Algorithms for Traffic Control Challenges in Softwarized Networks

Mahmoud Parham

28.11.2022

Thesis in a Nutshell

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.
- VNF life cycle is managed by special software.

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.
- VNF life cycle is managed by special software.
- **Softwarized Networking = SDN + NFV**; enables flexible **traffic control**

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.
- VNF life cycle is managed by special software.
- **Softwarized Networking = SDN + NFV**; enables flexible **traffic control**

Thesis in a Nutshell

- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.
- VNF life cycle is managed by special software.
- **Softwarized Networking = SDN + NFV**; enables flexible **traffic control**

- **Macro Goal**: adapting to "events" autonomously (algorithmically)
    1. Detecting Events
    2. <u>Making Decisions</u> => combinatorial problems
    3. Executing Decisions

Thesis in a Nutshell

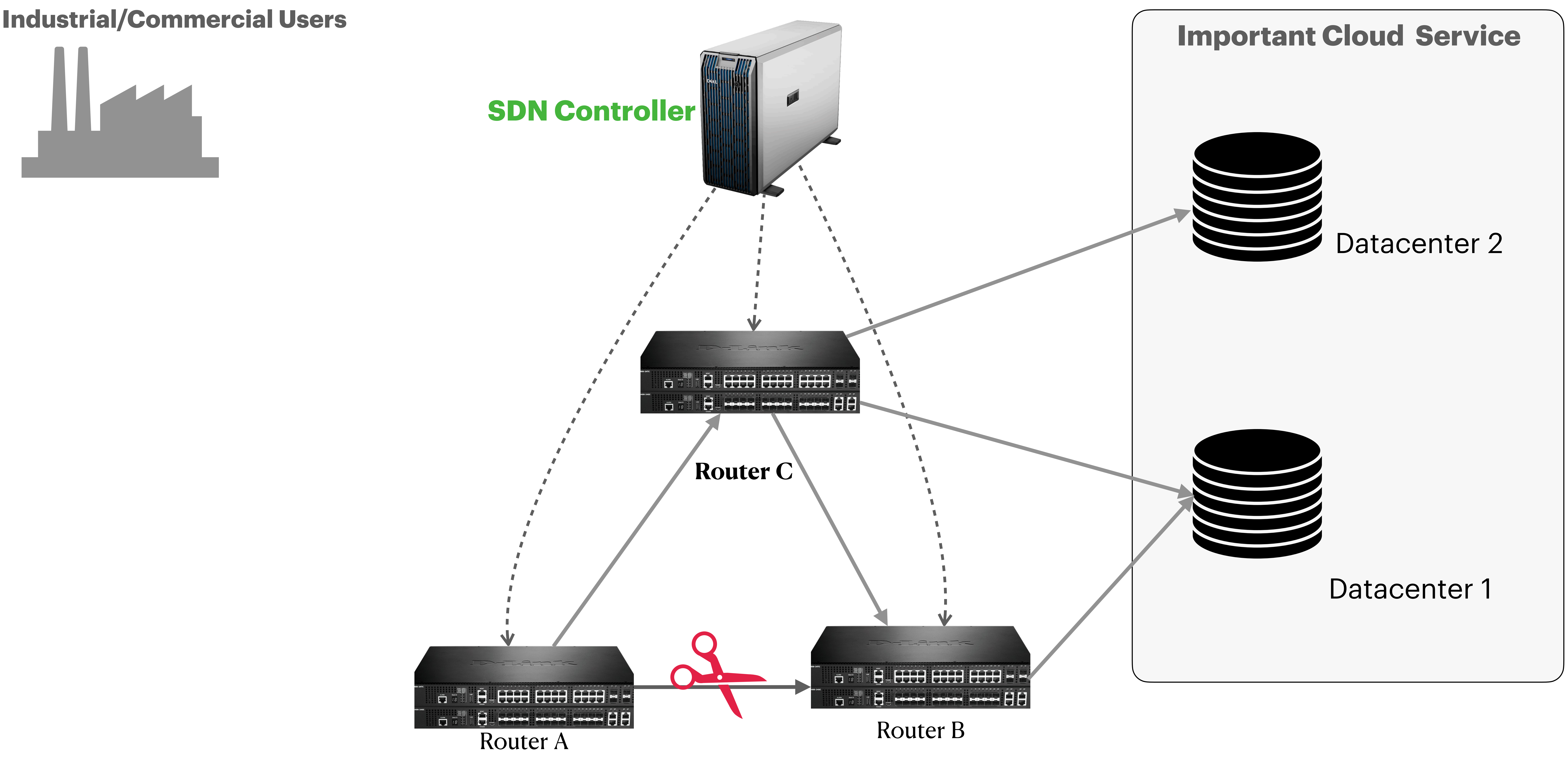
- **SDN**: a central controller (software) makes and executes routing decisions at runtime
- **NFV**: emulating network functions (e.g. firewall, DNS…) in software units, called **VNF**

- SDN enables flexible traffic **steering** using e.g. segment routing**.**
- NFV enables flexible **placement** of traffic endpoints.
- VNF life cycle is managed by special software.
- **Softwarized Networking = SDN + NFV**; enables flexible **traffic control**

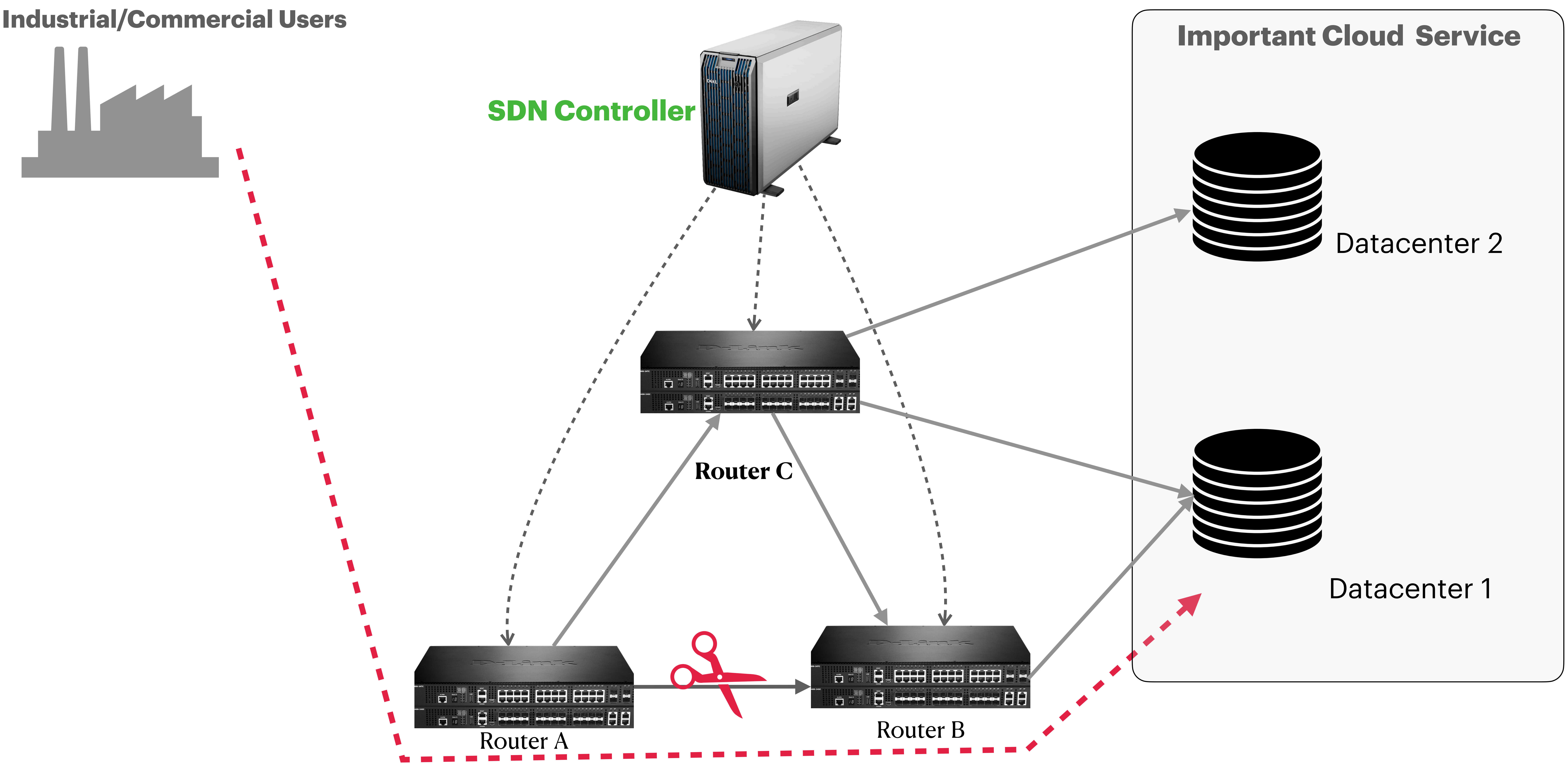
- **Macro Goal**: adapting to “events” autonomously (algorithmically)
  1. Detecting Events
  2. <u>Making Decisions</u> => combinatorial problems
  3. Executing Decisions

Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges. Kellerer, W.; Kalmbach, P.; Blenk, A.; Basta, A.; Reisslein, M.; and Schmid, S. *Proceedings of the IEEE*, 107(4): 711–731. April 2019.

