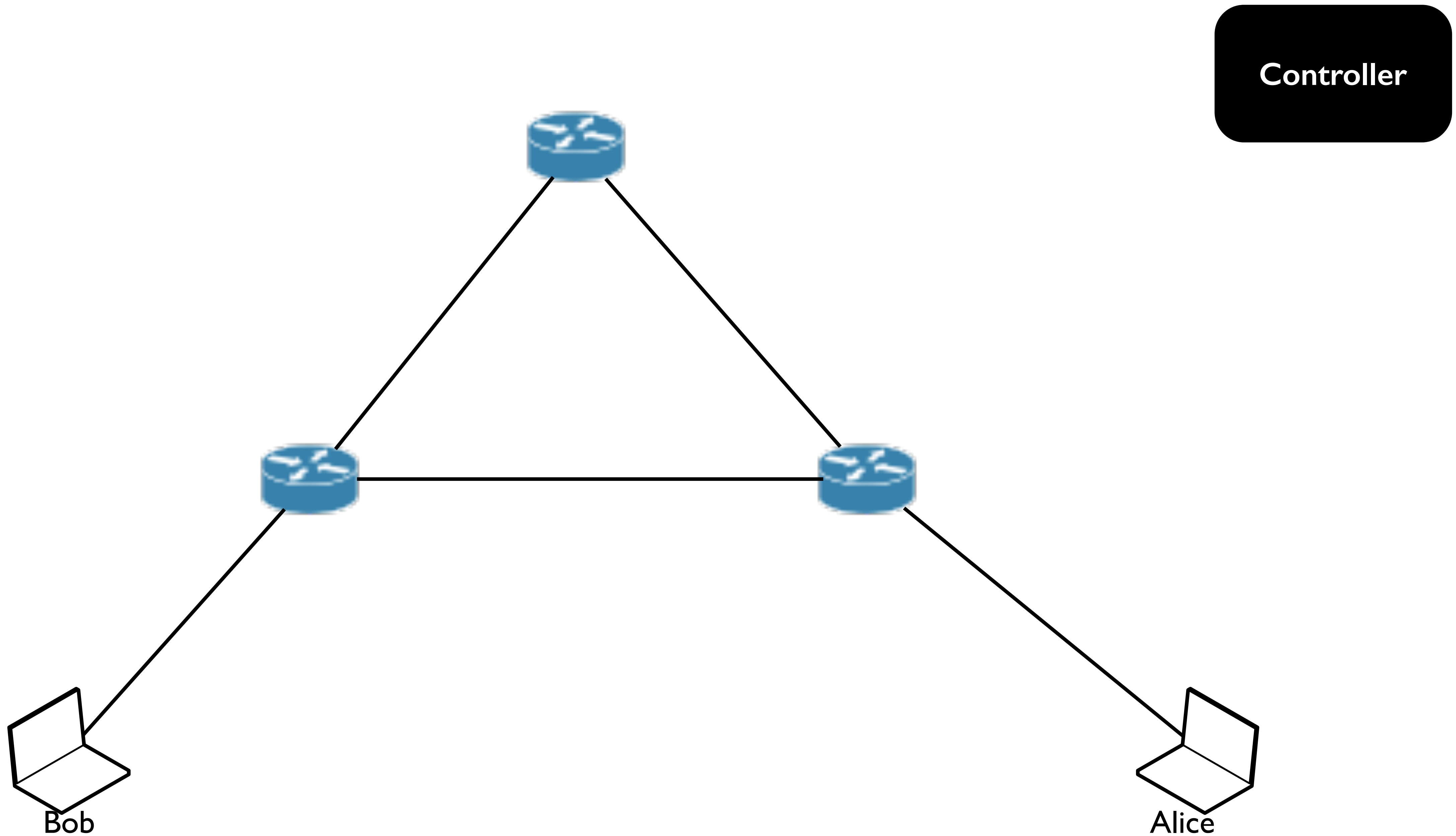
## Traditional approach



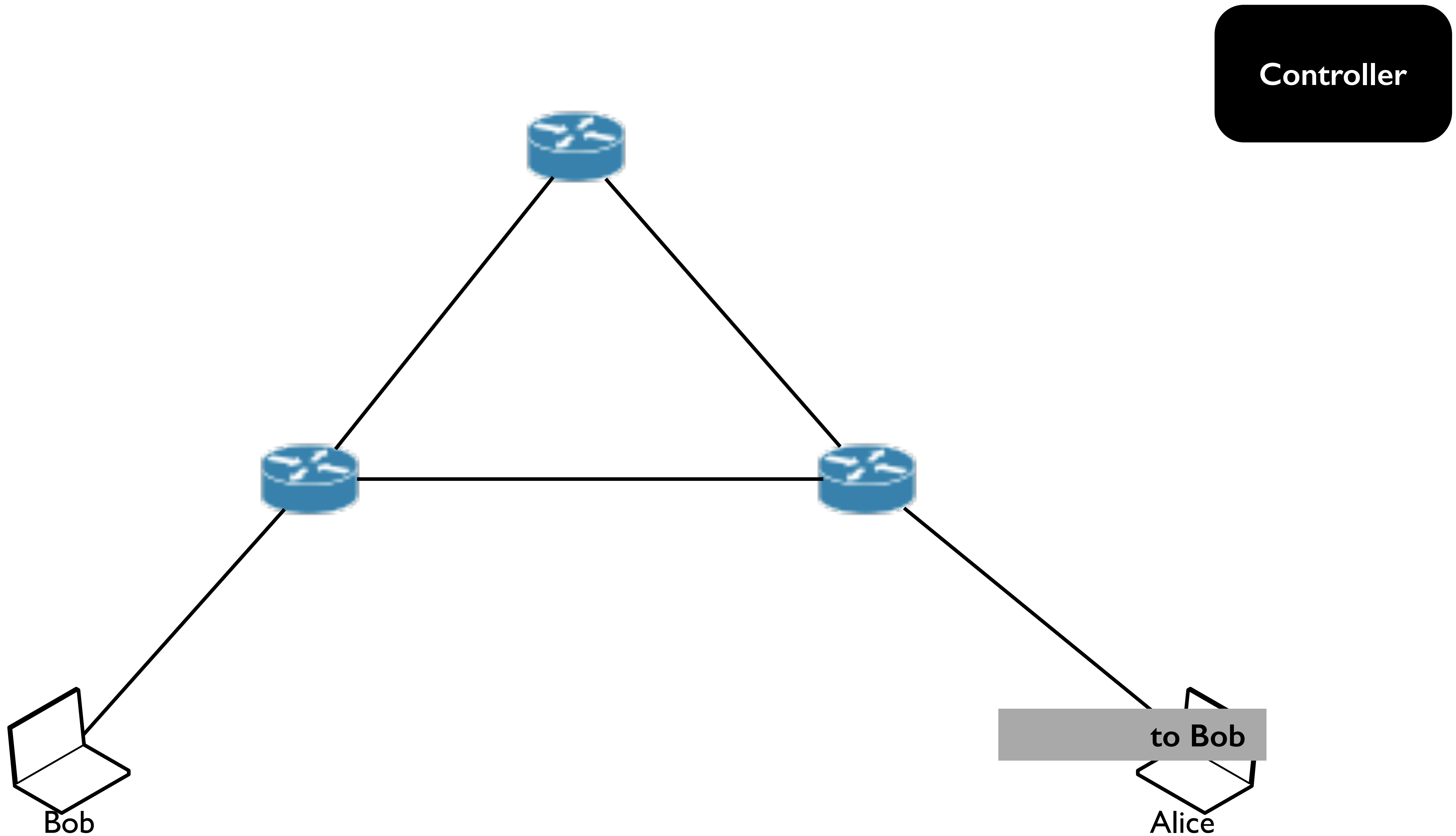

## OpenFlow approach



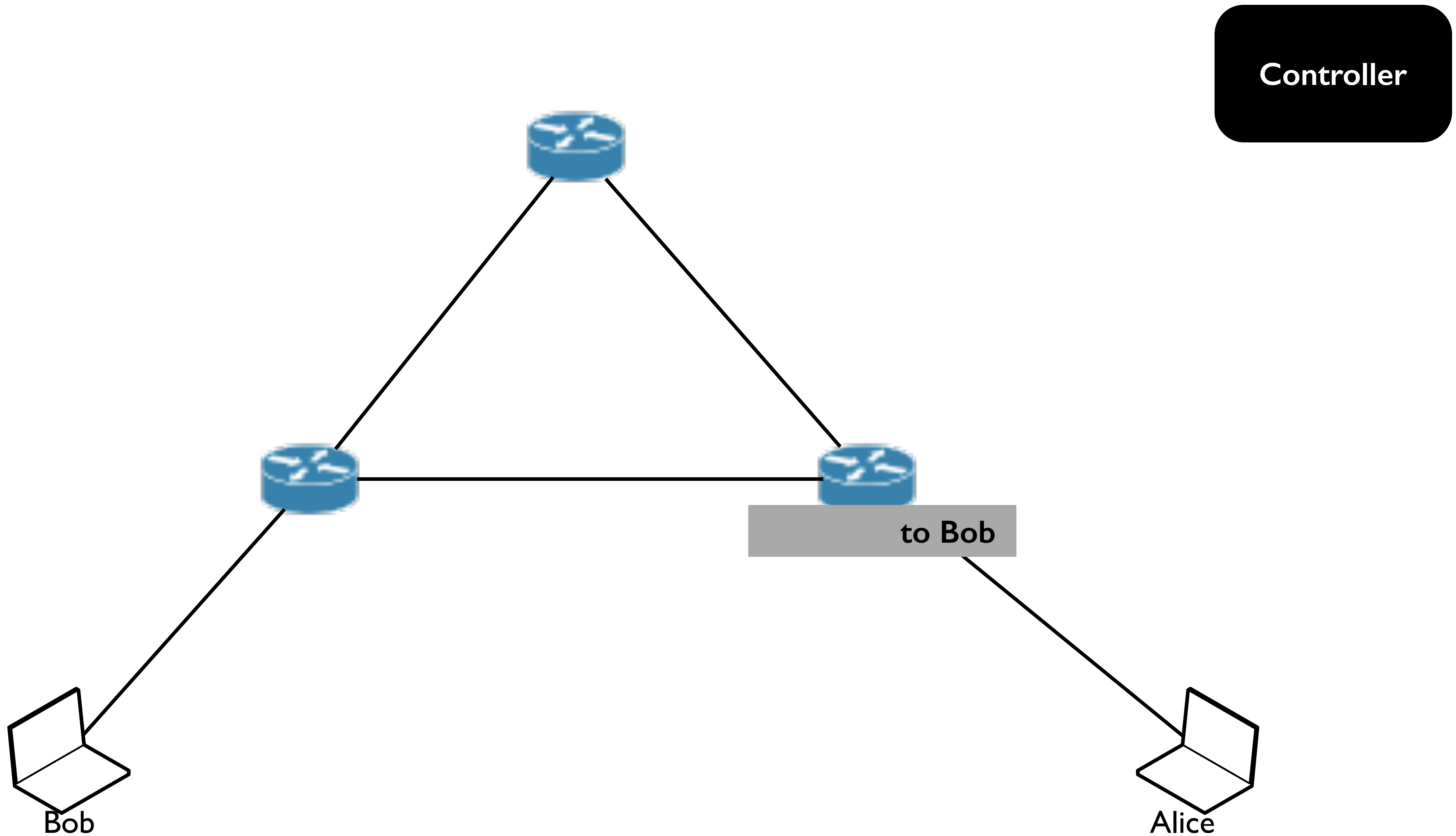
# OpenFlow workflow



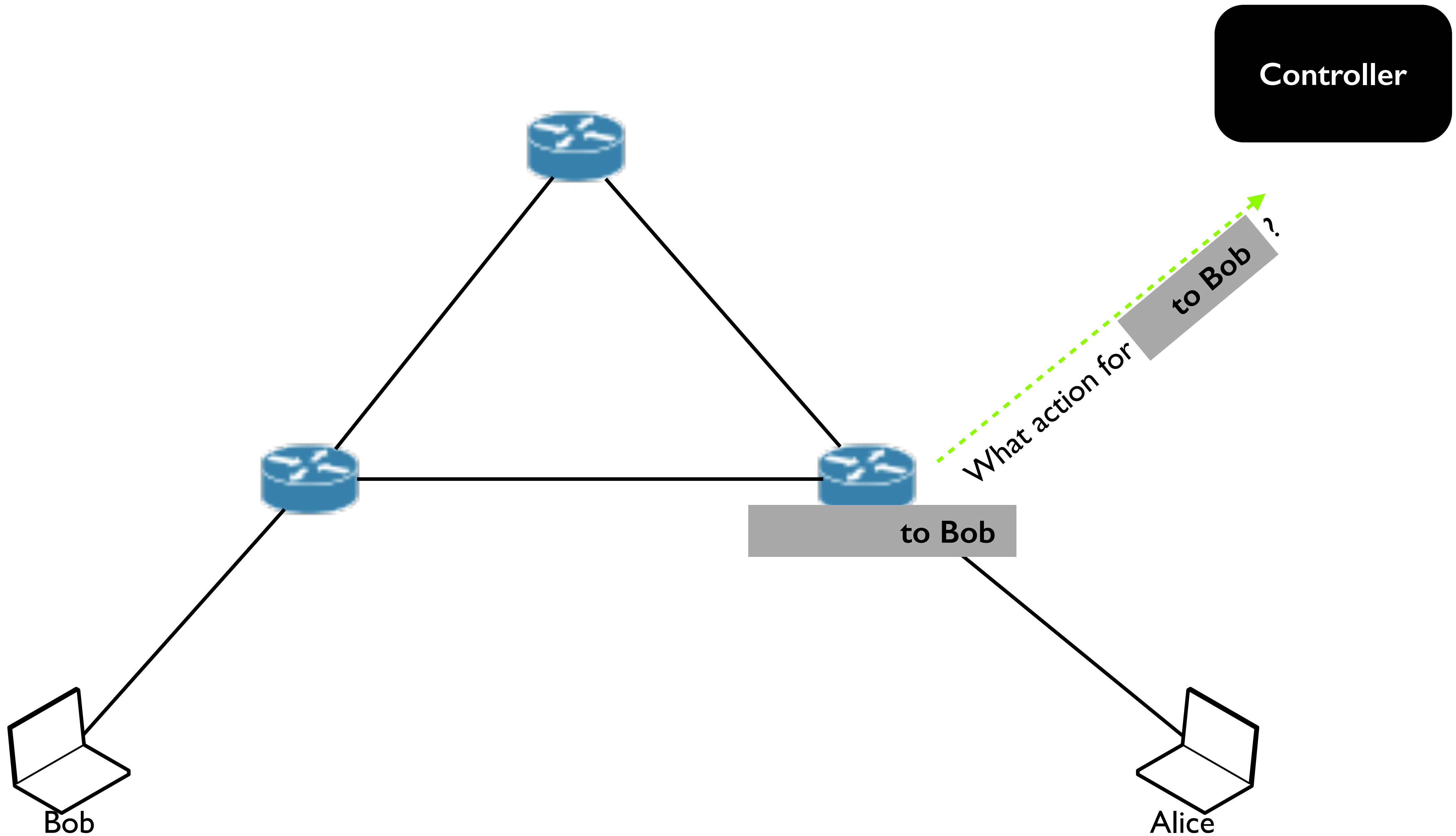
# OpenFlow workflow



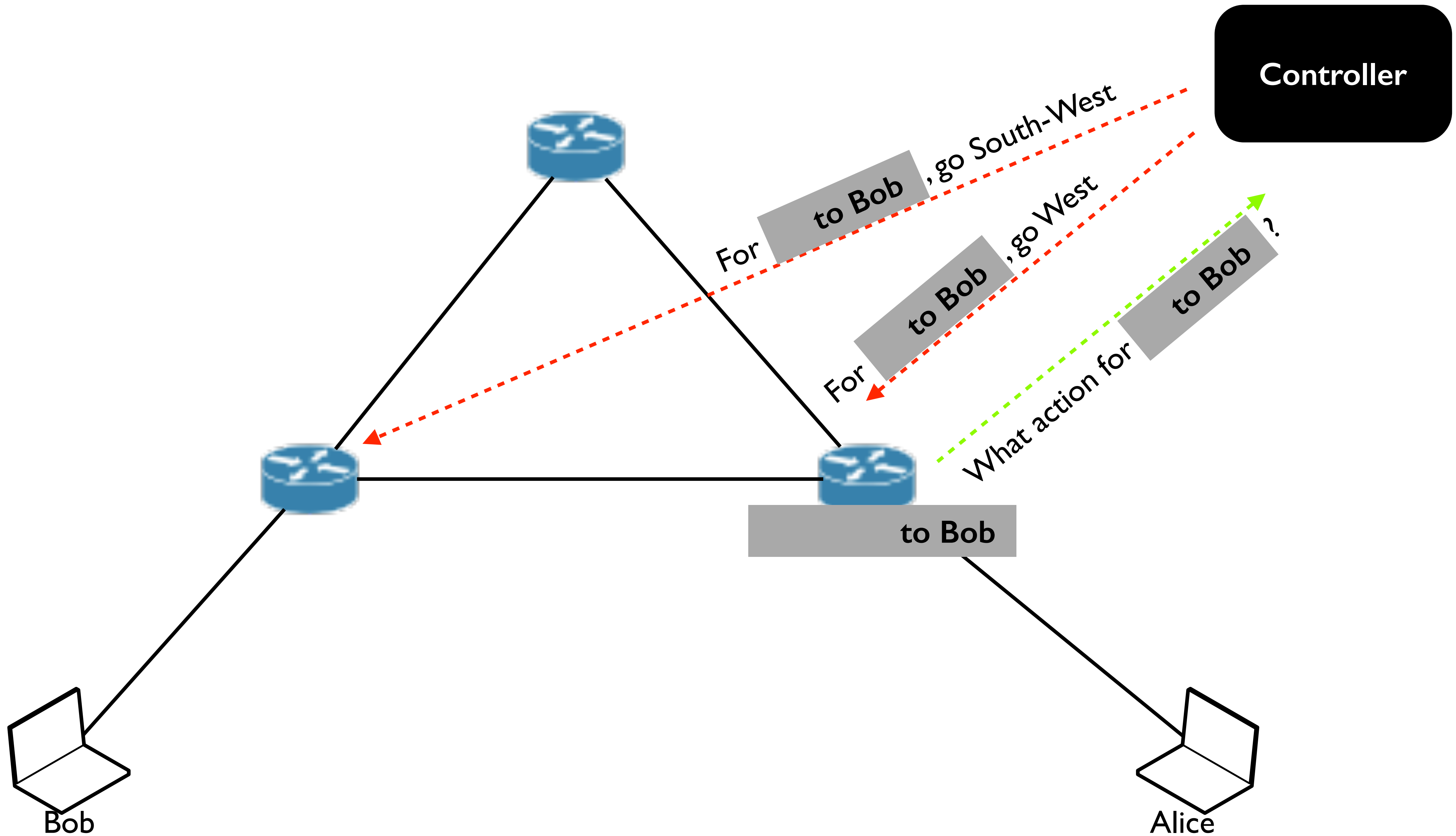
# OpenFlow workflow



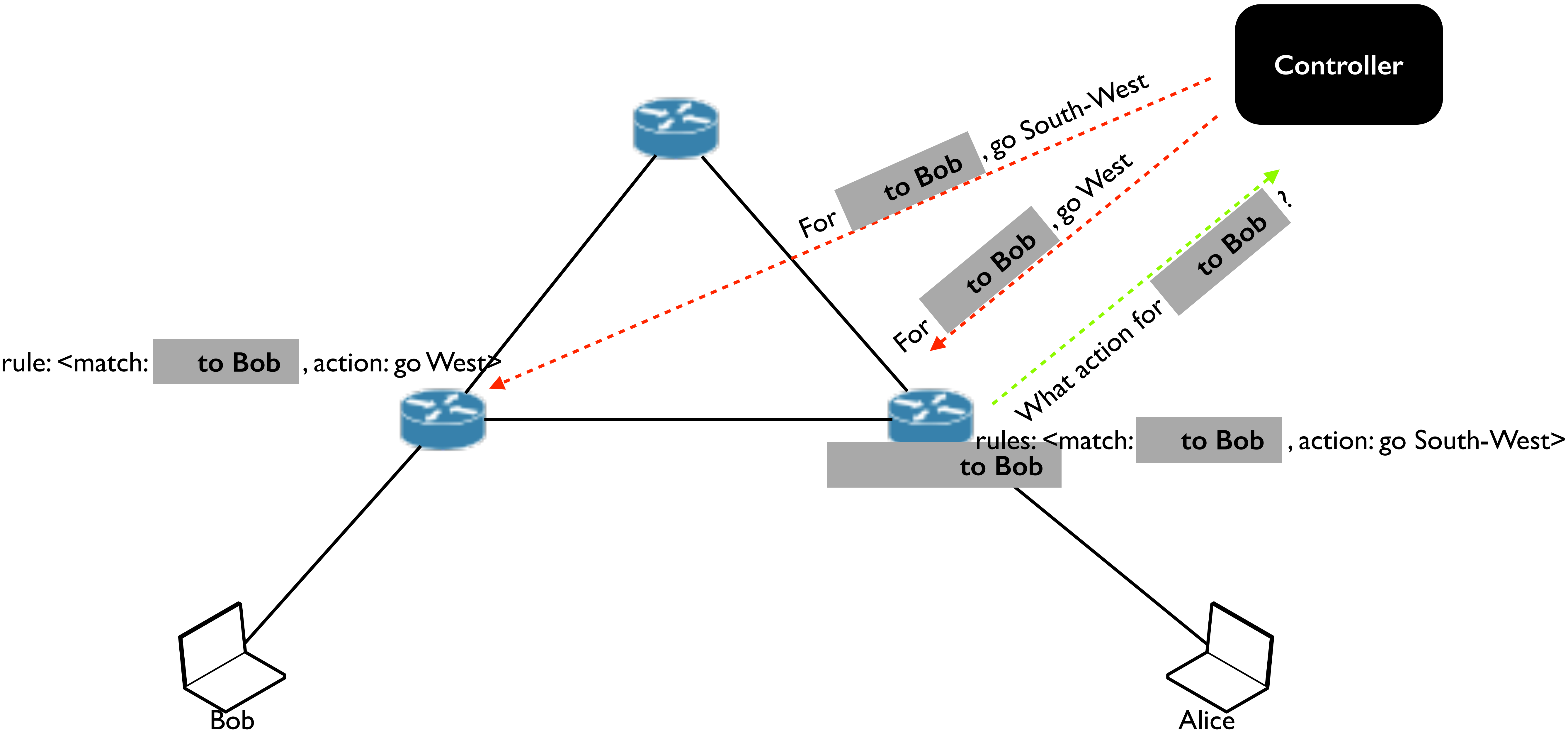
# OpenFlow workflow



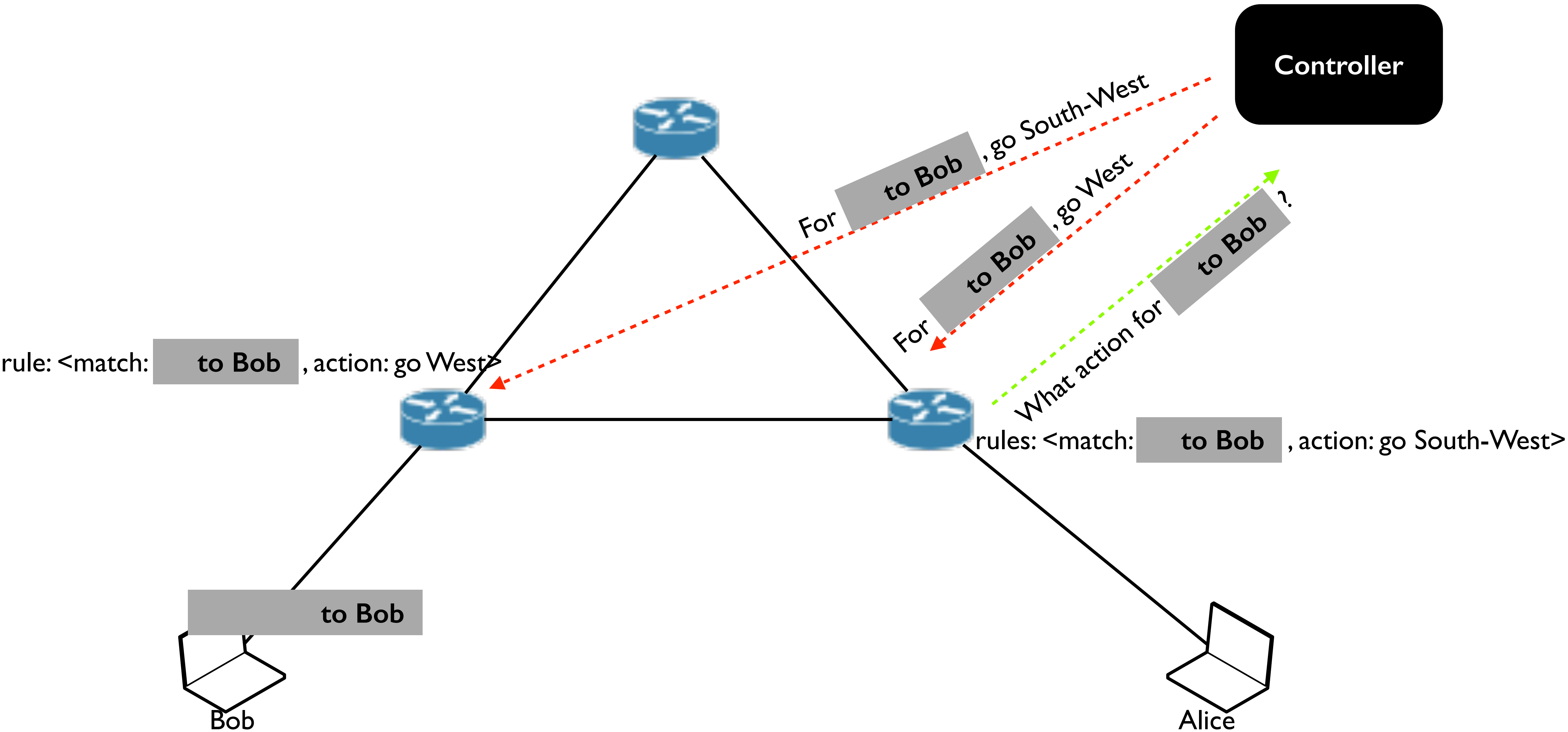
# OpenFlow workflow



# OpenFlow workflow



# OpenFlow workflow



# The OpenFlow Rules Placement Problem

# State of the art

- DevoFlow [2], DomainFlow [11], SwitchReduce [5]: aggressively use wildcard rules to minimise rule space consumption
- DIFANE [16], vCRIB [10]: cache important rules on additional devices
- Palette [8], OneBigSwitch [7]: network-wide optimisation, predefine the paths based on routing policy and place rules along these paths

# State of the art

- DevoFlow [2], DomainFlow [11], SwitchReduce [5]: aggressively use wildcard rules to minimise rule space consumption

**Isn’t that a bit too network’ish?**

- Palette [8], OneBigSwitch [7]: network-wide optimisation, predefine the paths based on routing policy and place rules along these paths

# Assumptions

- There exists one default point where packets can always be sent
  - e.g., OpenFlow controller, default egress point.
- Each switch knows how to reach this point
  - the path to the point is called the default path.
  - but all packets should be delivered to their appropriate endpoint instead of the default point.

