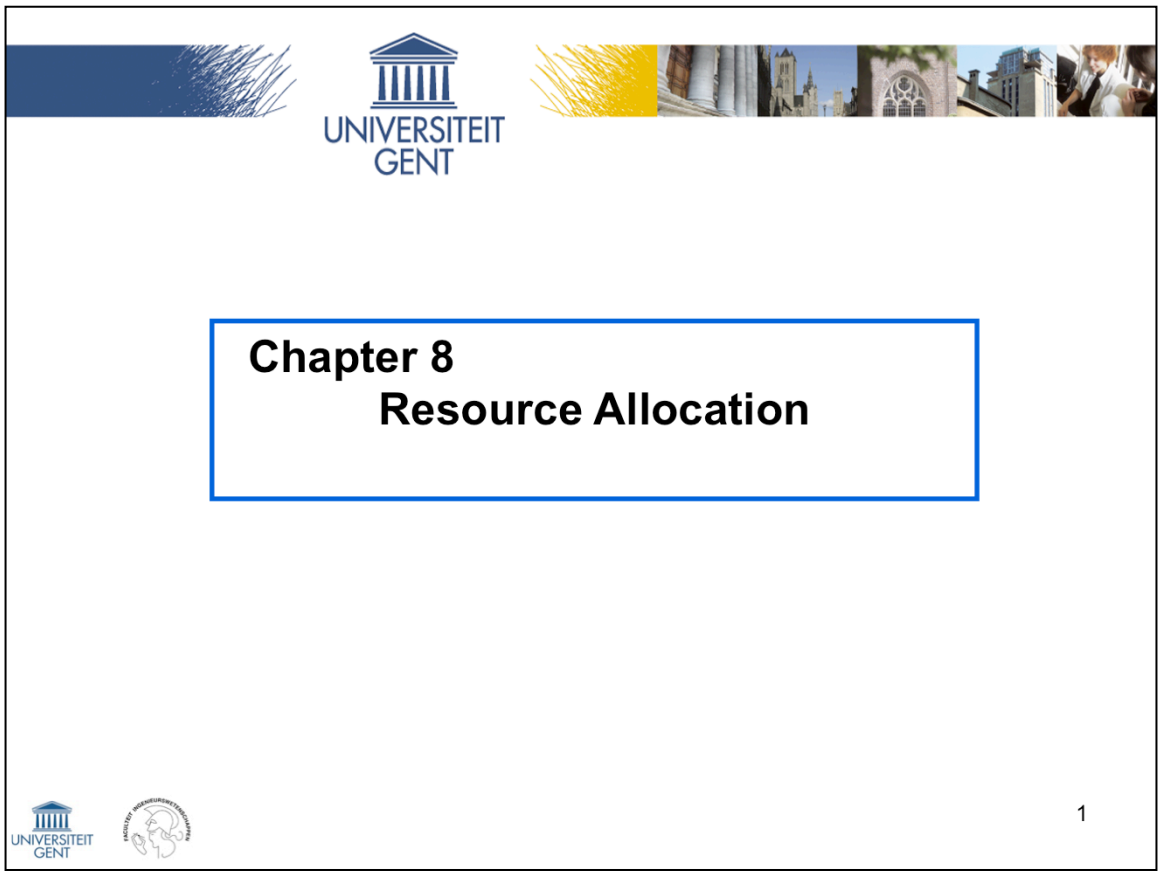

UNIVERSITEIT
GENT
Chapter 8
Resource Allocation
UNIVERSITEIT
GENT
1

Outline

1. Definition
2. Resource allocation algorithms
3. Autonomic resource allocation
4. Control theory
5. Performance models
6. Virtualized environments

Definition

Allocation of resources:

- Crucial in distributed systems
- To make sure applications run properly
- Feedback to applications required

Different type of resources:

- CPU, memory, bandwidth and storage

Objectives:

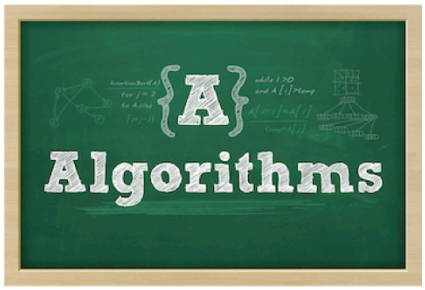
- Minimize response times of applications
- Maximize the quality of experience
- Minimize costs
- Maximize revenues

UNIVERSITEIT GENT

Resource allocation algorithms

Different types:

- Single request versus batch
- Optimal versus heuristical
- Online versus off-line
- Static versus dynamic



Models required:

- Resource models
- Request models

next:

- Integer Linear programming
- Metaheuristics
- Greedy algorithms

Integer Linear programming

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \\ \text{and} \quad & \mathbf{x} \in \mathbb{Z}, \end{aligned}$$

Optimal solution

Scalability ?!