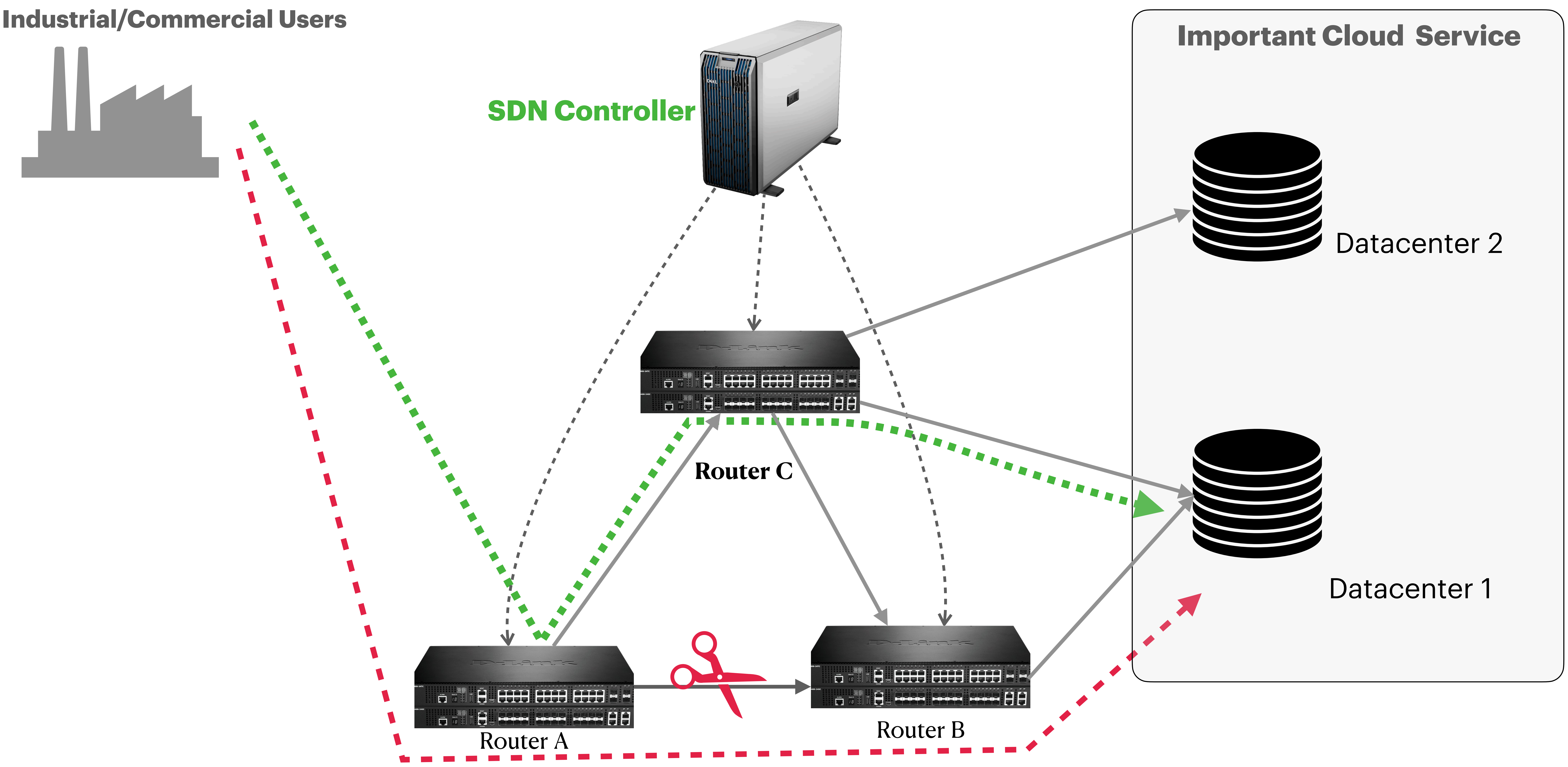
Softwarized Traffic Control



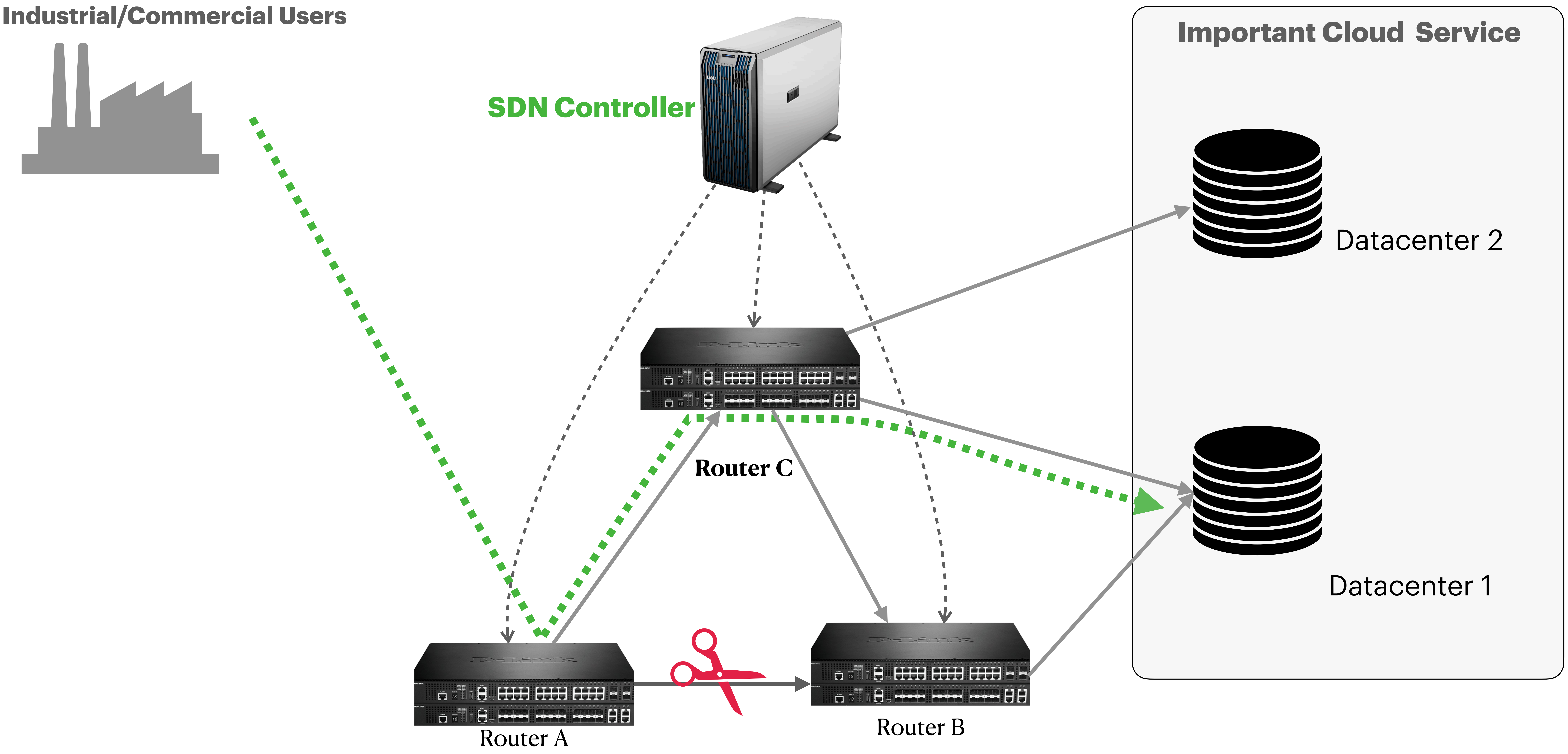
Softwarized Traffic Control



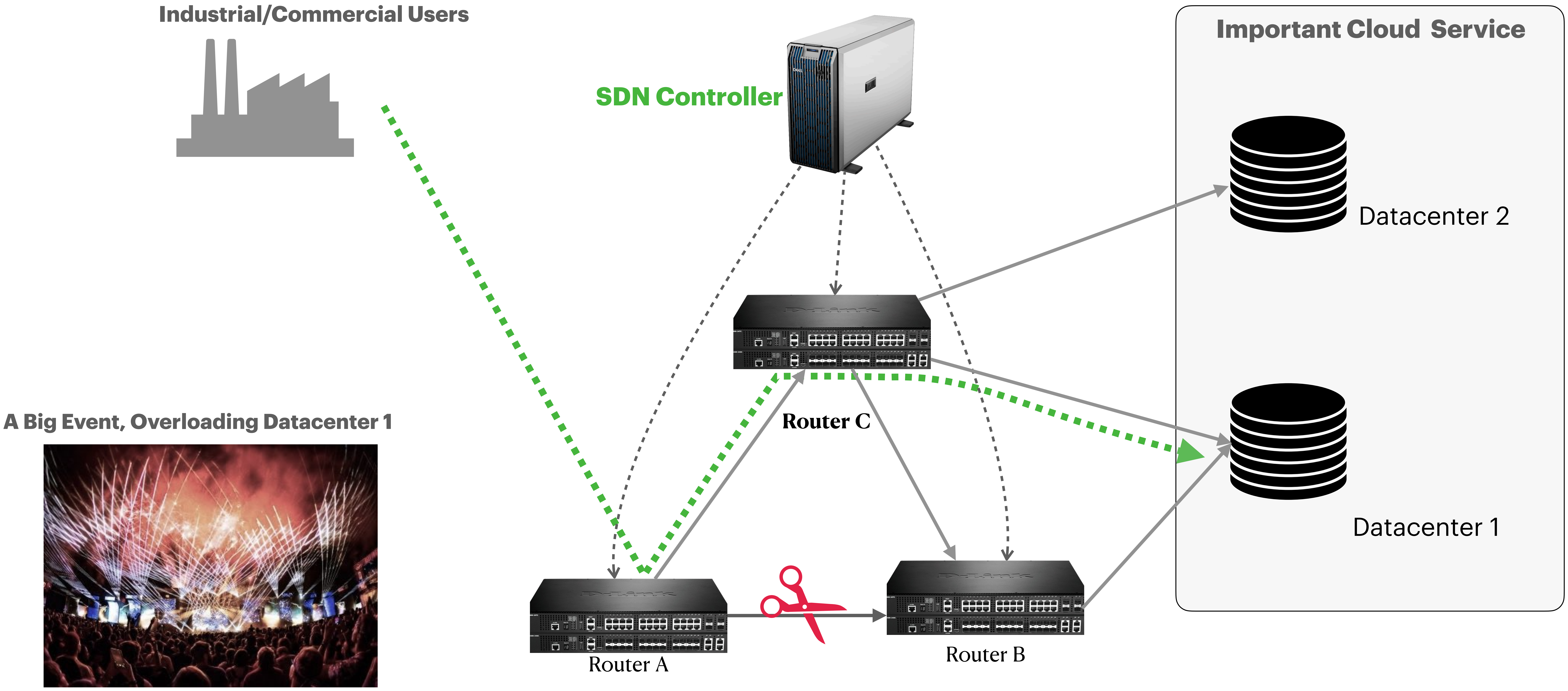
Softwarized Traffic Control



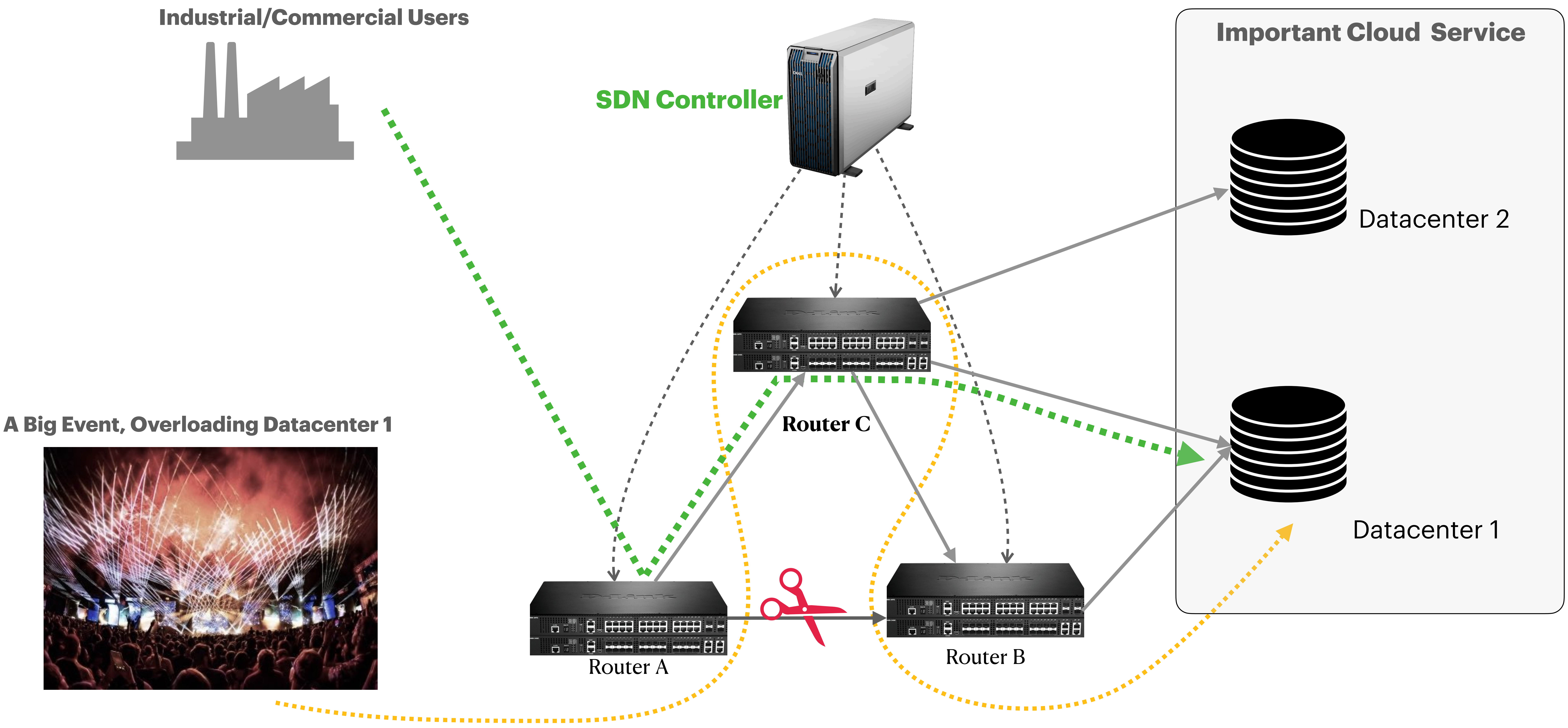
Softwarized Traffic Control



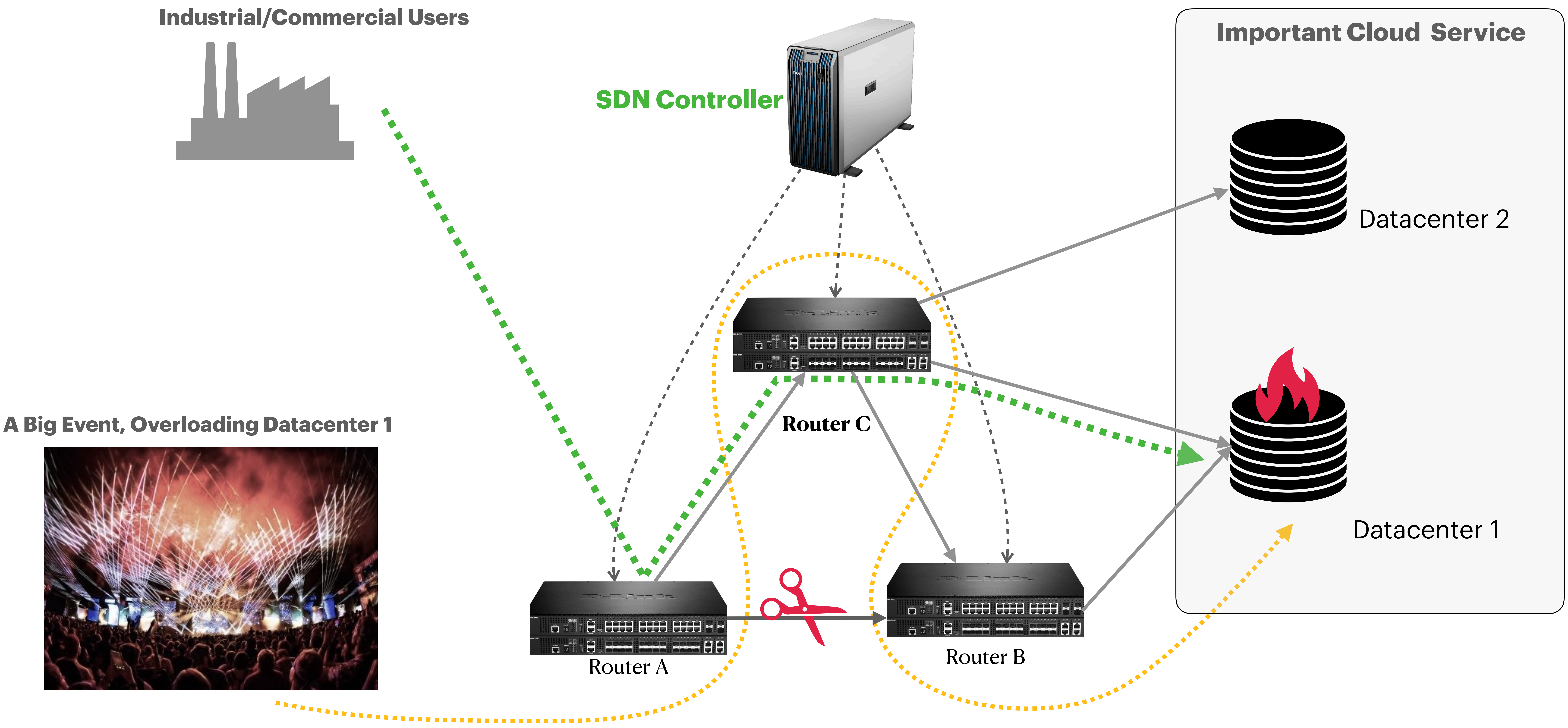
Softwarized Traffic Control



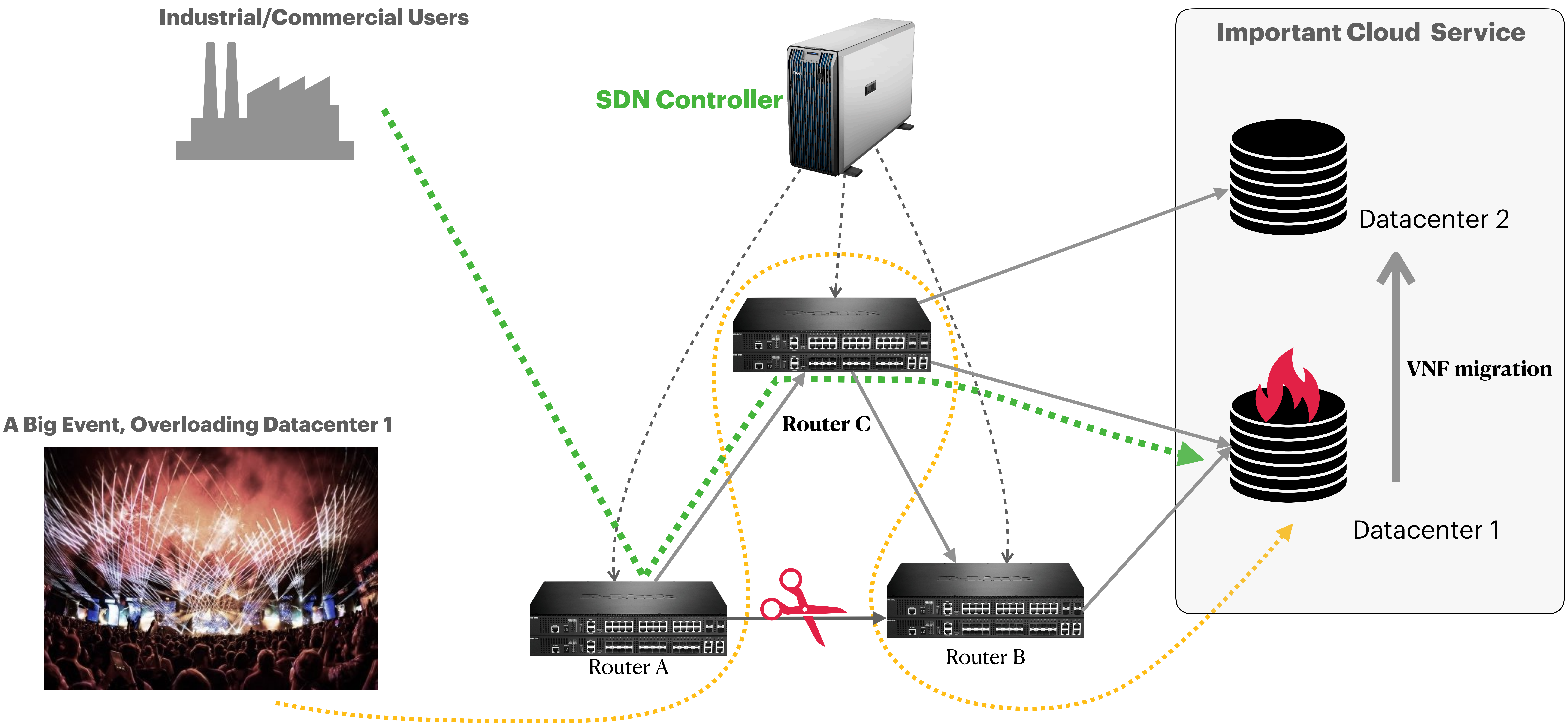
Softwarized Traffic Control



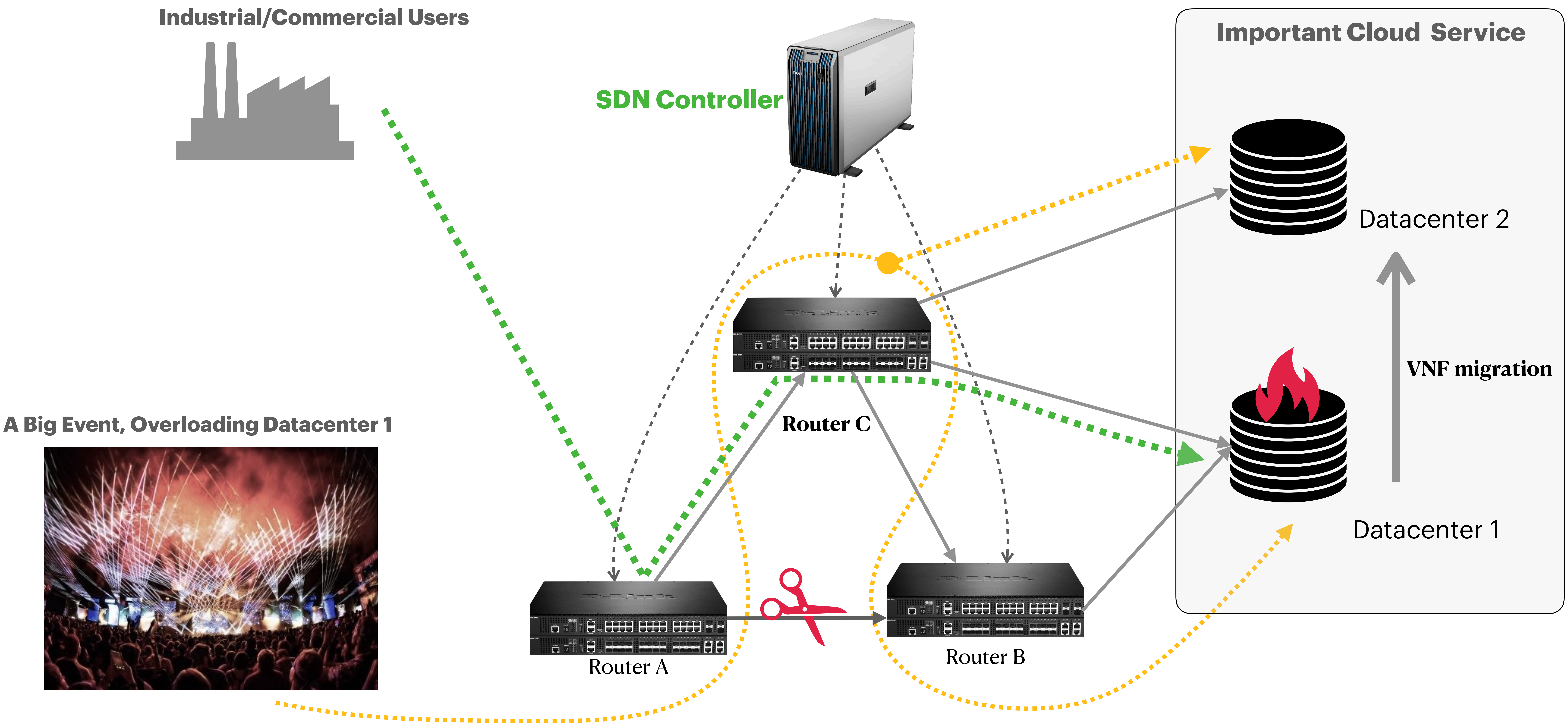
Softwarized Traffic Control



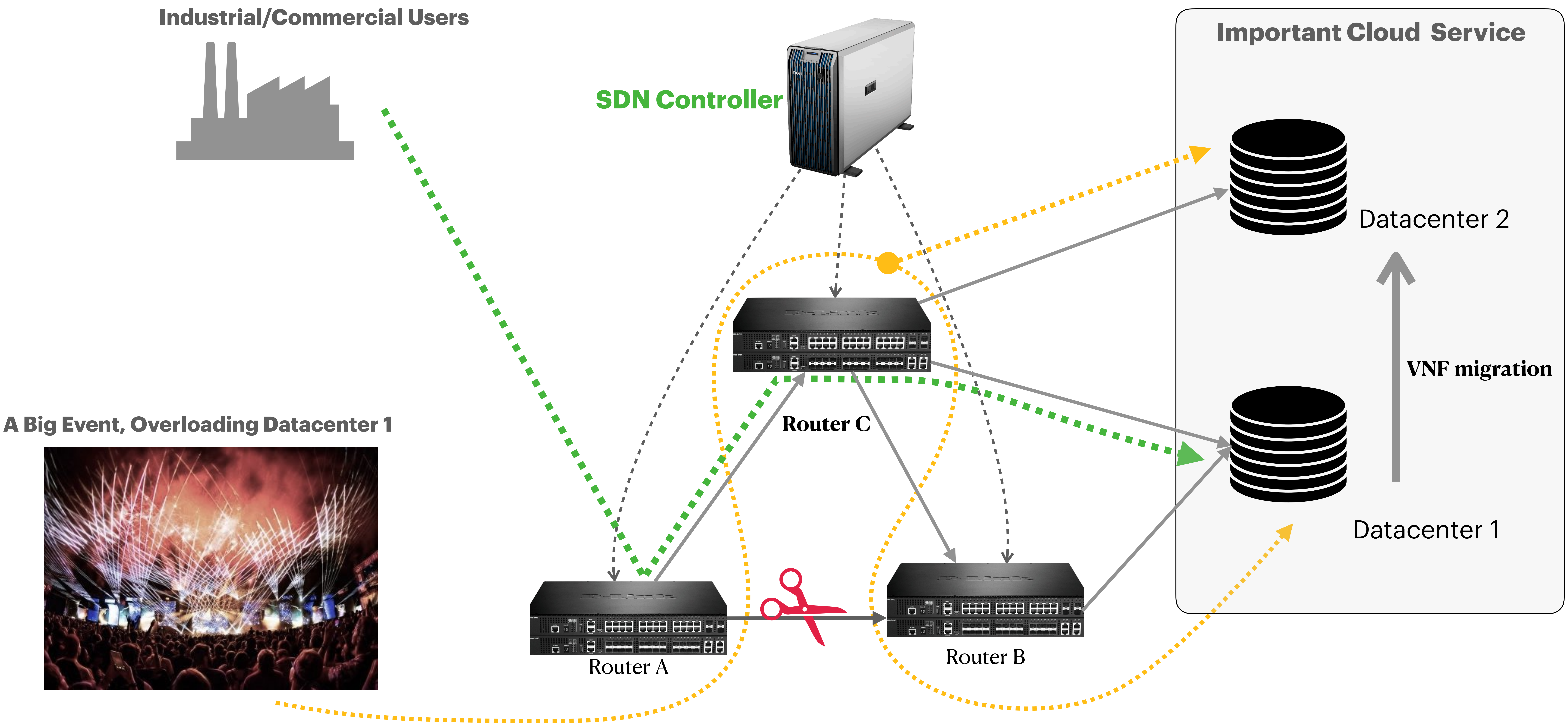
Softwarized Traffic Control



Softwarized Traffic Control