# $\mathcal{O}\left(|F| \cdot log(|F|)\right)$ greedy heuristic

**INPUT:** flow weights collection $W : F \times E \rightarrow \mathbb{R}_+$, set of network switches $S$, set of links $L^+$, set of default path per flow $DefaultPath$, a default path is a set of switches, annotated with a rank, on the path towards the controller.
**OUTPUT:** $A$, a $|F|$-by-$|L^+|$ binary matrix

```
A ← [0]_{F.L+}
M ← sort(W, descending)
for all (f, e) ∈ M do
    sequence ← sort(DefaultPath(f), ascending)
    for all s ∈ sequence do
        if canAllocate(A, f, e, s) then
            allocate(A, f, e, s)
            break
```

Try most promising flows first.

Try most promising deflection point first.

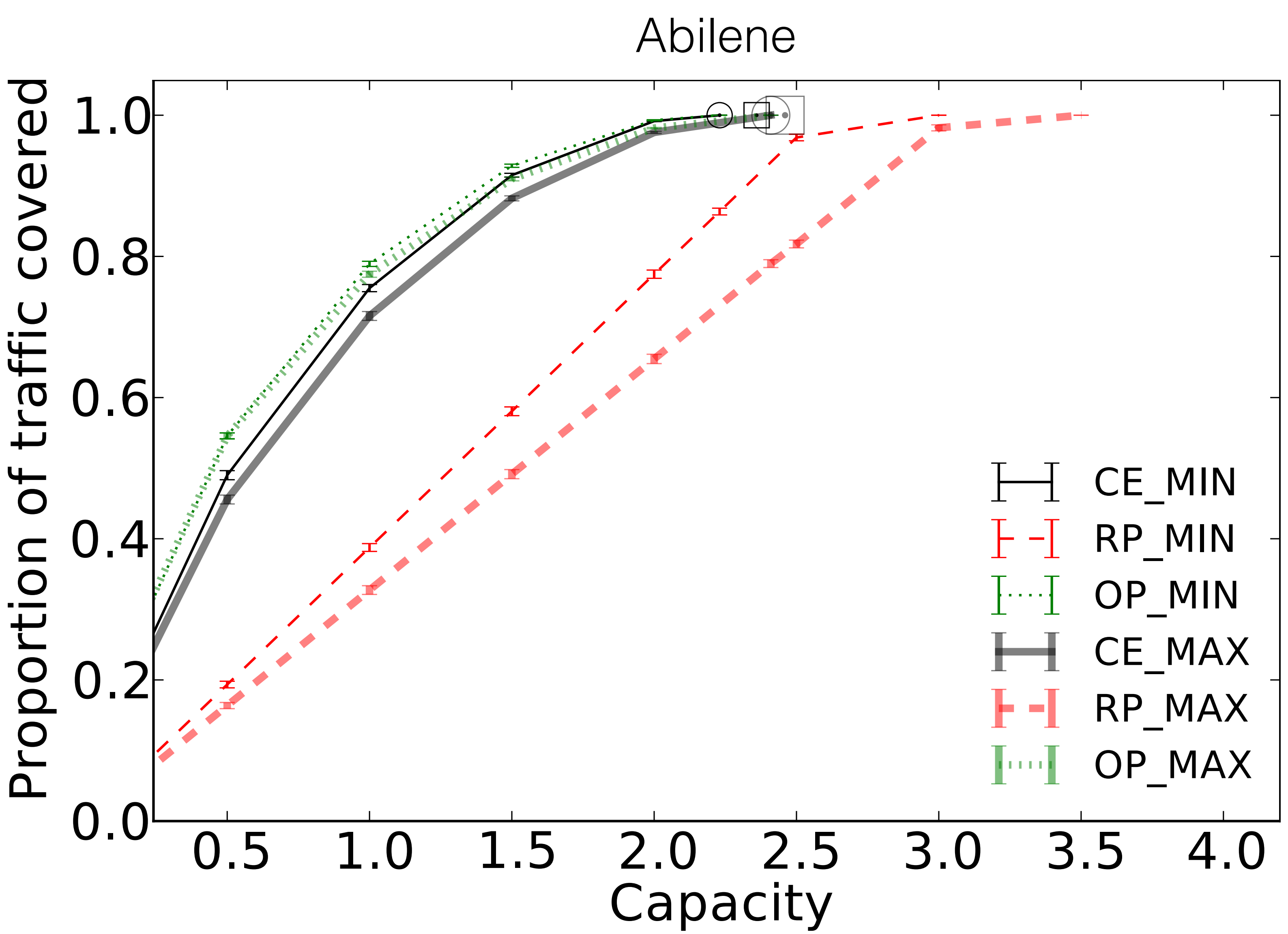
# Evaluation setup

- Numerical evaluation.
- Scenario: Machine-to-machine communications.
- Topologies:
  - ISP (Abilene with 12 nodes; scale free with 100 nodes).
  - Data center (8-fatTree with 80 nodes; 16-fatTree with 320 nodes).
- Workloads: 24 hours workloads generated by traffic generators [15][16].
- Focus on the impact of memory ( $B_l = \infty$)
  - uniform distribution of memory.

# Evaluation setup (contd.)

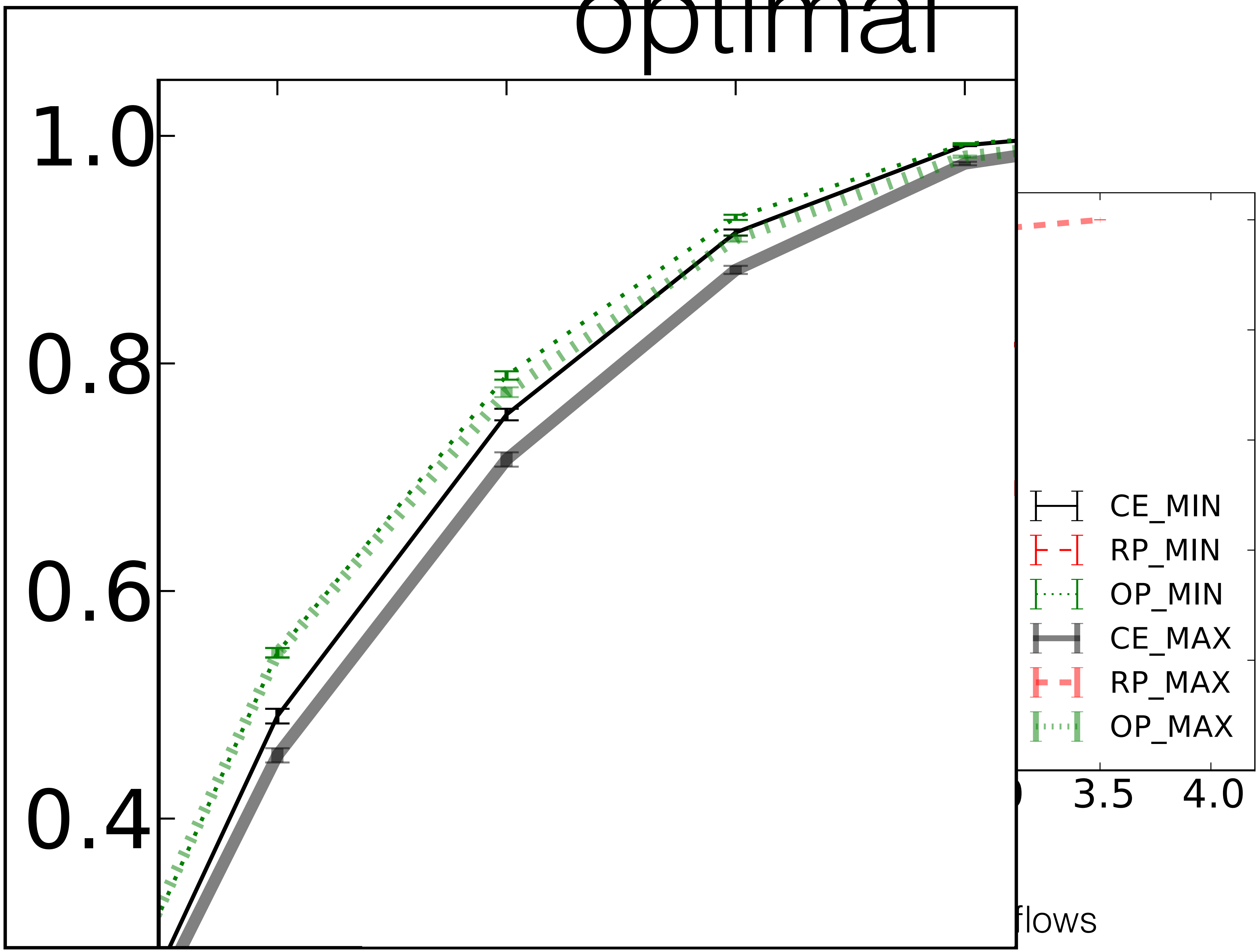
- Evaluated 3 rule placement algorithms
  - Optimum (OP),
  - Heuristic (CE),
  - Random placement (RP);
- and 2 controller placement techniques
  - Most centralised (MIN),
  - Least centralised (MAX).