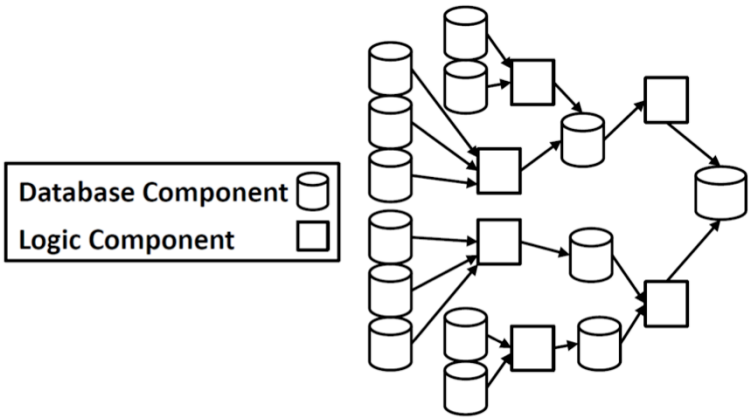
Example 1:

$$\begin{array}{lll} \min & \sum_{i\in L}\sum_{j\in F} c_{ij}x_{ij} + \sum_{j\in F} f_j y_j & \\ \text{s.t.} & \sum_{j\in F} x_{ij} = 1 & \forall i \in L \quad \text{(assign\_def)} \\ & x_{ij} \leq y_j & \forall i \in L, j \in F \quad \text{(link)} \\ & \sum_{i\in L} d_i x_{ij} \leq C y_j & \forall j \in F \quad \text{(capacity)} \\ & x_{ij} \in \{0,1\} & \forall i \in L, j \in F \\ & y_j \in \{0,1\} & \forall j \in F \end{array}$$

Example 2:

$$\begin{array}{ll} \min \sum_{i=0}^{n}\sum_{j\neq i, j=0}^{n} c_{ij}x_{ij} & \\ 0 \leq x_{ij} \leq 1 & i,j = 0,\cdots,n \\ u_i \in \mathbf{Z} & i = 0,\cdots,n \\ \sum_{i=0, i\neq j}^{n} x_{ij} = 1 & j = 0,\cdots,n \\ \sum_{j=0, j\neq i}^{n} x_{ij} = 1 & i = 0,\cdots,n \\ u_i - u_j + n x_{ij} \leq n-1 & 1 \leq i \neq j \leq n \end{array}$$

Request model

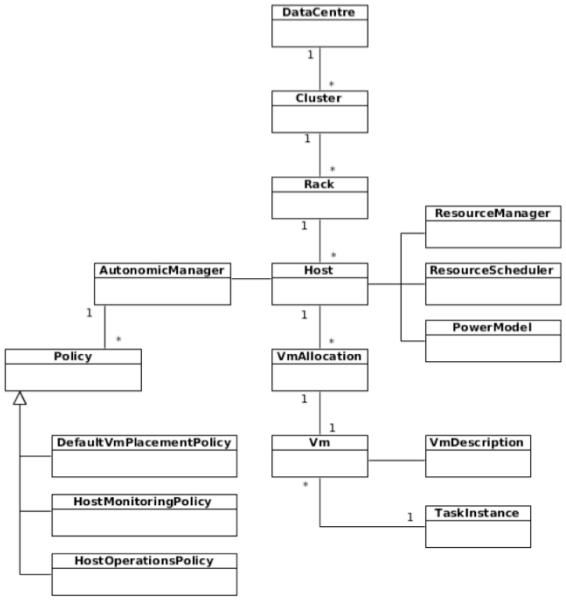


Input parameters

TABLE I: The model input variables.