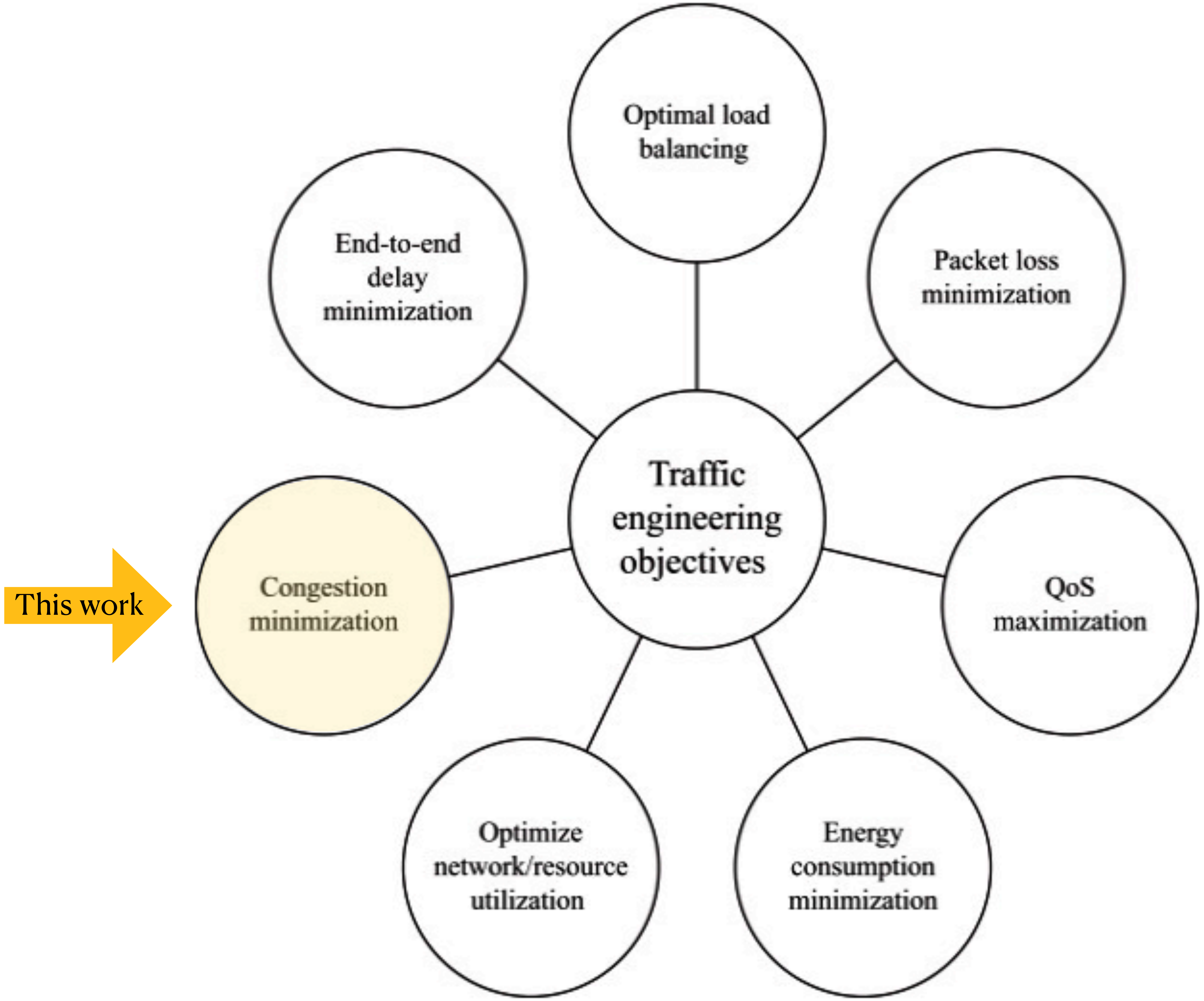


Softwarized Traffic Control



- **Traffic Engineering**
- Fast Traffic Rerouting
- Traffic Partitioning

Traffic Engineering (TE)



Traffic Engineering (TE)

Traffic Engineering (TE)

- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).

Traffic Engineering (TE)

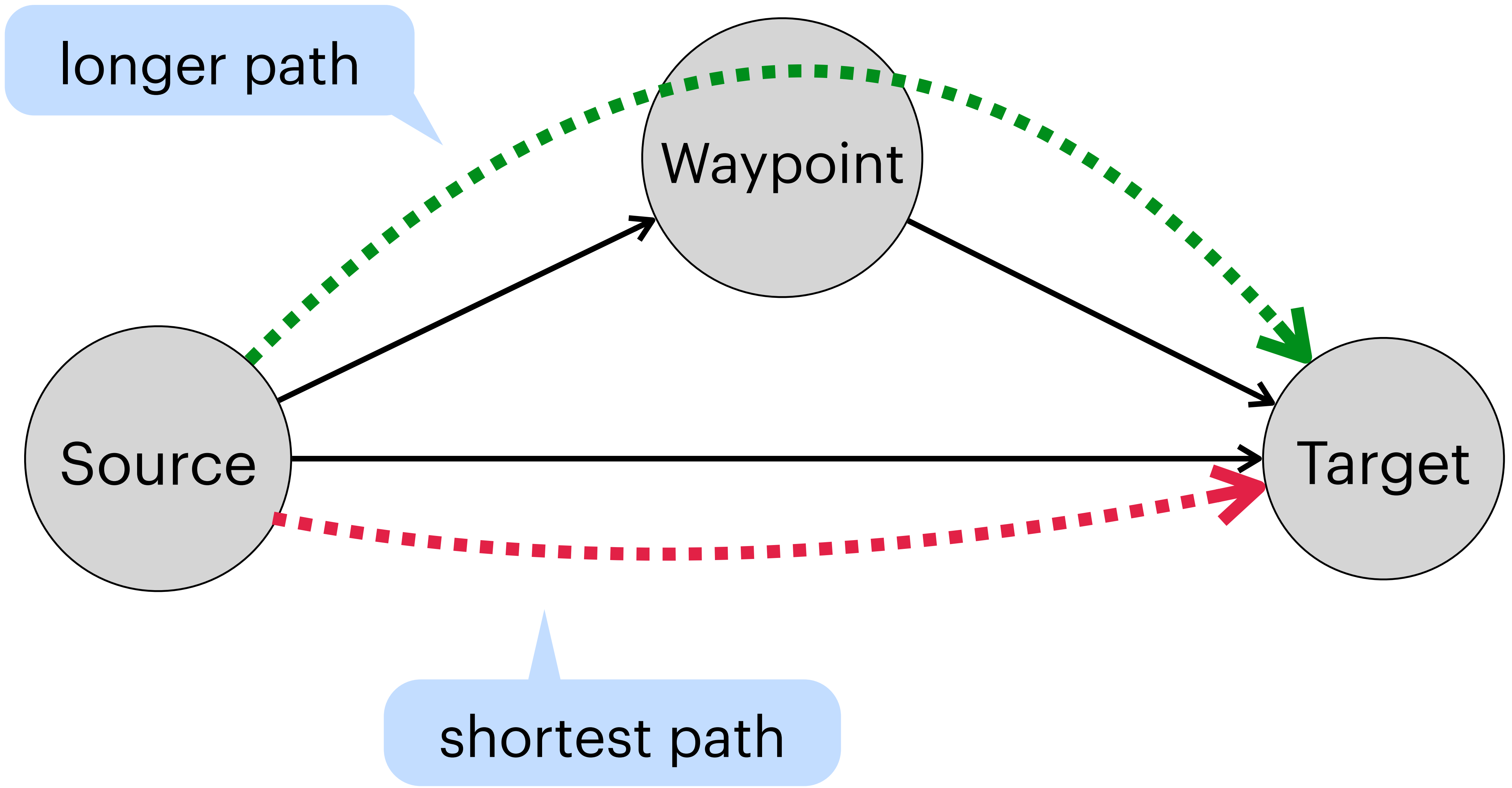
- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).
- **TE**: tricking the network to route a flow differently from its default (SP) routing

Traffic Engineering (TE)

- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).
- **TE**: tricking the network to route a flow differently from its default (SP) routing