# Greedy algorithm is close to optimal

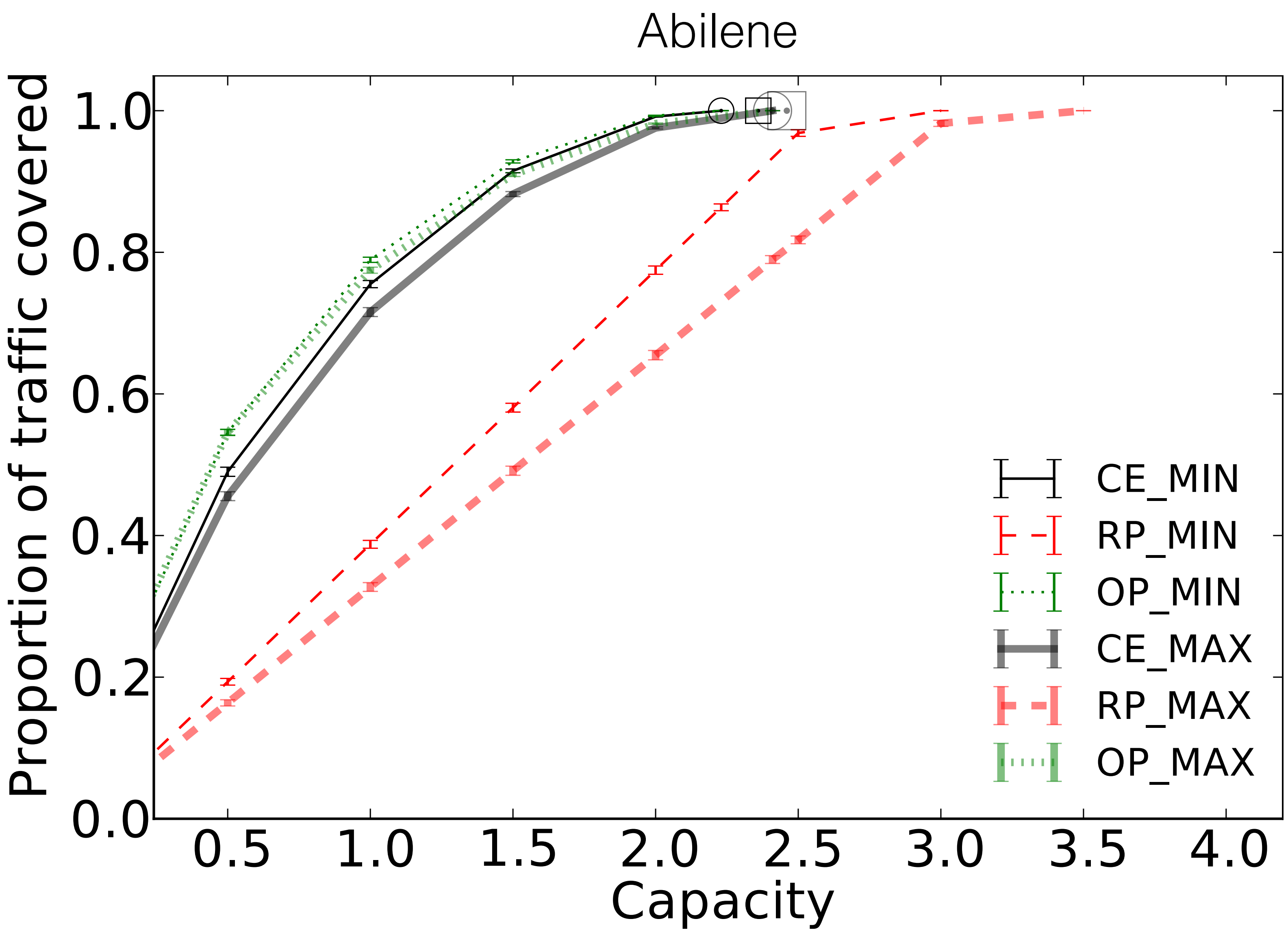



Capacity = # of entries / # of flows

# Greedy algorithm is close to optimal

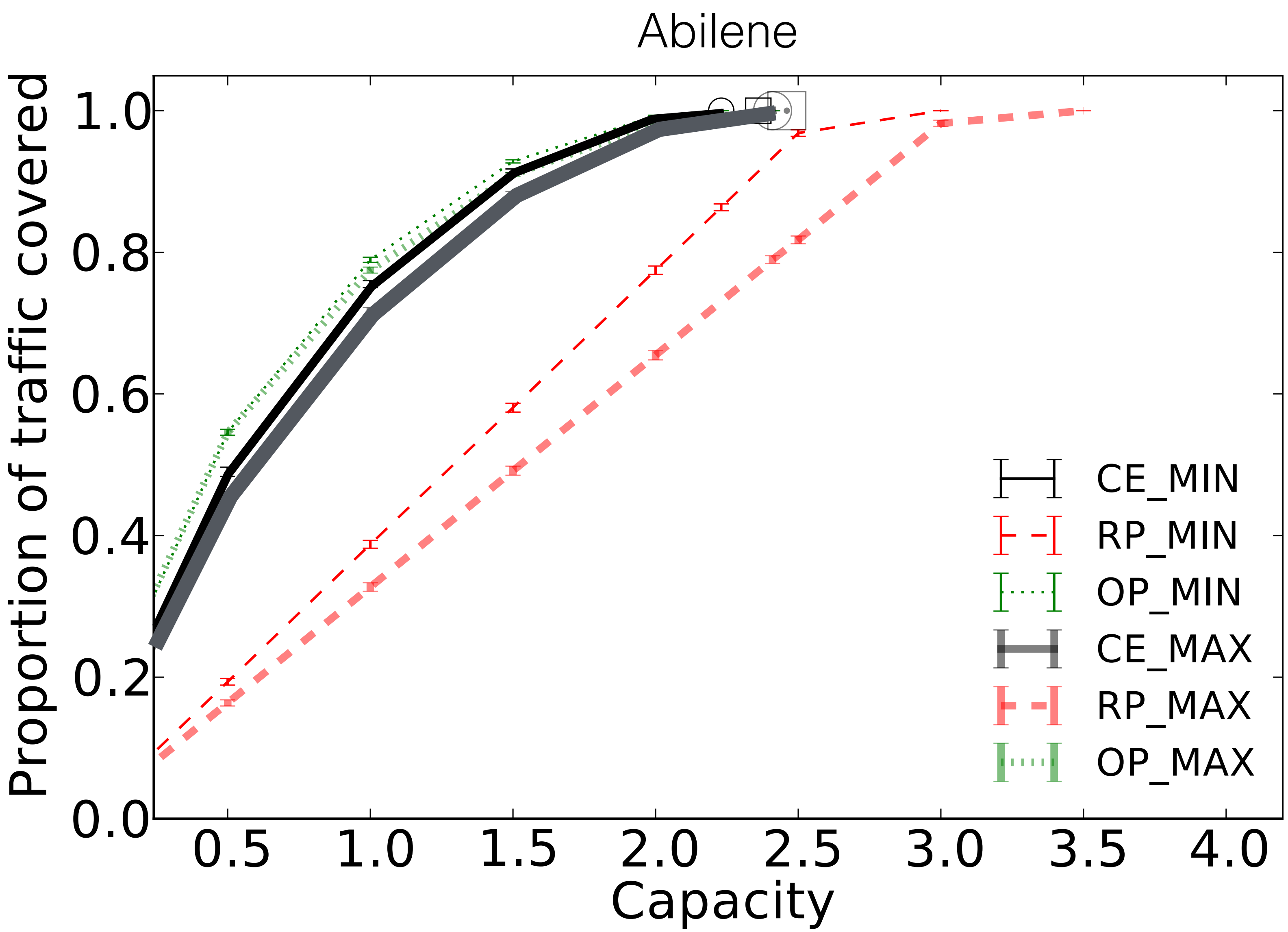


# Controller location has an impact



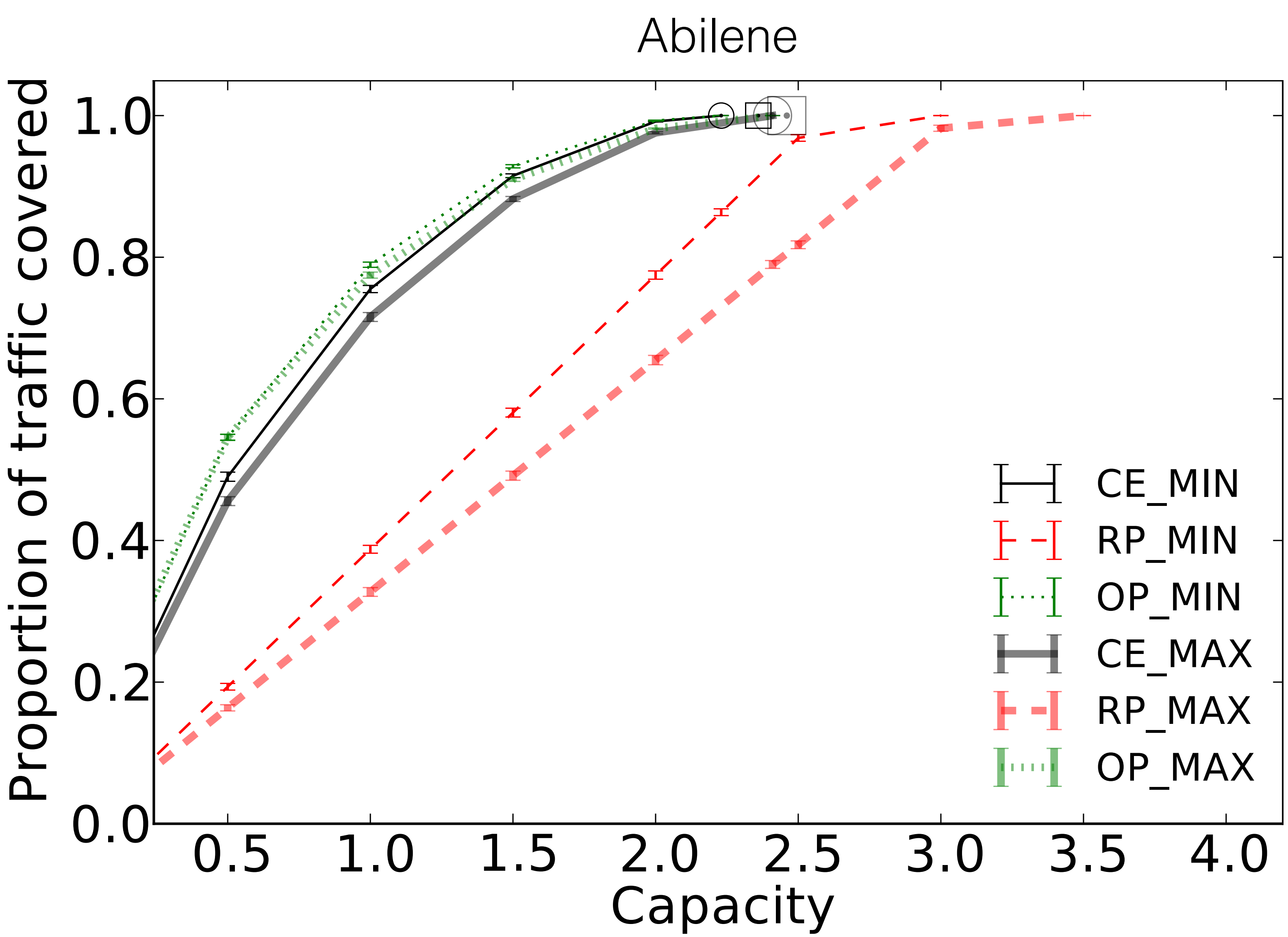

Capacity = # of entries / # of flows

# Controller location has an impact



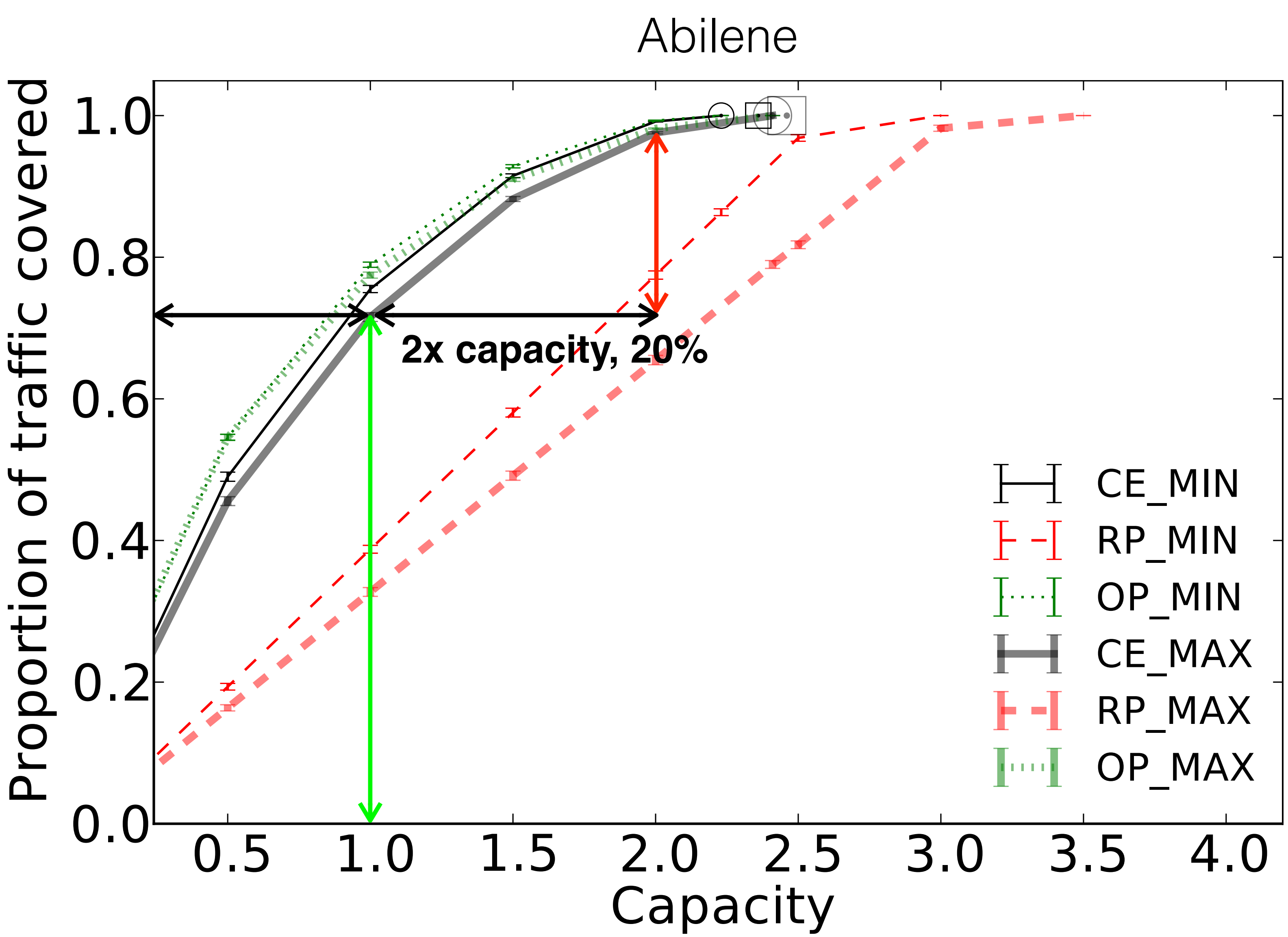

Capacity = # of entries / # of flows

# Marginal gain of increasing memory decreases with the total memory



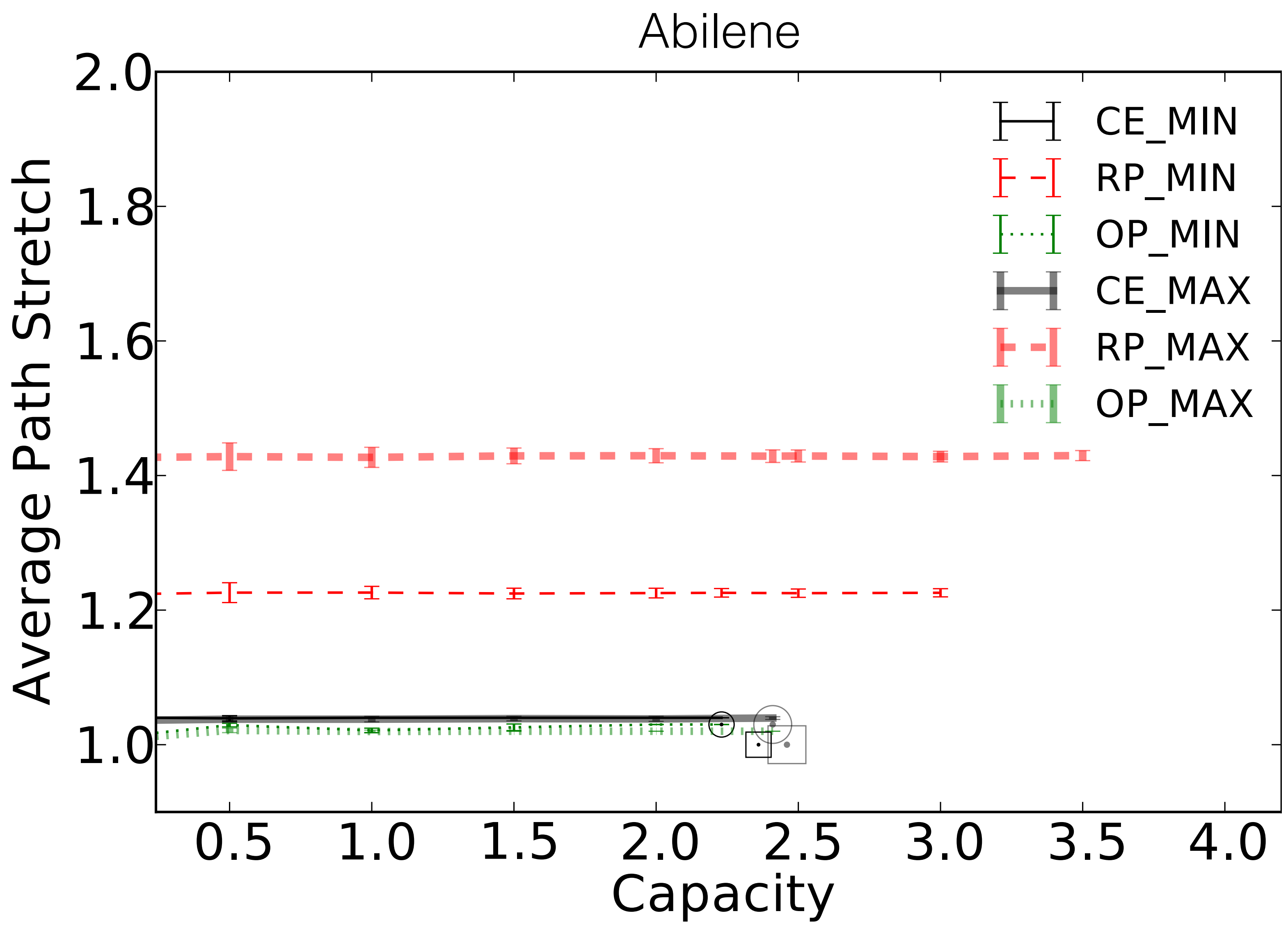

Capacity = # of entries / # of flows

