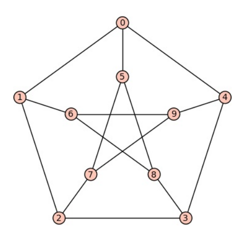
Overview of Networking Challenges for the Placement of Cloud Services

Frédéric Giroire

Université Côte d'Azur/CNRS/Inria COATI*, France

Journées Cloud 2019

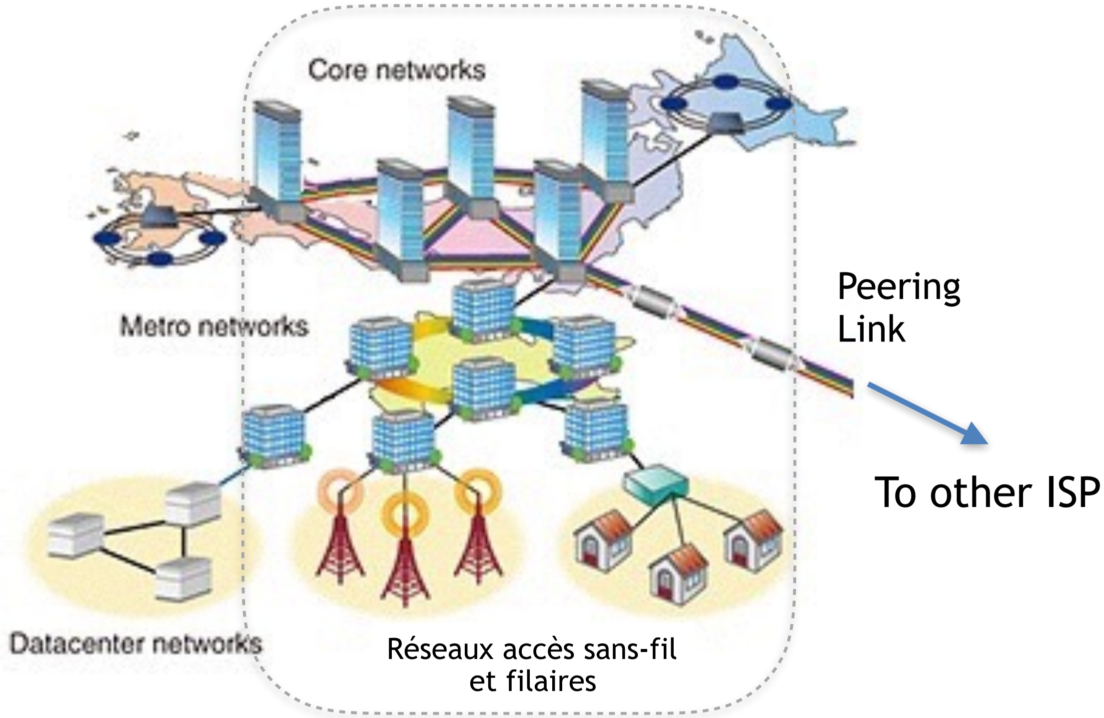
*Combinatorics, Optimisation et Algorithms For Telecommunications



$$
\begin{array}{llr}
\min & \sum_{e\in\mathcal{E}} y_e & \\
\text{s.t.} & \sum_{a\in A_i^+(u)} f_a^i - \sum_{a\in A_i^-(u)} f_a^i = \begin{cases} |V_i|-1 & \text{if } u = s_i \\ -1 & \text{if } u \neq s_i \end{cases} & \forall u\in V_i,\ V_i\in C \\
& f_a^i \le |V_i|\cdot x_a, & \forall V_i\in C, a\in A \\
& x_{(u,v)} \le y_{uv}, & \forall uv\in\mathcal{E} \\
& x_{(v,u)} \le y_{uv}, & \forall uv\in\mathcal{E}
\end{array}
$$

One slide on my research

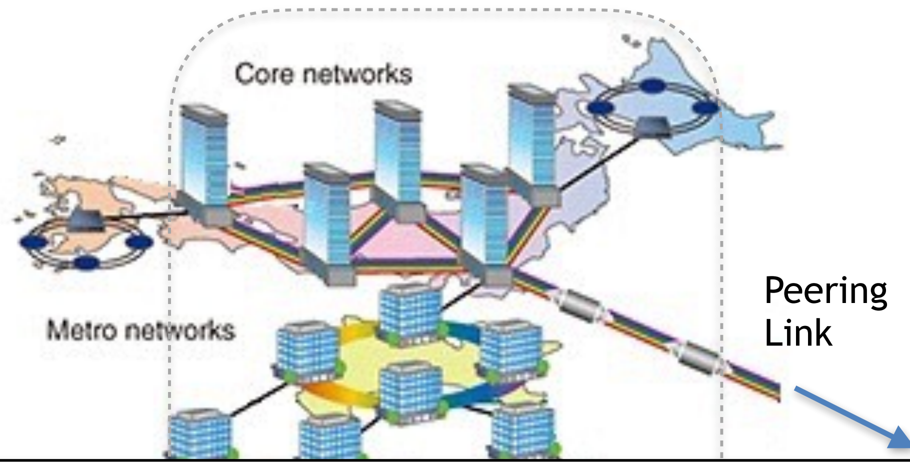
Optimization of network infrastructures.



Network of an
Internet Service
Provider (ISP)

One slide on my research

Optimization of network infrastructures.



Generic and recurring question: find the best tradeoff between

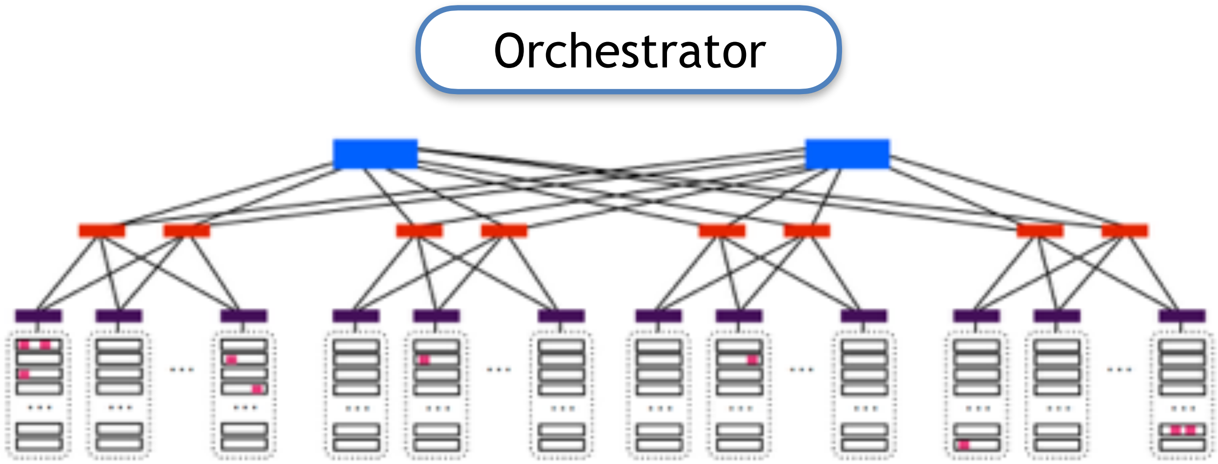
- where to store data,
- where to carry out computations or execute services,
- how much trafic to send in the network and by which route,

with diverse objectives: minimize the costs, the energy consumption, the failure probability or to maximize users' satisfaction.

Using tools from **algorithmics**, **optimization**, **combinatorics (graph theory)**, **simulations** and **experimentations**.

In the cloud

- Application or Services are run in Virtual Machines (VMs) or containers or Kata-containers
- An orchestrator assigns VMs to servers



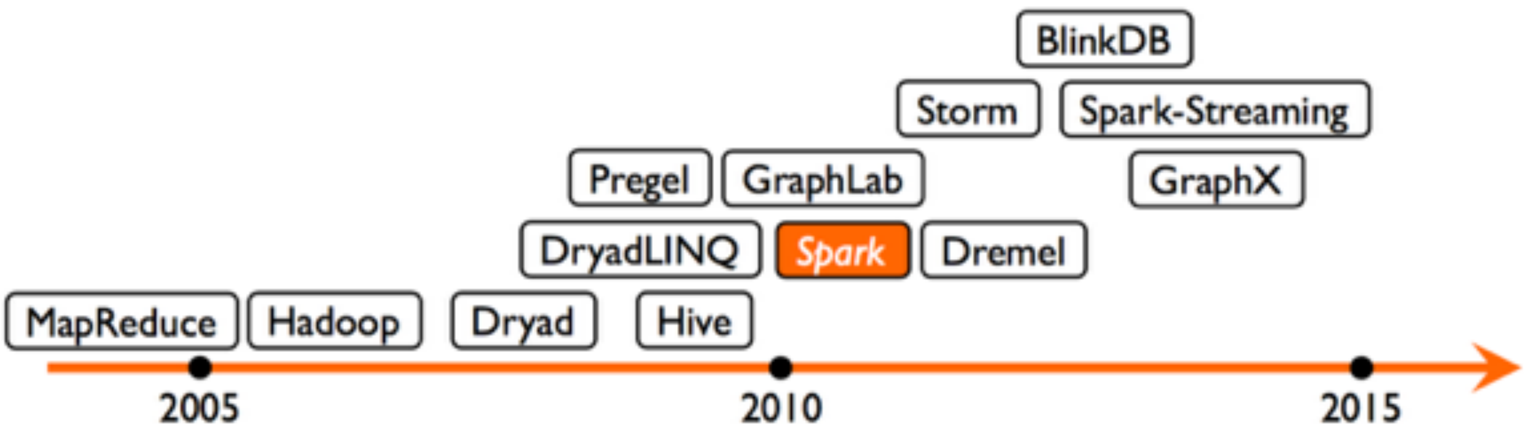
Classical optimization problem: VM placement satisfying CPU, memory, storage constraints while minimizing some cost

Big Data*

- The volume of data businesses want to make sense of is increasing
- Increasing variety of sources
  - Web, mobile, wearables, vehicles, scientific, …
- Cheaper disks, SSDs, and memory
- Stalling processor speeds

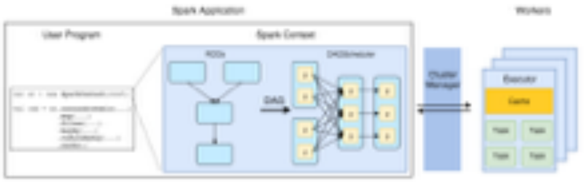
*Thanks: Some slides were borrowed from M. Chowdhury (University of Michigan)

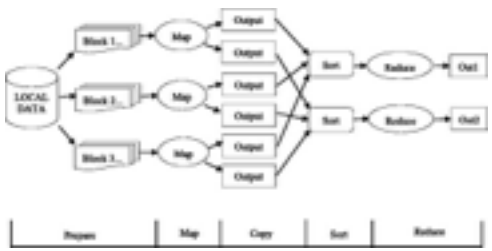
Solution: Big Data Centers for Massive Parallelism



Introduction

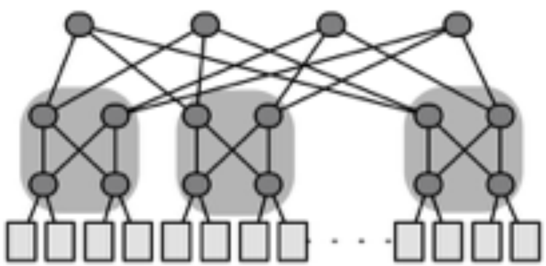
- More and more data-oriented parallel computing solutions (e.g., MapReduce, Dryad, CIEL, and Spark)





- Traditional scheduling consider properties of
  - **server** (e.g., CPU and memory usage)
  - **job** (e.g., execution time, deadline)

Network resources usually not taken into consideration



Communication is Crucial

- **Performance**
  - Facebook jobs spend ~**25%** of runtime on average in intermediate communications*
    [Chowdhury. Presentation in Dimacs. 2017]
  - For some workload, communications may account for up to 50% of job completion time [Chowdhury, et al. Orchestra SIGCOMM 2011]

As fast storage (e.g. SSD-based) systems proliferate, the network is likely to become an more and more important bottleneck

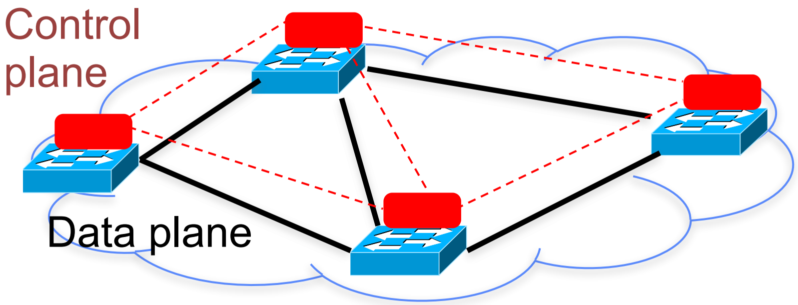
*Based on a month-long trace with 320,000 jobs and 150 Million tasks, collected from a 3000-machine Facebook production MapReduce cluster.

Legacy Networks

- However, network resources are usually not optimized.

- Why?
  - Network control is *very* difficult.

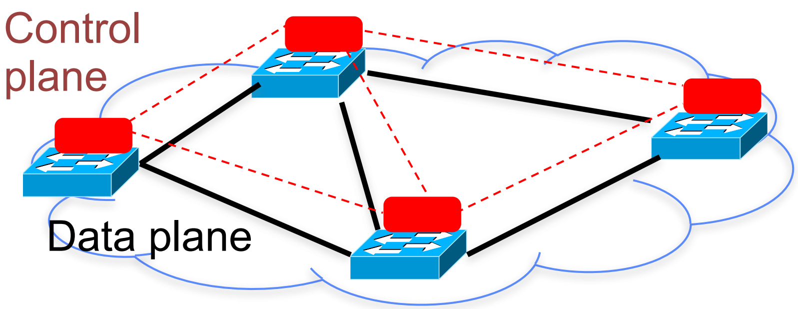
Legacy networks



- **Router=closed systems.** Any change has to be done **manually.**

- Networks are managed by complex **configurations.**

Legacy networks

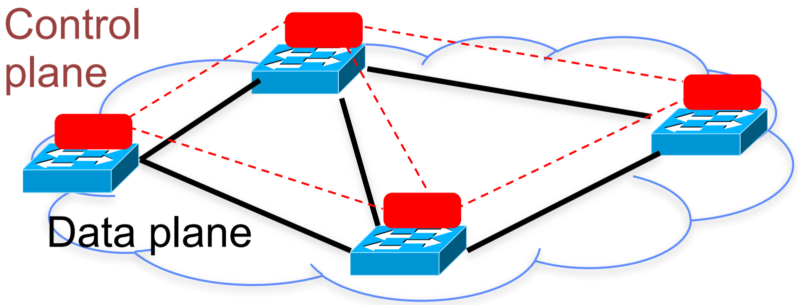


- Router=closed systems. Any change has to be done manually.

- Networks are managed by complex configurations.

—> Important difficulties to deploy new protocols

Legacy networks



- **Router=closed systems.** Any change has to be done **manually.**
- Networks are managed by complex **configurations.**

—> Important difficulties to deploy new protocols

-> Dynamic routing decision **not yet successfully implemented** in networks.

Question:

What can be done to improve network usage?

Outline

1. Motivation
2. A new situation: SDN and NFV
3. Placement of virtual network functions
   - Use case: Service Function Chaining
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. What next?

Some modeling tricks or algorithmic facts useful to know

Modeling Trick

- Trick 1: Layered graph
- Trick 2: Placement = set cover
- Fact 1: Efficient algorithms exist for SFC
- Trick 3: Modeling concurrent flows with co-flows
- Fact 2: Efficient algorithm exist for co-flows
- Trick 4: The big switch abstraction (and more generally finding the bottleneck)

Outline

1. Motivation
2. **A new situation: SDN and NFV**
3. Placement of virtual network functions
   - Use case: Service Function Chaining
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. What next?

A new context

However, arrival of two new network paradigms:

1. Software Defined Networking (SDN)

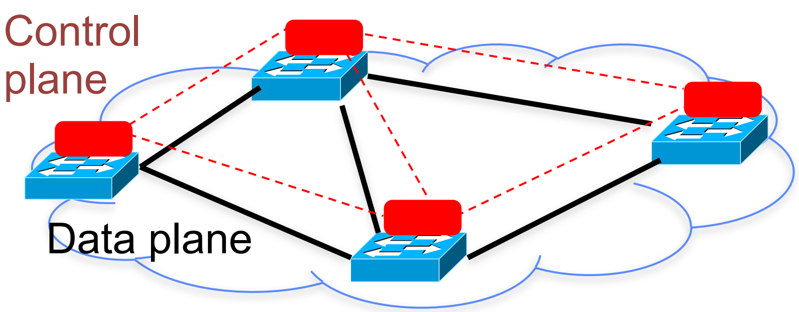
2. Network Function Virtualization (NFV)

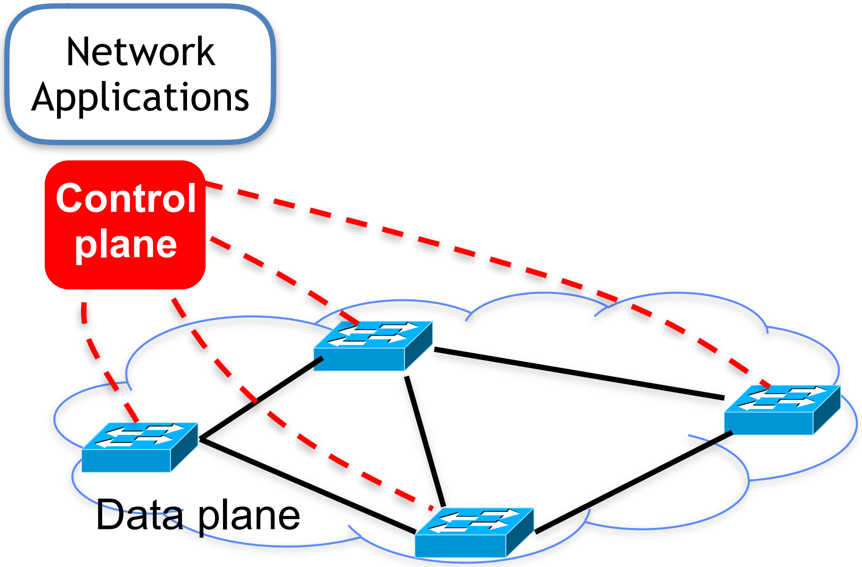
A new context

However, arrival of two new network paradigms:

1. Software Defined Networking (SDN)
2. Network Function Virtualization (NFV)

Software Defined Networks



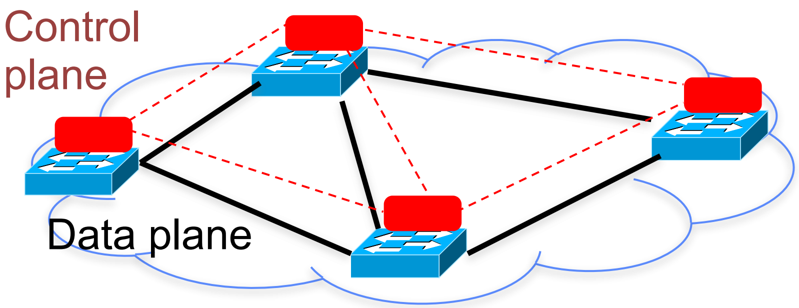


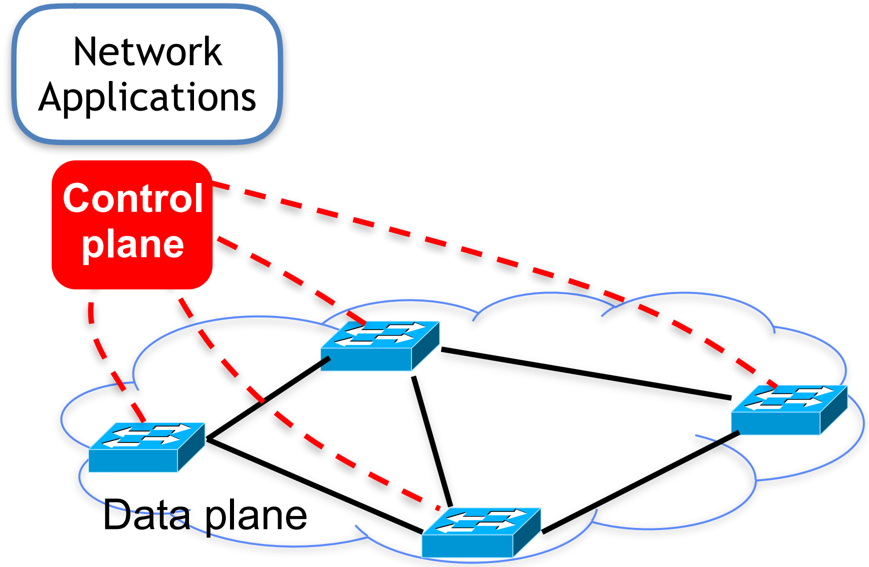
- **Router=closed systems.** Any change has to be done **manually.**
- Networks are managed by complex **configurations.**

—> **Important difficulties to deploy new protocols**

- Intelligence implemented by a **centralized controller** managing **elementary** switches
- SDN conceives the network as a **program.**

Software Defined Networks





- **Router=closed systems.** Any change has to be done **manually.**
- Networks are managed by complex **configurations.**

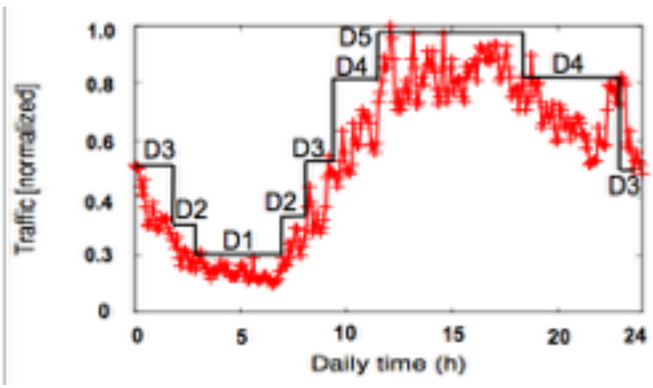
—> Important difficulties to deploy new protocols

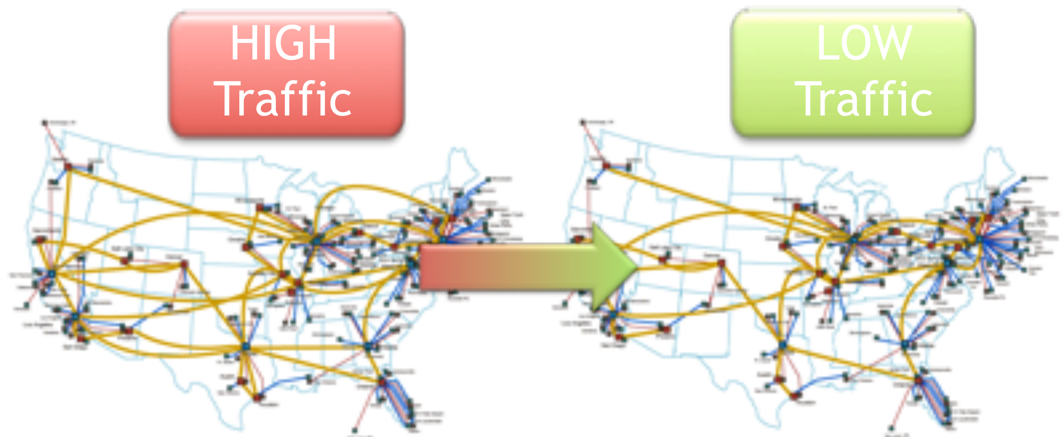
- Intelligence implemented by a **centralized controller** managing **elementary** switches
- SDN conceives the network as a **program.**

—>Allows the deployment of advanced (dynamic) protocols

Example: Energy Efficiency

- Core of solutions for energy efficiency: dynamic adaptation of resource usage to traffic changes.





Other applications: energy efficient data centers (virtual machine assignment), wireless networks (base-station assignment)…

Software Defined Networks

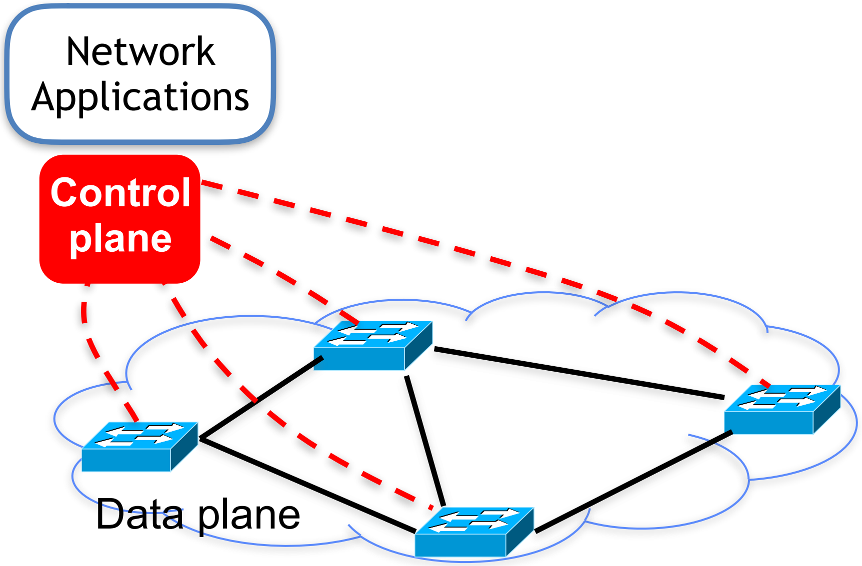
- Pushed by open source communities + large software and telecommunication companies.
  - **Large eco-system:** Open Flow / Open Day Light / Open Stack / Open vSwitch
  - **Software companies:** Google B4 large scale experiment on its inter-data center networks [Jain 2013].

B4 worldwide deployment (2011)

  - **Telcos:** e.g. AT&T targets 75% of network functions as a software by 2020.

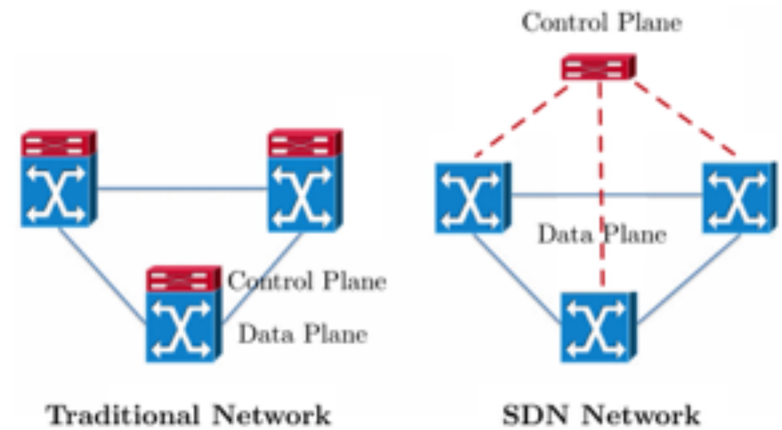
SDN Challenges

- **Defining the architecture.**
  - e.g. northbound **APIs** to enable real network programmability
- **Security**
  - e.g. single point of failure
- **Scalability of the SDN environment**
  - e.g. avoiding Control – Data Plane communications overhead



SDN in summary

Decoupling of network control and forwarding functions



Advantages:

- centralized management
- programmatically configured
- dynamic routing
- ...

A new context

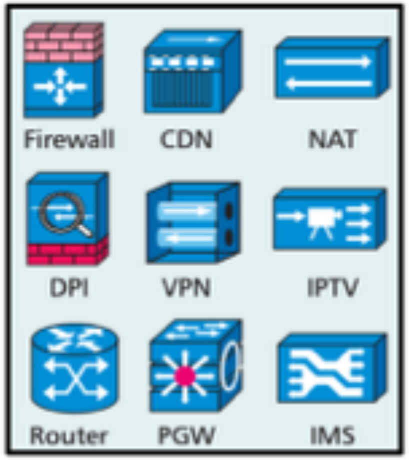
However, arrival of two new network paradigms:

1. Software Defined Networking (SDN)
2. Network Function Virtualization (NFV)

Network Function Virtualization

- Network flows have to be processed by a large number of network functions…

  …offering different services: security, traffic engineering, …



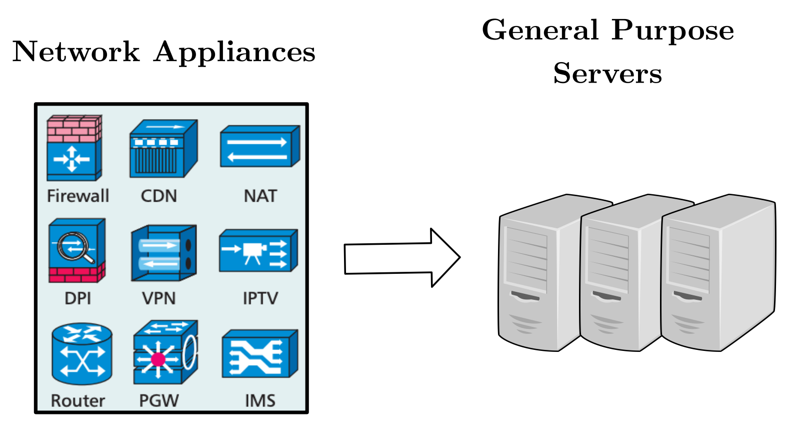
- Legacy networks implements network functions using expensive specific hardware called middleboxes.

Network Function Virtualization

- The NFV initiative decouples the network elements from underlying hardware

  by allowing functions to be run on general hardware using Virtual Machines.