- **Technique A:** tuning link weights
    - affects all paths indiscriminately; too blunt!

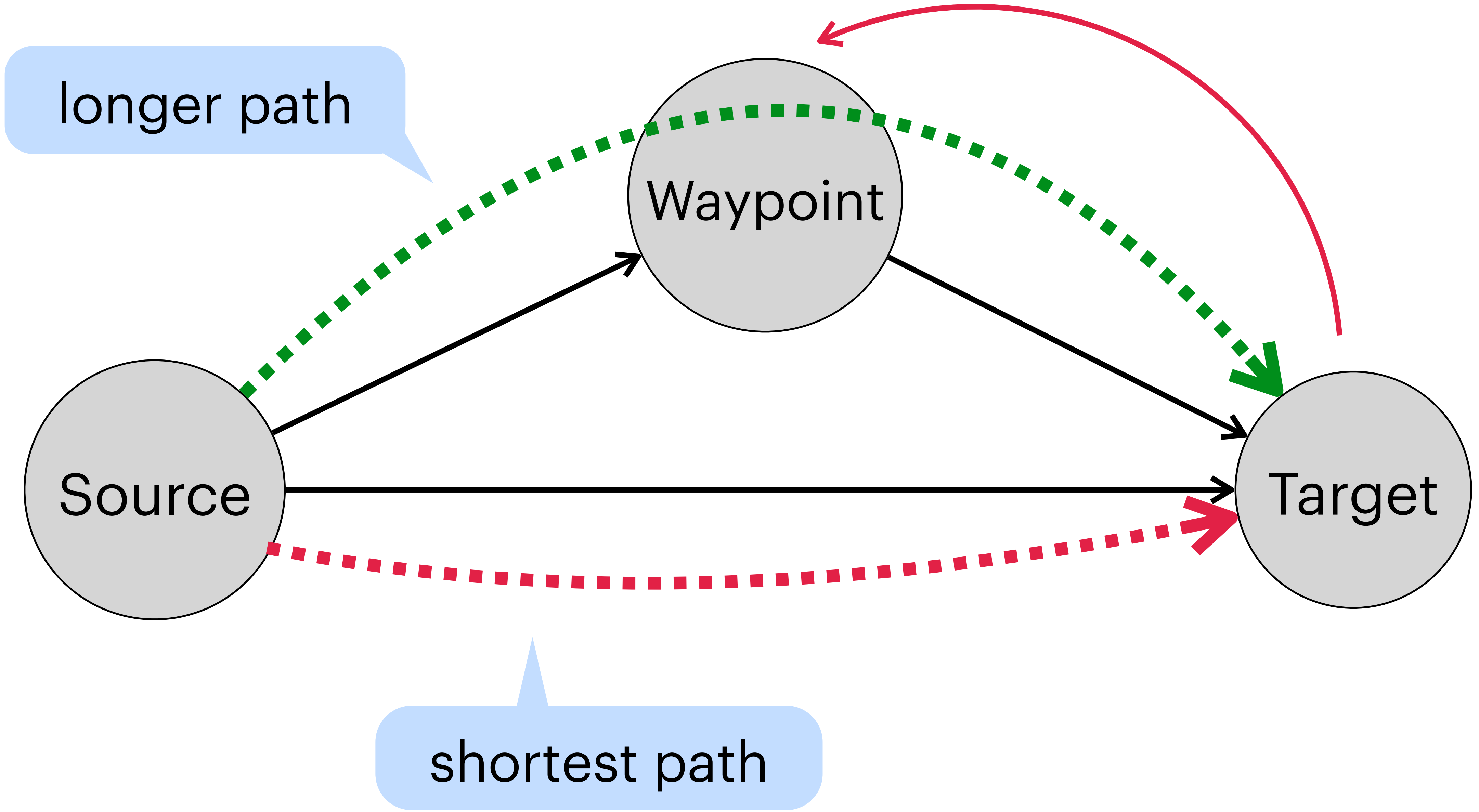
Traffic Engineering (TE)

- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).
- **TE**: tricking the network to route a flow differently from its default (SP) routing
- **Technique A:** tuning <u>link weights</u>
  - affects all paths indiscriminately; too blunt!
- **Technique B:** routing via intermediate nodes, or <u>waypoints</u> (segment routing)
  - requires appropriate link weights
  - ineffective under arbitrary weights