Symbol	Description
G	The graph $G = (N, E)$ representing the network.
N	The nodes within the network. This collection can be partitioned into a set of computational nodes N^c and a set of service nodes N^s.
E	The edges within the network.
E_n^{in}	The edges that are incoming in node $n \in N$.
E_n^{out}	The edges that are outgoing from node $n \in N$.
$C(e)$	The bandwidth capacity of an edge $e \in E$.
$L(e)$	The network latency of an edge $e \in E$.
$\mathcal{C}$	The service chains that must be allocated on the network.
$R^{VM}(C)$	The VM requests that are part of service chain $C \in \mathcal{C}$.
$R^s(C)$	The service requests that are part of service chain $C \in \mathcal{C}$.
$R(C)$	The resource requests that are part of service chain $C \in \mathcal{C}$. $R(C) = R^{VM}(C) \cup R^s(C)$.
S	The collection of all services that exist within the model.
$D^\gamma(r)$	The resources of type γ needed for a request r. This request can either be a service request or a VM request.
Γ^c	The resource types that are available on a physical computational node. Typically these resources are CPU and Memory.
Γ^s	The resource types that are provided by a service $s \in S$. Typically this is a service-specific resource such as requests per second.
$\mathcal{R}$	A collection containing all of the resource requests of all service chains in $\mathcal{C}$.
C_n^γ	The resource capacity of a node or service n. For computational nodes, $\gamma \in \Gamma^{VM}$. For service nodes offering service s, $\gamma \in \Gamma^s$. A service VM of a service s provides C_s^γ resources of types $\gamma \in \Gamma^s$.
$D(r_1, r_2)$	The network demand between requested VMs or services r_1 and r_2.
$L(r_1, r_2)$	The maximum network latency between requested VMs or services r_1 and r_2.

Resource model



ILP Solvers

e.g. iLog CPLEX solver

- Simplex algorithm
- Branch and Bound algorithm

- Execution times can be large
- Usage of HPC - after debugging!
- Stop criteria!

Metaheuristics

- Tabu search
- Simulated annealing
- Genetic programming (evolutionary algorithms)
- Ant-based algorithms or particle swarm algorithms (nature-inspired algorithms)

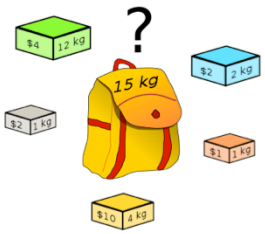
Can serve as benchmarks (lower bound) for the exact solutions and the greedy algorithms.

Bin packing algorithms

- objects of different volumes must be packed into a finite number of bins or containers, with a predetermined capacity C.
- goal: minimize the number of bins used.
- combinatorial NP-hard problem.
- greedy algorithm: **first fit algorithm**
  - fast but often non-optimal
  - placing each item into the first bin in which it will fit
  - can be made much more effective by first sorting the list of elements into decreasing order (also called first-fit decreasing algorithm)
  - there always exists at least one ordering of items that allows first-fit to produce an optimal solution.
- **best fit algorithm** places new object in the fullest bin that still has room.

Knapsack algorithms

- Special case of bin packing algorithm:
- number of bins is restricted to 1
- each item is characterised by both a volume and a value
- the problem of maximising the value of items that can fit in the bin



- approximation algorithms (greedy algorithms) are available

Autonomic resource allocation

Structure:

3.1 Autonomic Systems
- Goal
- Adaptive vs. Autonomic
- Use cases

3.2 Control Loops
- MAPE
- OODA
- FOCALE

3.3 Architecture for distributed autonomic systems

3.4 Knowledge representation

3.5 Reasoning

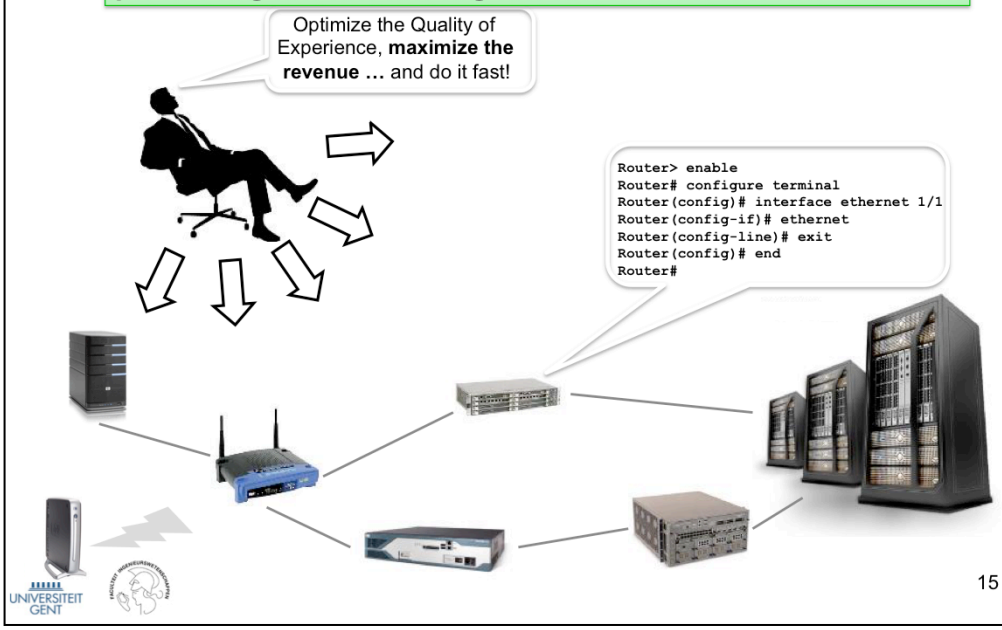
Autonomic Systems

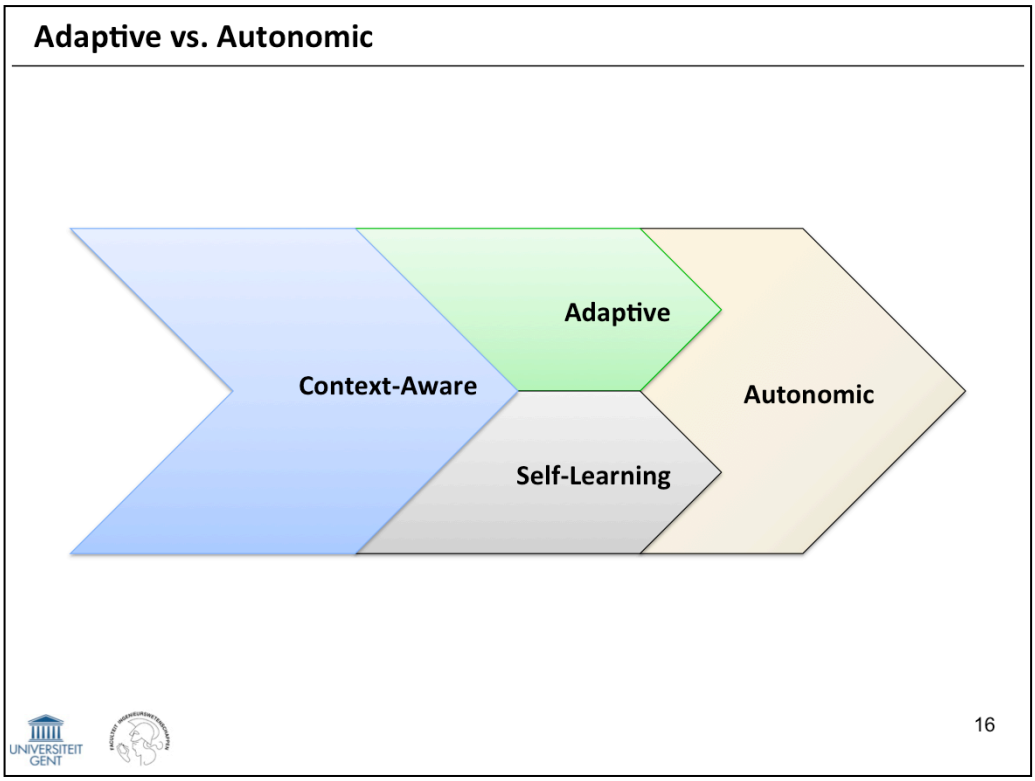
The system adapts its behavior to changes in
- The environment
- End-user needs
- Service requirements

It is governed by high-level policies
- Representing business goals
- Defined and managed by human operators

The goal of autonomic systems

Autonomic systems decrease management complexity by performing low-level configurations themselves