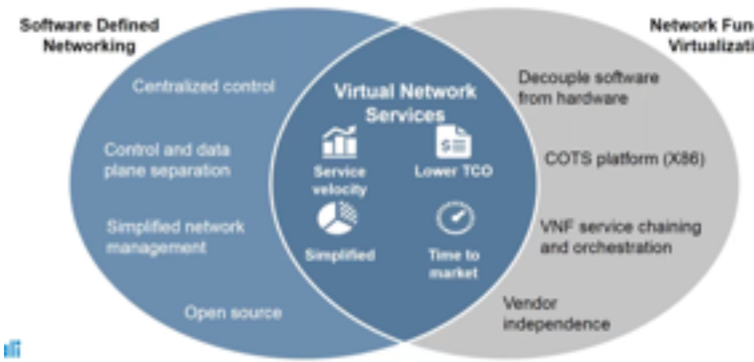
- **Advantages:**
  - flexibility,
  - cost,
  - scalability,
  - …



SDN+NFV = full Network Programmability

- NFV and SDN **independent of each other** but complementary

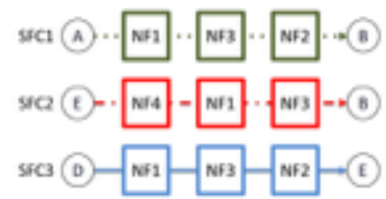
- A **symbiosis** between them can improve **resource management** and **service orchestration**:
  - Increased Efficiency and Lower Costs
  - Faster Innovation and Time to market
  - Agility - Automation & change faster
  - No Vendor Lock-in

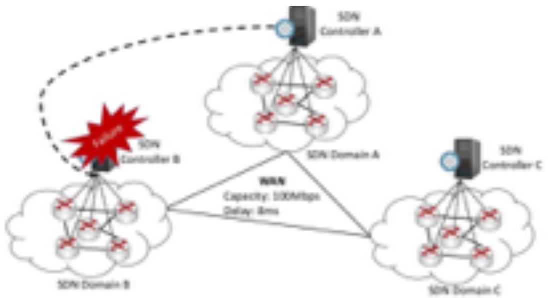


GOAL: exploit the benefits and potentials of both approaches

Research Challenges

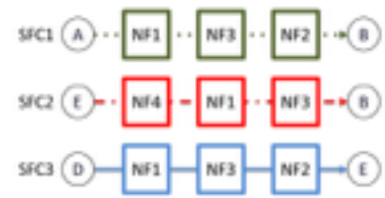
- Algorithmic Aspects of Resource Allocation
- Evaluation of SDN/NFV solutions
- New Protocols & Standardization
- Performance
- Resiliency
- Scalability
- Security
- …

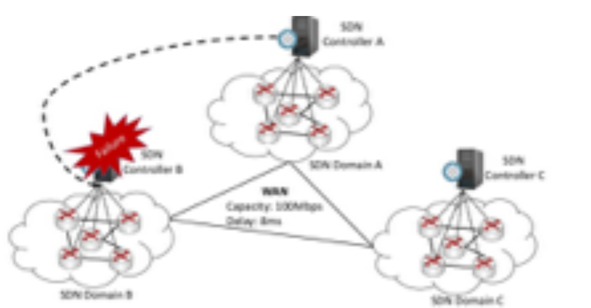




Research Challenges

- **Algorithmic Aspects of Resource Allocation**
- Evaluation of SDN/NFV solutions
- New Protocols & Standardization
- Performance
- Resiliency
- Scalability
- Security
- …



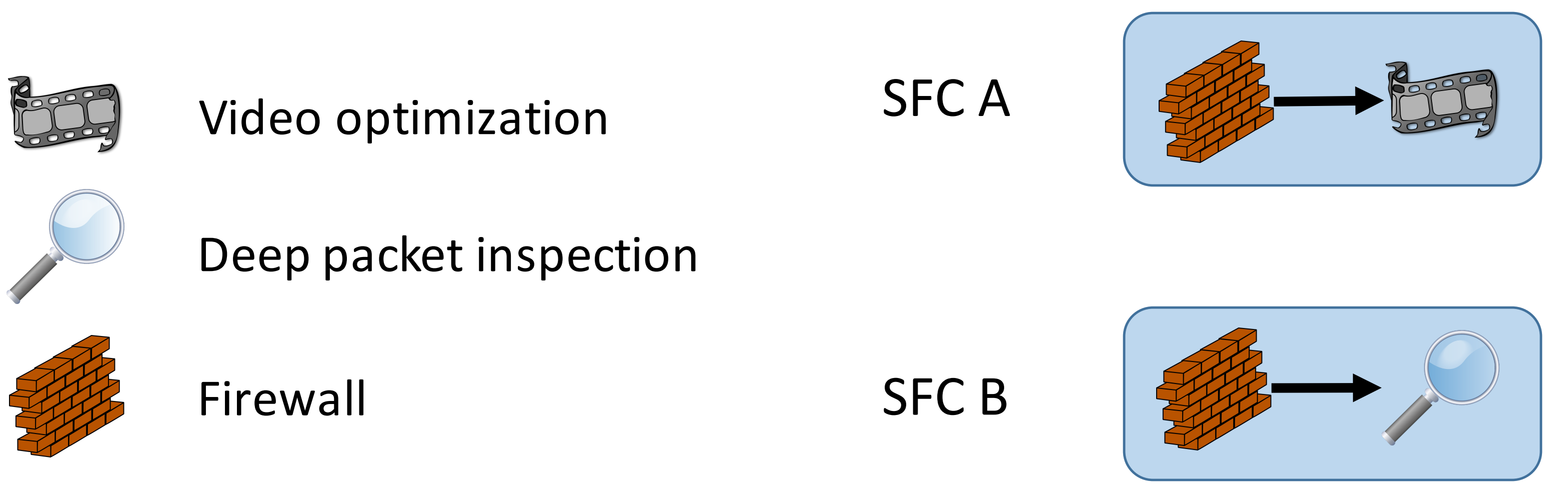


Outline

1. Motivation
2. A new situation: SDN and NFV
3. **Placement of virtual network functions**
   - **Use case: Service Function Chaining**
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. What next?

Service Function Chaining

- Network flows are often required to be processed by an ordered sequence of network functions defining a service
- Different customers can have different requirements in terms of the sequence of network functions



Service Function Chaining

- **Legacy Networks**: new service —> new hardware
  - impractical to change the locations of physical middleboxes

- **SDN/NFV-enabled Networks**: easier and cheaper SFCs deployment and provisioning:
  - simplified middlebox traffic steering (**SDN**)
  - flexible and dynamic deployment of network functions (**NFV**)

Flows can be managed **dynamically from end-to-end** and the network functions can be installed **only along the paths for which and when** they are necessary.

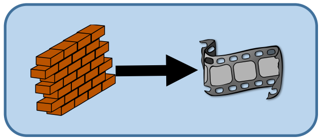
NFV Placement

- NFV: more efficient and **flexible network management.**
- Hence, **placing network functions** in a cost effective manner is an essential step toward the full adoption of the NFV paradigm.
- **Problem:** place VNFs to satisfy the **ordering constraints** of the flows with the goal of **minimizing the total setup cost** (such as license fees, network efficiency, or energy consumption)

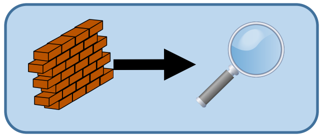
Example of Service Function Chains

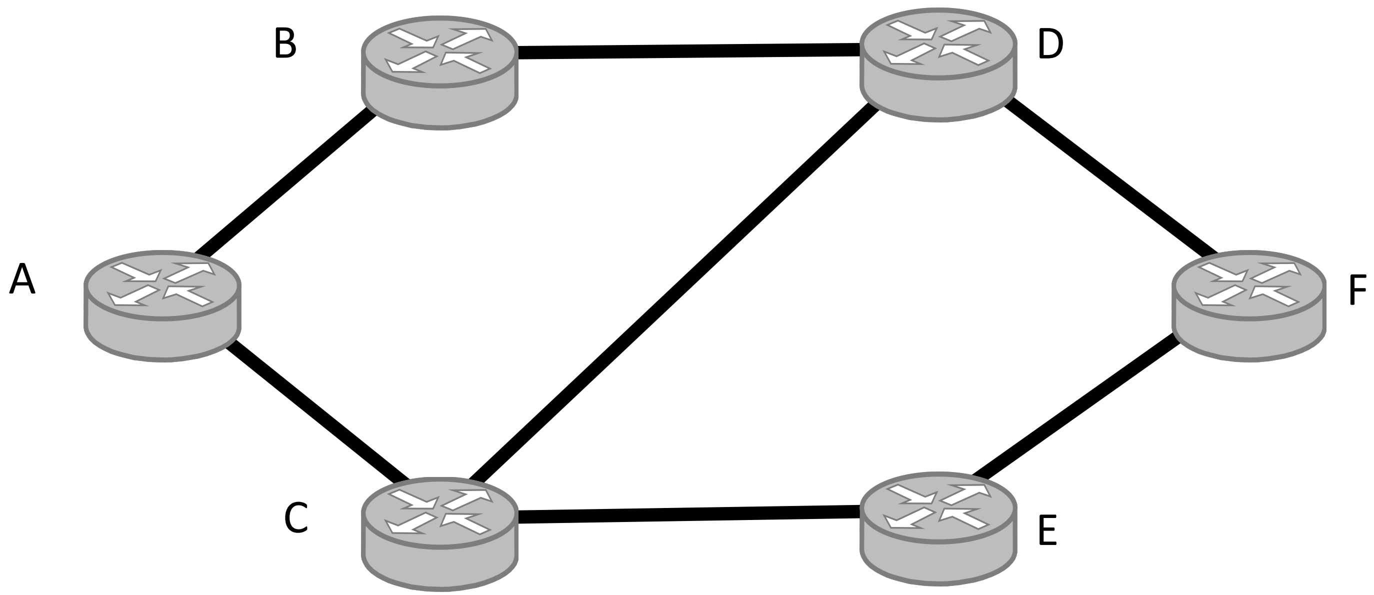
3 flows: A to F
A to E
F to C

SFC A



SFC B



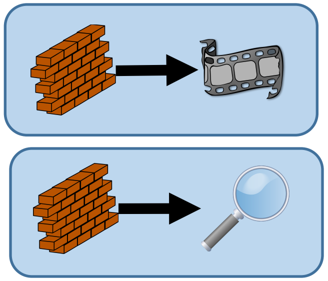


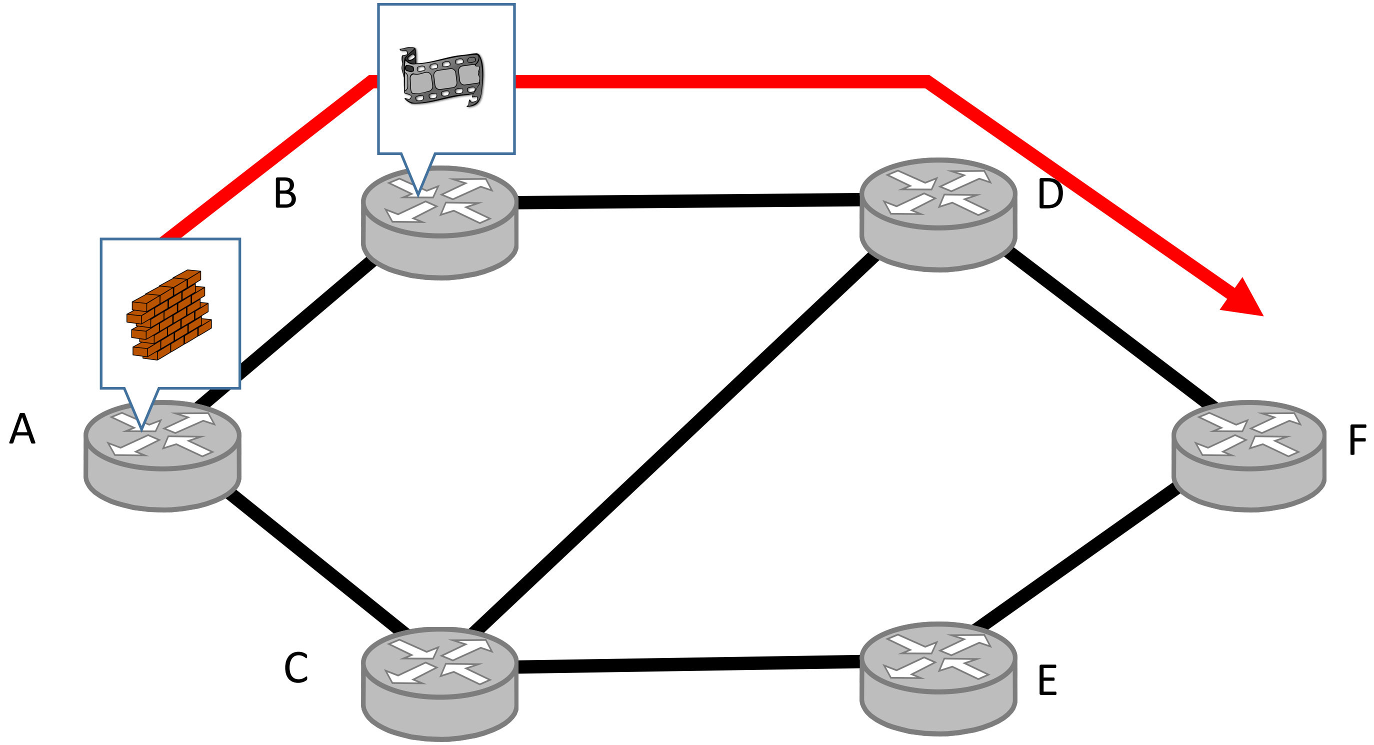
Example of Service Function Chains

3 flows: A to F
A to E
F to C

SFC A

SFC B

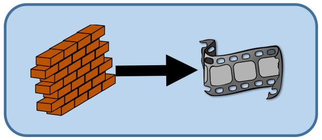


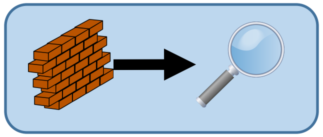


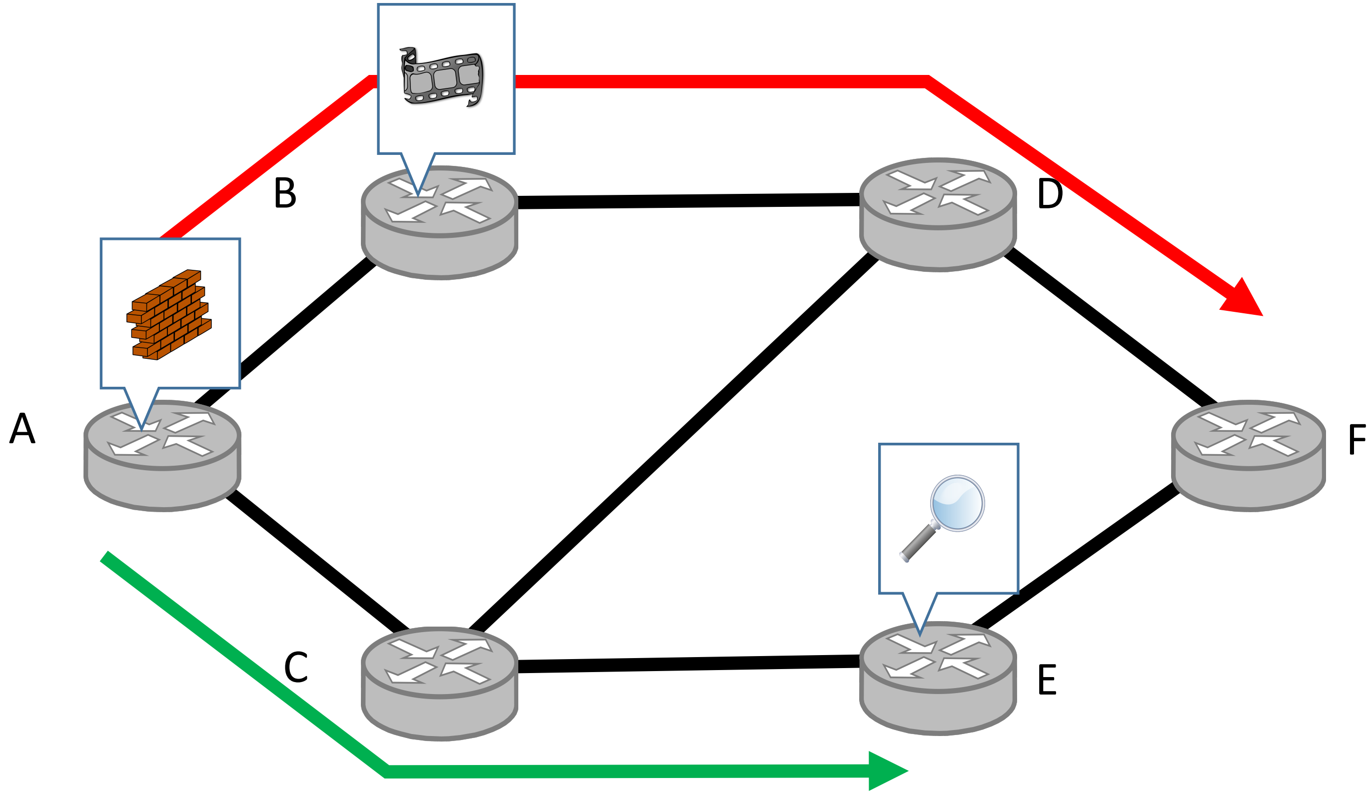
Example of Service Function Chains

3 flows: A to F — SFC A

A to E, F to C — SFC B

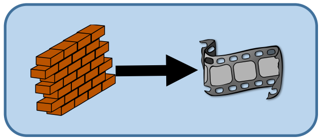


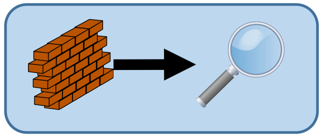


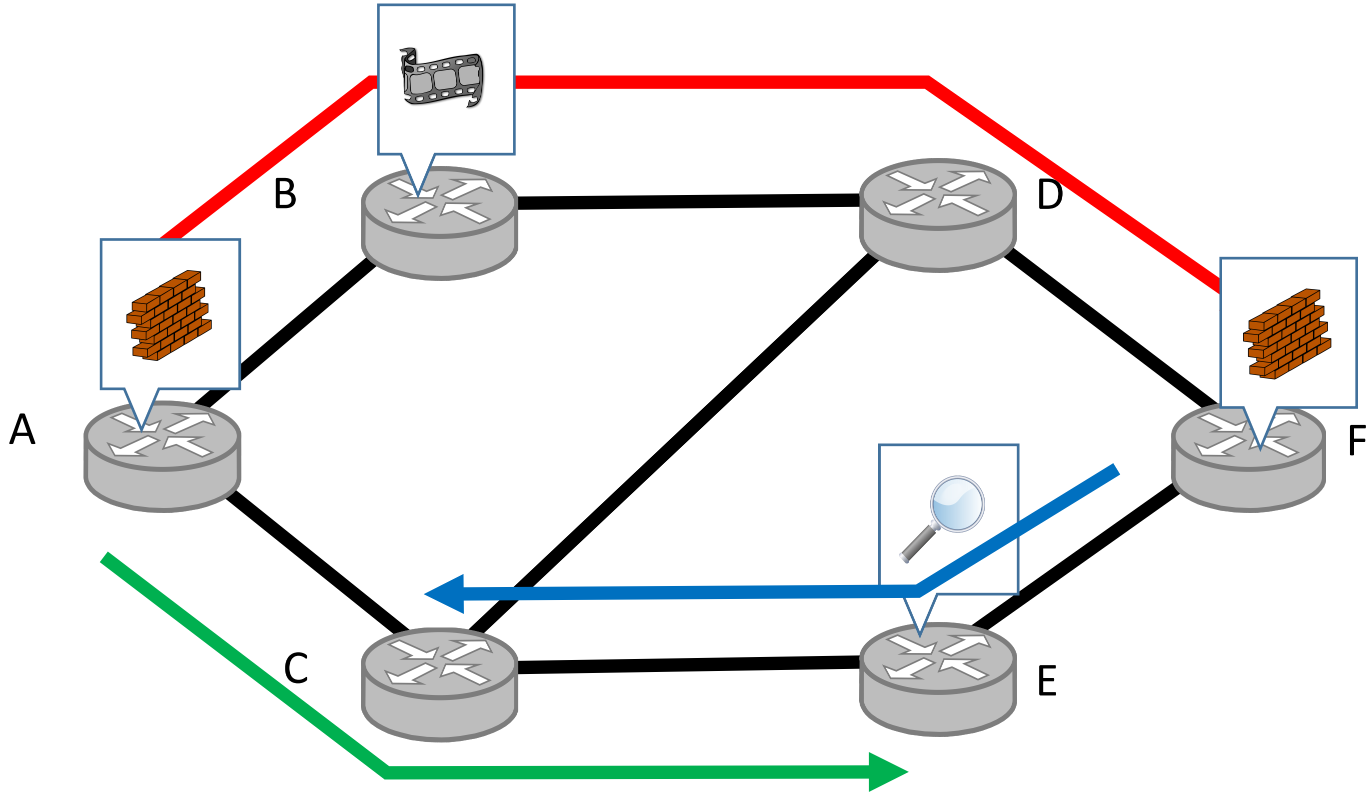


Example of Service Function Chains

3 flows: A to F — SFC A
A to E, F to C — SFC B







SFC Placement

- Challenges:
  - Optimizing routing **AND** NVF provisioning
  - Modeling order between functions

- Outline:
  1. Trick 1: The layered graph
     [Dwaraki and Wolf, in HotMIddlebox, 2016]

  2. Approximation algorithms for SFC
     [Tomassilli, Giroire, Huin, Perennes, in INFOCOM 2018]

     - Trick 2: NVF placement = Set Cover
       [Sang et al. in Infocom 2017]

SFC Placement

- Challenges:
  - Optimizing routing **AND** NVF provisioning
  - Modeling order between functions

- Outline:
  1. **Trick 1: The layered graph**
     [Dwaraki and Wolf, in HotMIddlebox, 2016]

  2. Approximation algorithms for SFC
     [Tomassilli, Giroire, Huin, Perennes, in INFOCOM 2018]

     - Trick 2: NVF placement = Set Cover
       [Sang et al. in Infocom 2017]

SFC Placement

Modeling Trick 1

- Classic way to model the problem of routing & provisioning SFC is using Integer Linear Programming (ILP) with
  - Introduction of large number of binary variables to model the function placement.
  - Introduction of large number of binary variables to model the order ("function f2 cannot appear on the path before function f1").

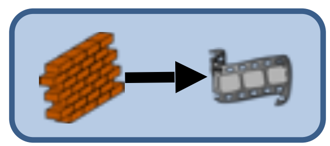
- Leads to not efficient optimization solutions and algorithms

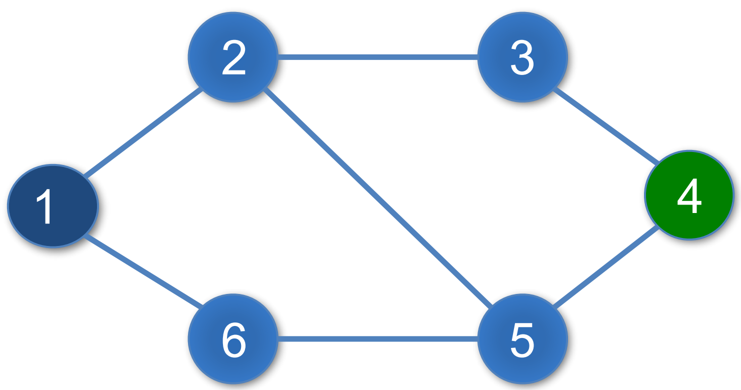
Layered Graph[1]

Modeling Trick 1

- Proposes an alternate way to find Service Path (path & placement of function)
  - *Transforms a **problem of routing and placement** into a **problem of routing**,*
  - *While taking into account the **order** between functions.*

Example:
Request between 1 and 4 for SFC



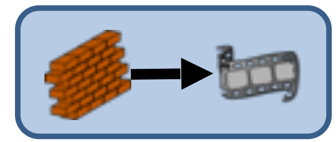


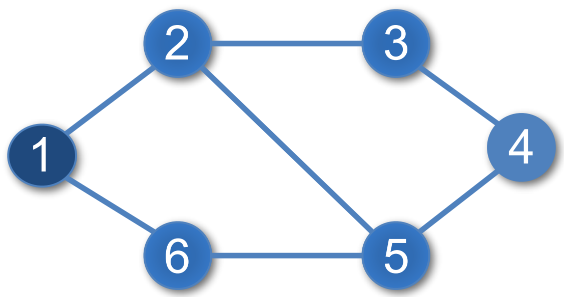
[1] Dwaraki and Wolf. Adaptive service-chain routing for virtual network functions in software-defined net- works," in Workshop on Hot topics in HotMIddlebox, 2016]

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC



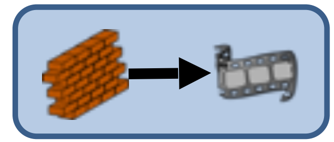


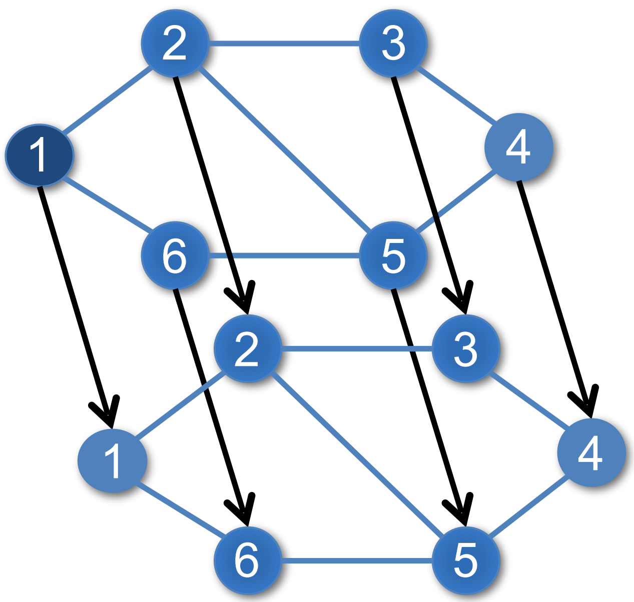
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC



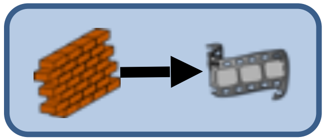


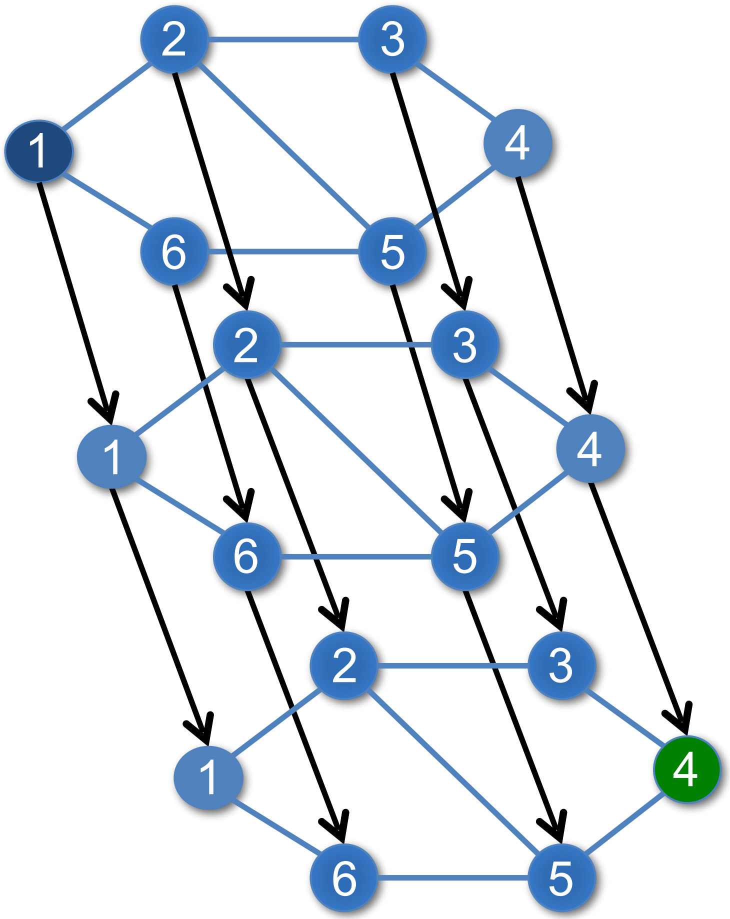
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC



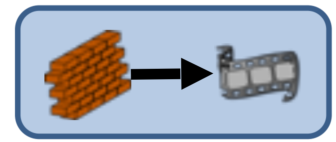


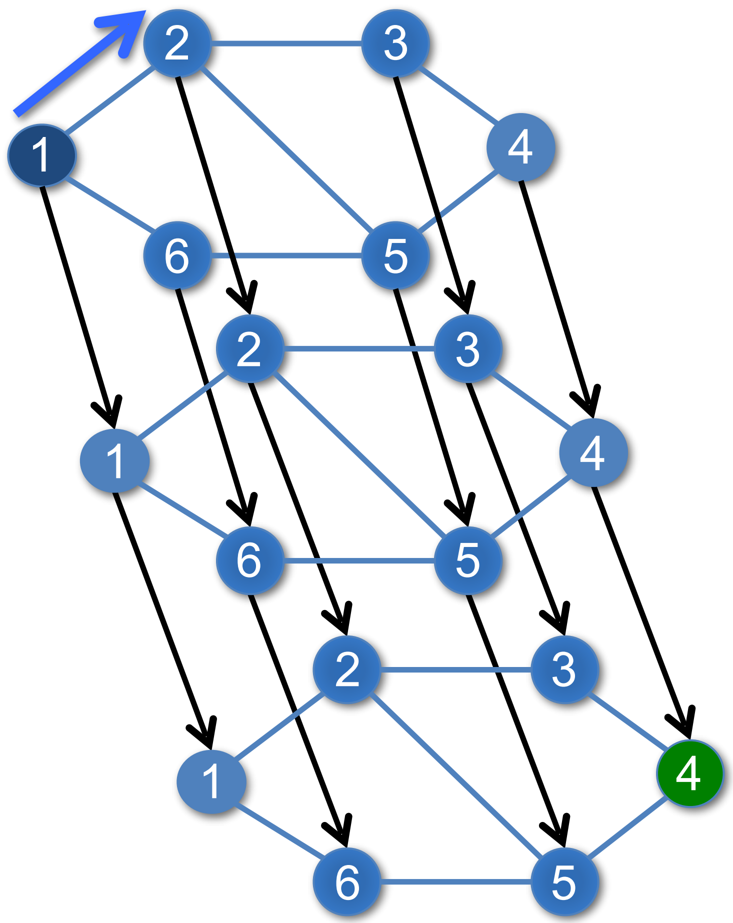
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Modeling Trick 1