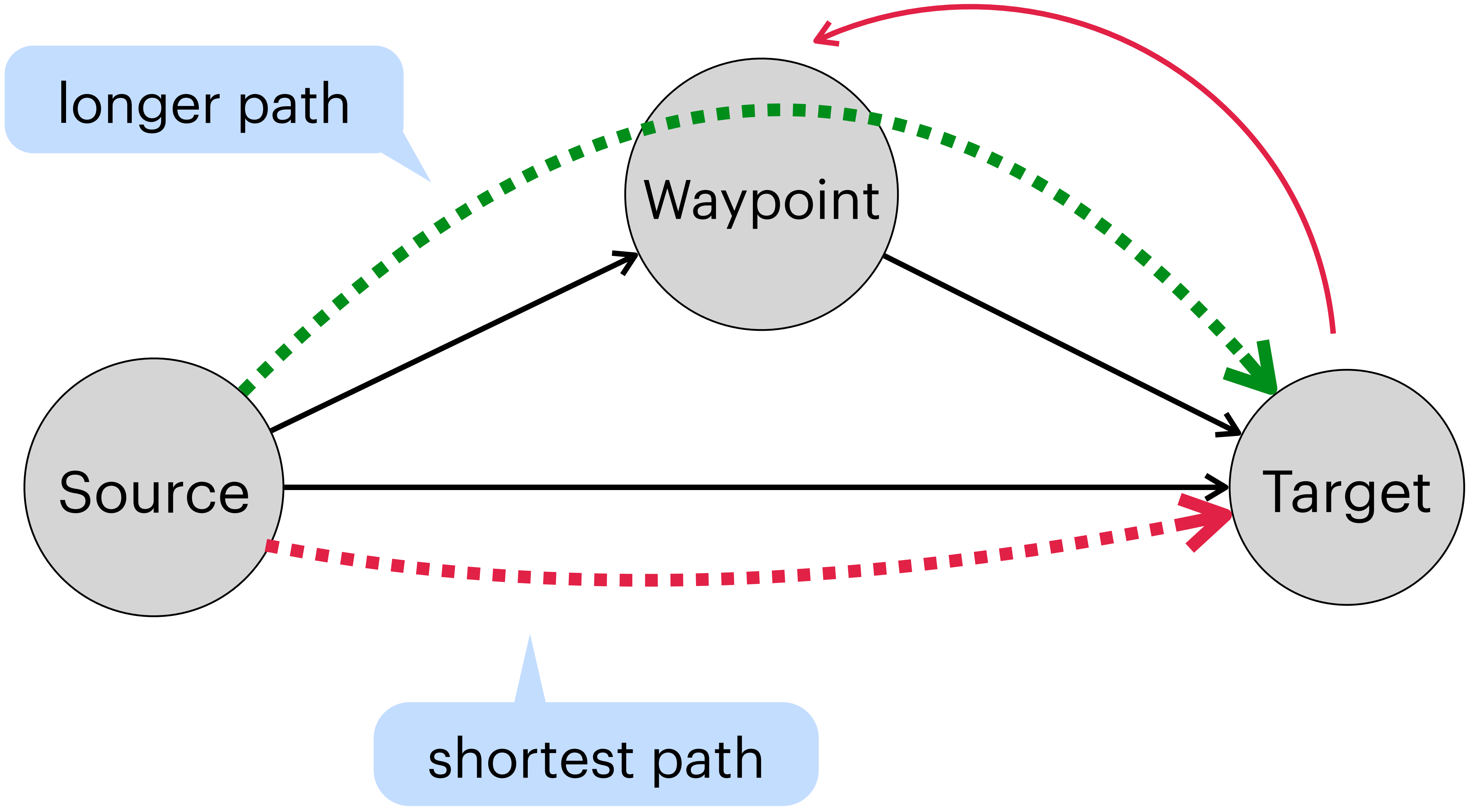
Traffic Engineering (TE)

- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).
- **TE**: tricking the network to route a flow differently from its default (SP) routing
- **Technique A:** tuning link weights
  - affects all paths indiscriminately; too blunt!
- **Technique B:** routing via intermediate nodes, or waypoints (segment routing)
  - requires appropriate link weights
  - ineffective under arbitrary weights



Traffic Engineering (TE)

- **Shortest path:** the default routing, is oblivious to link capacities (e.g., OSPF).
- **TE**: tricking the network to route a flow differently from its default (SP) routing
- **Technique A:** tuning link weights
  - affects all paths indiscriminately; too blunt!
- **Technique B:** routing via intermediate nodes, or waypoints (segment routing)
  - requires appropriate link weights
  - ineffective under arbitrary weights
- **R.Q.**: can we combine A and B for a more fine-grained TE?



The Joint Optimization

- Link Weight Optimization (**LWO**): find link weights minimizing MLU
- Waypoint Optimization (**WPO**): find waypoints (per flow) minimizing MLU
- **JOINT**: find a link-weight-waypoint setting minimizing MLU

$$\text{Link Utilization} = \frac{load}{capacity}$$

MLU = maximum link utilization over all links

The Joint Optimization

- Link Weight Optimization (**LWO**): find link weights minimizing MLU
- Waypoint Optimization (**WPO**): find waypoints (per flow) minimizing MLU
- **JOINT**: find a link-weight-waypoint setting minimizing MLU

$$\text{Link Utilization} = \frac{load}{capacity}$$

MLU = maximum link utilization over all links

JOINT $\neq$ LWO $\cup$ WPO

JOINT vs (LWO ∪ WPO)

- The **Gap** is a ratio comparing JOINT with the best of the other two.
- For any network instance I:

JOINT vs LWO

$$R_{\text{LWO}}(\mathcal{I}) := \frac{\text{LWO}(\mathcal{I})}{\text{JOINT}(\mathcal{I})}, \text{ and } \qquad R_{\text{WPO}}(\mathcal{I}) := \frac{\text{WPO}(\mathcal{I})}{\text{JOINT}(\mathcal{I})}$$

JOINT vs WPO

JOINT vs (LWO ∪ WPO)

- The **Gap** is a ratio comparing JOINT with the best of the other two.
- For any network instance I:

JOINT vs LWO

$$R_{\mathrm{LWO}}(\mathcal{I}) := \frac{\mathrm{LWO}(\mathcal{I})}{\textsc{Joint}(\mathcal{I})}, \text{ and } \quad R_{\mathrm{WPO}}(\mathcal{I}) := \frac{\mathrm{WPO}(\mathcal{I})}{\textsc{Joint}(\mathcal{I})}$$

JOINT vs WPO

the gap

$$R^* := \max_{\mathcal{I}} \min\{R_{\mathrm{LWO}}(\mathcal{I}), R_{\mathrm{WPO}}(\mathcal{I})\}$$

JOINT vs (LWO ∪ WPO)

- The **Gap** is a ratio comparing JOINT with the best of the other two.
- For any network instance I:

JOINT vs LWO

$$R_{\mathrm{LWO}}(\mathcal{I}) := \frac{\mathrm{LWO}(\mathcal{I})}{\textsc{Joint}(\mathcal{I})}, \text{ and } \qquad R_{\mathrm{WPO}}(\mathcal{I}) := \frac{\mathrm{WPO}(\mathcal{I})}{\textsc{Joint}(\mathcal{I})}$$

