Joint
SDN & NFV
Optimization

SDNRG @ IETF-93

Presented by:

- **Yaakov (J) Stein**

Tel Aviv University team

- **Boaz Patt-Shamir**
- **Guy Even**

Research funded under the Neptune consortium

Recap: rich communications services

Traditional communications services are pure *connectivity* services

transport data from A to B

with constraints (e.g., minimum bandwidth, maximal delay)

with maximal efficiency (minimum cost, maximized revenue)

Modern communications services are richer

combining connectivity and network functionalities

e.g., firewall, NAT, load balancing, CDN, parental control, ...

We deal with a service provider that

- maintains a network of communications and computational resources
- maintains an inventory of VNFs
- dynamically sets up and tears down services
- charges based on
  - service requirements
  - time between set-up and tear-down

The service provider employs an orchestrator to maximize profits

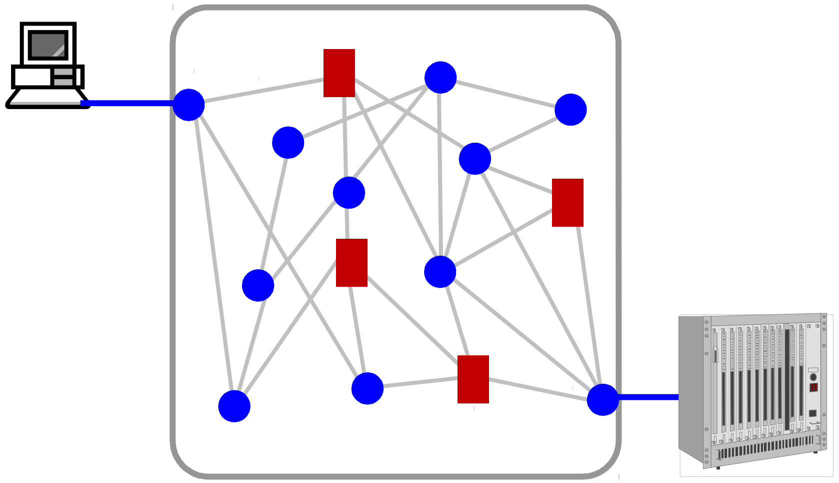
(profit is the difference between revenue and expenses)

Recap SDN/NFV Optimization Problems

Given a network with distributed computational resources for VNFs
we distinguish three SDN/NFV optimization problems

- pure SDN path computation (no VNFs needed)
- pure NFV placement (routing performed separately)
- joint SDN/NFV path/placement

The first problem has many classic solutions (see the PCE literature)
The second problem (NFV placement) has been studied recently
The third problem is as yet unsolved



On-line optimization

A batch (off-line) algorithm receives the list of all services to be set up and simultaneously finds all the allocations for a *clean* network

We require an on-line algorithm that services requests of unknown duration as they come in

We do not allow pre-emption or re-optimization of service already set-up

The on-line case is harder since we don't know ahead of time whether it is worthwhile to use up resources for a given request and risk having to deny some later request that may be more profitable

Simple example

- first request requires 100% of a resource and pays **x**
- later requests require some of that resource and together pay **y**>>**x**

How do we know whether to accept or deny the first request ?

- if we accept, we lose **y-x** if later requests *do* arrive
- if we deny and later requests never arrive, we lose **x**

Formal definition of the joint problem

Known

- full network topology graph
  - link and node current resource loading information
- places where computational resources are available
  - resource availability
- other SDN or NFV criteria and constraints

Service request definition

- traffic ingress and egress points
- service data-rate and delay requirements
- sequence of VNF(s) to be installed

Note: we do not yet support partial ordering of VNFs other than by exhaustively testing every possible order

  - computational (including memory, storage, etc.) requirements
- service set-up or tear-down ?

Find

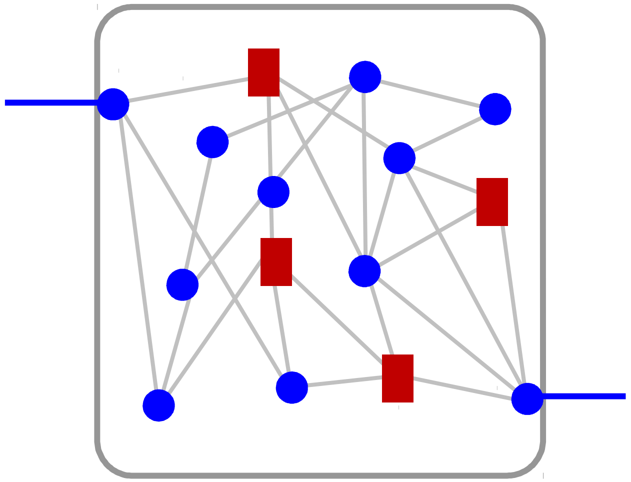
- the optimal path and VNF placement(s)

The solution

Our solution combines two ideas:

1. use of Cartesian Graph Product
2. use of an ACCEPT/STANDBY mechanism

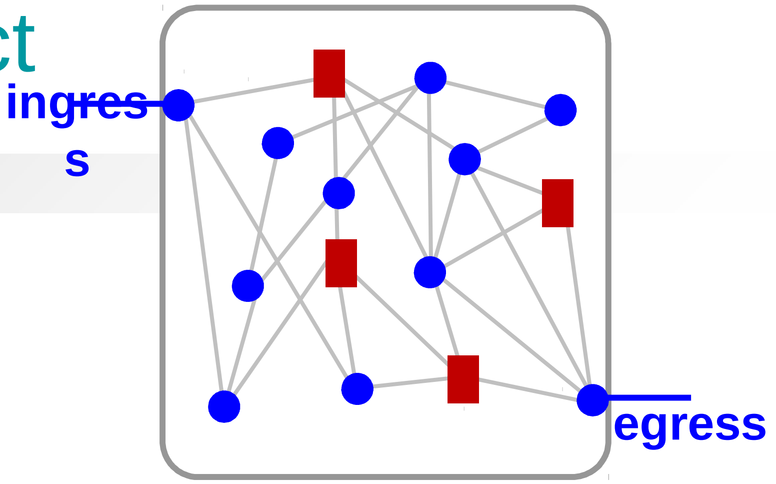
The first idea is a method of transforming the joint problem into a conventional path computation problem on a single network graph



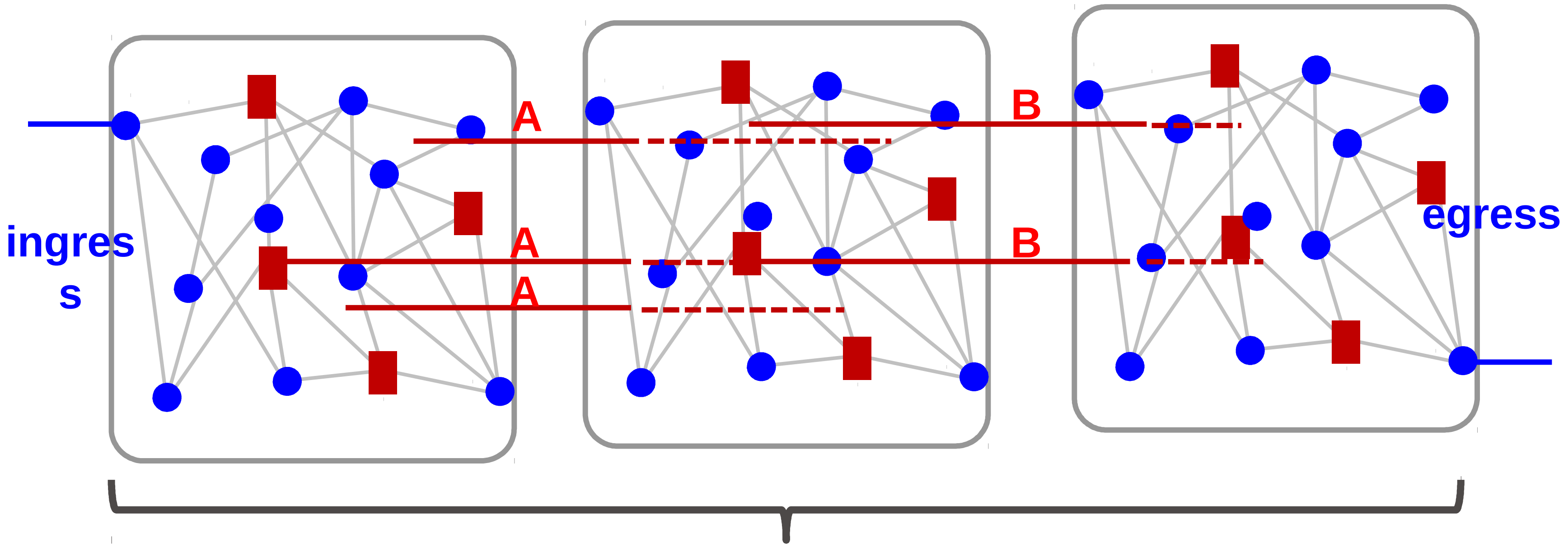
● **switch (network resource)**

■ **server (compute resource)**

Step 1 : Cartesian Product



Assume 2 VNFs : **A** and **B**



PRODUCT GRAPH

The product graph is much larger than the original graph, but still manageable

Performance of competitive algorithms

- The standard on-line mechanism receives service requests, and returns
  - ACCEPT + service routing and placement
  - DENY

Given an on-line optimization problem
we can quantify the (*worst case*) performance of an algorithm ALG as follows

For each input I define

- $\mathrm{OPT}(I)$: profit of best possible solution
  one that knows the future, can pre-empt/reroute/load-balance, etc.
- $\mathrm{ALG}(I)$: profit obtained by the algorithm

The **algorithm's competitive ratio** is defined to be $\mathbf{C} = \underset{\text{all inputs } I}{\mathbf{max}} \left\{ \frac{\mathrm{OPT}(I)}{\mathrm{ALG}(I)} \right\}$

This means that the algorithm's profit is at least 1/C times the optimal profit
Good competitive algorithms have small C ! (Beware of alternative definitions!)

The *AAP algorithm* has competitive factor $O(\log n)$ n= number of network nodes
assuming

- small demands – no service request consumes the majority of any resource
- requests are of known finite durations

assuming

Step 2: ACCEPT/STANDBY response

- In our problem, services may potentially indefinite duration
  leading to potentially dismal worst case performance

Simple example

- reject service request that would have indefinitely paid x per unit time
- no further service requests are ever received

To avoid this problem, we never reject a request, instead we return

- ACCEPT + service routing and placement
- STANDBY – service request placed on hold until can be serviced

note that the service request may be rescinded before it is ever serviced!

We still assume that no request consumes a sizeable amount of any resource

The mechanism has a competitive ratio of $O(k \log n)$
  where k is the maximum number of VNFs in a service request
New ideas may improve this to $O(\log(nk))$

Summary

By combining the two ideas

1. use of Cartesian Graph Product
2. use of an ACCEPT/STANDBY mechanism

we obtain a tractable on-line joint SDN/NFV optimization algorithm