Layered Graph

Example: Request between 1 and 4 for SFC



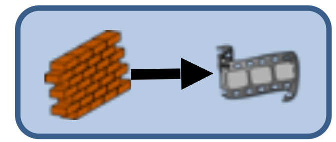


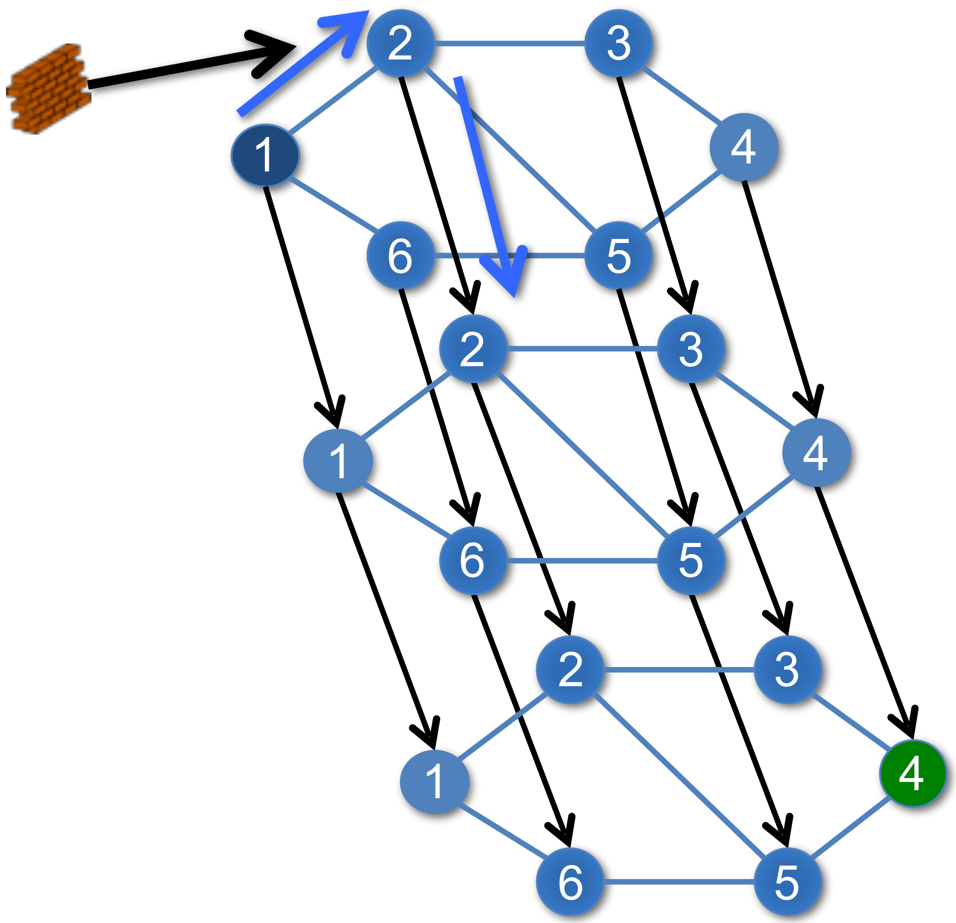
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC



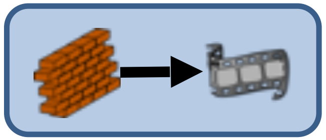


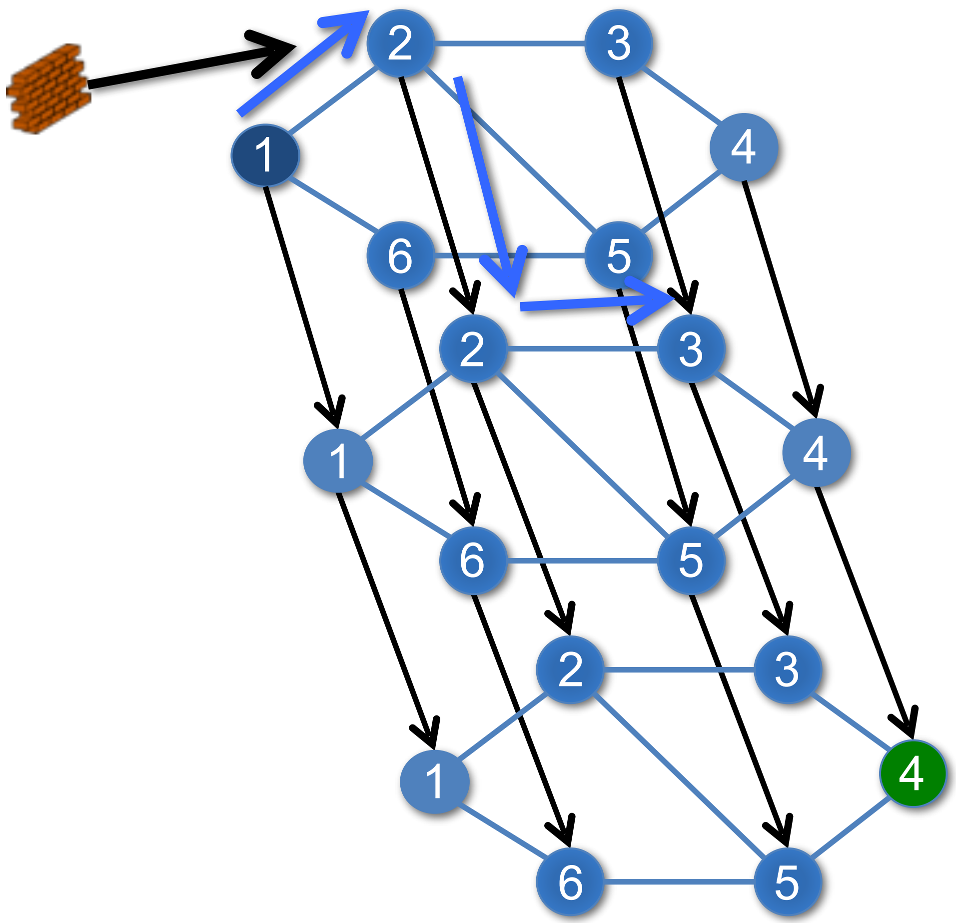
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC

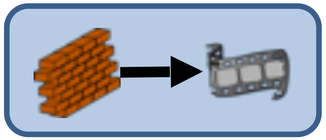


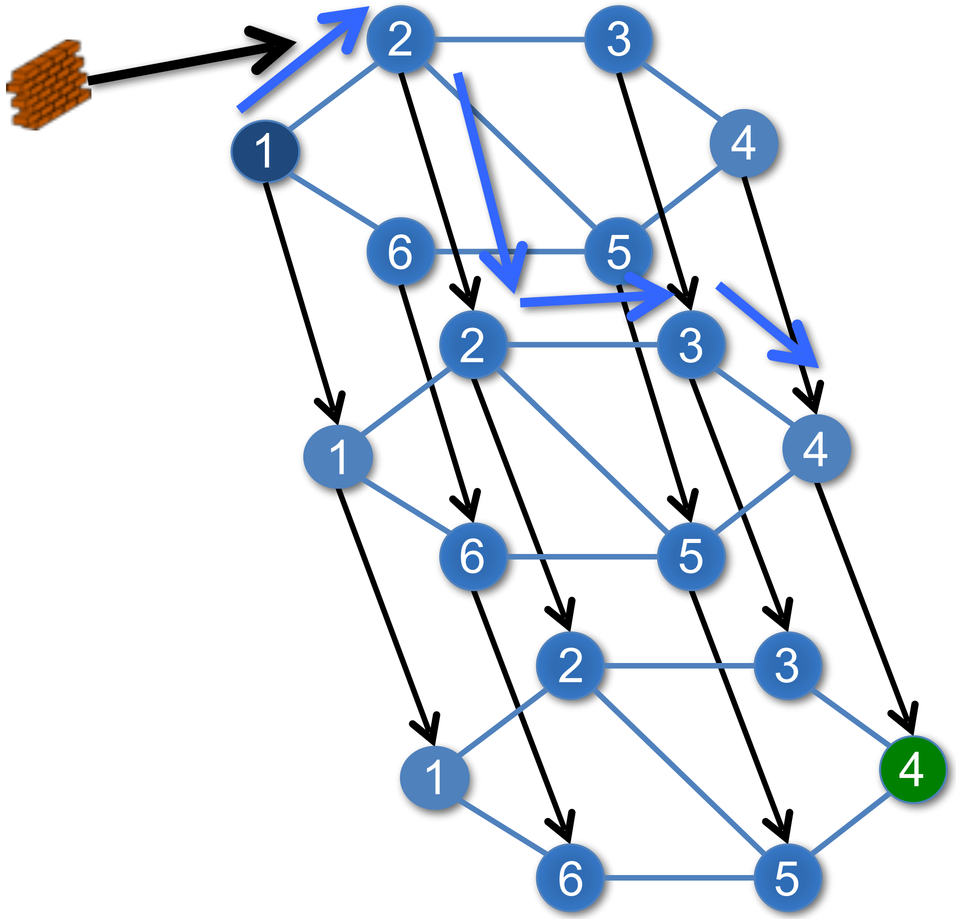


- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Example: Request between 1 and 4 for SFC



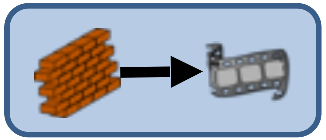


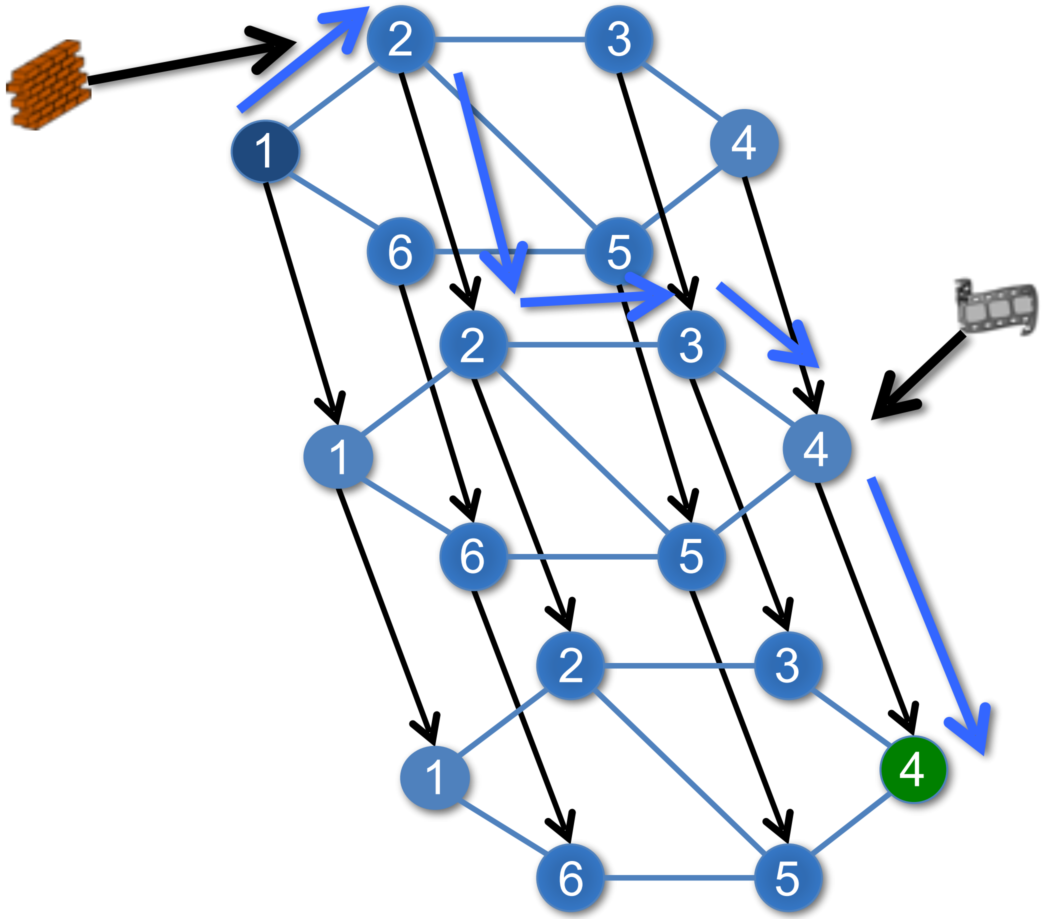
- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Modeling Trick 1

Example: Request between 1 and 4 for SFC





- # layers = # functions + 1
- Link **between layers** gives the **placement**
- Link **inside layers** gives the **routing**
- Path from first to last layer

Layered Graph

Algorithmic Fact 1

- Finding a Service Path boils down now to find
  - a constrained shortest path (because of shared capacity) in the layered graph, using **fast pseudo-polynomial algorithms** e.g. [1]
  - or even a simple shortest path (often sufficient in practice), using a very fast algorithm like Dijkstra.

[1] Irnich and Desaulniers. Shortest path problems with resource constraints. Column generation. 2005.]

SFC Placement

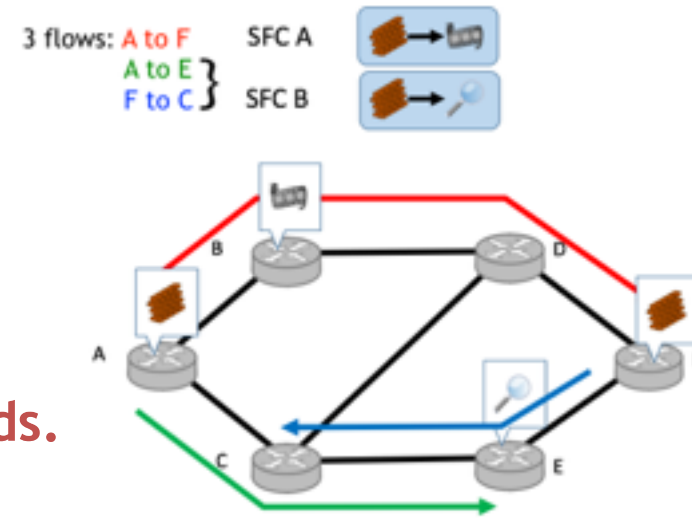
- Challenges:
  - Optimizing routing **AND** NVF provisioning
  - Modeling order between functions

- Outline:
  1. Trick 1: The layered graph
     [Dwaraki and Wolf, in HotMIddlebox, 2016]

  2. **Approximation algorithms for SFC**
     [Tomassilli, Giroire, Huin, Perennes, in INFOCOM 2018]

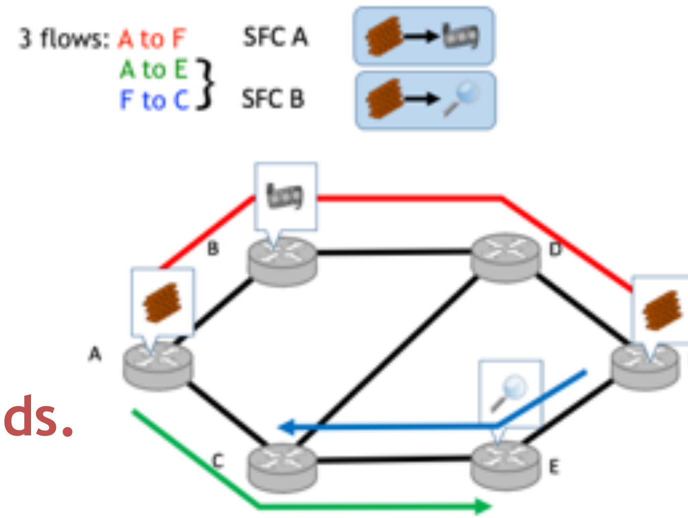
     - **Trick 2: NVF placement = Set Cover**
       [Sang et al. in Infocom 2017]

Problem



- **Input:** A digraph G = (V,E), a set of functions F, and a **collection D of demands.**
- A demand d ∈ D is modeled by a couple :
  - **a path** path(d) of length l(d) and
  - **a service function chain** sfc(d) of length s(d).
- A **setup cost** c(v,f) of function f in node v ∈ V .
- **Output:** A function placement Π ⊂ V × F
- **Objective:** minimize total setup cost $\sum_{(v,f)\in\Pi} c(v,f)$

Problem



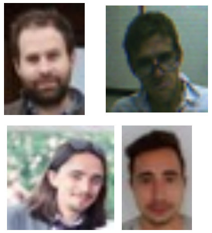
- **Input:** A digraph G = (V,E), a set of functions F, and a **collection D of demands.**
- A demand d ∈ D is modeled by a couple :
  - **a path** path(d) of length l(d) and
  - **a service function chain** sfc(d) of length s(d).
- A **setup cost** c(v,f) of function f in node v ∈ V .
- **Output:** A function placement Π ⊂ V × F
- **Objective:** minimize total setup cost $\sum_{(v,f)\in\Pi} c(v,f)$
- Similarly to [Sang et al. Infocom 2017], we consider the case of an operator which has **already routed its demands** and which now wants to optimize the placement of network functions.

Related Work

- Roughly two categories Heuristic-Based and ILP based
  - [Kuo et al. Infocom 2016] Maximizing the total number of admitted demands
  - [Mehraghdam et al. Cloudnet 2014] Minimizing the number of used nodes or the latency of the paths.
- Works closest to us, Approximation Algorithms
  - [Cohen et al. Infocom 2015] Minimizing setup cost near-optimal approximation algorithms with theoretically proven performance. However, no execution order of the network functions
  - [Sang et al. Infocom 2017] Minimizing the total number of network functions. But one single network function and *leave the placement of virtual functions with chaining constraint as an open problem for future research.*

Contributions

“First approximation algorithms taking into account ordering constraints.”



+ optimal on trees + validation

[Tomassilli, Giroire, Huin, Perennes INFOCOM 2018]

Preliminaries: Chains of Length 1

Modeling Trick 2

- **Direct equivalence** with the **Minimum Weight Hitting Set Problem**

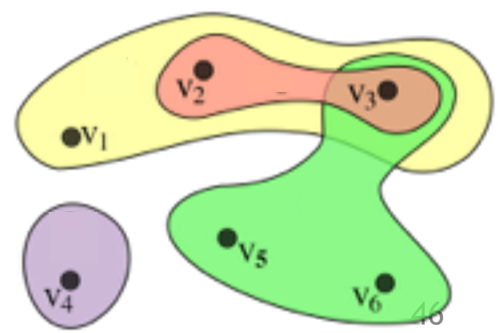
[Sang et al. Infocom 2017]

Preliminaries: Chains of Lengt 1

Modeling Trick 2

- Direct equivalence with the Minimum Weight Hitting Set Problem

- Input: Collection C of subsets of a finite set S.
  Output: A hitting set for C, i.e., a subset S′ ⊆ S such that S′ contains at least one element from each subset in C.
- Objective: Minimize the cost of the hitting set, i.e., $\sum_{x \in S'} c_x$

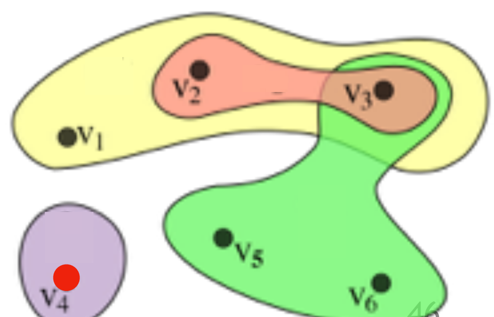


Preliminaries: Chains of Length 1

Modeling Trick 2

- **Direct equivalence** with the **Minimum Weight Hitting Set Problem**

- Input: Collection C of subsets of a finite set S.
  Output: A hitting set for C, i.e., a subset S′ ⊆ S such that S′ contains at least one element from each subset in C.
- Objective: Minimize the cost of the hitting set, i.e., $\sum_{x \in S'} c_x$

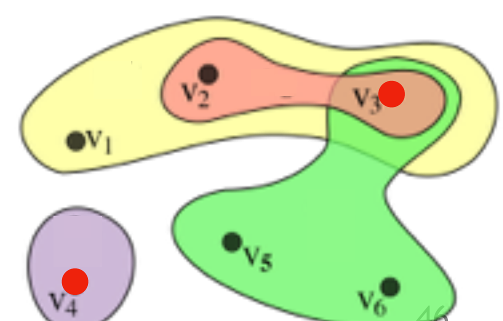


Preliminaries: Chains of Length 1

Modeling Trick 2

- **Direct equivalence** with the **Minimum Weight Hitting Set Problem**

- Input: Collection C of subsets of a finite set S. Output: A hitting set for C, i.e., a subset S′ ⊆ S such that S′ contains at least one element from each subset in C.
- Objective: Minimize the cost of the hitting set, i.e., $\sum_{x \in S'} c_x$



Preliminaries: Chains of Length 1

Modeling Trick 2

- Elements of S: possible **function locations**, i.e., the **vertices** in V . Each element has cost c(v).
- Sets in C: **paths** of the demands in D. Set = all path nodes $\{u_1, \ldots, u_{l(d)}\}$.

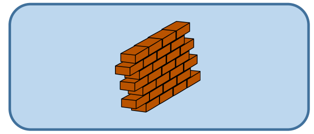
-> Placement of minimum cost covering all demands corresponds to a minimum cost hitting set.

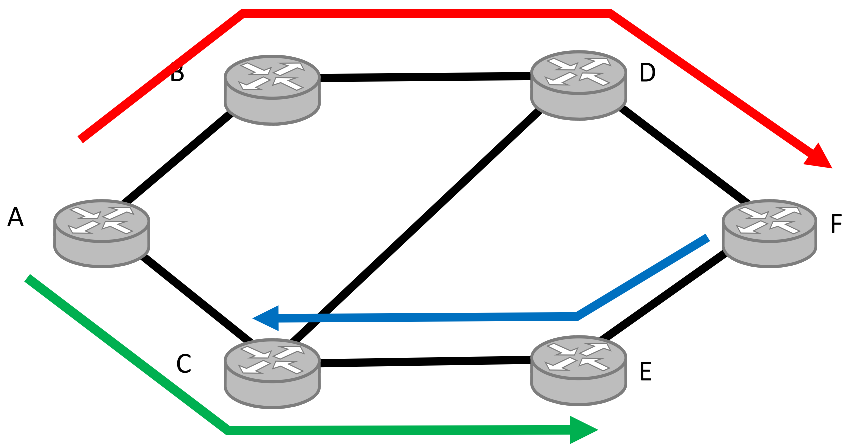
Preliminaries: Chains of Length 1

Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC

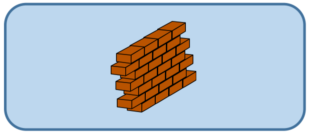


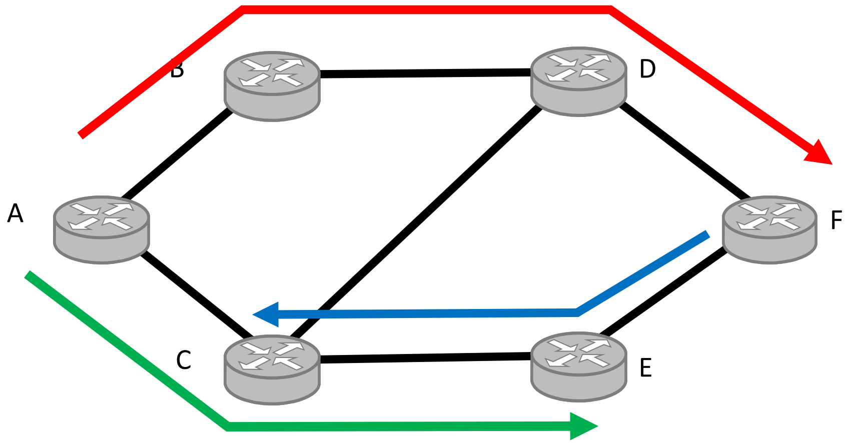


Preliminaries: Chains of Length 1

3 flows: A to F
A to E
F to C

SFC





ABDF

ACE

FEC

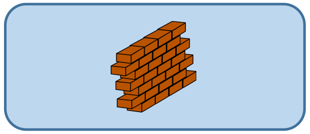
A
B
C
D
E
F

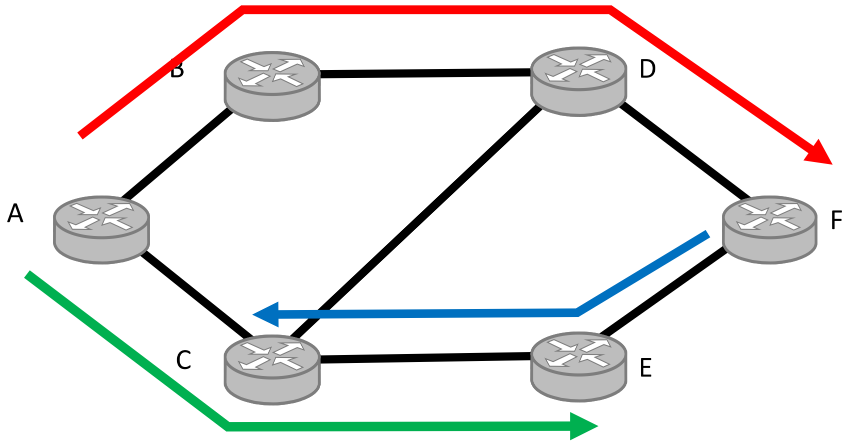
Preliminaries: Chains of Length 1

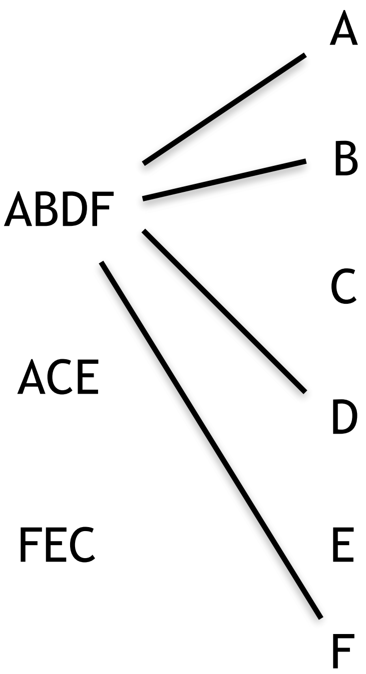
Modeling Trick 2

3 flows: A to F
A to E
F to C

} SFC





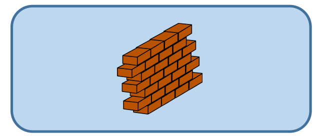


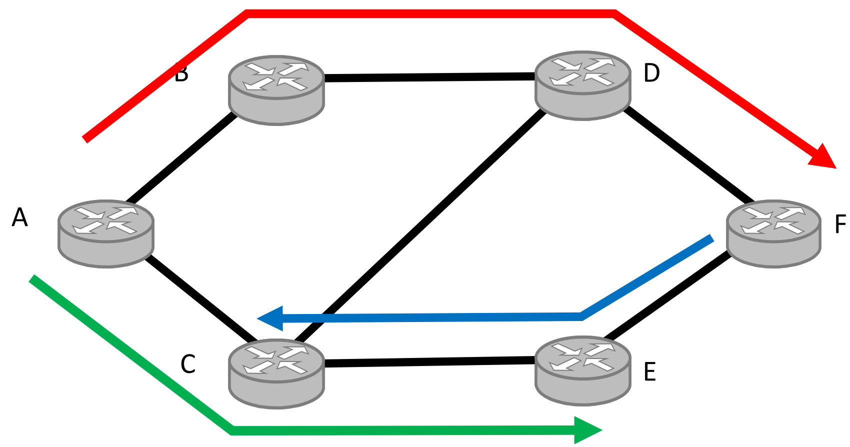
Preliminaries: Chains of Length 1

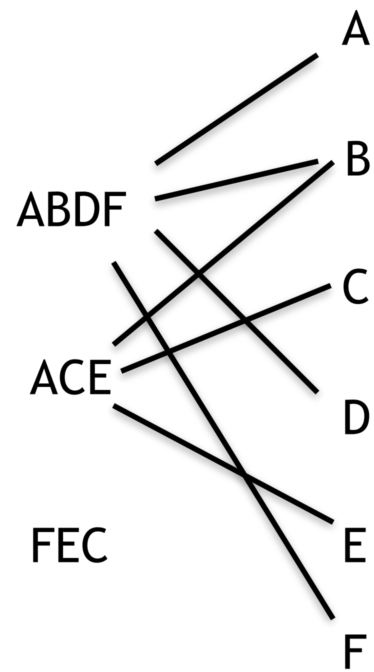
Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC