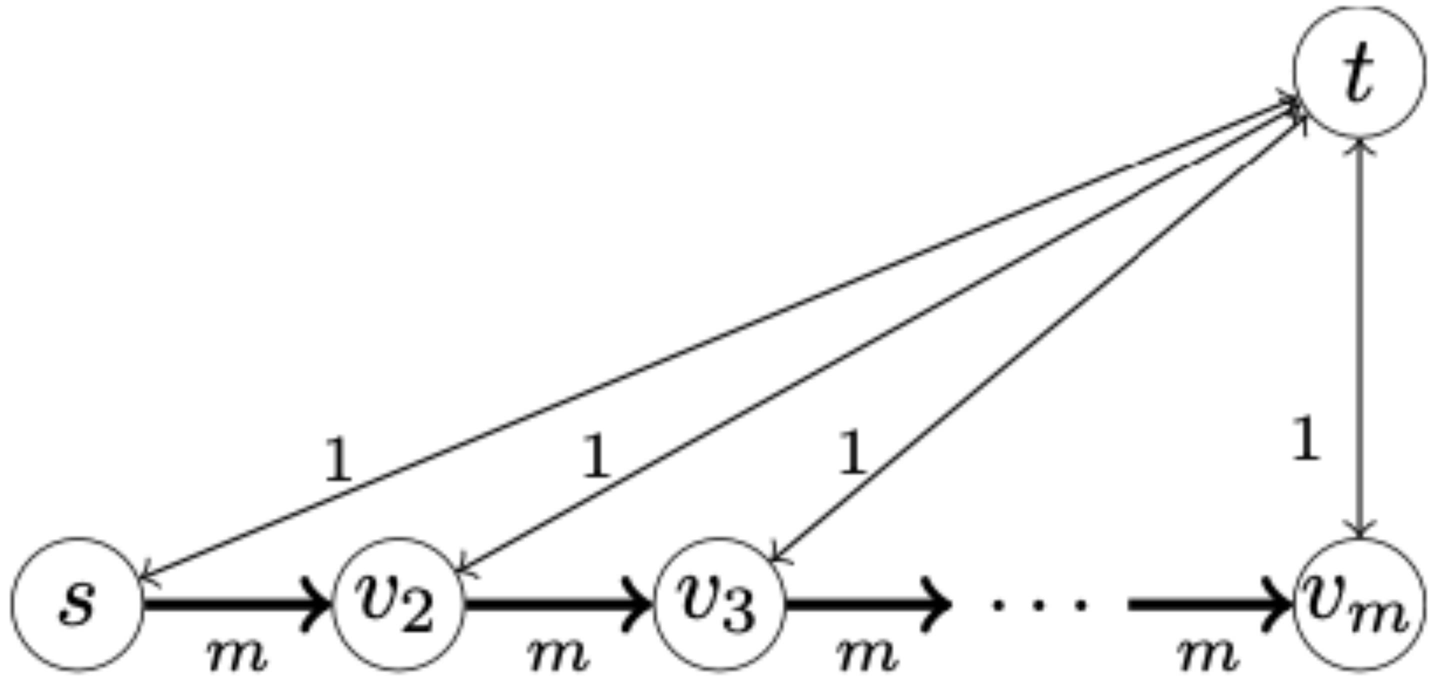
JOINT vs WPO

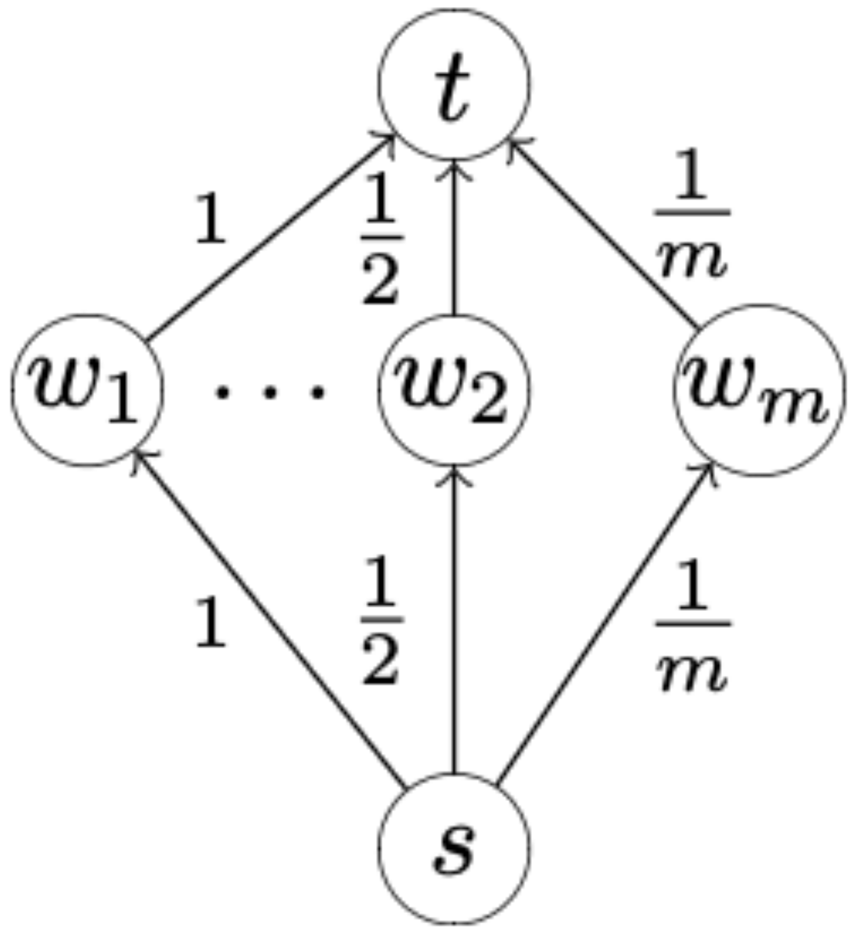
the gap

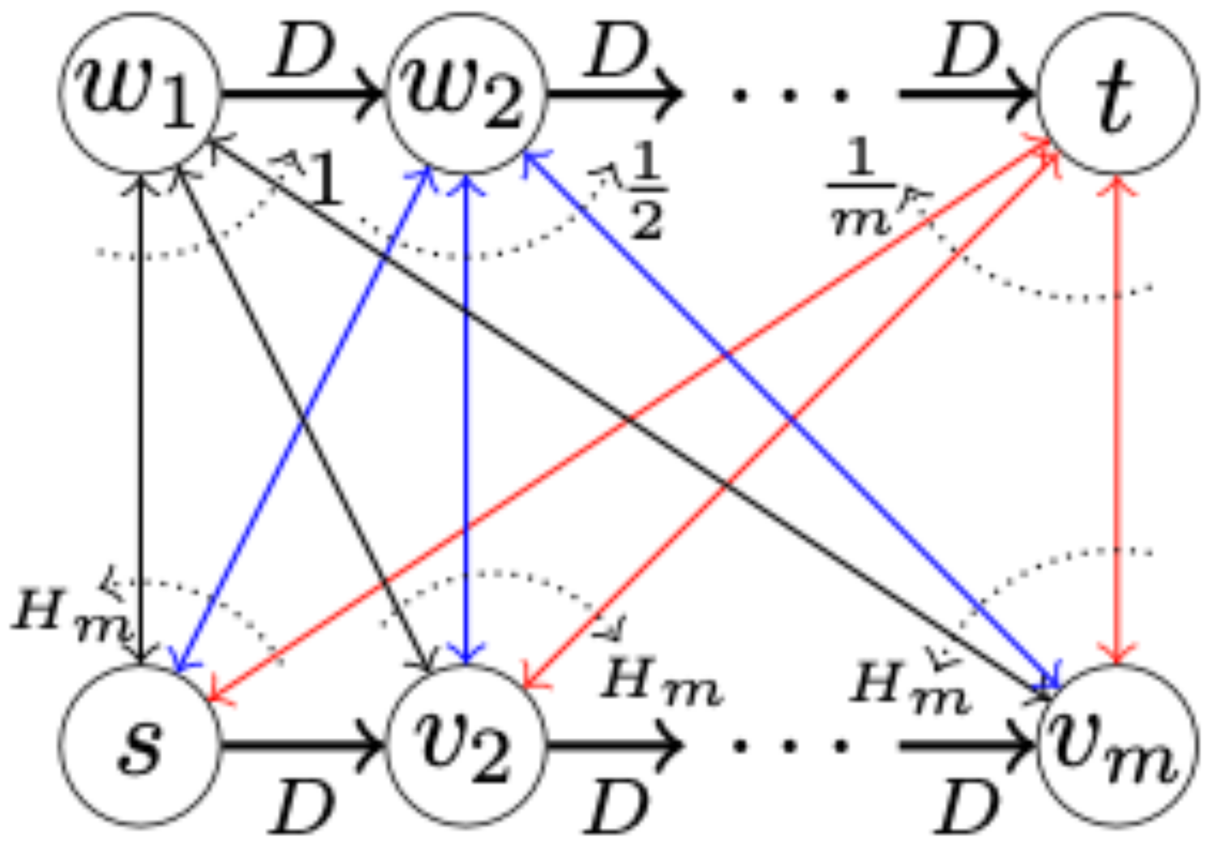
$$R^* := \max_{\mathcal{I}} \min\{R_{\mathrm{LWO}}(\mathcal{I}), R_{\mathrm{WPO}}(\mathcal{I})\}$$

- A larger gap means JOINT is more powerful than LWO and WPO <u>unified</u>.
- Next: find the instance I that brings the largest gap ratio (asymptotically)

Improving Gap Ratio



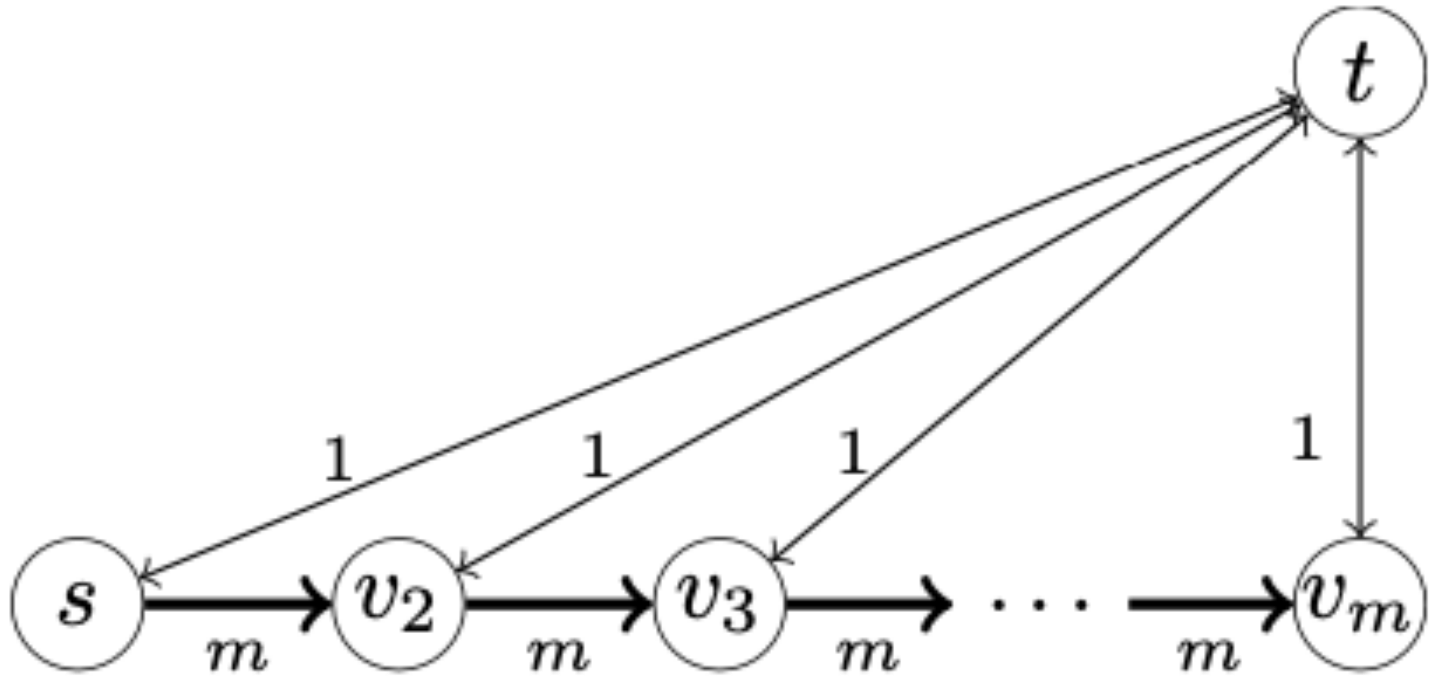


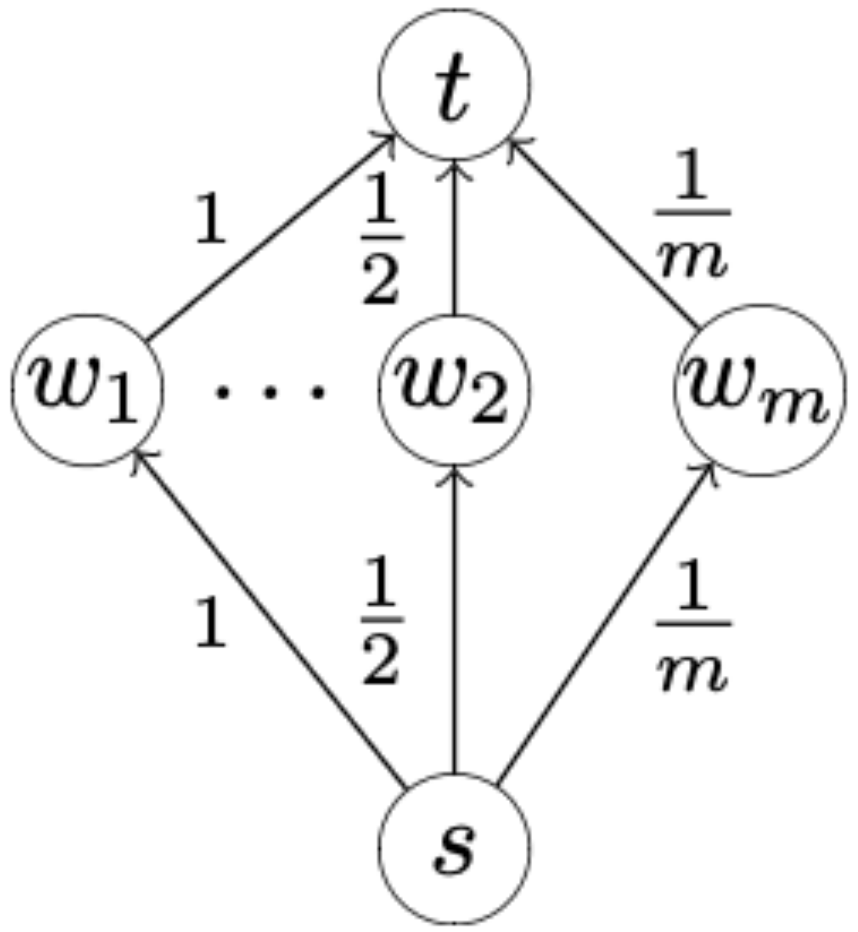


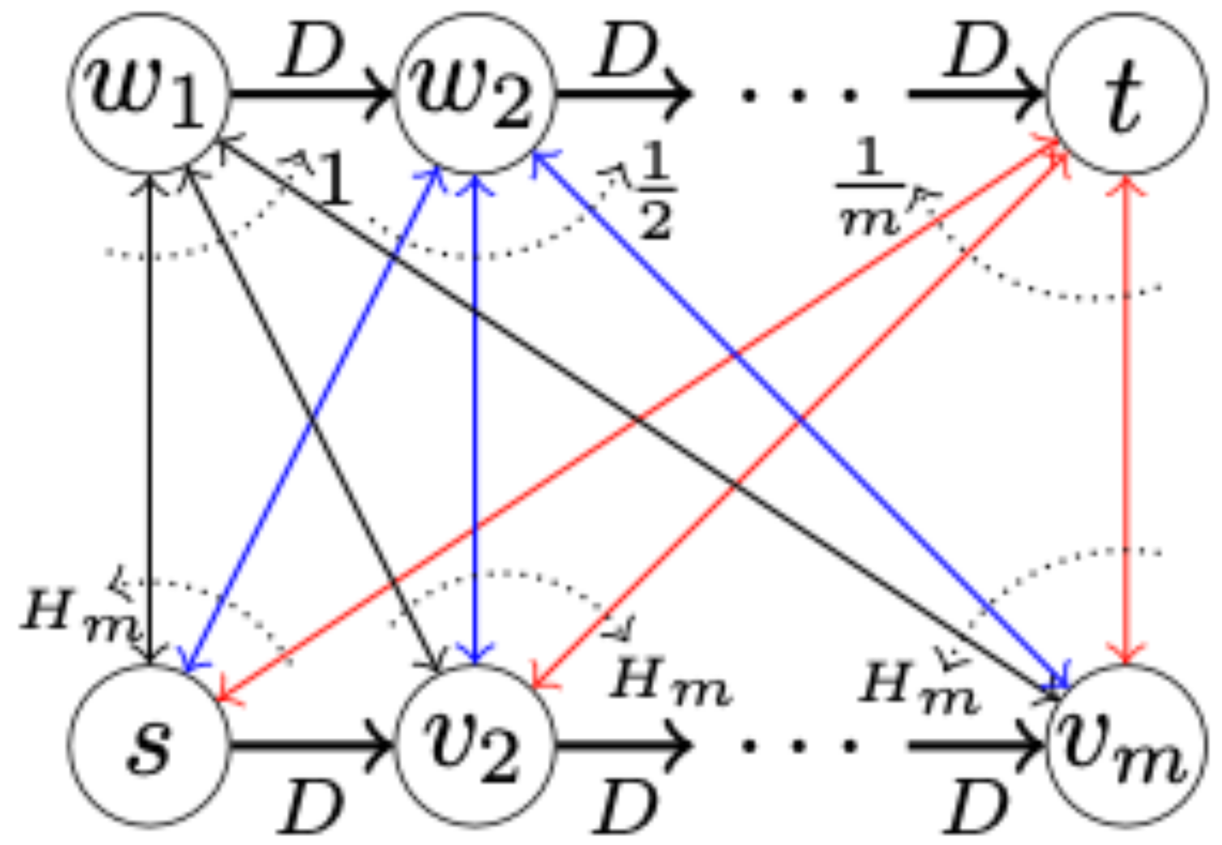
$$\Omega(n) \times \Omega(\log n) = \Omega(n \log n)$$