# Marginal gain of increasing memory decreases with the total memory



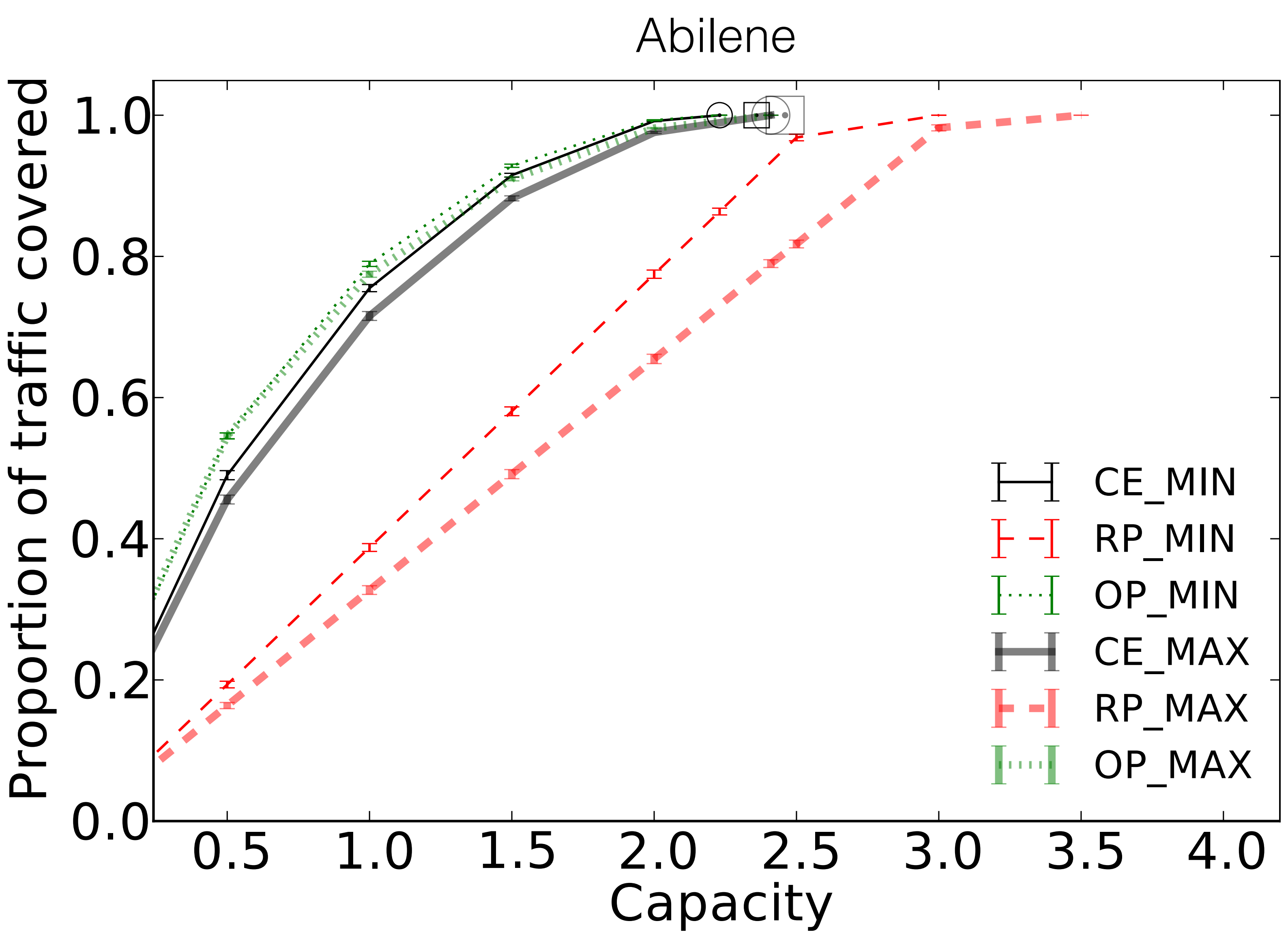

Capacity = # of entries / # of flows

# Path stretch is reasonable



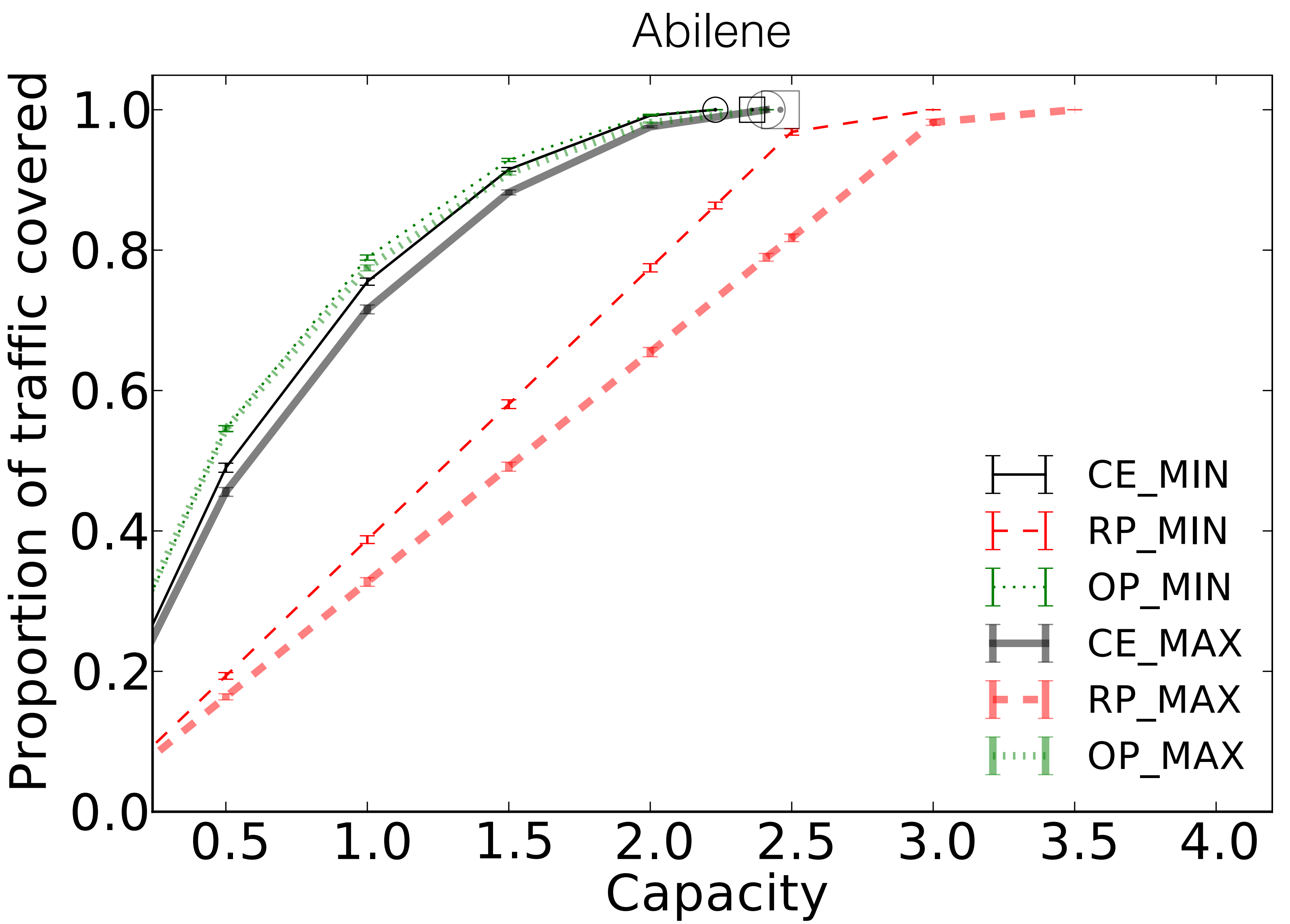

Capacity = # of entries / # of flows

# Traffic satisfaction vs memory



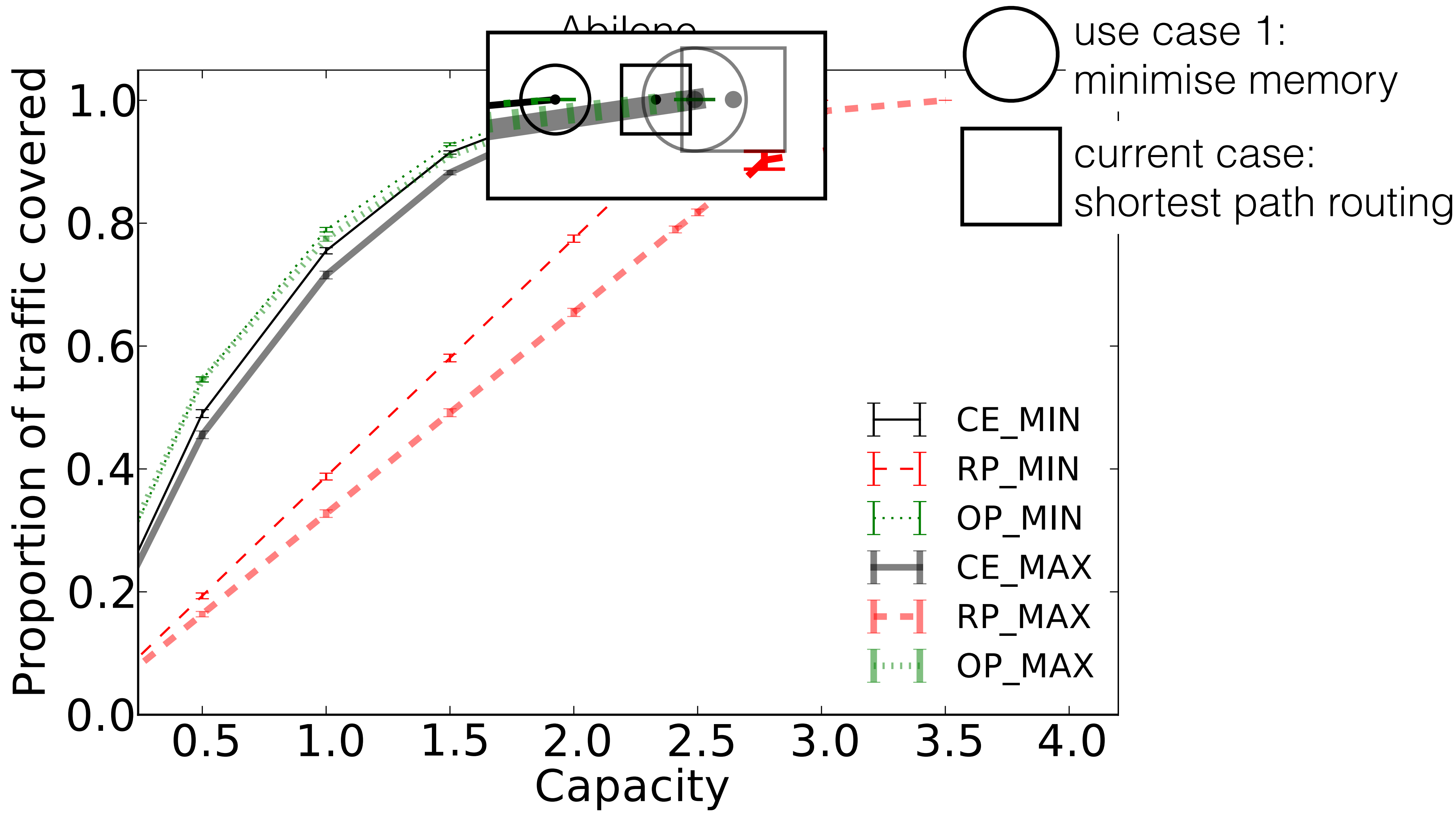

Capacity = # of entries / # of flows

# Trading routing reduces memory consumption




Capacity = # of entries / # of flows

# Trading routing reduces memory consumption




Capacity = # of entries / # of flows

# … and reluctant to changes

- Middleboxes are everywhere  [SHC+12]