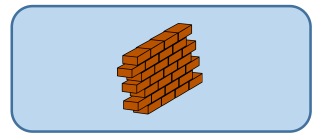


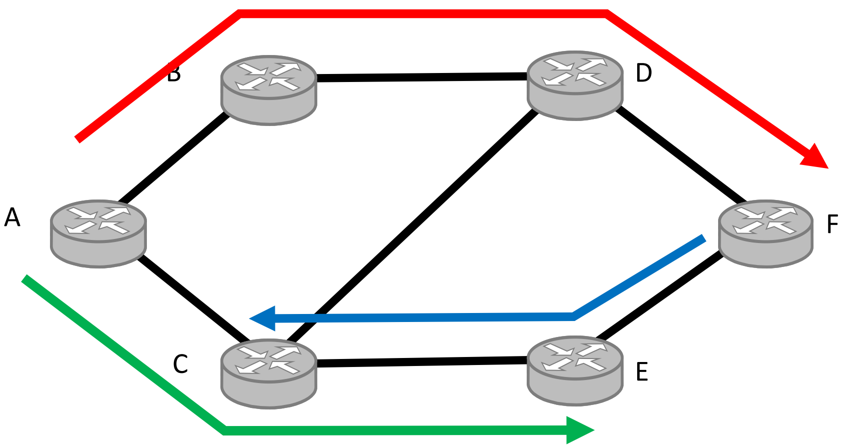
Preliminaries: Chains of Length 1

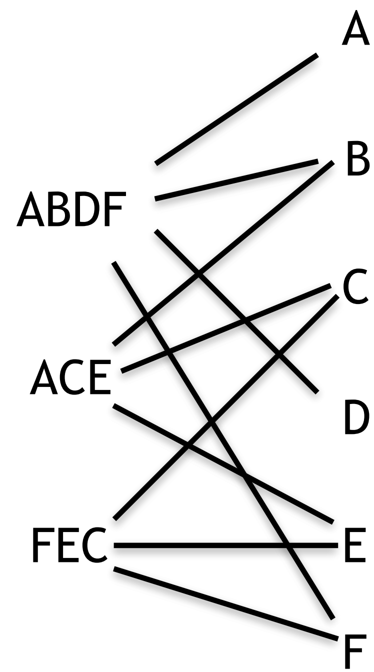
Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC





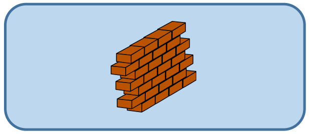


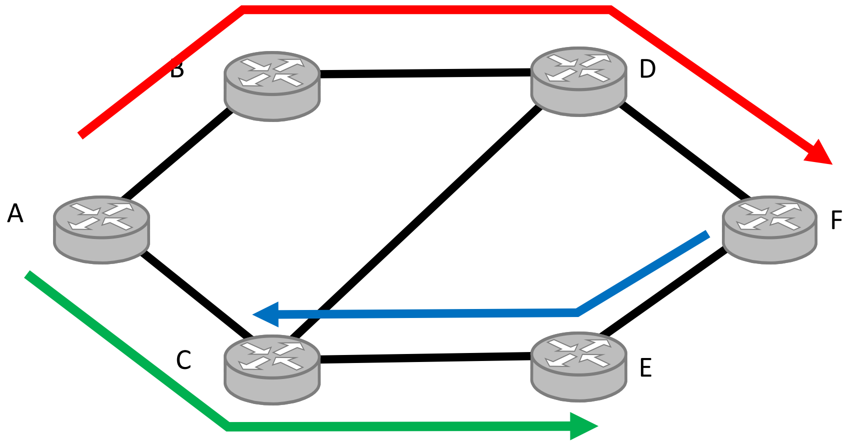
Preliminaries: Chains of Length 1

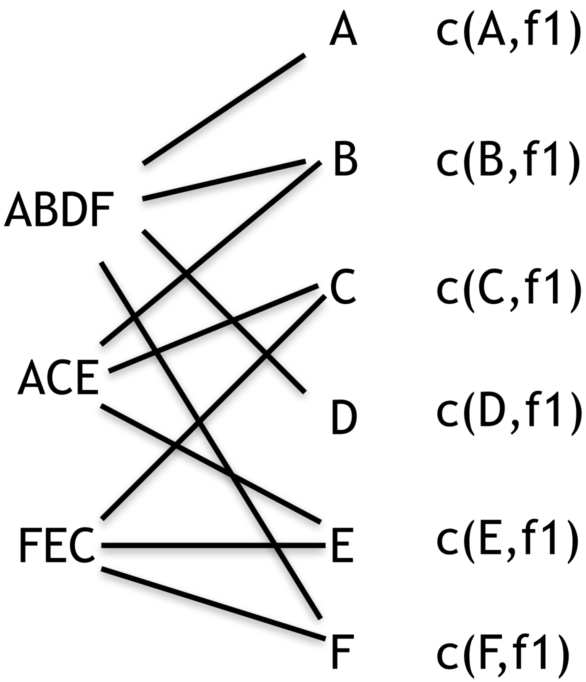
Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC





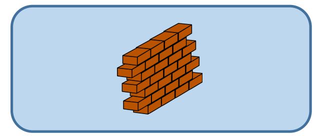


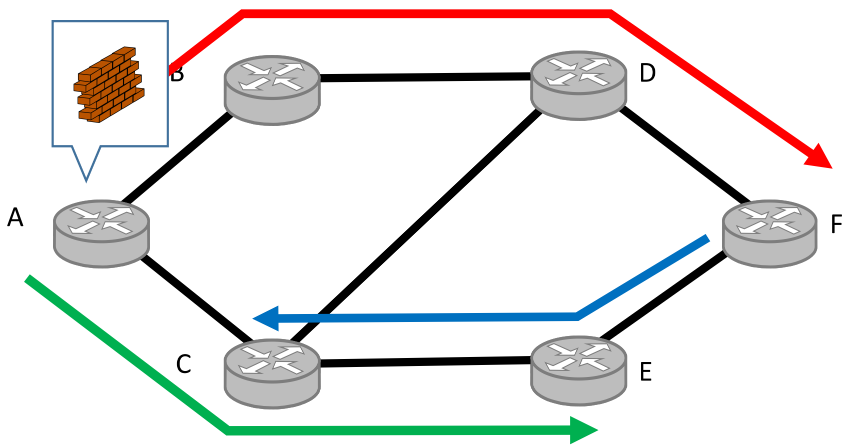
Preliminaries: Chains of Length 1

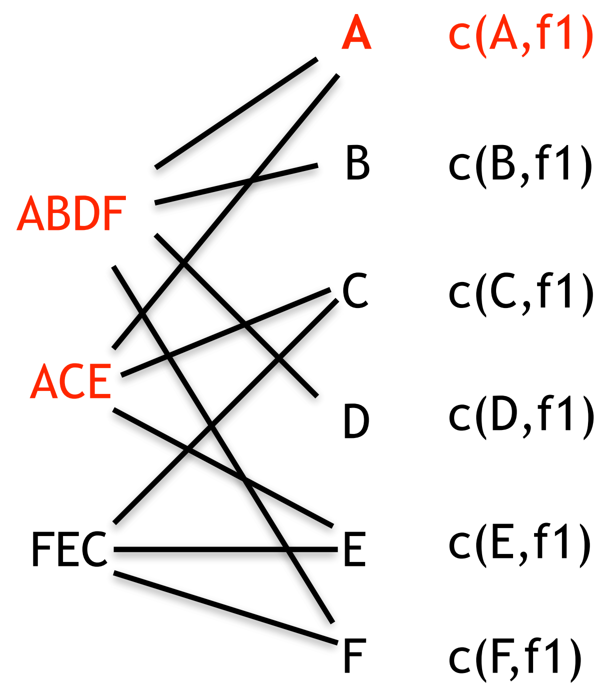
Modeling Trick 2

3 flows: A to F
A to E
F to C

} SFC





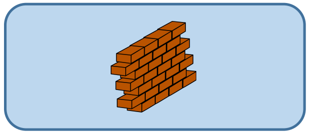


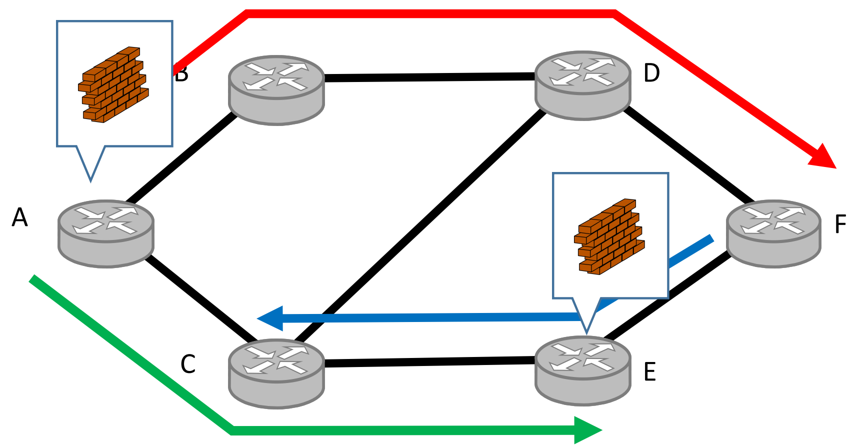
Preliminaries: Chains of Length 1

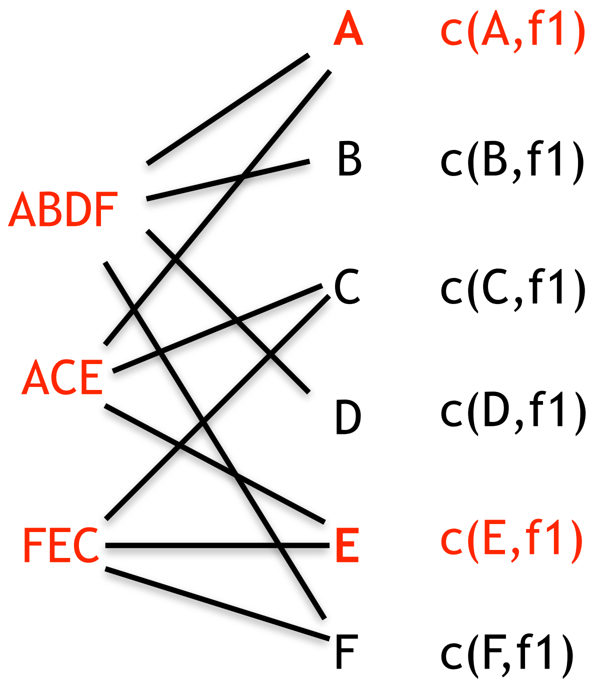
Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC





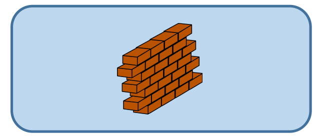


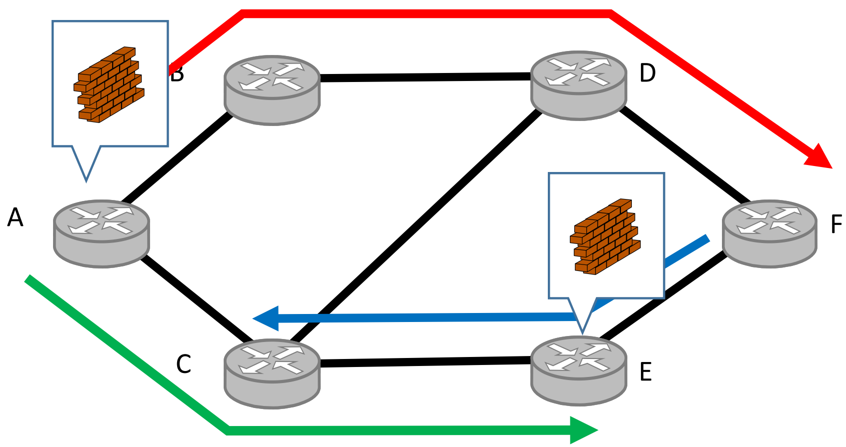
Preliminaries: Chains of Length 1

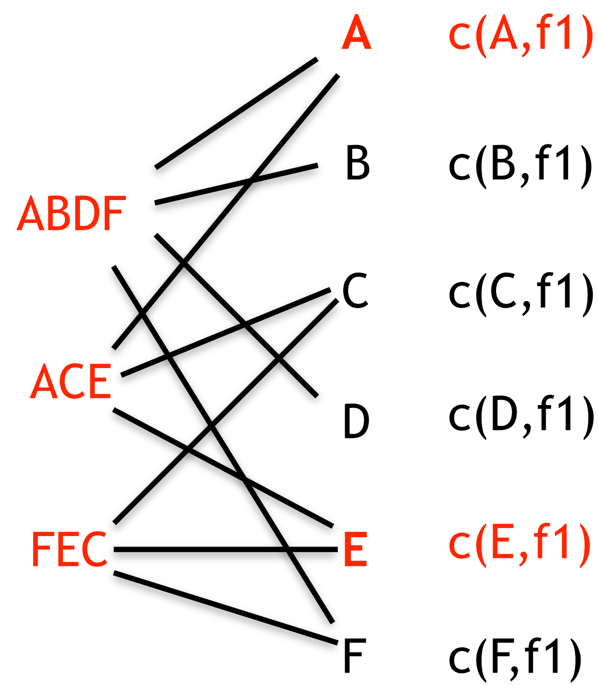
Modeling Trick 2

3 flows: A to F
A to E
F to C

SFC







Cost= c(A,f1)+c(E,f1)

Preliminaries: Chains of Length 1

Modeling Trick 2

- The equivalence directly gives:
  - On the positive side, an **H(|D|)-approximation** using the greedy-algorithm for Set Cover [Chvatal 1979].
  - On the negative side, SFC Placement Problem is **hard to approximate within ln(|D|)** [Alon et al. 2006].

General Case

- When length of the chain >= 2,
  Extension is not direct ***even for a single chain*.**

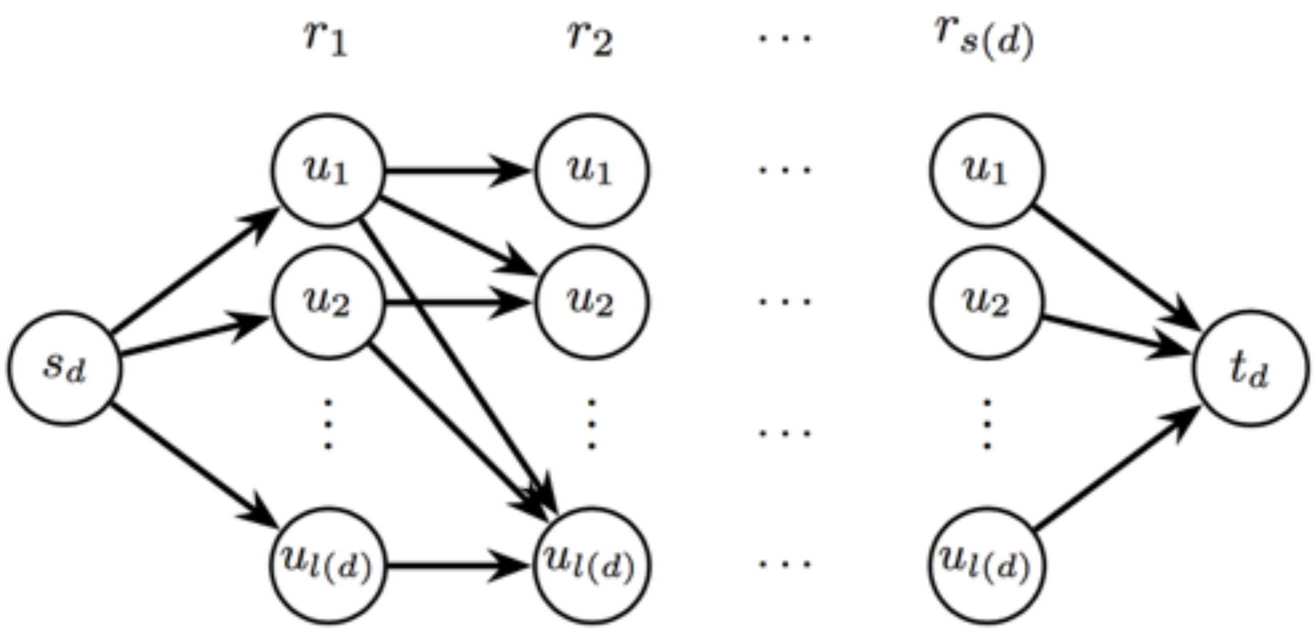
How to deal with the general case?

Associated Network

- **A key concept:** an **associated network** for each demand

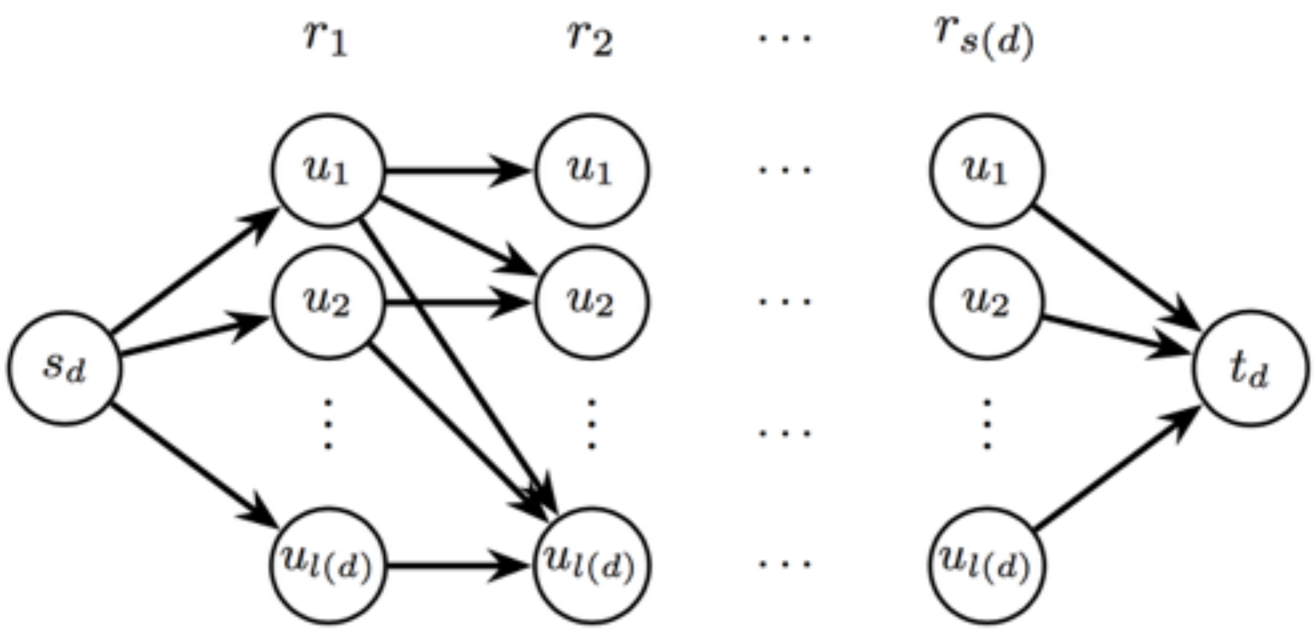
Associated Network

- **Definition:** **Associated Networks** H(d) for demand d with path(d) = $u_1, u_2, \ldots, u_{l(d)}$ and chain sfc(d) = $r_1, r_2, \ldots, r_s(d)$



Associated Network

- **Definition:** **Capacited** Associated Network
  H(d,Π) of demand d and **function placement Π**:
  - All arcs have infinite capacity.
  - **Capacity of node u** of layer i is 1 if $(u, r_i) \in \Pi$
  ar



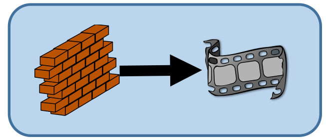
Associated Network

- **Key property:** A **demand $d \in D$ is satisfied by Π** if and only if there exists a **feasible st − path** in the capacitated associated network H(d,Π).