Improving Gap Ratio

- Easy to deduce a gap in $\Omega(n)$ from previous works (Fortz et. al)



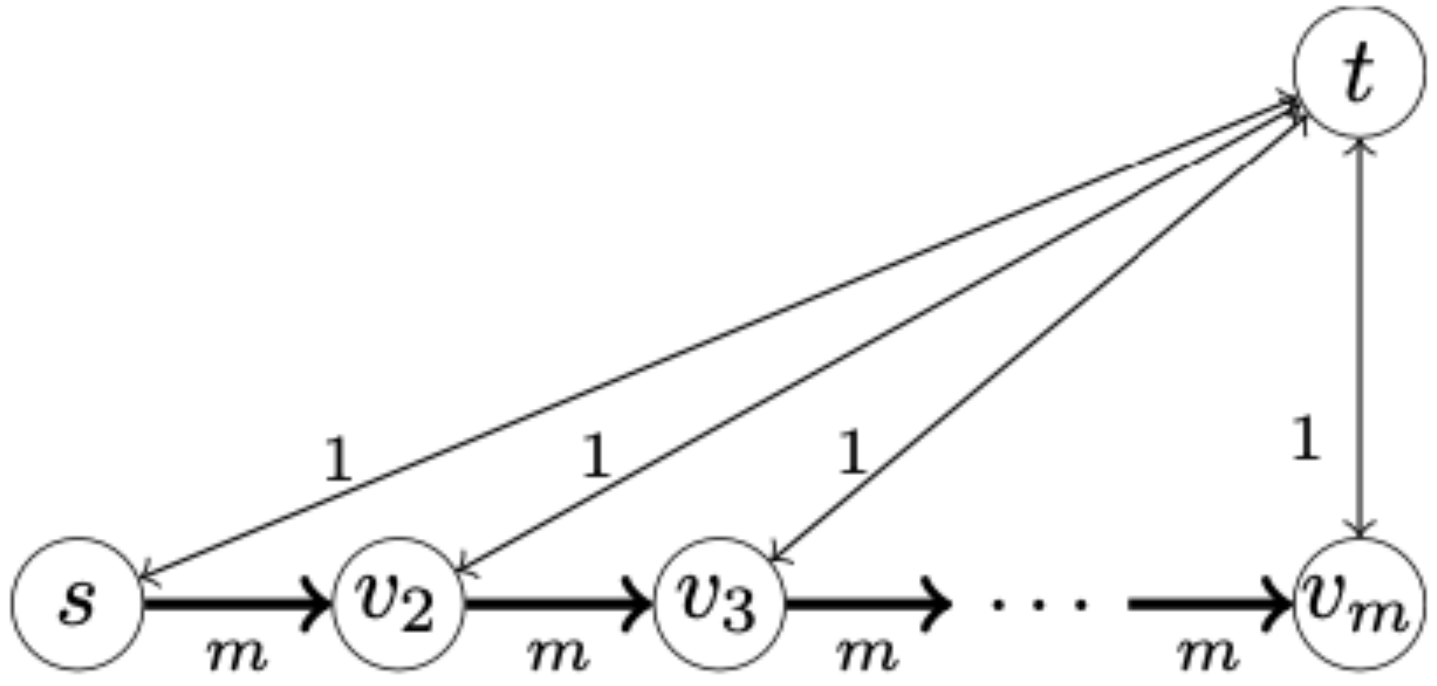


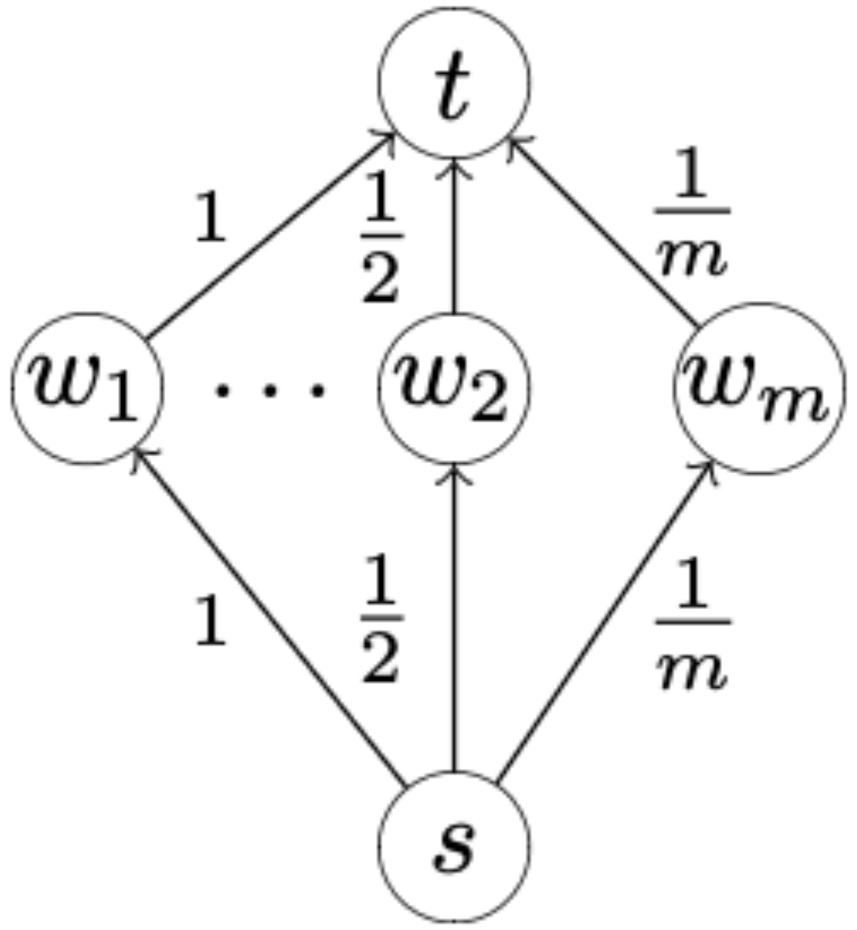


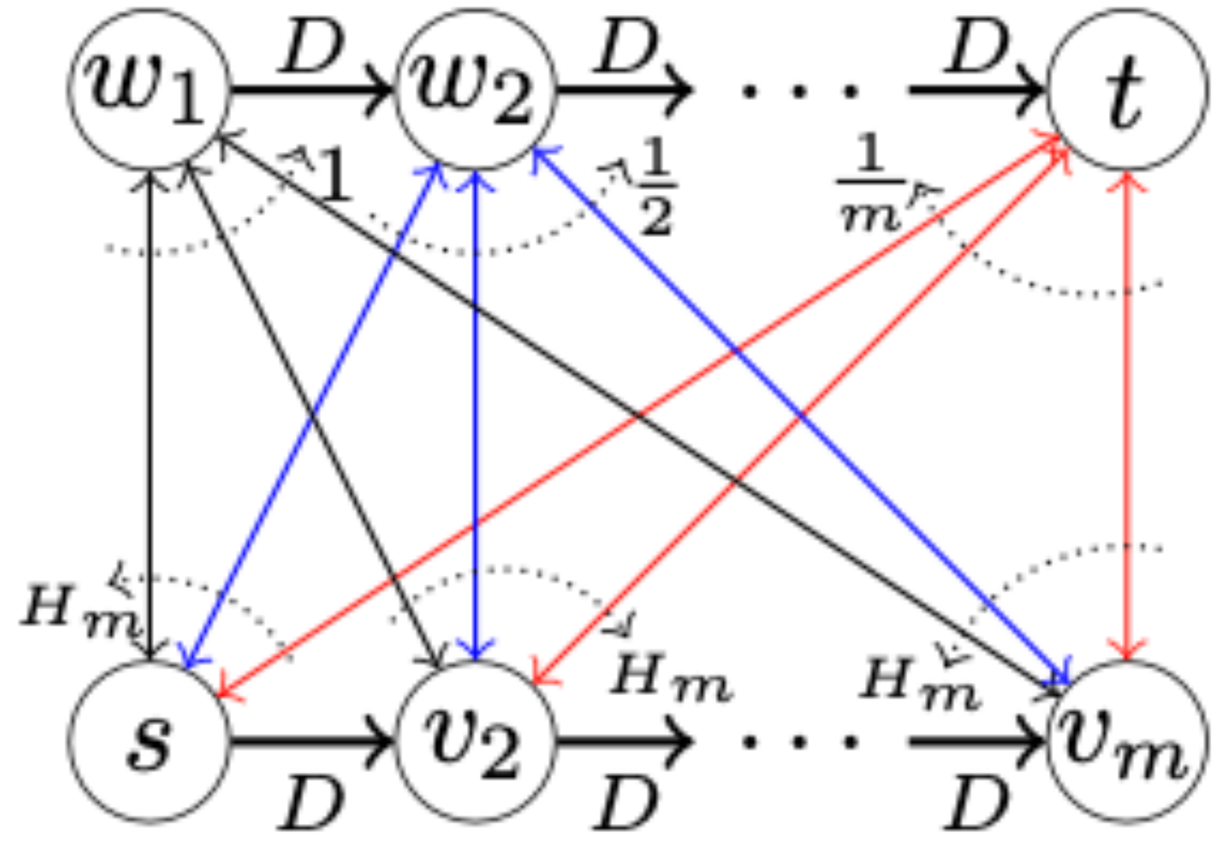
$\Omega(n)$ $\times$ $\Omega(\log n)$ $=$ $\Omega(n \log n)$

Improving Gap Ratio

- Easy to deduce a gap in $\Omega(n)$ from previous works (Fortz et. al)
- Combined the existing construction (left) with a new one (middle)