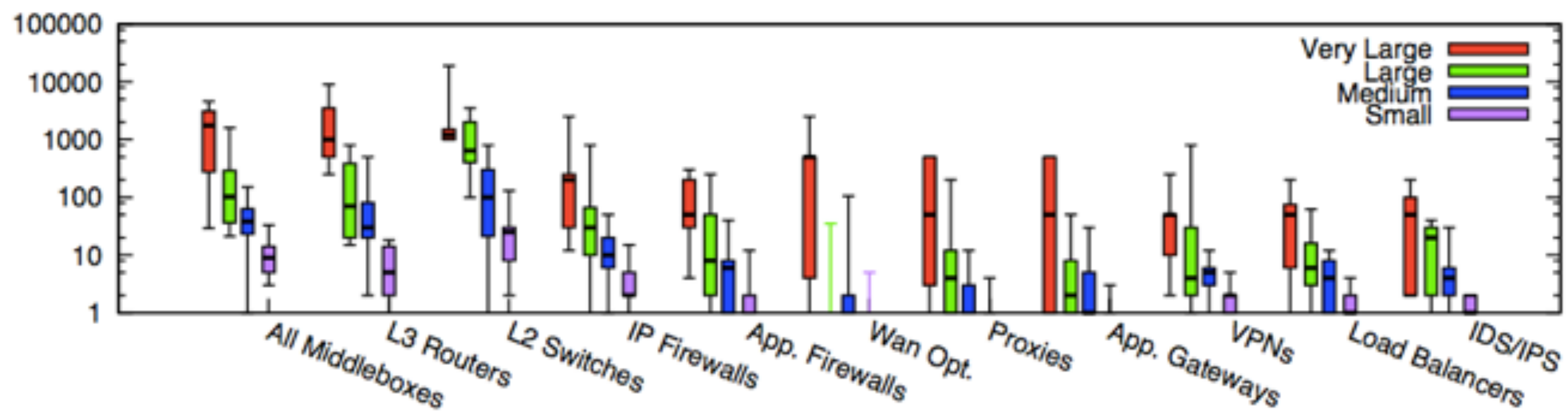



**Figure 1: Box plot of middlebox deployments for small (fewer than 1k hosts), medium (1k-10k hosts), large (10k-100k hosts), and very large (more than 100k hosts) enterprise networks. Y-axis is in log scale.**

  - very likely that your packet will be touched by a middlebox before reaching its destination [HNR+11],
  - Middleboxes limit deployment of new protocols in the Internet [HNR+11].
- Middleboxes can be used against user interests.



[HNR+11] Honda et al., Is it Still Possible to Extend TCP?
[SHC+12] Sherry et al. 2012. Making middleboxes someone else's problem: network processing as a cloud service

# Methodology

- Observe:
  - scrutinise for operational networking problems.
- Generalisation:
  - what is the general problem hidden behind it? Find the root-cause of the problem.
- Solve:
  - design a solution that is as efficient as possible and that can work in practice.
- Validate:
  - experiment the solution with real deployment whenever possible.
- Impact:
  - proof of concept in conferences/workshops followed by complete study in journals; standardisation and industrial transfers when relevant.