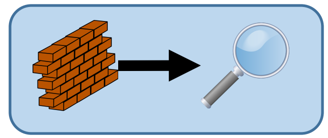
Associated Network: An Example

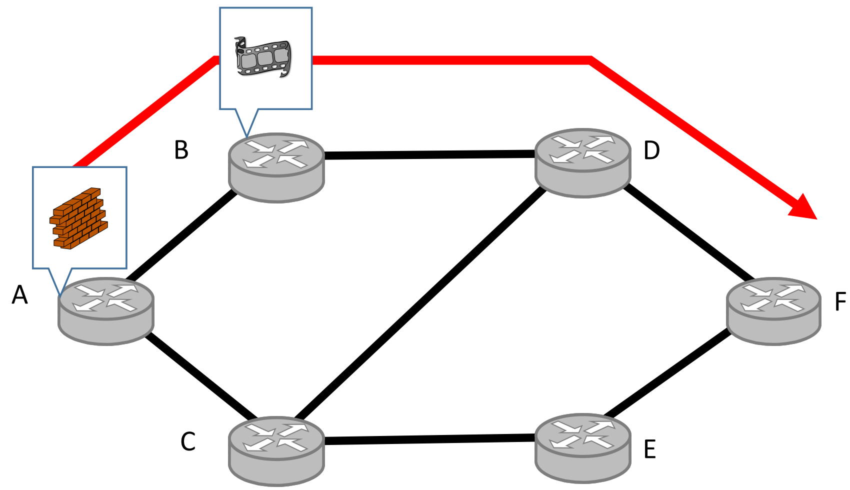
3 flows: A to F
A to E
F to C

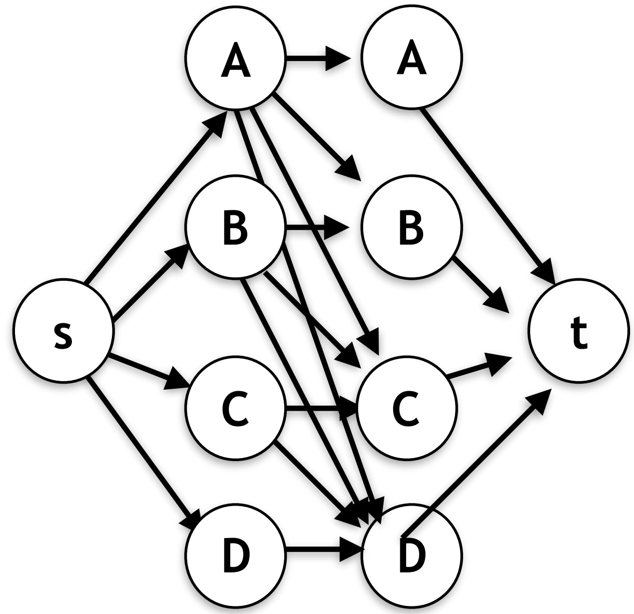
SFC A



SFC B

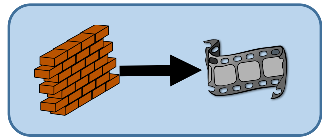


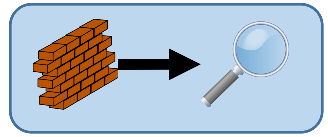


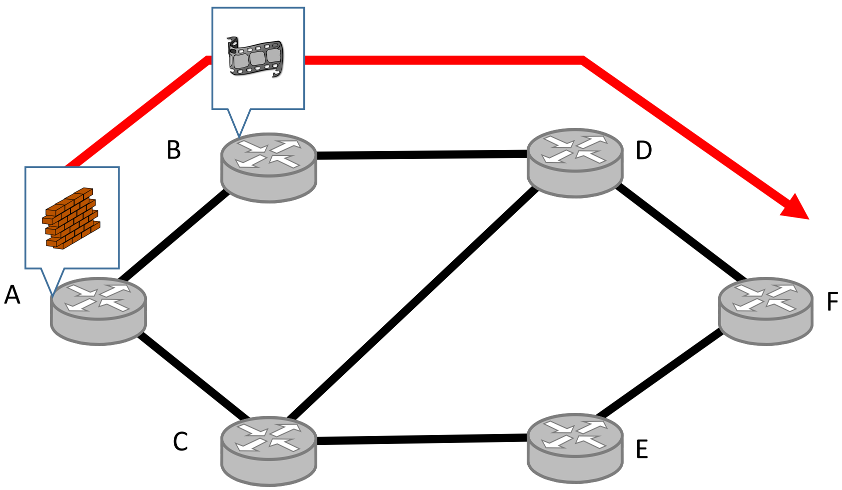


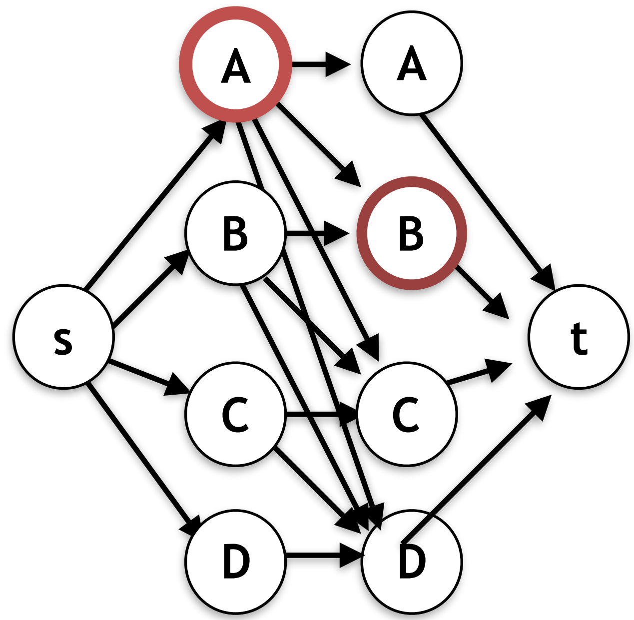
Associated Network: An Example

3 flows: A to F SFC A
A to E }
F to C } SFC B



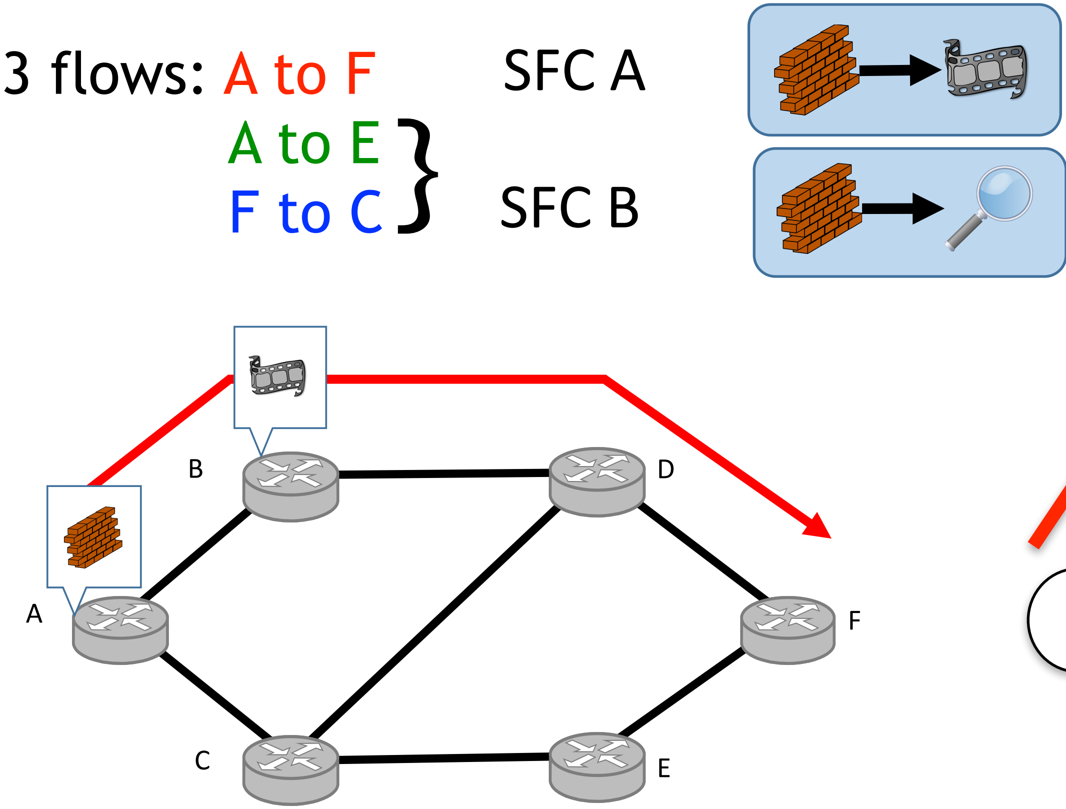


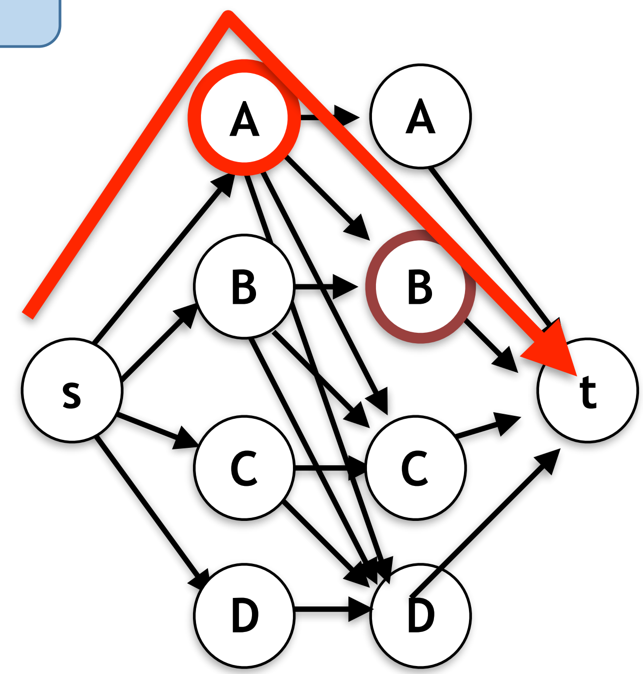




Associated Network: An Example

3 flows: A to F — SFC A
A to E, F to C — SFC B

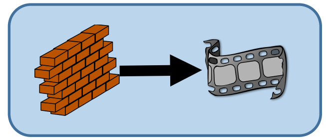


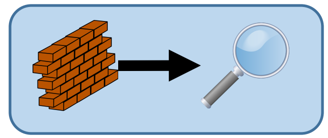


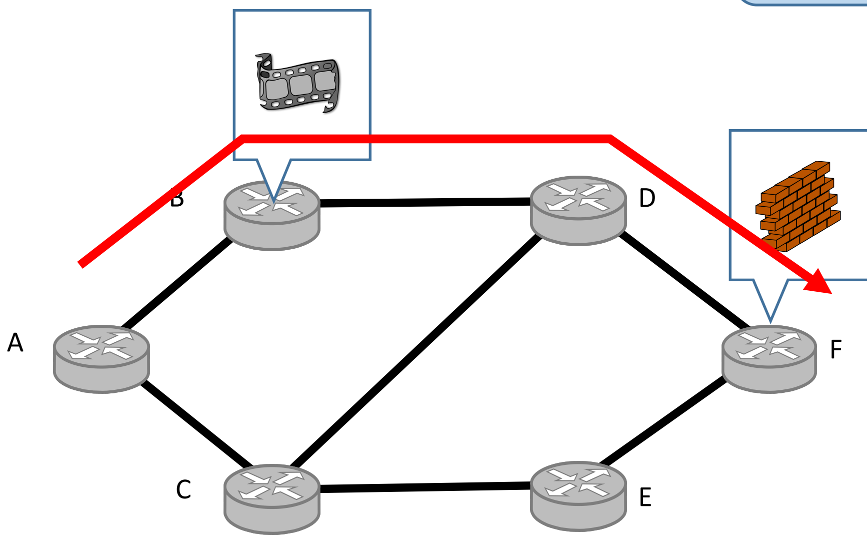
Associated Network: An example

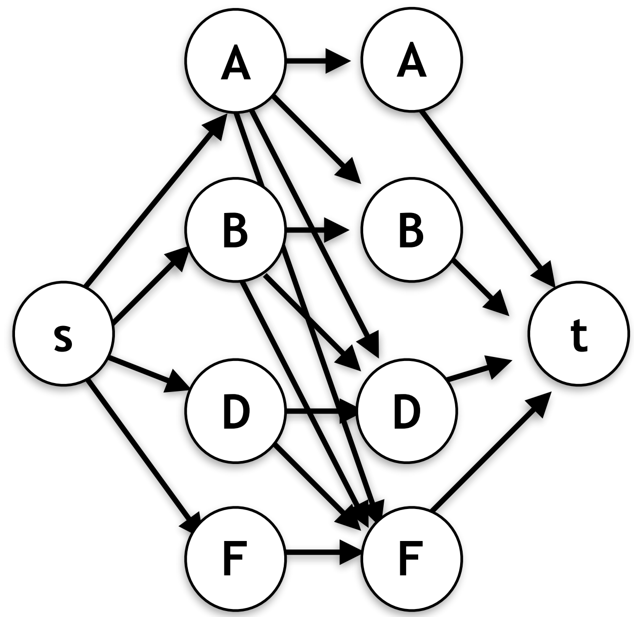
3 flows: A to F — SFC A

A to E
F to C — SFC B





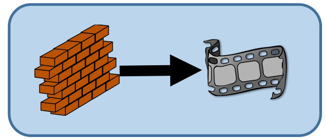


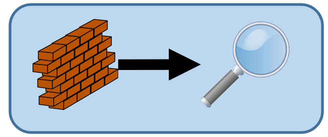


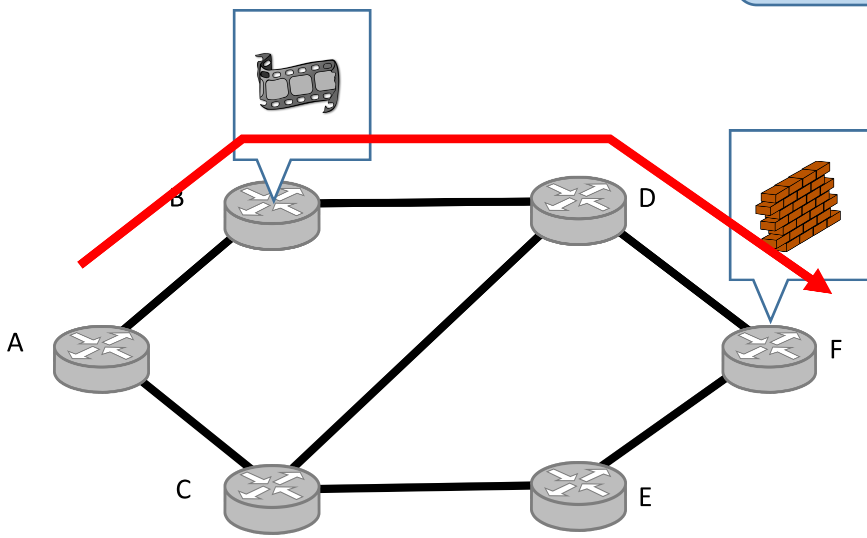
Associated Network: An Example

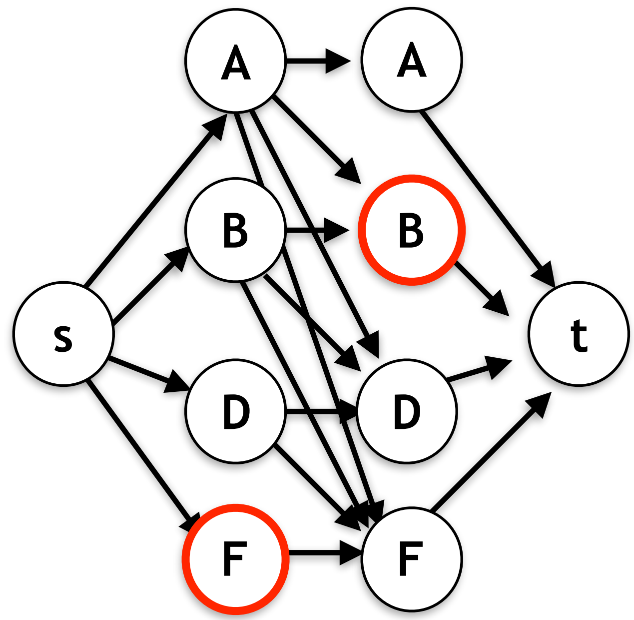
3 flows: A to F — SFC A

A to E }
F to C } — SFC B





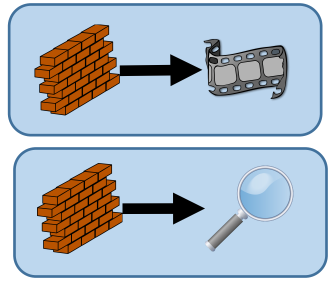


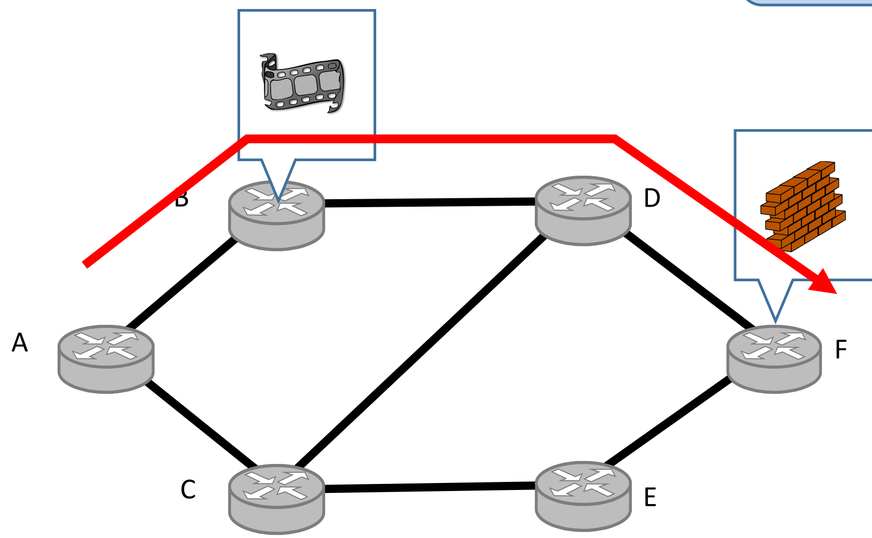


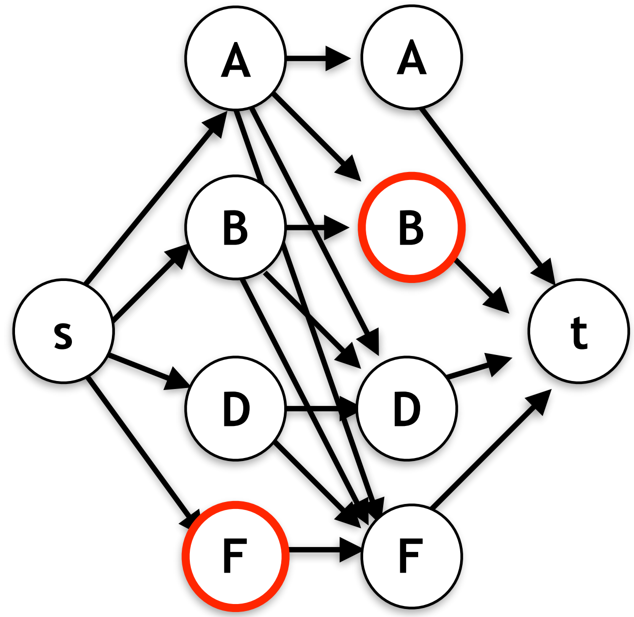
Associated Network: An Example

3 flows: A to F
A to E
F to C

SFC A
SFC B







Order not respected = No st-paths

New Formulation of the Problem

- **Goal:** *Link our problem with the Hitting Set Problem.*

- **Tool:** **Menger's theorem** for digraphs (max flow-min cut)
  "number of st – paths in a digraph is equal to the minimum st-vertex cut"

  -> **Existence of st-paths** <=> **cost >= 1** of minimum st-vertex cut

  -> **All cuts of the associated networks have to be hit.**

Approximation Algorithms

Proposition 2. *The problem* SFC-PLACEMENT $(\mathcal{D}, c)$ *is equivalent to a Hitting Set Problem with* $\sum_{d \in \mathcal{D}} \binom{l(d)+s(d)-1}{s(d)-1}$ *sets as an input. If each demand requires at most* $s_{\max}$ *network functions and is associated with a path of length smaller than* $l_{\max}$*, then the size of the instance is at most* $O(|\mathcal{D}| \cdot (l_{\max})^{s_{\max}-1})$.

-> leads to two approximation algorithms with logarithmic factor

- a greedy one (naive and fast versions)
- one using LP-rounding (naive and fast versions)

Contributions

Algorithmic Fact 1

- Investigated the problem of **placing VNFs** to satisfy the **ordering constraints** of the flows with the goal of **minimizing the total setup cost.**

- We proposed **two algorithms** that achieve a **logarithmic approximation factor.**

- For the special case of **tree network topologies** with only upstream and downstream flows, we devised an optimal algorithm.

An optimal algorithm for tree topologies

Modeling Trick 3

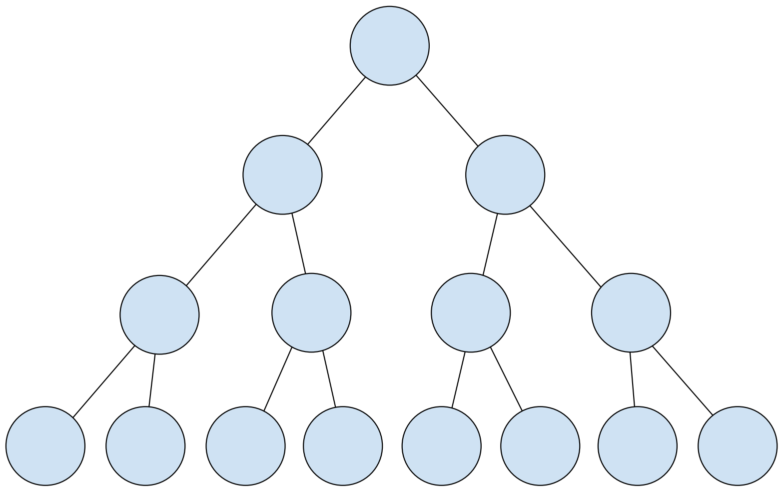
- Finding efficient algorithms for some class of graphs (such as trees)

  -> often important in practice e.g. for Mobile Edge Computing or FOG computing (specific topology of access networks)

An optimal algorithm for tree topologies

Modeling Trick 3

- **Tree topology.**
  - Physical network of any shape,
  - But clients communicating through a **logical tree** (e.g. CDNs, sensor networks, …)



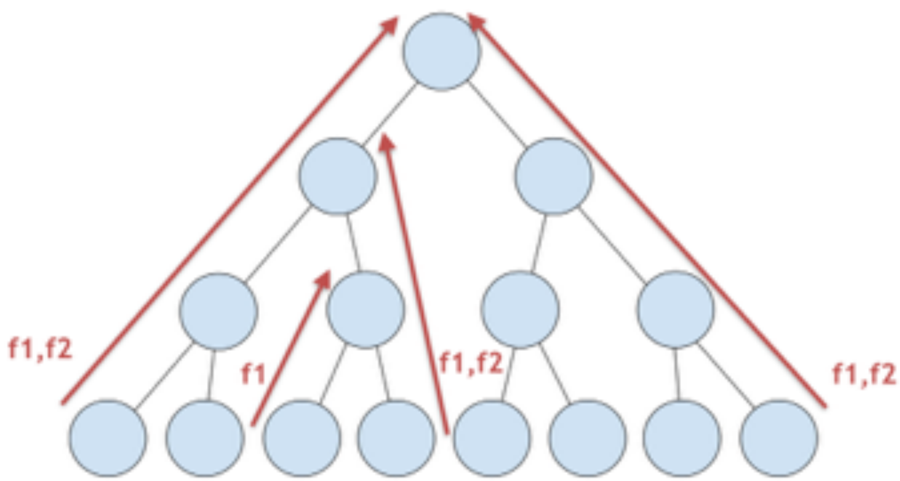
An optimal algorithm for tree topologies

Modeling Trick 3

- **Theorem:** SFC Placement Problem **NP-hard** even on a **tree** and with a **single network function.** (**Proof:** Reduction from Vertex Cover)

Algorithmic Fact 1

- **Polynomial exact algorithm** for **upstream or downstream flows** based on dynamic programming.



SFC - Conclusions

- **Efficient algorithms** proposed for SFC provisioning
- **Theoretical framework** for studying the placement problem **with ordering constraints.**

- Unaddressed issues:
  - accounting of practical constraints such as **soft capacities** on network functions or **hard capacities** on network nodes.
  - Affinity/anti-affinity rules
  - Partial order
  - Latency

Future research direction: possible to efficiently approximate these problems?

SFC - Conclusions

- **SDN** and **NFV** bring several **benefits**:
  - simplify management
  - enhance flexibility of the network
  - reduce the network cost
- But also several **challenges** that need to be addressed to fully attain their benefits

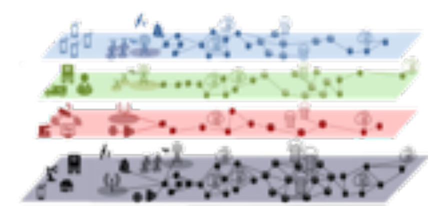
SDN-NFV enabled network has the potential to boost NFV deployment and support new efficient and cost-effective services

Future Directions

- Several major revolutions:
  - 5G
  - IoT
  - Mobile Edge Computing
  - …

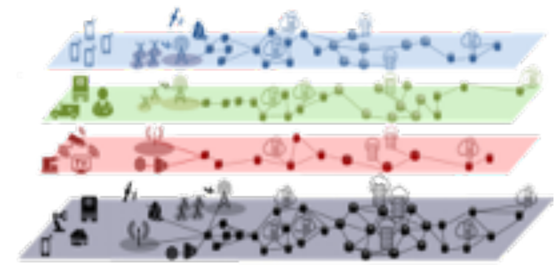
New challenges

- Assign slices to capacity slots of physical links -> **slicing**
- Dynamic SFC Placement
- Network Reconfiguration



New algorithmic problems to be solved

Network Slicing



- Assign slices to capacity slots of physical links
  - each slice is independent from each other
  - each slice may have different QoS requirements
- 2 different network slicing strategies:
  - **SOFT**: traffic is multiplexed in queuing systems: high load may affect other slices
  - **HARD**: each slice has dedicated resources at physical and MAC layers

(Parallel with isolation problems VM vs Containers)

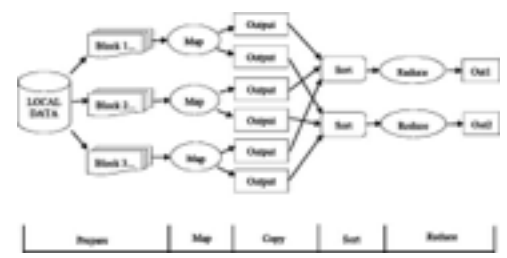
Outline: Summary

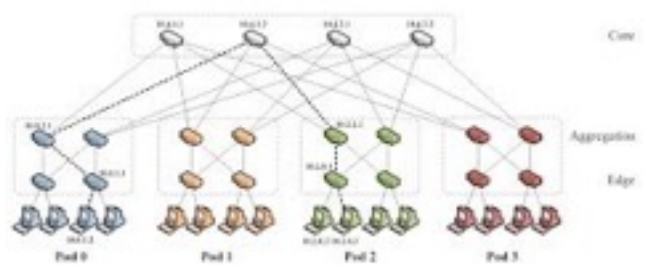
1. Motivation
2. A new situation: SDN and NFV
3. Placement of virtual network functions
    - Use case: Service Function Chaining
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. What next?

Convergence Data Centers/Networks

- Convergence
  - of infrastructures,
  - of their control with the next generation SDN/NFV networks
- Allows a joint optimization of applications and network trafic.
- Revisit the fundamental problems of scheduling in data centers.

Topic of a joint lab between Orange and Inria "Big OS"



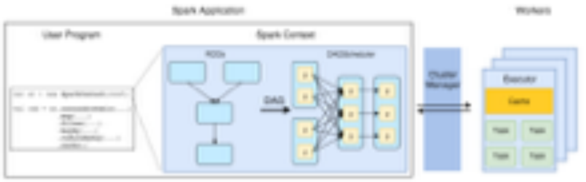


Outline

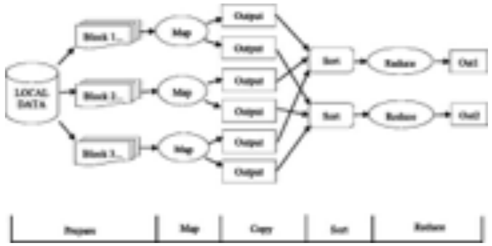
1. Motivation
2. A new situation: SDN and NFV
3. Placement of virtual network functions
    - Use case: Service Function Chaining
4. **Coflows for datacenters**
5. **Scheduling with network tasks**
6. Tools to evaluate solutions
7. What next?

Reminder

- More and more data-oriented parallel computing solutions (e.g., MapReduce, Dryad, CIEL, and Spark)

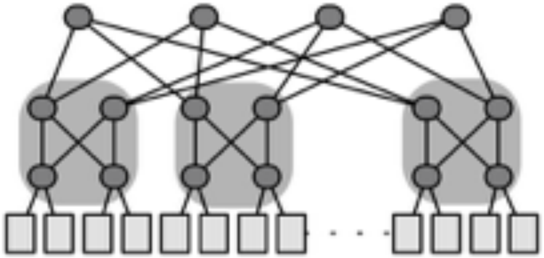


- Traditional scheduling consider properties of
  - **server** (e.g., CPU and memory usage)
  - **job** (e.g., execution time, deadline)



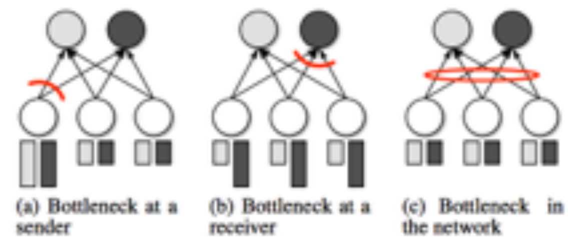
Network resources usually not taken into consideration

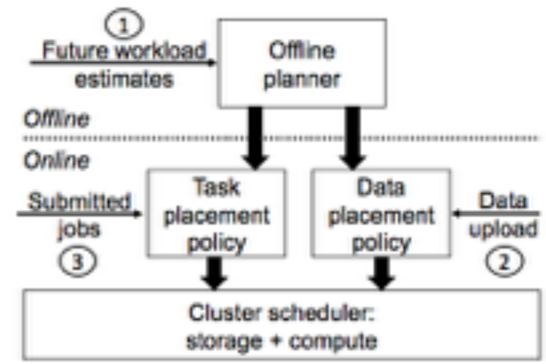
- Communications account for up to 50% of job completion time [Chowdhury, et al. Orchestra SIGCOMM 2011]



Related Work

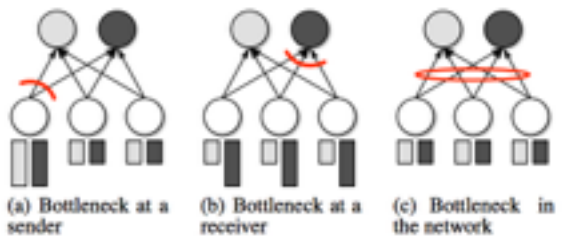
- **Optimizing data center communications.**
  - [Chowdhury et al. Sigcomm 2011] *Orchestra*. Load balancing mechanisms to improve the shuffle phase.
  - [Jalaparti et al. Sigcomm Rev. 2015] *Corral*. Using job recurrence to place data and large computation locality .

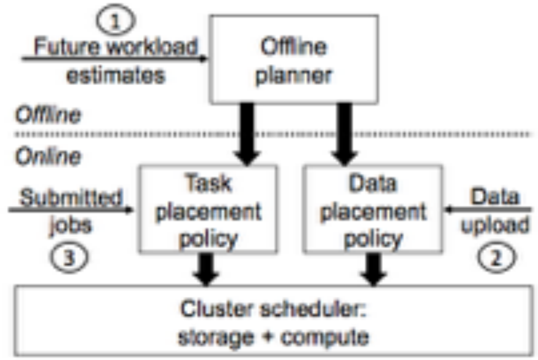




Related Work

- **Optimizing data center communications.**
  - [Chowdhury et al. Sigcomm 2011] *Orchestra*. Load balancing mechanisms to improve the shuffle phase.
  - [Jalaparti et al. Sigcomm Rev. 2015] *Corral*. Using job recurrence to place data and large computation locality .

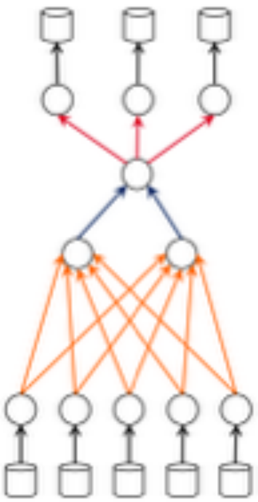




Few theoretical frameworks and provably efficient algorithms

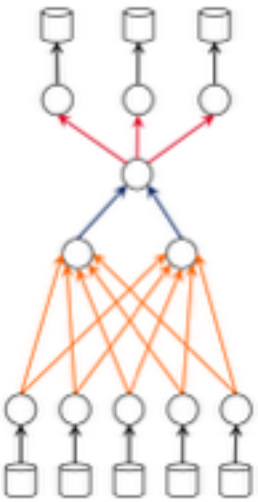
Related Work

- **Theoretical frameworks for Scheduling of complex workflows**
  - [Graham Bell System Tech. Journal 1966] Scheduling with precedence constraints or list scheduling.
    Main result: 2-1/m-approx.
  - In the 90s, scheduling with communication delays. Minimizing makespan still an open problem.
    However, 2-approx if uniform delays and task replication [Papadimitriou Yannakakis SIAM J. of Computing 1990] or if unitary costs [Rayward-Smith DAM 1987]



Related Work

- **Theoretical frameworks for Scheduling of complex workflows**
  - [Graham Bell System Tech. Journal 1966] Scheduling with precedence constraints or list scheduling.
    Main result: 2-1/m-approx.
  - In the 90s, scheduling with communication delays. Minimizing makespan still an open problem.
    However, 2-approx if uniform delays and task replication [Papadimitriou Yannakakis SIAM J. of Computing 1990] or if unitary costs [Rayward-Smith DAM 1987]



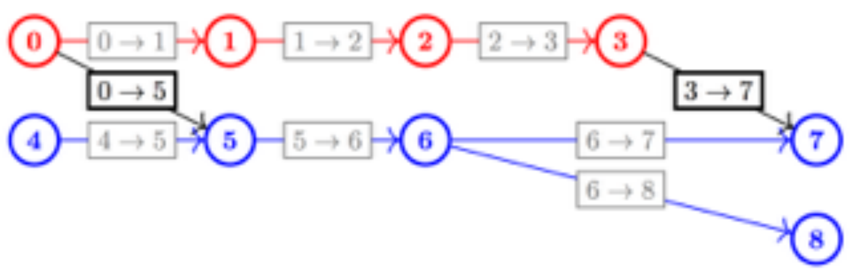
> No Network Capacity is assumed: ***all communications*** can be done ***at the same time*** without changing the delay!

Two Theoretical Frameworks

1. **Coflows**
   or scheduling group of dependant flows

[Chowdhury,Stoica Hotnets 2012]

2. **Network tasks**
   or scheduling while optimizing network resources



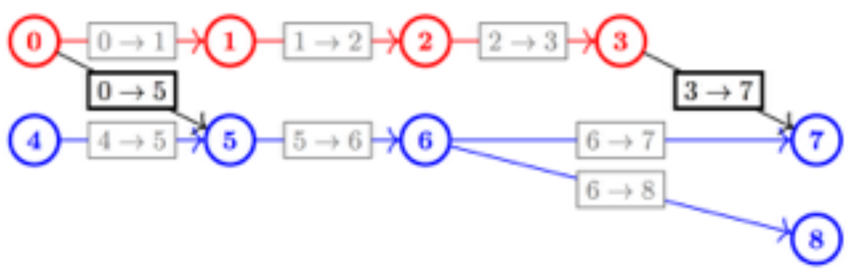
[Giroire,Huin,Tomassilli,Pérennes, INFOCOM 2019]

Two Theoretical Frameworks

1. **Coflows**
   or scheduling group of dependant flows

[Chowdhury,Stoica Hotnets 2012]

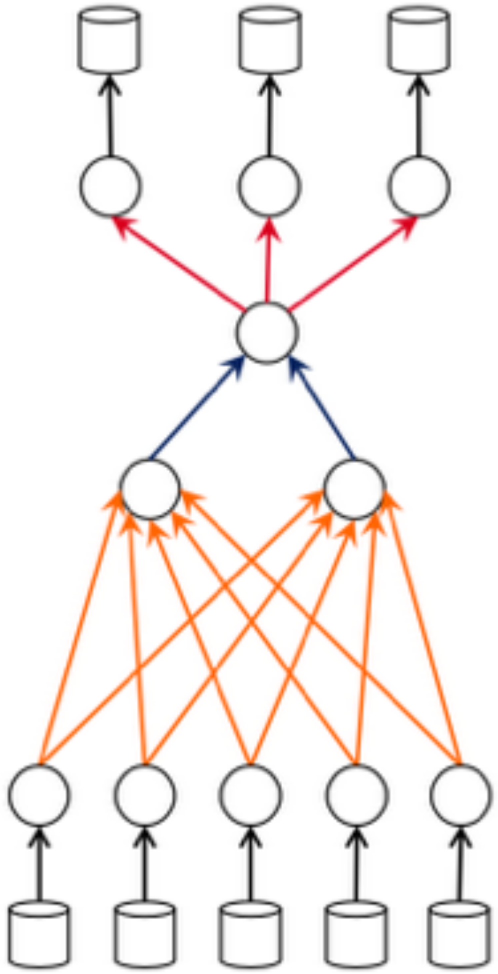
2. **Network tasks**
   or scheduling while optimizing network resources



[Giroire,Huin,Tomassilli,Pérennes, INFOCOM 2019]

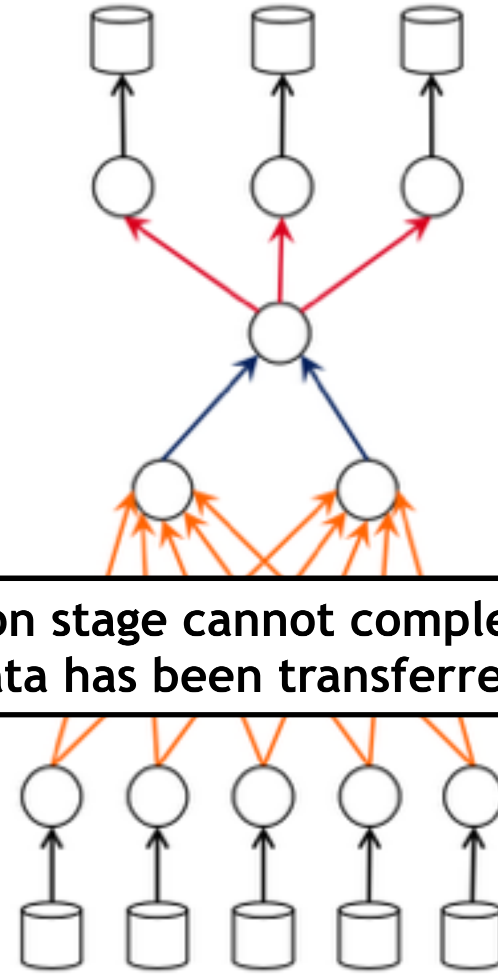
Distributed Data-Parallel Applications

- Multi-stage dataflow
  - Computation interleaved with communication
- Computation Stage (e.g., Map, Reduce)
  - Distributed across many machines
  - Tasks run in parallel
- Communication Stage (e.g., Shuffle)
  Between successive computation stages



Distributed Data-Parallel Applications

- Multi-stage dataflow
  - Computation interleaved with communication
- Computation Stage (e.g., Map, Reduce)
  - Distributed across many machines
  - Tasks run in parallel
- Communication Stage (e.g., Shuffle)
  Between successive computation stages



A communication stage cannot complete until all the data has been transferred

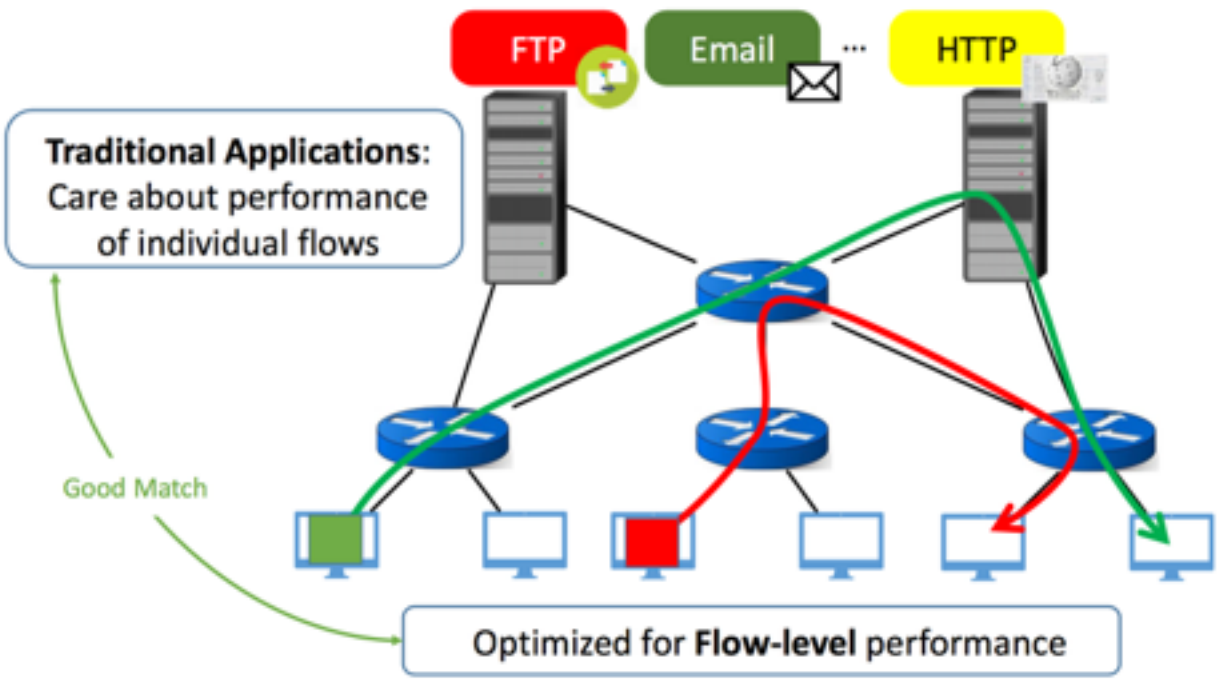
Question

How to design the network for data parallel applications?