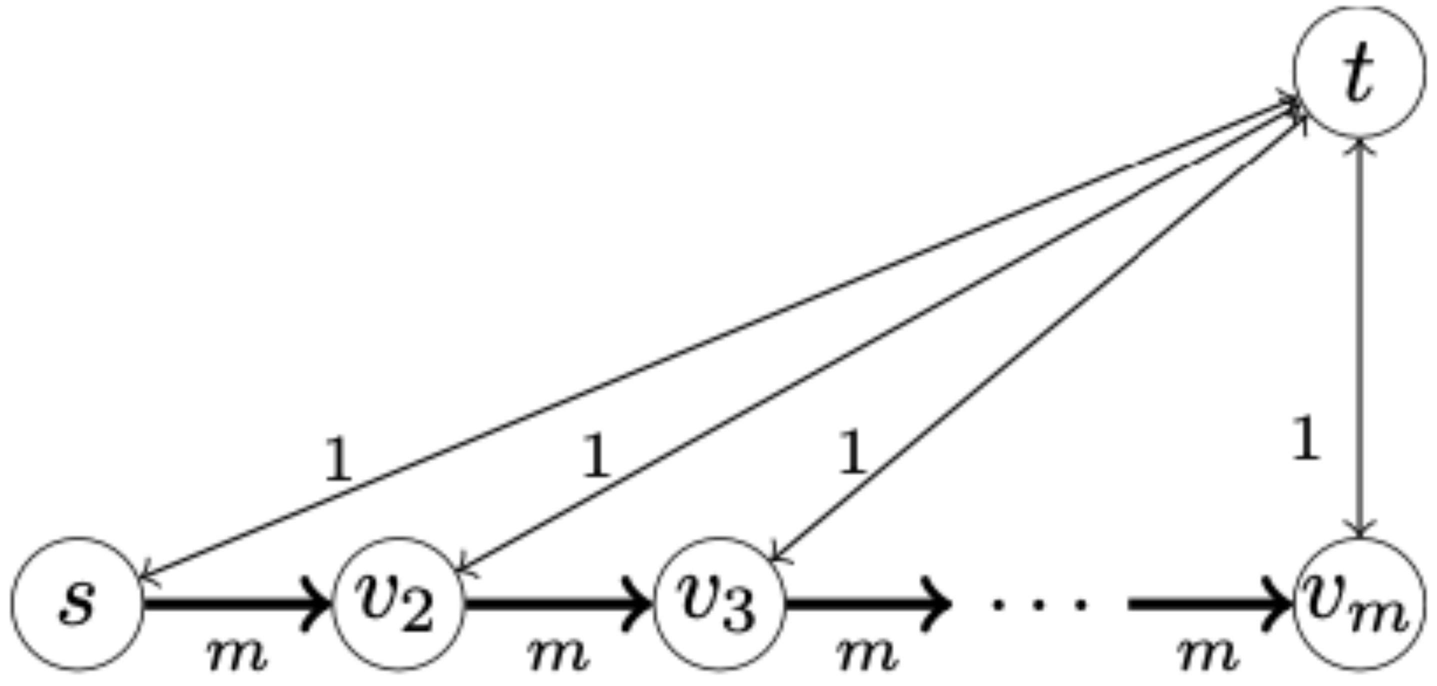


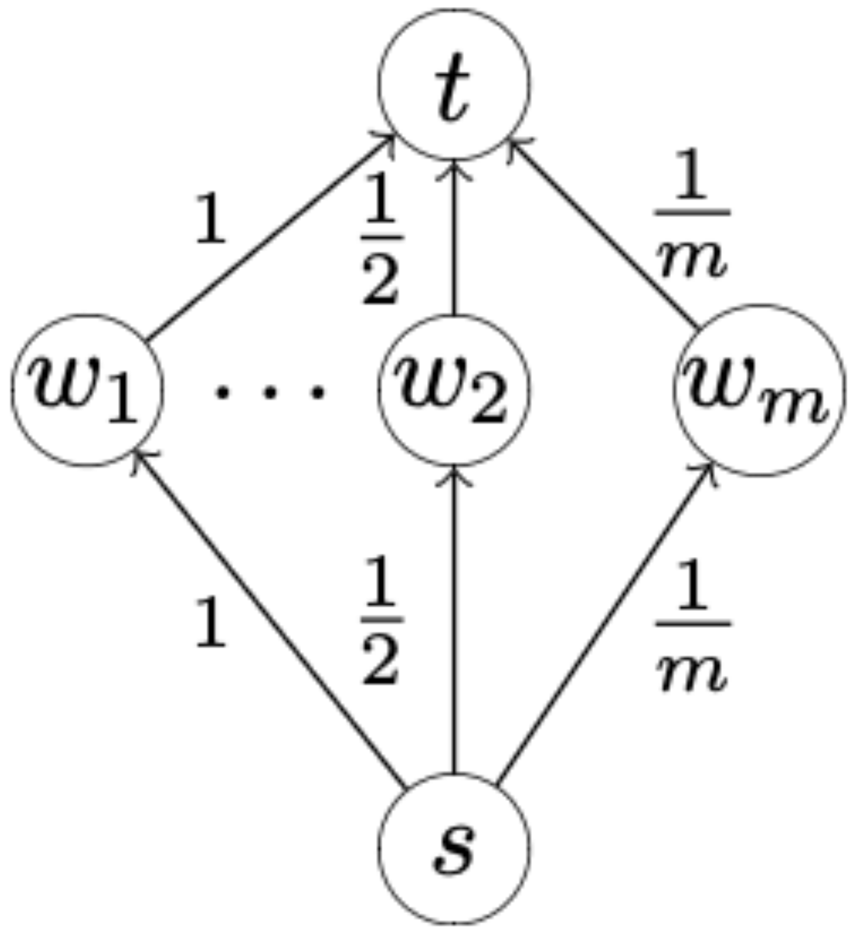


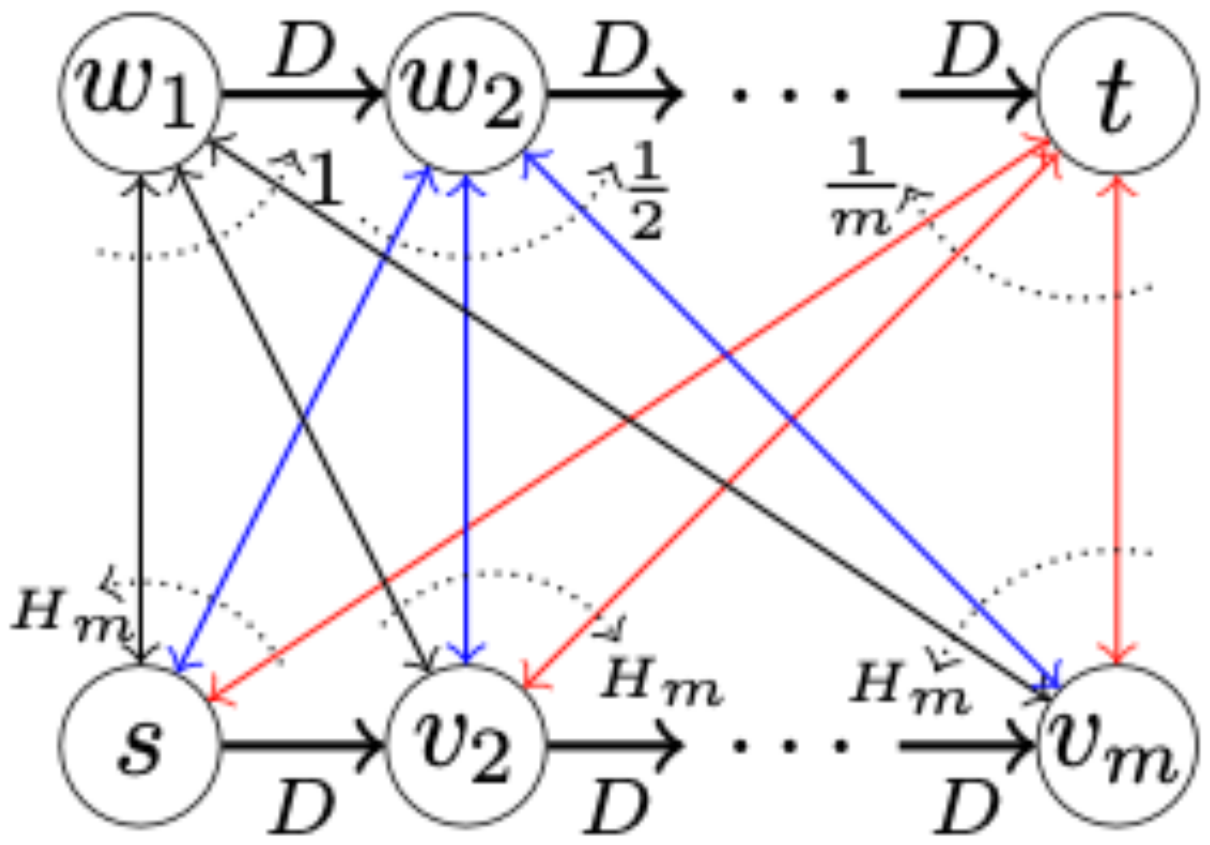
$$\Omega(n) \quad \times \quad \Omega(\log n) \quad = \quad \Omega(n \log n)$$

Improving Gap Ratio

- Easy to deduce a gap in $\Omega(n)$ from previous works (Fortz et. al)
- Combined the existing construction (left) with a new one (middle)
- JOINT is $\Omega(n \log n)$ times better than LWO ∪ WPO







$\Omega(n)$ $\times$ $\Omega(\log n)$ $=$ $\Omega(n \log n)$

TE Contributions

TE Contributions

- **LB**: The worst-case gap ratio is in $\Omega(n \log n)$.

TE Contributions

- **LB**: The worst-case gap ratio is in $\Omega(n \log n)$.
- **UB**: It cannot be worse than $O(n \log n)$ in single source-target cases.

TE Contributions

- **LB**: The worst-case gap ratio is in $\Omega(n \log n)$.
- **UB**: It cannot be worse than $O(n \log n)$ in single source-target cases.
- $O(n \log n)$-approximation algorithm for single source-target instances

TE Contributions

- **LB**: The worst-case gap ratio is in $\Omega(n \log n)$.
- **UB**: It cannot be worse than $O(n \log n)$ in single source-target cases.
- $O(n \log n)$-approximation algorithm for single source-target instances
- Mixed linear integer program (MILP) and evaluations

TE Contributions

- **LB**: The worst-case gap ratio is in $\Omega(n \log n)$.
- **UB**: It cannot be worse than $O(n \log n)$ in single source-target cases.
- $O(n \log n)$-approximation algorithm for single source-target instances
- Mixed linear integer program (MILP) and evaluations
- A heuristic for the JOINT problem, an extension of Fortz-Thorup's

Softwarized Traffic Control

Softwarized Traffic Control

- Traffic Engineering

Softwarized Traffic Control

- Traffic Engineering
- **Fast Traffic Rerouting**

- Traffic Engineering
- **Fast Traffic Rerouting**
- Traffic Partitioning

Fast Traffic Rerouting

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for single-link failures

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for <u>single-link</u> failures
- Challenging for <u>multi-link</u> failures and without a global view

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for <u>single-link</u> failures
- Challenging for <u>multi-link</u> failures and without a global view
- Permanent forwarding loops if BP are chosen arbitrarily

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for single-link failures
- Challenging for multi-link failures and without a global view
- Permanent forwarding loops if BP are chosen arbitrarily
- **R.Q.**: can we have BPs that guarantee a resiliency up to edge-connectivity? => **maximally resilient**

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for single-link failures
- Challenging for multi-link failures and without a global view
- Permanent forwarding loops if BP are chosen arbitrarily
- **R.Q.**: can we have BPs that guarantee a resiliency up to edge-connectivity? => **maximally resilient**

Fast Traffic Rerouting

- Fast (Traffic) Rerouting (**FRR**): link failure recovery under 50 ms
- A **backup path** replaces the failed link immediately
- BPs are computed and installed in advance (e.g. by the SDN controller)
- Trivial for single-link failures
- Challenging for multi-link failures and without a global view
- Permanent forwarding loops if BP are chosen arbitrarily
- **R.Q.**: can we have BPs that guarantee a resiliency up to edge-connectivity? => **maximally resilient**

- Contributions:
  - Introducing *maximal resiliency* formally with ILP
  - maximally resilient backup paths for hypercubes, complete graphs, torus, and grids
  - Extension of TI-LFA for multi-link failures and its SR analysis

Softwarized Traffic Control

Softwarized Traffic Control

- Traffic Engineering

Softwarized Traffic Control

- Traffic Engineering
- Fast Traffic Rerouting

- Traffic Engineering
- Fast Traffic Rerouting
- **Traffic Partitioning**

Traffic Partitioning

Traffic Partitioning

Traffic Partitioning

- **Context**: a datacenter with ℓ server machines each hosting k VNFs (server capacity); $k \times \ell = n$

Traffic Partitioning

- **Context**: a datacenter with ℓ server machines each hosting k VNFs (server capacity); $k \times \ell = n$
- VNFs may talk frequently to each other while providing a macro service
  - costly if a talkative VNF pair live on different servers; **communication cost**
  - collocating them is also costly; **migration cost**
  - the communication pattern is not known in advance
  - the communication graph is revealed one edge at a time, i.e., **online**
  - swapping is necessary to make space

Traffic Partitioning

- **Context**: a datacenter with ℓ server machines each hosting k VNFs (server capacity); $k \times \ell = n$
- VNFs may talk frequently to each other while providing a macro service
  - costly if a talkative VNF pair live on different servers; **communication cost**
  - collocating them is also costly; **migration cost**
  - the communication pattern is not known in advance
  - the communication graph is revealed one edge at a time, i.e., **online**
  - swapping is necessary to make space
- **R.Q.**: when and on which server we should collocate a VNF pair?
  - s.t. the overall cost is competitive to the optimal cost
  - respecting the capacity constraint

Traffic Partitioning

- **Context**: a datacenter with ℓ server machines each hosting k VNFs (server capacity); $k \times \ell = n$
- VNFs may talk frequently to each other while providing a macro service
  - costly if a talkative VNF pair live on different servers; **communication cost**
  - collocating them is also costly; **migration cost**
  - the communication pattern is not known in advance
  - the communication graph is revealed one edge at a time, i.e., **online**
  - swapping is necessary to make space
- **R.Q.**: when and on which server we should collocate a VNF pair?
  - s.t. the overall cost is competitive to the optimal cost
  - respecting the capacity constraint
- Contributions:
  - lower bound in $\Omega(n)$
  - upper bound (online algorithm) in $O(n)$ for a special case called *perfect partitioning*
  - 6-competitive algorithm for the special case $k = 2$

Traffic Partitioning (all results)

Variant	Lower bound	Upper bound
Learning, $k \geq 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 4.3.1)	$O(k\ell), \epsilon = 0$ (Thm. 4.3.4)
Learning, $k \geq 3$	$\Omega(\ell \log k), \epsilon \leq \frac{1}{32}$ [105]	$O(\ell \log k), \epsilon \leq 1$ [105]
Learning, $k \geq 3$	$\Omega(\ell), \epsilon < 1/3$ (Thm. 4.3.2)	$O(\log k), \epsilon > 1$ [105]
General, $k = 3$	$\Omega(\ell), \epsilon = 0$ (Thm. 4.4.1)	$O(\ell), \epsilon = 0$ (Thm. 4.4.3)
General, $k = 2$	$3, \epsilon = 0$ [19]	$6, \epsilon = 0$ (Thm. 4.4.6)
General, $k > 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 4.4.1)	$O(2^{O(k)}\ell), \epsilon = 0$ [28]

Additional Works

Additional Works

- A “Time-local” online problem closest related to partitioning

Additional Works

- A “Time-local” online problem closest related to partitioning
- Waypoint Routing

به یاد مادربزرگ مهربان و دلسوزم

In Memory of My Loving and Selfless Grandma