- What are good communication abstractions?

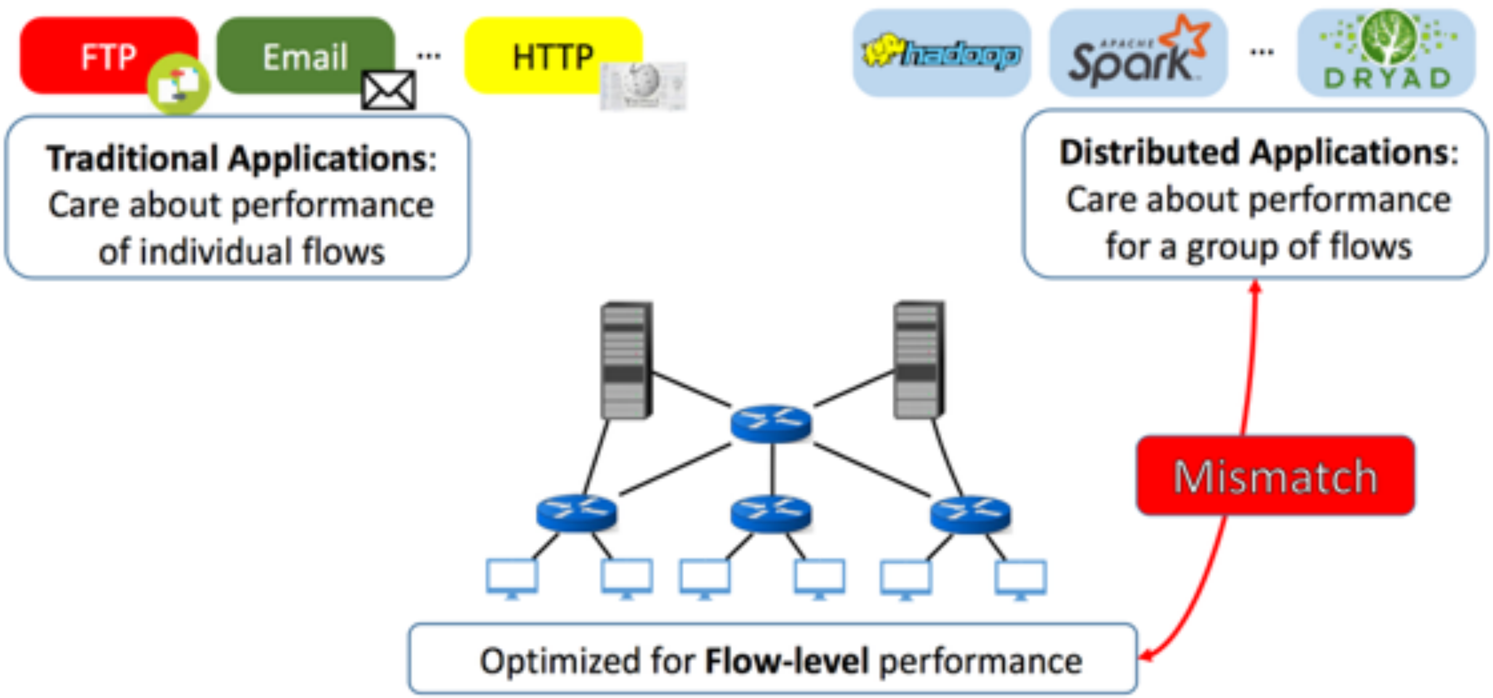
Traditional solution: The flow abstraction

Flow: Transfer of data from a source to a destination



E.g., Lots of work to ensure Per-Flow Fairness and/or minimize Flow Completion Time

Is Flow Still the Right Abstraction?

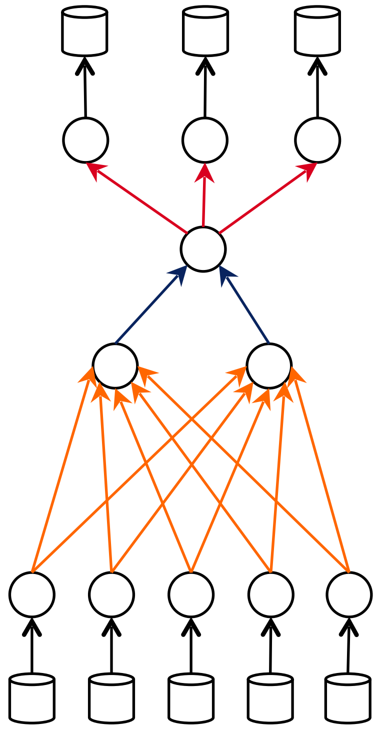


Independent flows cannot capture the collective communication behavior common in data-parallel applications

The Coflow abstraction

- Coflow = Collection of semantically related flows [1]

- Communication abstraction for data-parallel applications to express their performance goals

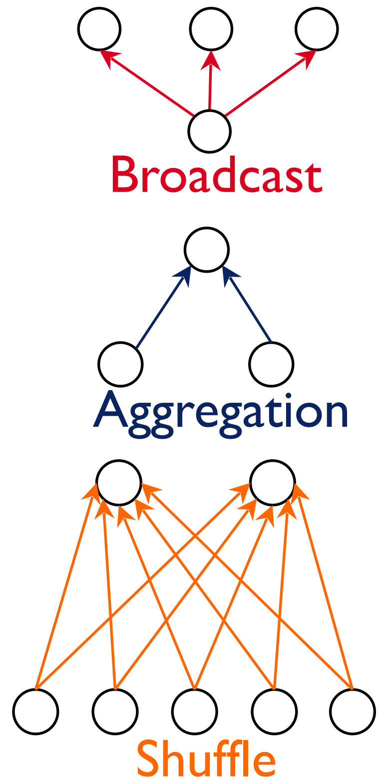


[1] Chowdhury,Stoica Hotnets 2012

The Coflow abstraction

- Coflow = Collection of semantically related flows [1]

- Communication abstraction for data-parallel applications to express their performance goals

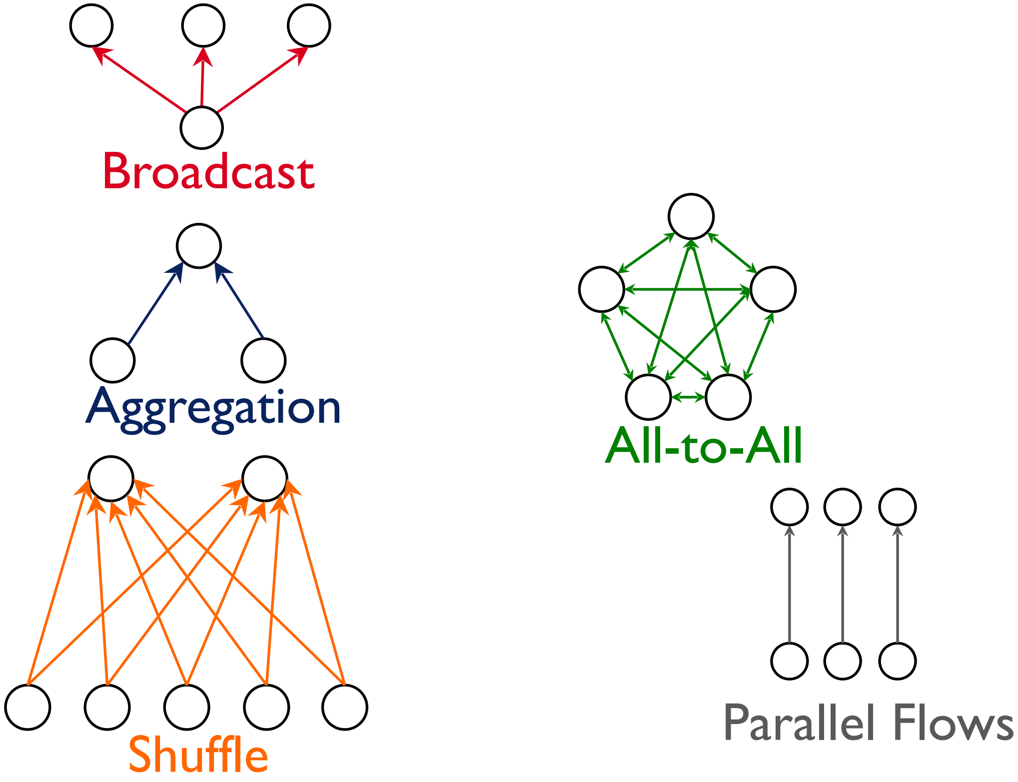


[1] Chowdhury,Stoica Hotnets 2012

The Coflow abstraction

- Coflow = Collection of semantically related flows [1]

- Communication abstraction for data-parallel applications to express their performance goals

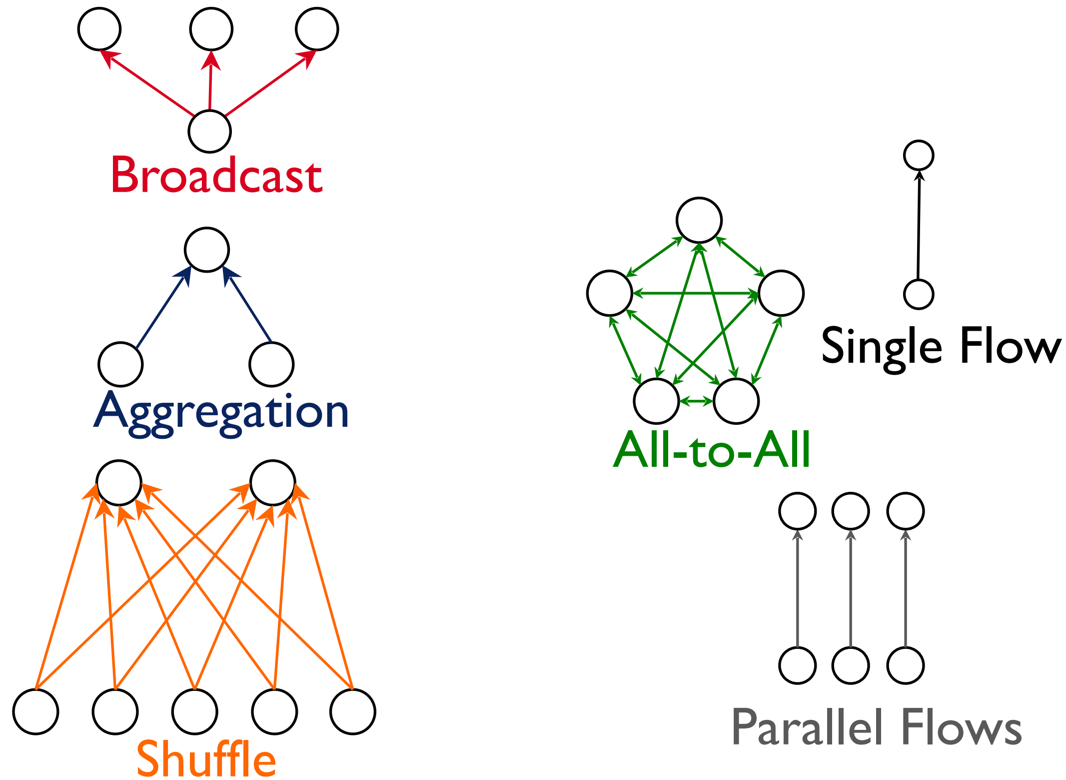


[1] Chowdhury,Stoica Hotnets 2012

The Coflow abstraction

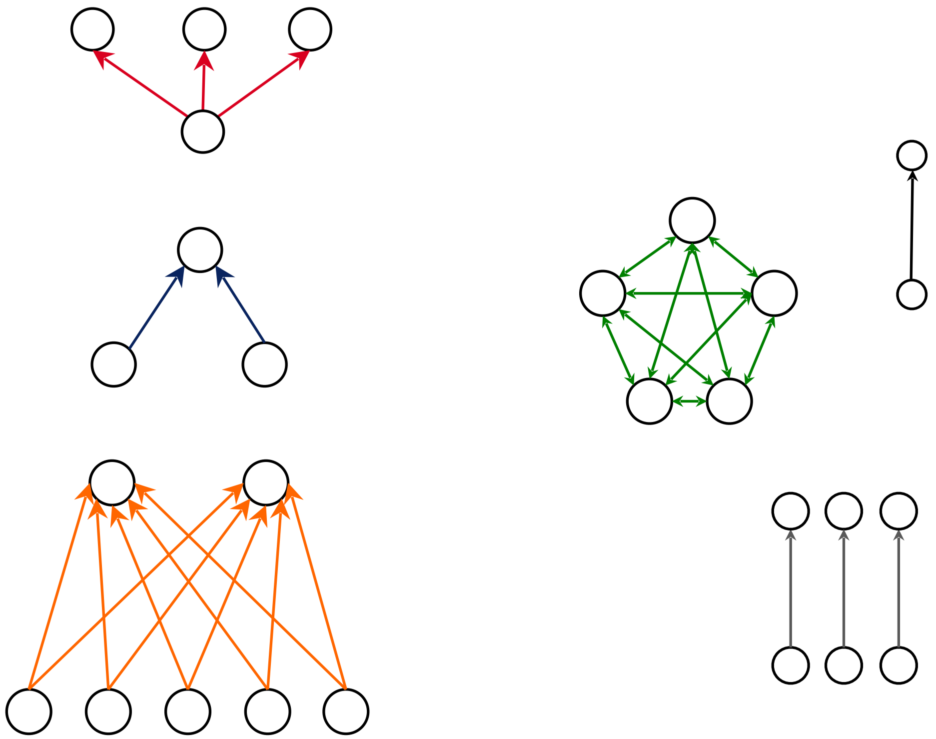
- Coflow = Collection of semantically related flows [1]

- Communication abstraction for data-parallel applications to express their performance goals



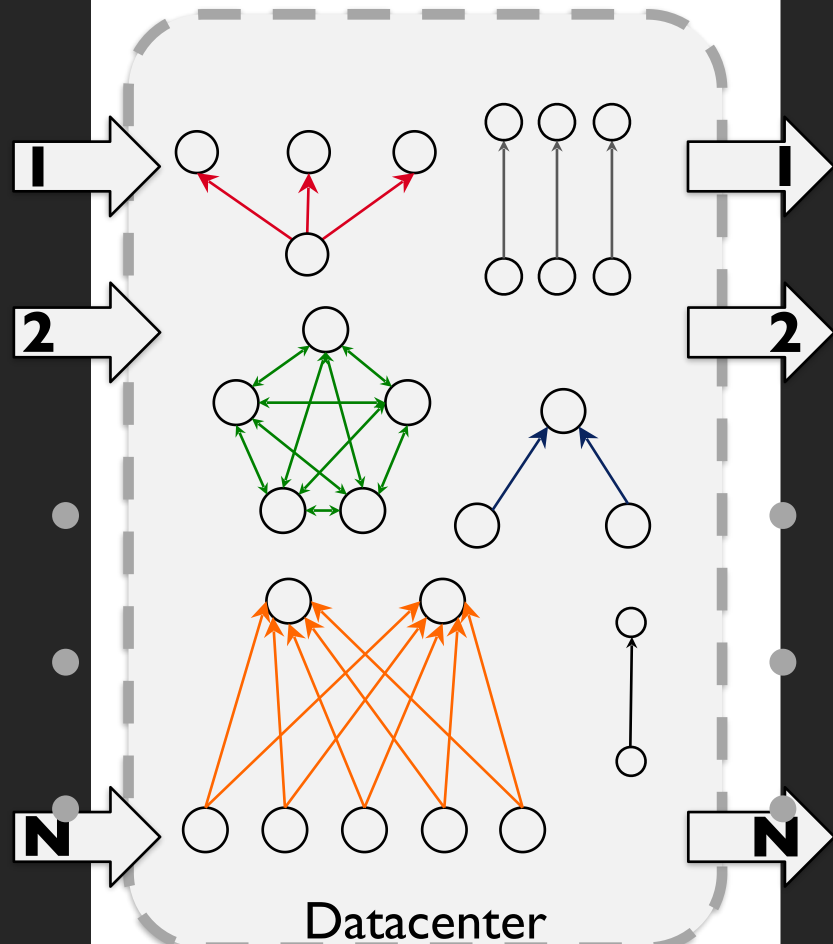
[1] Chowdhury,Stoica Hotnets 2012

The Coflow abstraction



The Coflow abstraction

How to schedule coflows online …



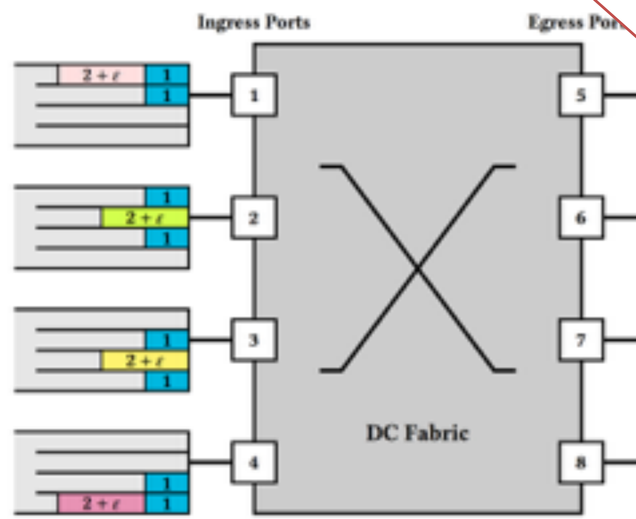
… for faster #1 completion of coflows?

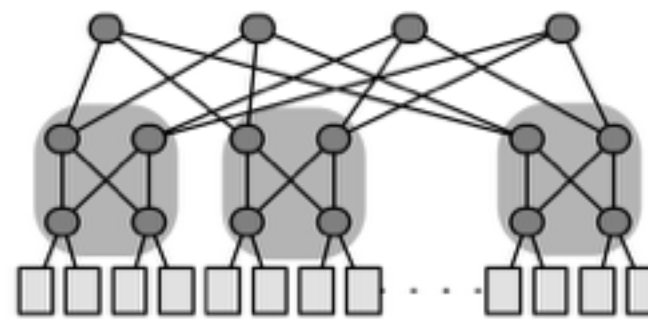
… to meet #2 more deadlines?

Network and Coflow Model

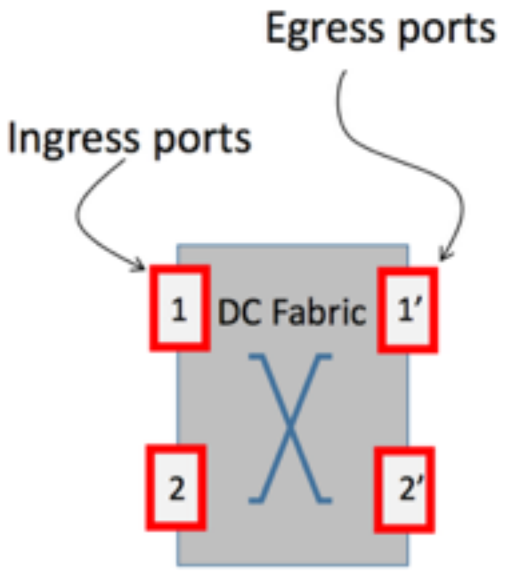
Modeling Trick 4

- "Big switch" conceptual model = abstract out the datacenter network fabric as one big switch interconnecting servers.
- Assumption: ***the fabric core can sustain 100% throughput*** and only the ingress (NICs) and egress (TOR switches) queues are potential congestion points.
- Indeed: most data center network architecture (e.g. Fat Tree) have full bissection bandwidth and are permutation networks.





Network and Coflow Model



- **Big-switch model**

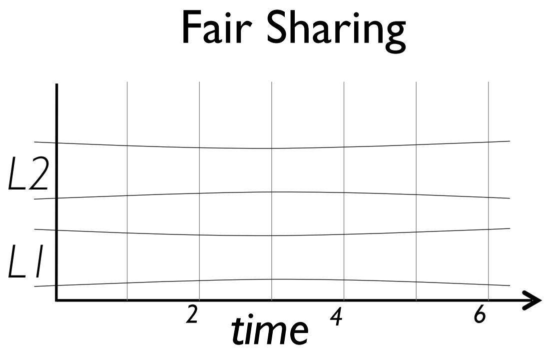
- **Clairvoyant scheduler** = Coflow details known at arrival time:
  - Source-destination for each flow
  - Size of each flow
  - Coflow weight

- **Considered Metric:** Coflow Completion Time (CCT) = Time when all flows of a coflow have completed

Goal: Minimize Average Weighted CCT

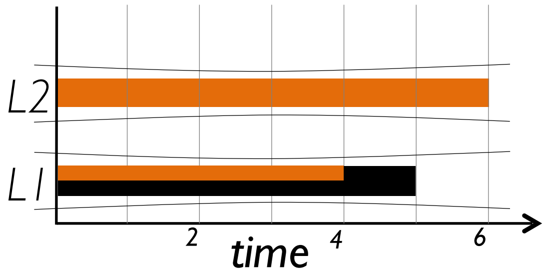
Benefit of inter-coflow scheduling

Benefit of inter-coflow scheduling



Benefit of inter-coflow scheduling

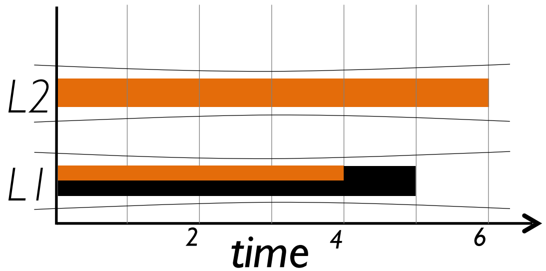
Fair Sharing



Coflow1 comp. time = 5
Coflow2 comp. time = 6

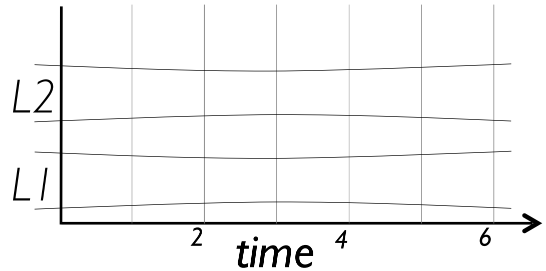
Benefit of inter-coflow scheduling

Fair Sharing



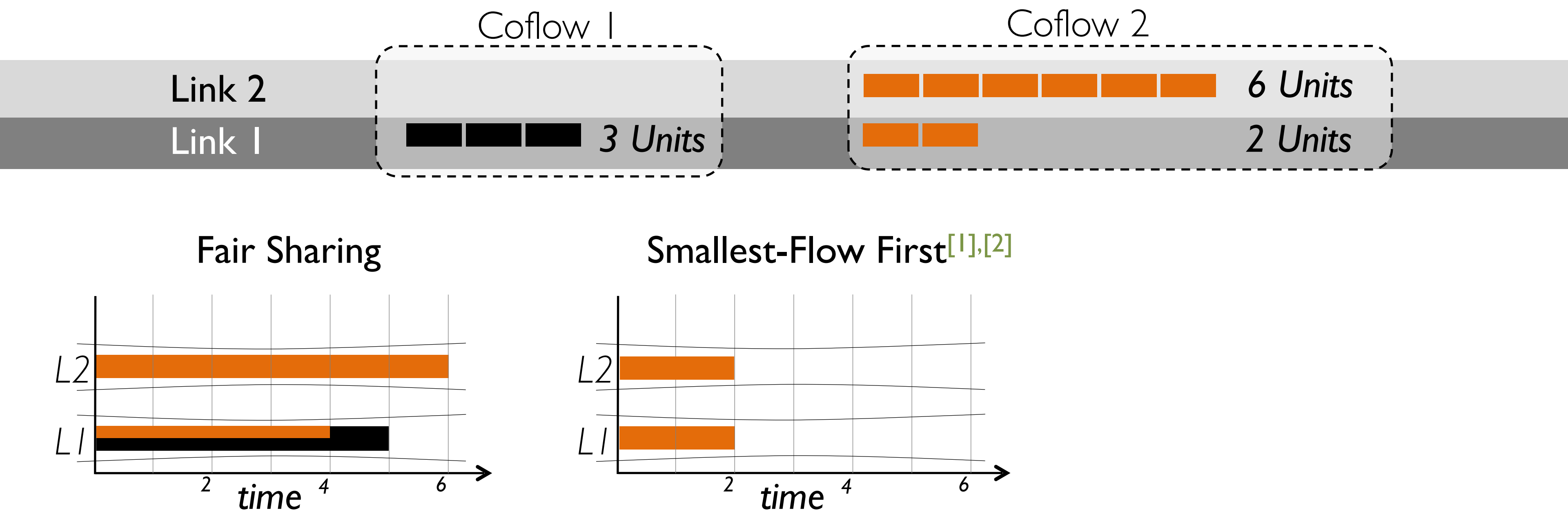
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1],[2]



[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Benefit of inter-coflow scheduling

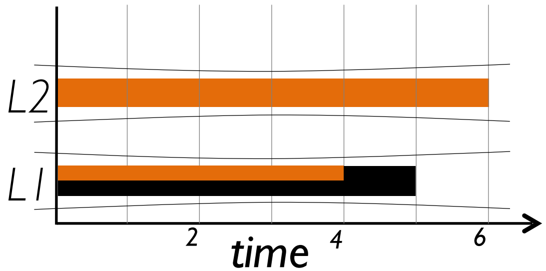


Coflow1 comp. time = 5
Coflow2 comp. time = 6

[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

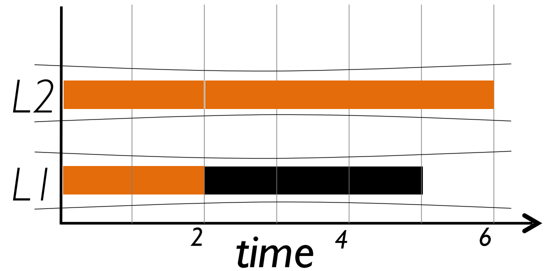
Benefit of inter-coflow scheduling

Fair Sharing



Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1],[2]

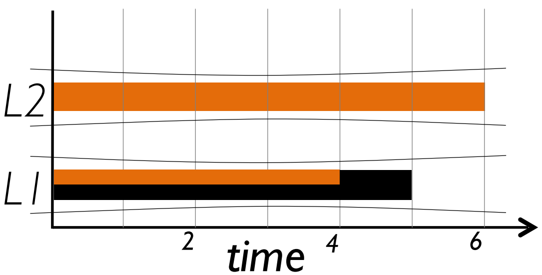


Coflow1 comp. time = ***5***
Coflow2 comp. time = ***6***

[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

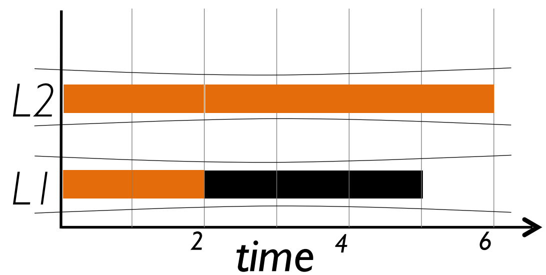
Benefit of inter-coflow scheduling

Fair Sharing



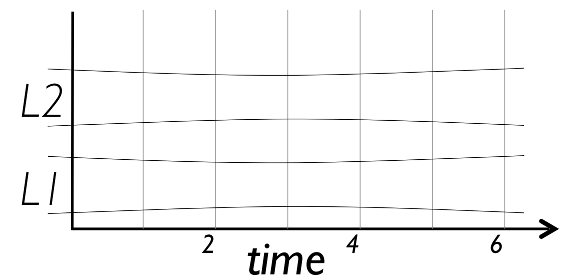
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1],[2]



*Coflow1 comp. time = **5***
*Coflow2 comp. time = **6***

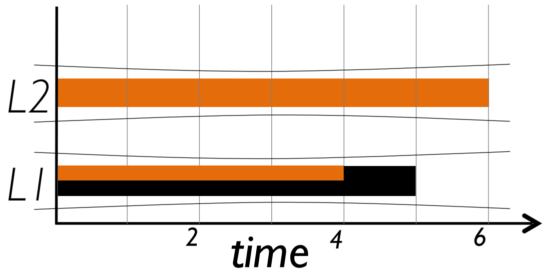
The Optimal



[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

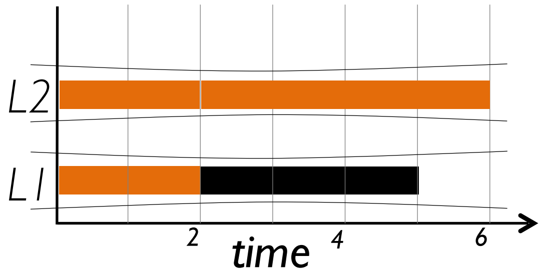
Benefit of inter-coflow scheduling

Fair Sharing



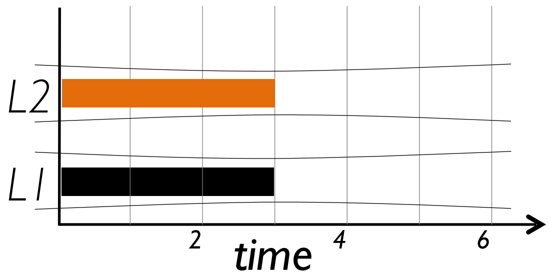
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1],[2]



Coflow1 comp. time = ***5***
Coflow2 comp. time = ***6***

The Optimal

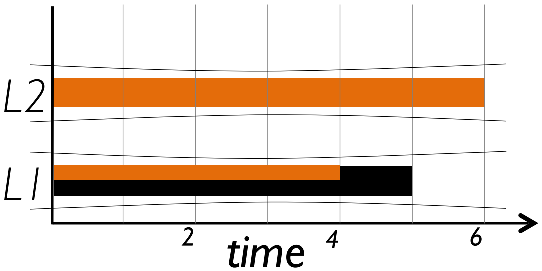


Coflow1 comp. time = ***3***

[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

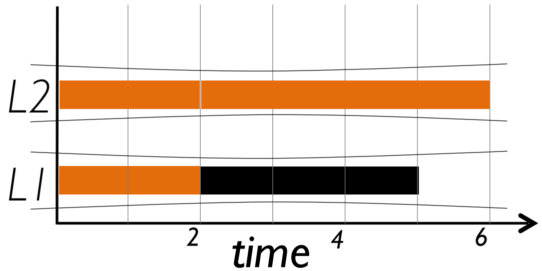
Benefit of inter-coflow scheduling

Fair Sharing



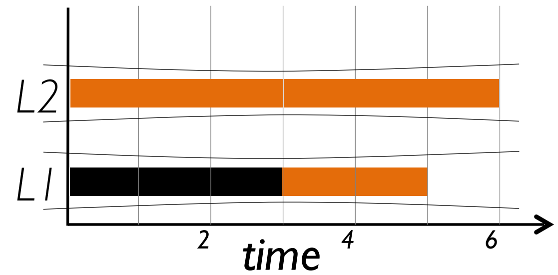
Coflow1 comp. time = 5
Coflow2 comp. time = 6

Smallest-Flow First[1],[2]



Coflow1 comp. time = ***5***
Coflow2 comp. time = ***6***

The Optimal



Coflow1 comp. time = ***3***
Coflow2 comp. time = ***6***

[1] Finishing Flows Quickly with Preemptive Scheduling, SIGCOMM'2012.
[2] pFabric: Minimal Near-Optimal Datacenter Transport, SIGCOMM'2013.

Coflows: Main Known Results

Problem of min. avg CCT - Negative Algorithmic Results:

- **NP-Hardness** (reduction from concurrent open-shop scheduling). [Chowdhury, Zhong, and Stoica. Varys. In ACM SIGCOMM 2014] Thus, best hope for = approximation algorithms.
- Lower Bounds: **Inapproximimibility** within a factor of $2 - \varepsilon$. [Bansal and Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In EATCS ICALP 2010.]
- **Necessity for Coordination:** Without $\Omega(\sqrt{n})$ of the optimal. [Chowdhury and Stoica. Efficient coflow scheduling without prior knowledge. In ACM SIGCOMM 2015]

Coflows: Main Known Results

Algorithmic Fact 2

Problem of min. avg CCT - Positive Algorithmic Results:

Lots of **coflow schedulers** proposed:

- Baraat [Dogar et al. in ACM SIGCOMM 2014]
- Varys [Chowdhury, Zhong, and Stoica. Efficient coflow scheduling with varys. In ACM SIGCOMM 2014]
- Sincronia [Agarwal et al. Sincronia: near-optimal network design for coflows. In ACM SIGCOMM 2018]

- **Best known approximation algorithm:** 4-approximation [Agarwal, Rajakrishnan, Narayan, Agarwal, Shmoys, Vahdat, Sincronia: near-optimal network design for coflows. In ACM SIGCOMM 2018]

Key open challenges

- Better theoretical understanding
- Efficient solutions to deal with
  - decentralization,
  - more complex topologies,
  - estimations over DAG,
  - etc.
- Extensions to
  - non-clairvoyant scheduler,
  - other performance metrics, e.g. tail completion time, fairness.
  - co-designing routing along with scheduling of coflows.

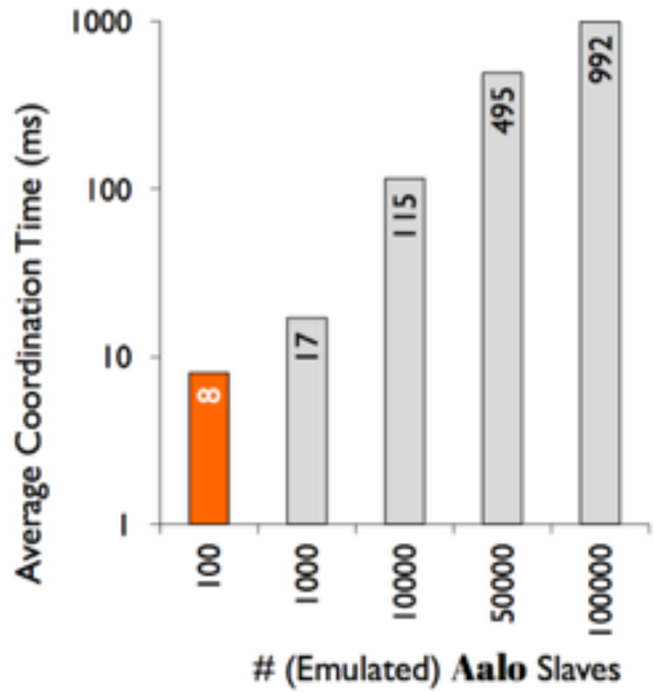
[Coflow Recent Advances and What's Next? M. Chowdhury Dimarcs 2017]

Key open challenges

- Better theoretical understanding
  - Gap between lower and upper bounds: $2 - \varepsilon$ vs 4-approx.
  - Improved competitive ratio for **online** coflow scheduling (best known 12).

Key open challenges

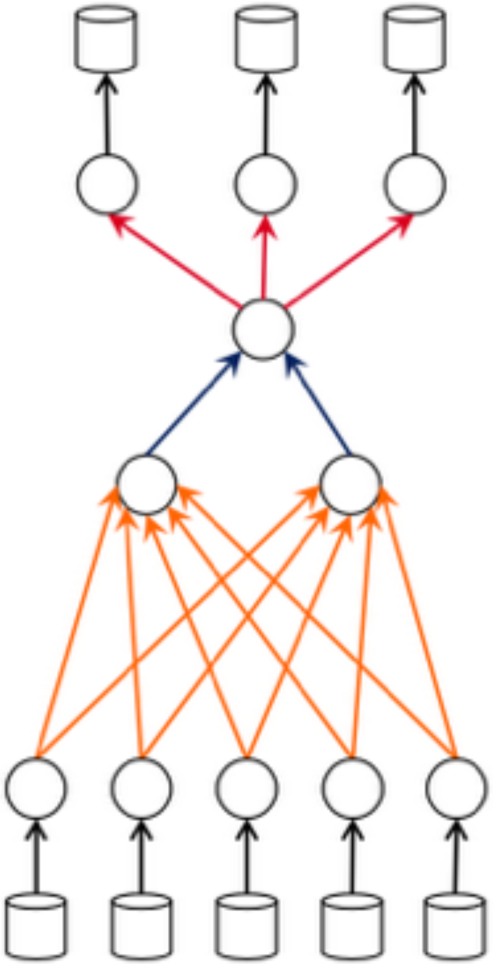
- **Coordination is necessary to determine realtime**
  - Coflow size (sum);
  - Coflow rates (max);
  - Partial order of coflows (ordering);
- **Can be a large source of overhead**
  - Does not impact too much for large coflows in slow networks, but …
- How to perform decentralized coflow scheduling?
  - Some centralization necessary with strong lower bound of $\Omega(\sqrt{n})$
  - But, which “amount of coordination” is unclear
    - e.g. Sincronia does not need per flow rate adaptation



Key open challenges

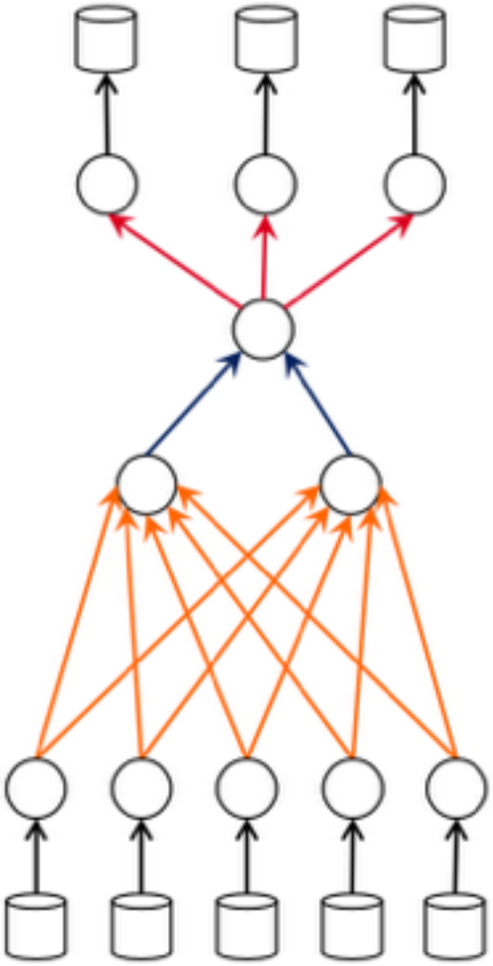
- Schedule a DAG of coflows

- Consider both network and server resources (cores)



Key open challenges

- Schedule a DAG of coflows
- Consider both network and server resources (cores)



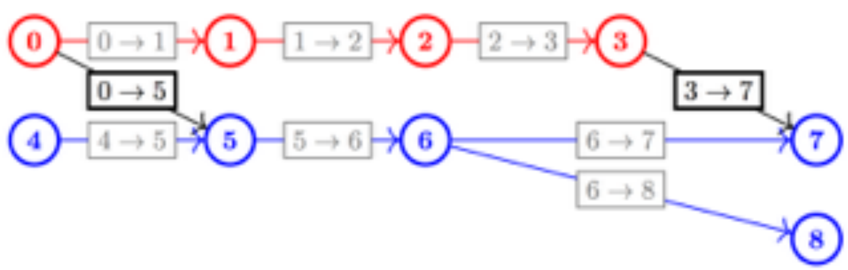
-> Introduction of a new theoretical framework

Two Theoretical Frameworks

1. **Coflows**
   or scheduling group of dependant flows

[Chowdhury,Stoica Hotnets 2012]

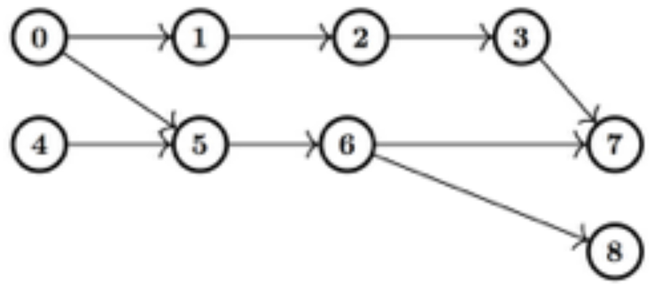
2. **Network tasks**
   or scheduling while optimizing network resources



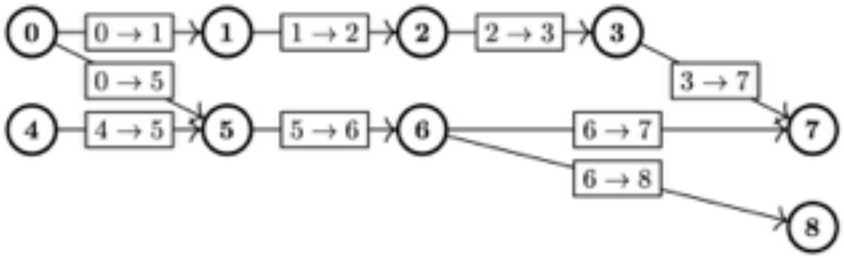
[Giroire,Huin,Tomassilli,Pérennes, INFOCOM 2019]

A new scheduling framework

- **Goal:** schedule workflows while taking into account the limited communication bandwidth

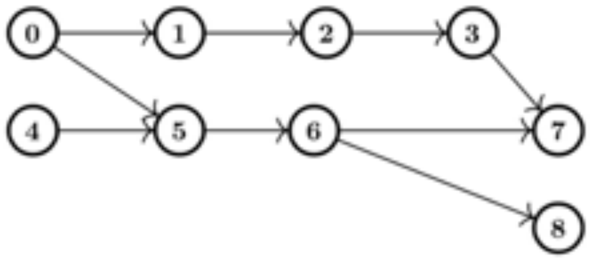


- 2 kinds of tasks:
  - CPU tasks: to be executed by servers
  - Network tasks: to be executed by network machines



- Network tasks <u>may or may not</u> *be executed* depending on the placement of the CPU tasks

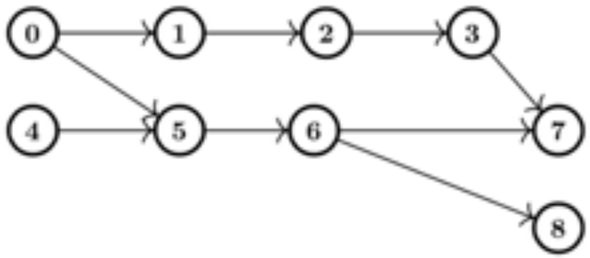
A simple example with 2 Servers and 1 Network Machine)



Dependency Digraph of a Job with 9 CPU Tasks

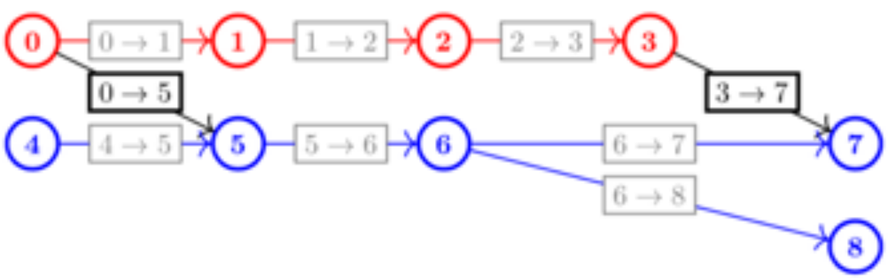
2 Servers (P1 and P2)

A simple example with 2 Servers and 1 Network Machine)

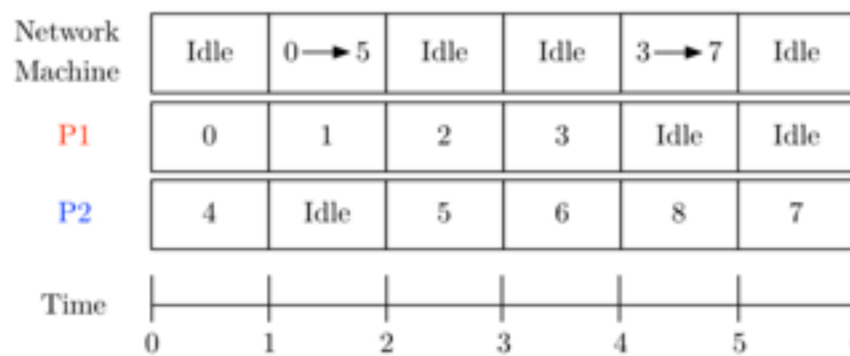


Dependency Digraph of a Job with 9 CPU Tasks

2 Servers (P1 and P2)



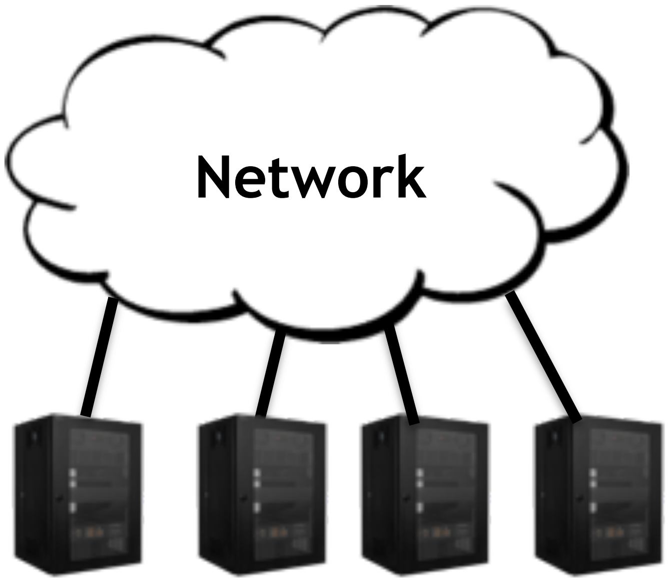
Workflow with network tasks



A possible schedule

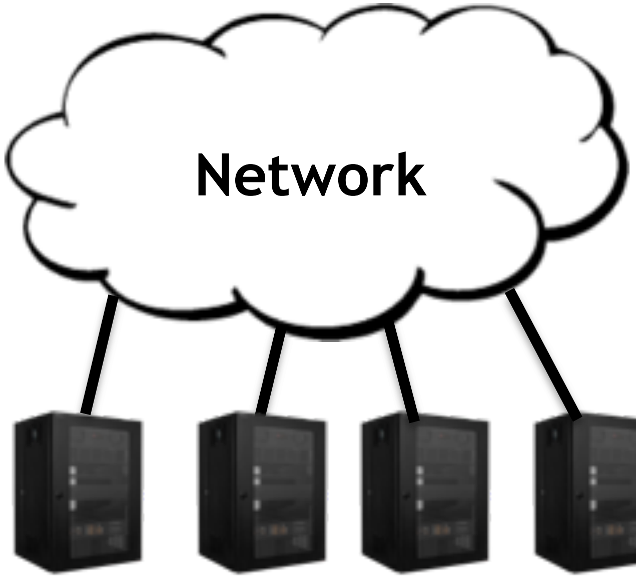
Modeling Data Center Networks

- **Simple Networks** (machines connected via a bus or via an antenna)
  - one network machine per channel

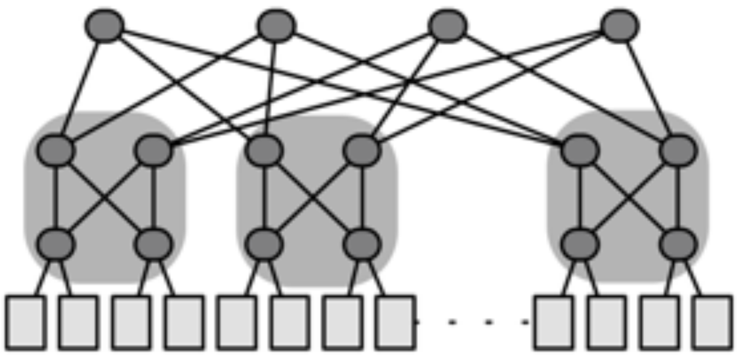


Modeling Data Center Networks

- **Simple Networks** (machines connected via a bus or via an antenna)
  - one network machine per channel

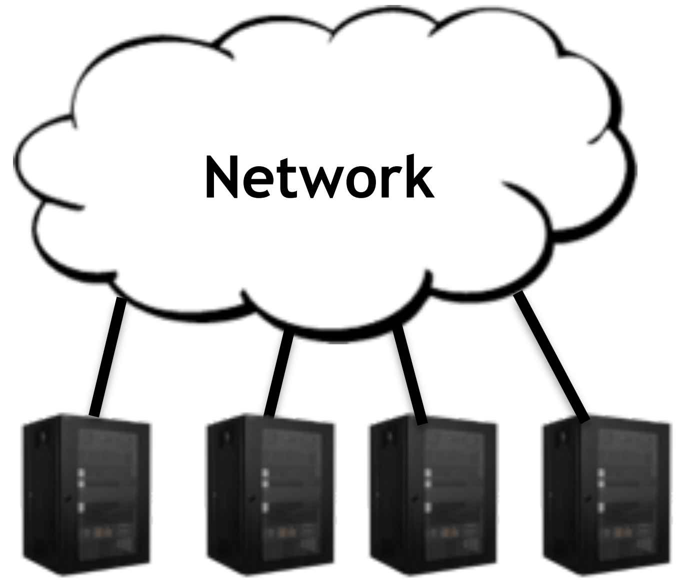


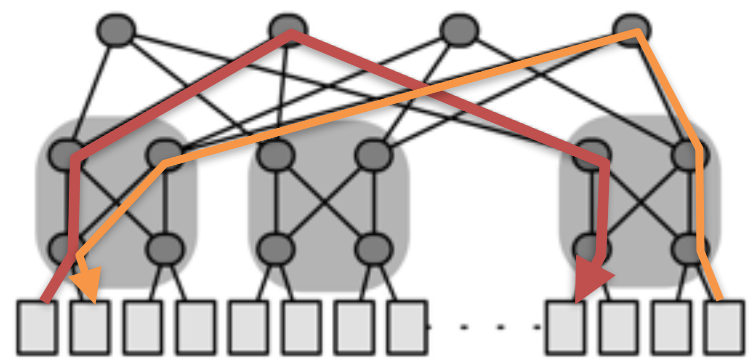
- **Data Center Networks**
  - key property: large bisection bandwidth (full for VL2 and Fat Trees) [Chen et al. JPDC 2016]
  - only border links (i.e., links between the servers and the ToR switches) have to be taken into account



Modeling Data Center Networks

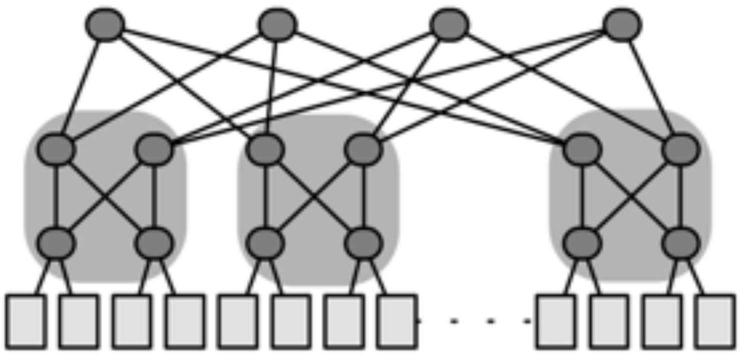
- **Simple Networks** (machines connected via a bus or via an antenna)
  - one network machine per channel
- **Data Center Networks**
  - key property: large bisection bandwidth (full for VL2 and Fat Trees) [Chen et al. JPDC 2016]
  - only border links (i.e., links between the servers and the ToR switches) have to be taken into account





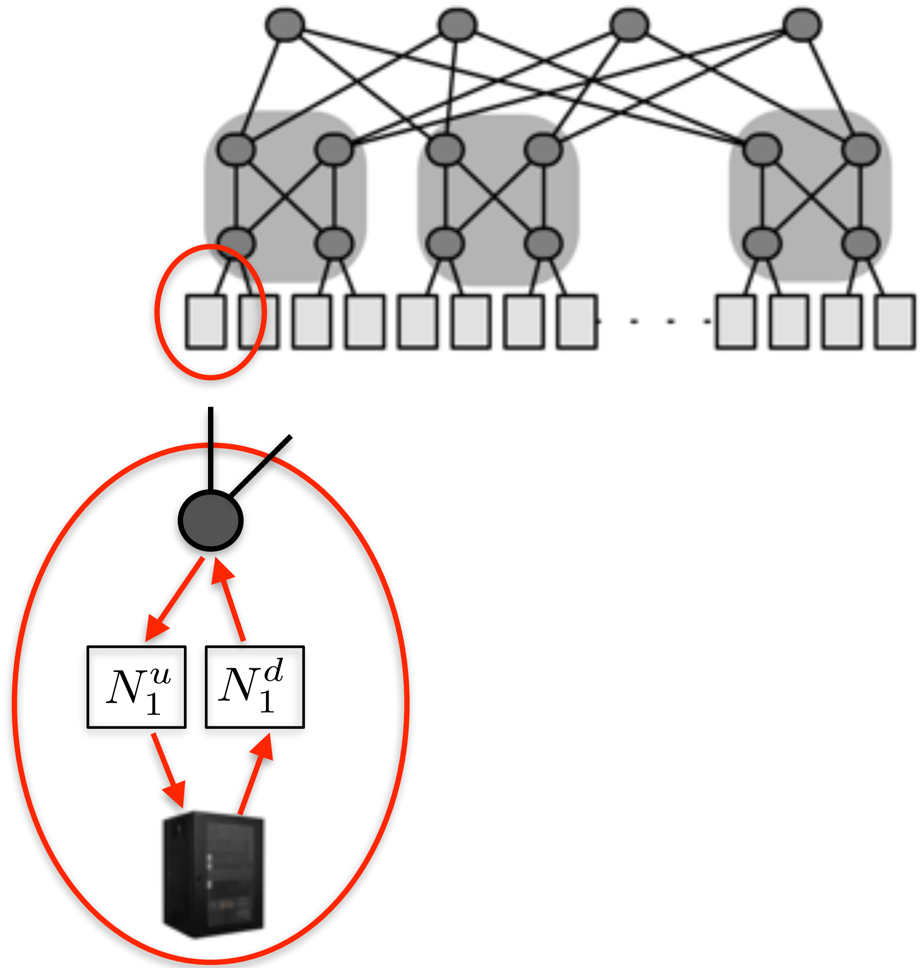
Modeling Data Center Networks

- Only inter-rack bandwidth to be modeled



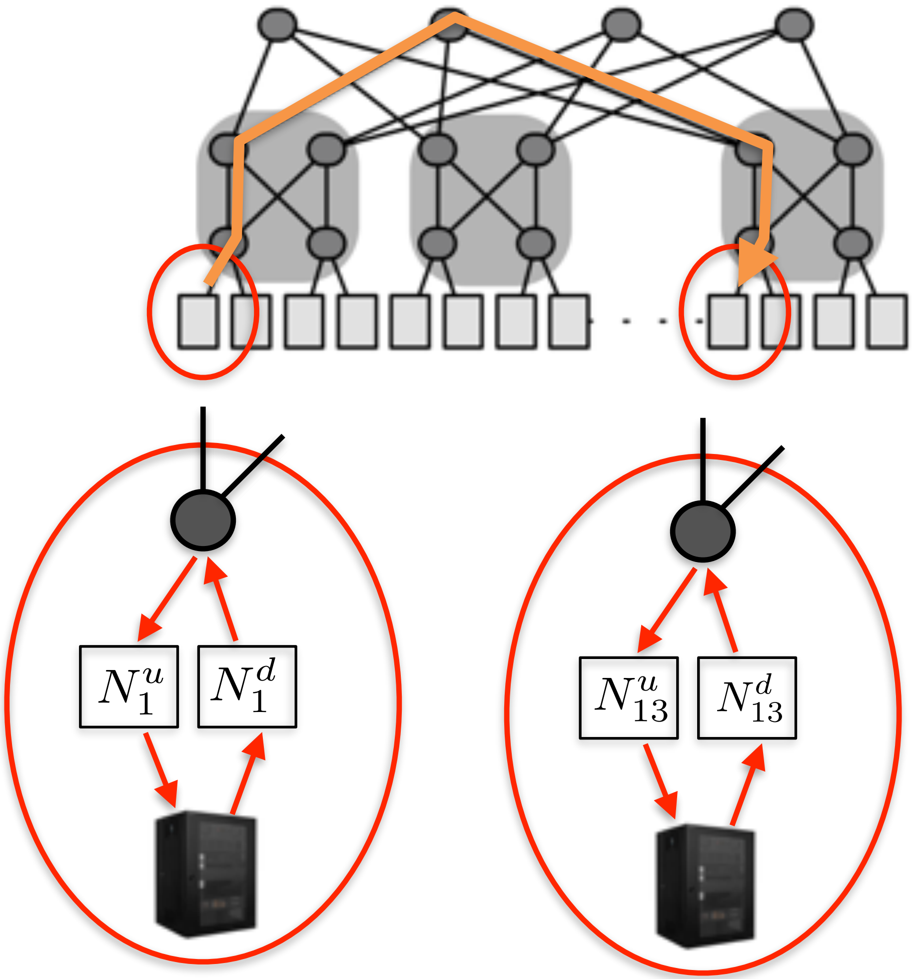
Modeling Data Center Networks

- Only inter-rack bandwidth to be modeled
- 2 network machines per link:
  - one for **upload**
  - one for **download**



Modeling Data Center Networks

- Only inter-rack bandwidth to be modeled
- 2 network machines per link:
  - one for **upload**
  - one for **download**
- Network transfer between $\mathbf{M_1}$ and $\mathbf{M_{13}}$
  - job in download machine of $\mathbf{M_1}$
  - job in upload machine of $\mathbf{M_{13}}$

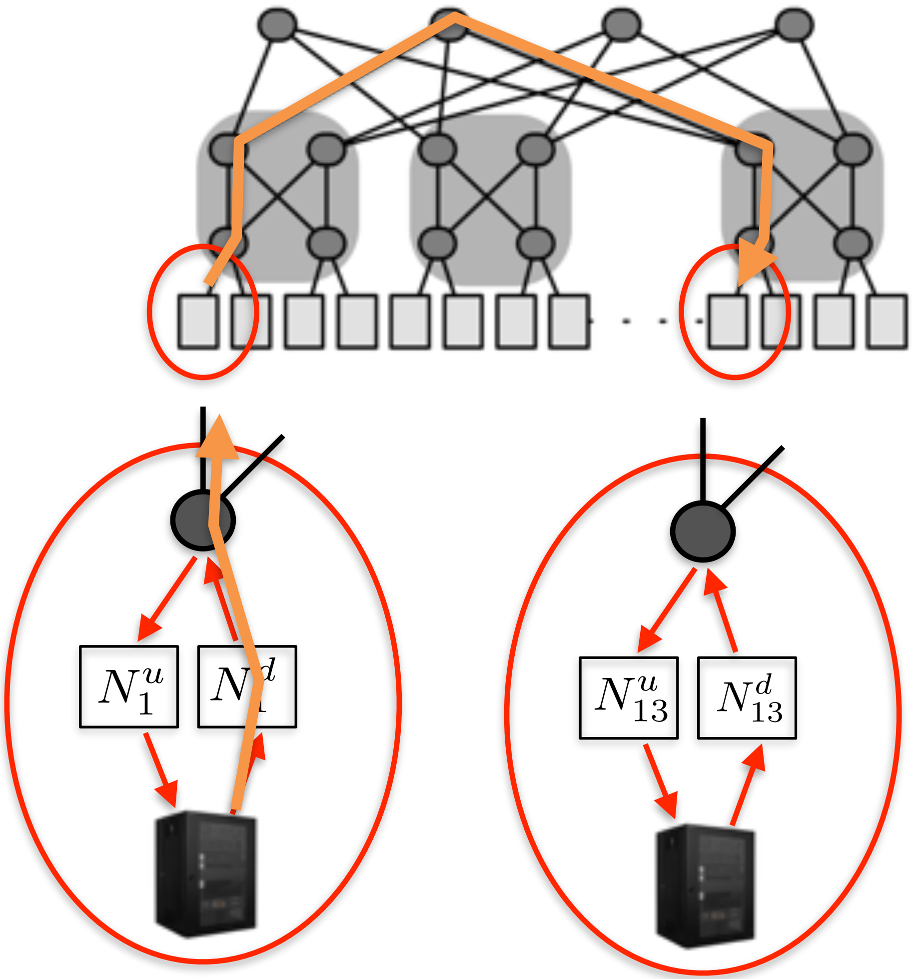


Modeling Data Center Networks

- Only inter-rack bandwidth to be modeled

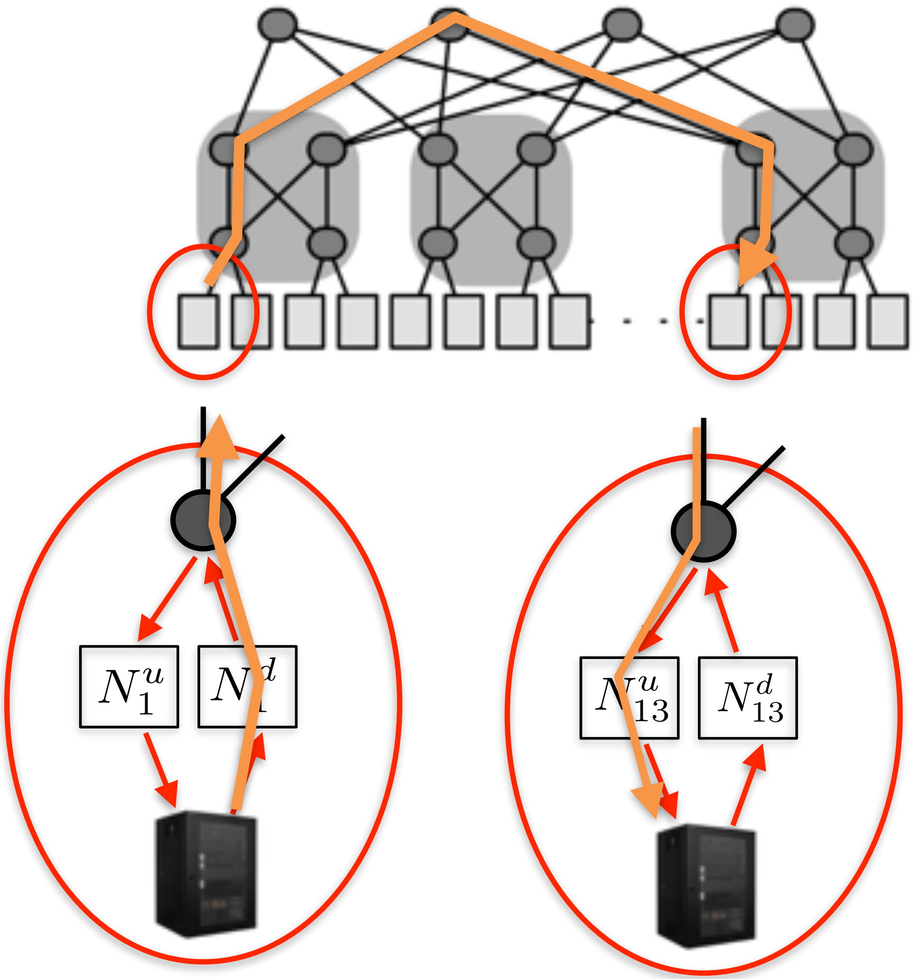
- 2 network machines per link:
  - one for **upload**
  - one for **download**

- Network transfer between **M_1** and **M_{13}**
  - job in download machine of **M_1**
  - job in upload machine of **M_{13}**



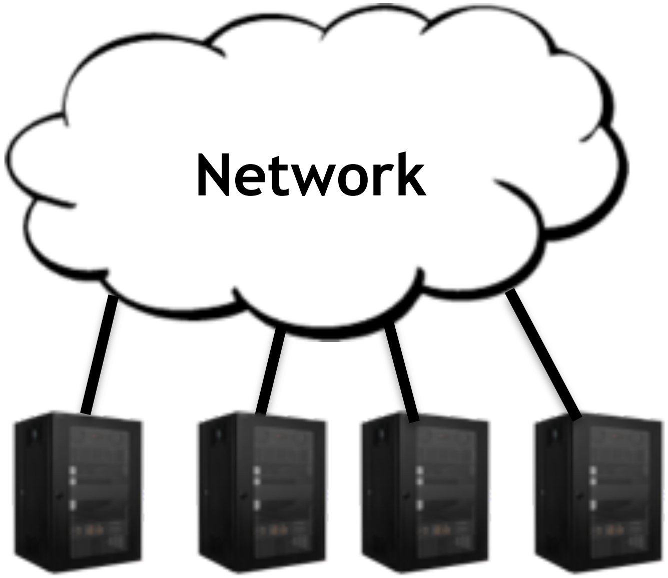
Modeling Data Center Networks

- Only inter-rack bandwidth to be modeled
- 2 network machines per link:
  - one for **upload**
  - one for **download**
- Network transfer between $\mathbf{M_1}$ and $\mathbf{M_{13}}$
  - job in download machine of $\mathbf{M_1}$
  - job in upload machine of $\mathbf{M_{13}}$

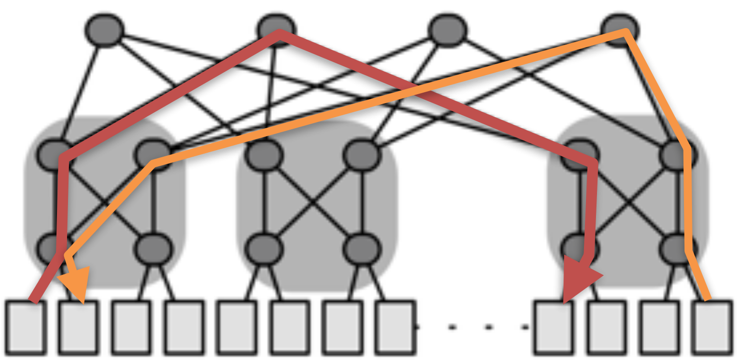


Modeling Data Center Networks

- **Simple Networks** (machines connected via a bus or via an antenna)
- **Data Center Networks**
  - modeling border links



- **More general networks** (without full bissection bandwidth) leads to $\frac{C}{O(m \log m)}$ -approximation with C minimum network multicut [Garg et al STOC 1993]



Contributions

1. Introduction of new scheduling framework to model *communication delays when tasks are competing for a limited network bandwidth.*

2. Show how to schedule data center jobs while routing their communications

3. Hardness results of SCHEDULING WITH NETWORK TASKS problem

4. Two efficient scheduling algorithms, G-LIST and PARTITION

5. Extensive evaluation using workflows based on Google trace [Reiss et al. White paper]

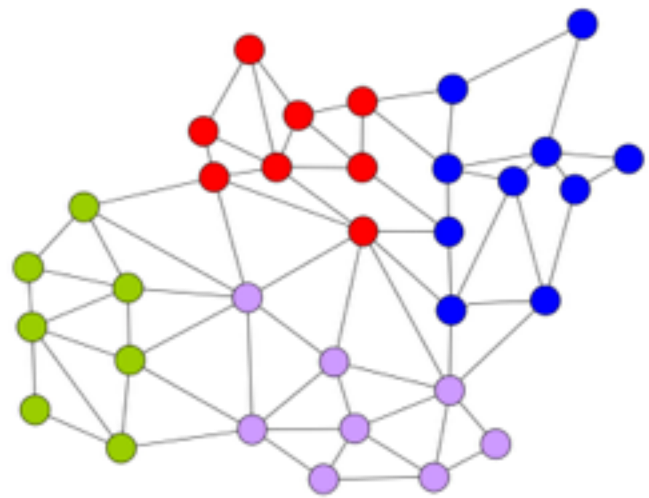
Two Efficient Algorithms

- G-LIST: **greedy** algorithm
  - Generalization of the List Scheduling algorithm
  - Idea: place a task where there is most needed data **and** only if needed network tasks can all be done
  - **Proposition 1.** G-LIST *is optimal on simple MapReduce workflows.*

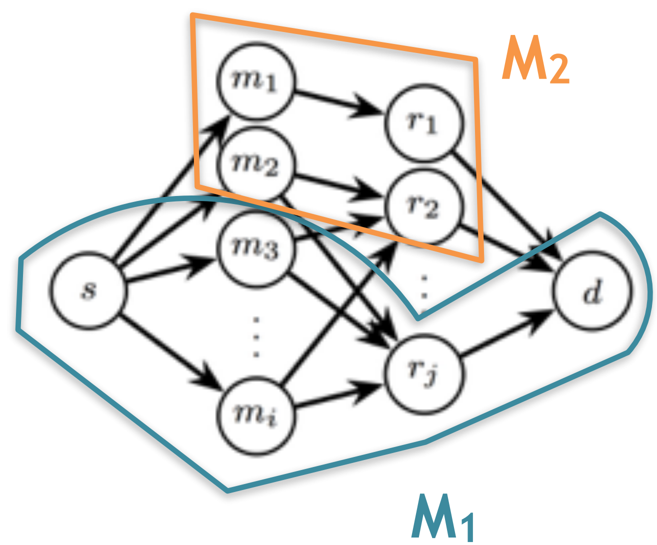
- PARTITION: a **2-phase** algorithm
  1. assign the tasks to machines while minimizing the CPU and the networking work
  2. compute a schedule for the tasks

PARTITION: 2-phase approach

- Phase 1: Distribute tasks into machines minimizing communications

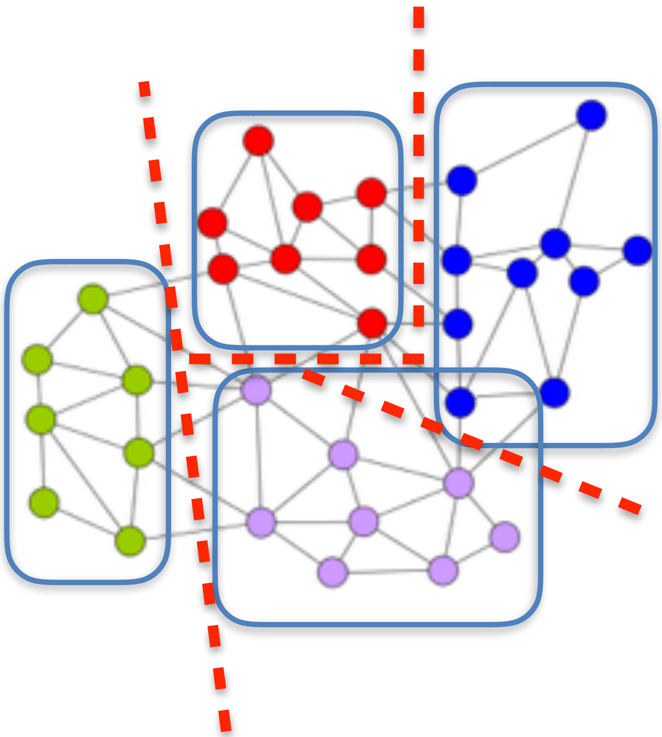


- Phase 2: Schedule the tasks when placed minimizing makespan



PARTITION: Phase 1

- Based on the **k-balanced graph partitioning** problem:



Goal: Partition vertices of input graph G into k *equally sized components*, while *minimizing* the total weight of the *border edges*

Known results: $O(\sqrt{\log n \log k})$-approximation algorithm [Krauthgamer SODA 2009]

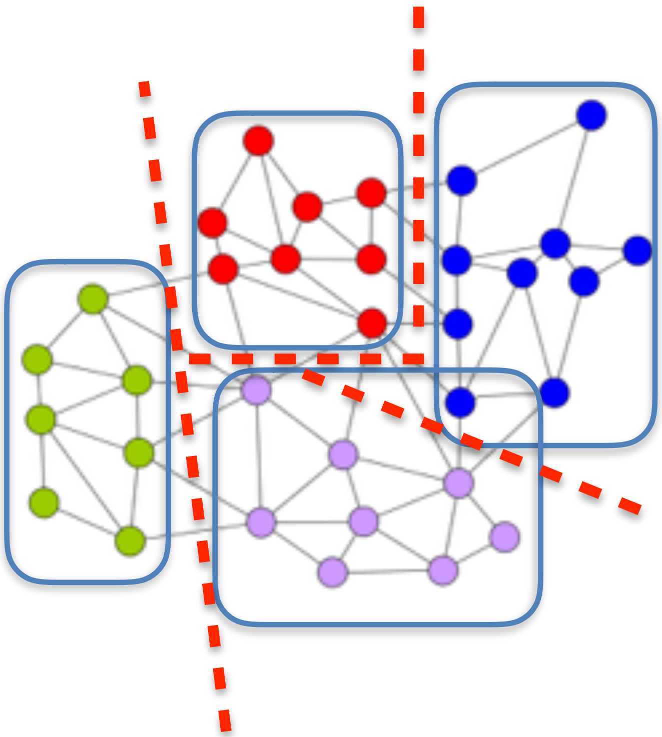
Beware! Best solution is not necessarily with the largest number of machines

PARTITION: Phase 1

Beware! Best solution is not necessarily with the largest number of machines

Principle of **PARTITION-ASSIGN Algo:**

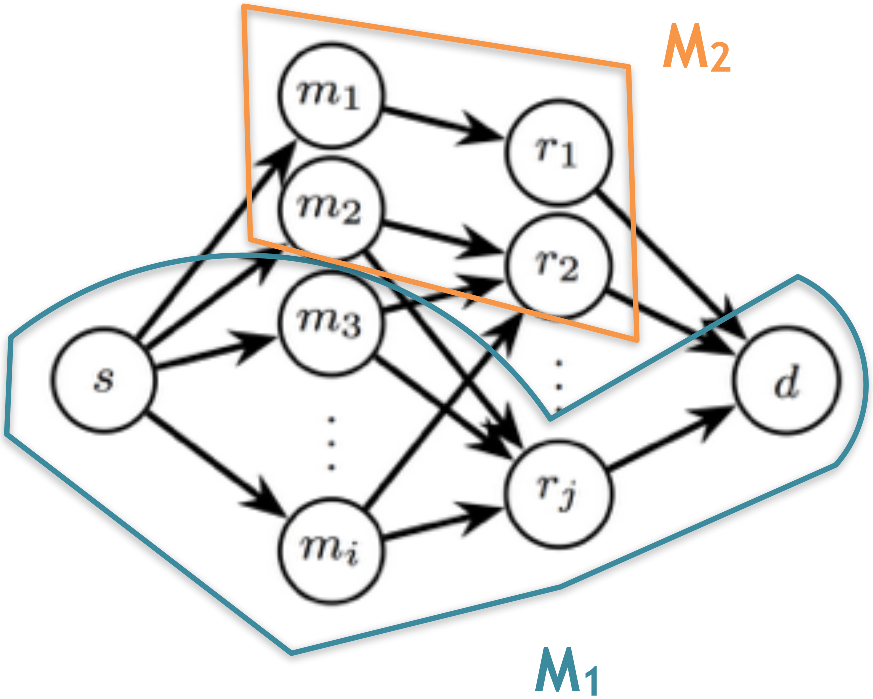
1. Choose a number of machines, k.
2. Solve a **k-balanced partitioning** problem
3. Do it for all possible $1 \leq k \leq m$



Theorem 2. PARTITION-ASSIGN *provides a* $O(\sqrt{\log n \log m})$ *-approximation algorithm of the* PARTITIONING TO SCHEDULE *problem.*

PARTITION: Phase 2

- SchedulingWhen Placed problem.
- Results:
  - Hardness: NP-complete and inapproximability 5/4 (reduction from 3SAT)
  - Approximation algorithm, PARTITION-SCHEDULE.



Network tasks: Conclusion

- Proposition of a new framework to model the orchestration of tasks in a datacenter for scenarios in which the network bandwidth is a limiting resource.

- Two algorithms to solve the problem, for which we derive some theoretical guarantees.

- Demonstration of the effectiveness of our algorithms using datasets built using statistics from Google data center traces.

Network tasks: Conclusion

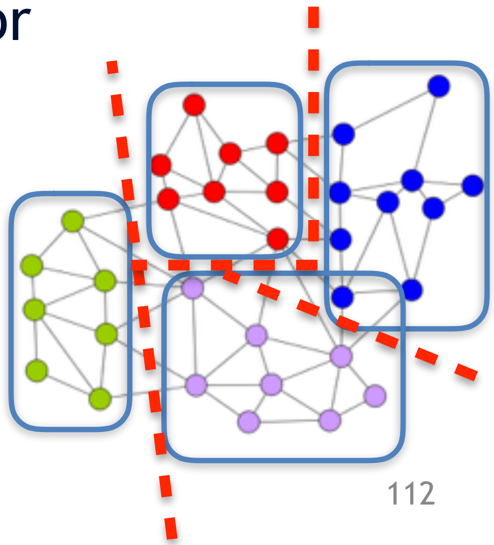
A lot of open questions:

- Main one: inapproximability of the general problem?

Reminder: Without network, scheduling with a dependency digraph not approximable within a factor 4/3 [Lenstra Rinnooy Kan 78] and 2-approximation.

Goal: With network, approximation algorithm or inapproximability (with a constant>4/3 or log factor)

-> Study of variants of k-balanced partition.



Conclusion

A lot of open questions:

- On the practical side: study of behaviors of the algorithms on a testbed, comparing them with practical solutions proposed for data centers.

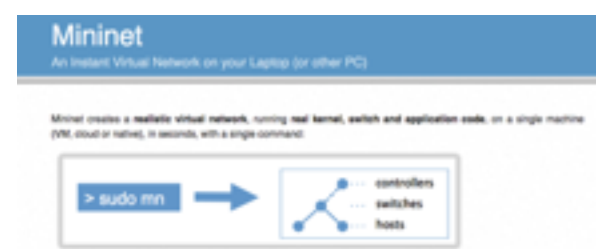
Outline

1. Motivation
2. A new situation: SDN and NFV
3. Placement of virtual network functions
   - Use case: Service Function Chaining
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. What next?

Validating solutions

- Theoretical results (explain main parameter dependencies, but often too simplistic hypothesis)
- Simulations (represent more complex phenomena, but bad fidelity to real networks, implementation different from actual application)
- Emulations (fast and good scalability, can run actual application, can interact width a live environment)
- Experimentations (Wide-area implementation not always possible, too few nodes may be available, not reproducible)

- **Most used tool for SDN/NFV networks: Mininet.**

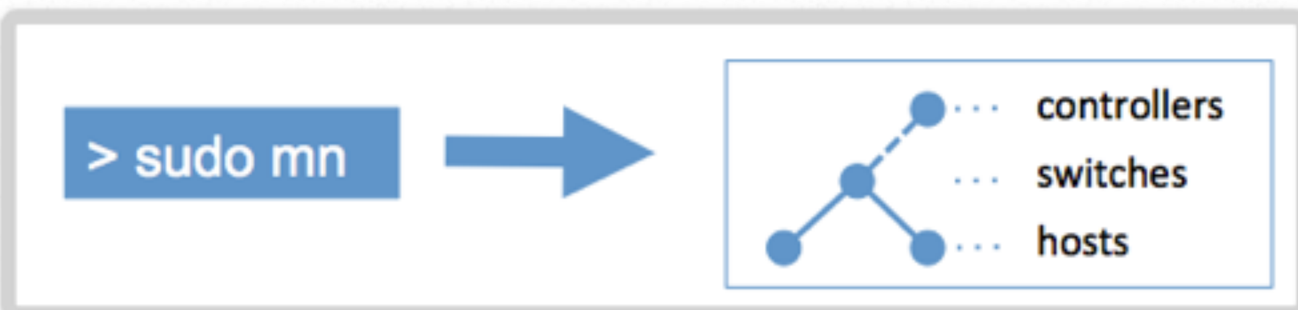


Mininet

Mininet

An Instant Virtual Network on your Laptop (or other PC)

Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command:



Because you can easily interact with your network using the Mininet CLI (and API), customize it, share it with others, or deploy it on real hardware, Mininet is useful for development, teaching, and research.

Mininet Limitations

- **Mininet** provides a flexible and cost-efficient experimental platform to evaluate SDN applications.

- But it has several **limitations**:
  - resources limits (CPU, bandwidth) if experiments are run on a single host.
  - no strong notion of virtual time (timing measurements based on system clock)

- When the physical host is overloaded, Mininet
  - may return wrong results or
  - not be able to run the experiments

Mininet Limitations

- Mininet provides a flexible and cost-efficient experimental platform to evaluate SDN applications.

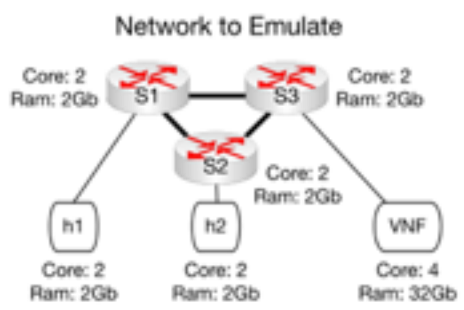
- But it has several **limitations**:
  - resources limits (CPU, bandwidth) if experiments are run on a single host.
  - no strong notion of virtual time (timing measurements based on system clock)

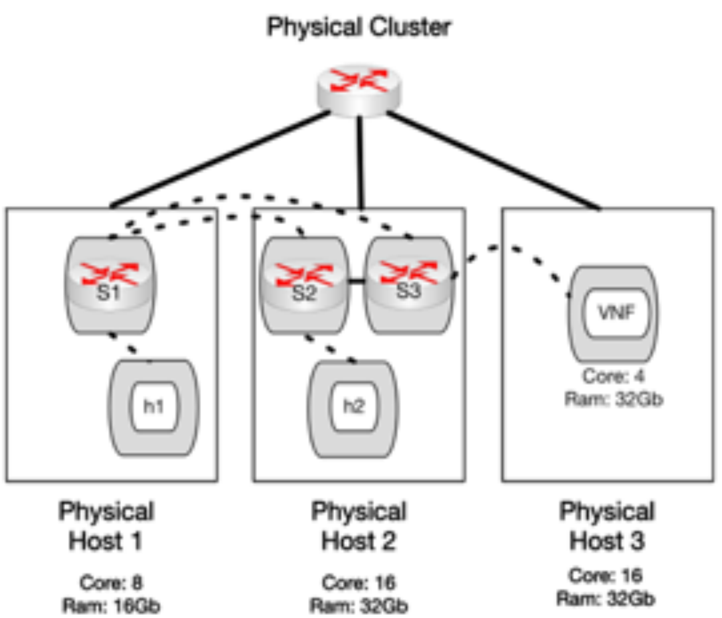
- When the physical host is overloaded, Mininet
  - may return wrong results or
  - not be able to run the experiments

Need to **overcome** Mininet Limitations and **increase** the performance fidelity of network experiments

Distributed Network Emulation

Solution: distribute the load for resource intensive experiments.





Existing tools:

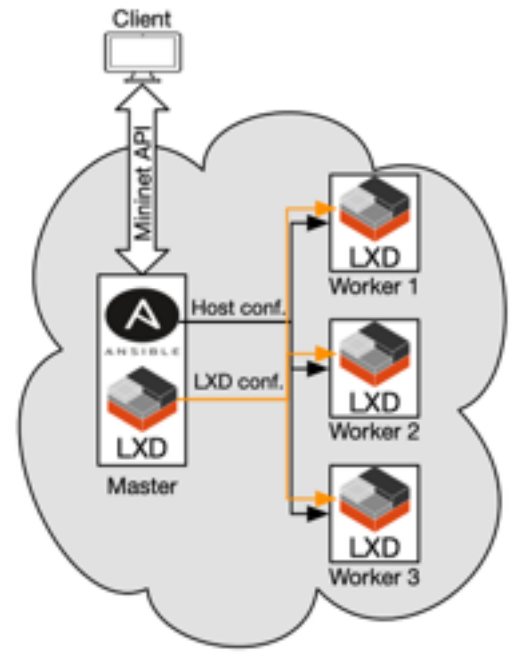
- Mininet Cluster Edition: [1]
- Maxinet: [Wette et al. IFIP Networking 2014]

[1] https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype

A new tool: Distrinet

- Limitations of existing tools:
  - No performance guarantees
  - New API

- Distrinet Work in progress [2]
  - + Fully compatible with Mininet API.
  - + Automatic deployment in private infrastructures (linux machines and Grid5000) or public cloud (AWS).
  - + Some guarantees that resources requirements (e.g. cores, memory, network) are satisfied.
  - + Minimization of resource utilization for private infrastructures and costs for public cloud.



Linux Servers / AWS Cloud / Grid 5000

[2] **Di Lena**, Tomassilli, **Saucez**, **Giroire**, Turletti and Lac. Mininet on steroids: exploiting the cloud for Mininet performance IEEE CloudNet 2019

Outline

1. Motivation
2. A new situation: SDN and NFV
3. Placement of virtual network functions
   - Use case: Service Function Chaining
4. Coflows for datacenters
5. Scheduling with network tasks
6. Tools to evaluate solutions
7. **What next?**

Challenge 1

- Lots of open algorithmic problems
  - For SFCs
  - For coflows
  - For variants of scheduling

Challenge 2

- Scheduling beyond the cloud
  - Fog Computing and Mobile Edge Computing

Fog/Edge Computing

- **PROBLEM**: Interactive applications require ultra-low network latencies (< 20 ms) … but latencies to the closest data center are 20-30 ms using wired networks, up to 50-150 ms on 4G mobile networks
- **SOLUTION**: **Exploit distributed and near-edge computation**:
  - Reduce latency and network traffic
  - improve power consumption
  - increase scalability and availability

FOG COMPUTING

Analyze most IoT data near the devices that produce and act on that data

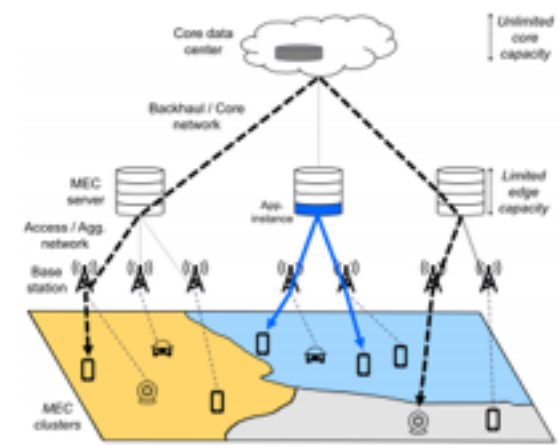
Fog/Edge Computing - Challenges

How to assign the IoT applications to computing nodes (fog nodes) which are distributed in a Fog environment?

- Computing and networking resources are:
  - heterogeneous
  - not always available
- Service cannot be processed everywhere
- Demands and resources are dynamic

Mobile Edge Computing

- **IDEA**: Offloading to improve latency and alleviate congestion in the core -> Push the content (application servers) close to the users using **MEC servers** (small datacenter collocated with the base station) in the infrastructure close to the edge of the network
- **PROBLEM:** assign users, application, and share of traffic to the MEC servers
- **Constraints:**
  - mobile traffic depends on time and locality
  - geographical constraints
  - mobility of the users
  - budget



Challenge 3

- Getting realistic scenarios with
  - data (application and networks)
  - architecture

Challenge 3

- Getting realistic scenarios with
  - data (application and networks)
  - architecture

THANKS FOR YOUR ATTENTION!