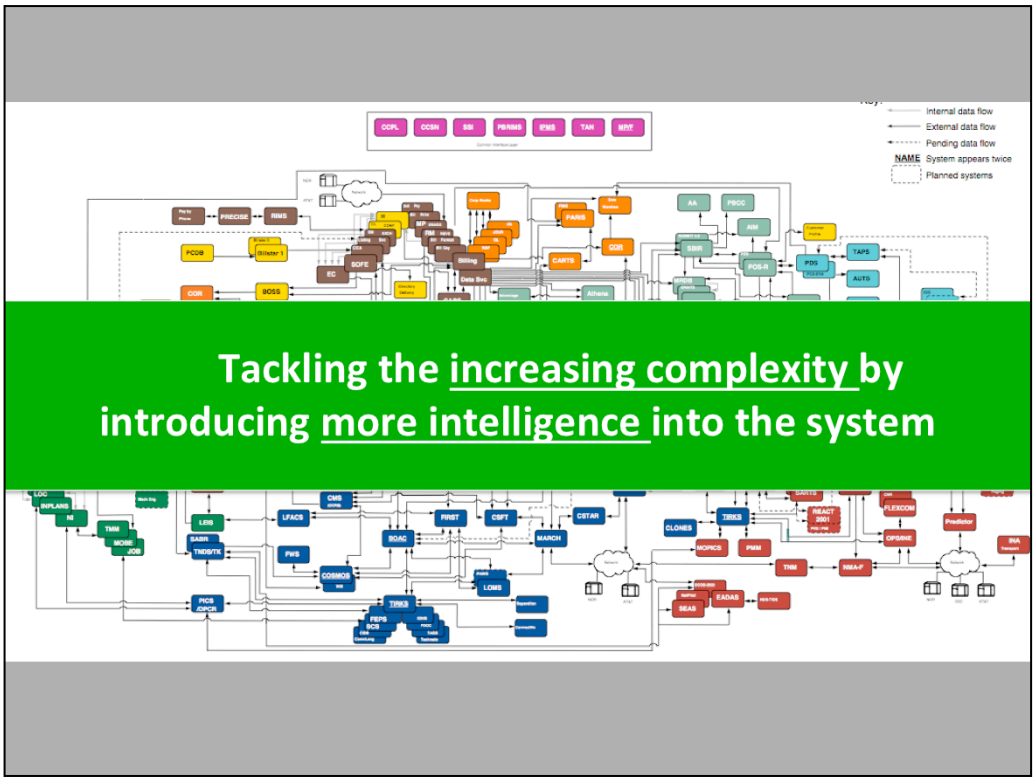
• There exists some ambiguity about the difference between “adaptive” and “autonomic”. This figure shows the relationship between some of the terms often used when classifying networked systems.

• Context awareness is at the basis of any system that wants to adapt its behavior to the environment. These systems must be able to monitor the environment, before they can react to it.

• An adaptive system, uses the monitored context in order to change its behavior when the state of the system is no longer considered valid. This is often achieved by using a set of predefined “rules” that are triggered when certain events occur.

• A self-learning system is capable of deriving new knowledge and reasoning on existing knowledge.

• An autonomic system combines both. It adapts itself to the current context, but is also able to learn from the decisions it takes, in order to derive new rules that can be used in the future. Therefore, it is an intelligent adaptive system that evolves in time and can cope with events that were unknown, or unexpected at design time. This can be achieved by deriving new policies at runtime, but also by more complex techniques such as dynamic code-generation.



In practice, we have whole range of services, devices, technologies,… which leads to an overwhelming complexity.

The goal of autonomic communications is to tackle this increasing complexity by introducing more intelligence into the network.

Use Cases

eHealth

Future Internet

Robotics

Intelligent Transportation

Cloud Computing

18

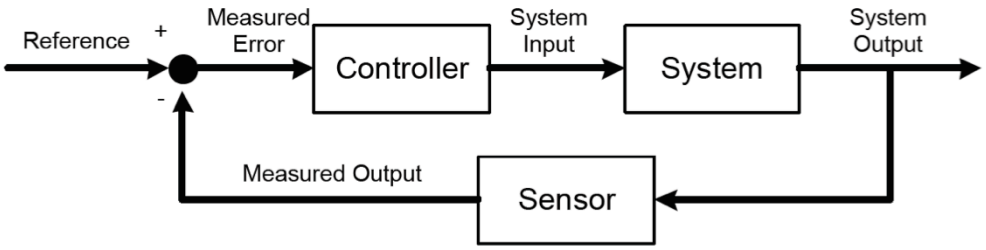
How do we design and implement an autonomic system?

Definition – 4 characteristics

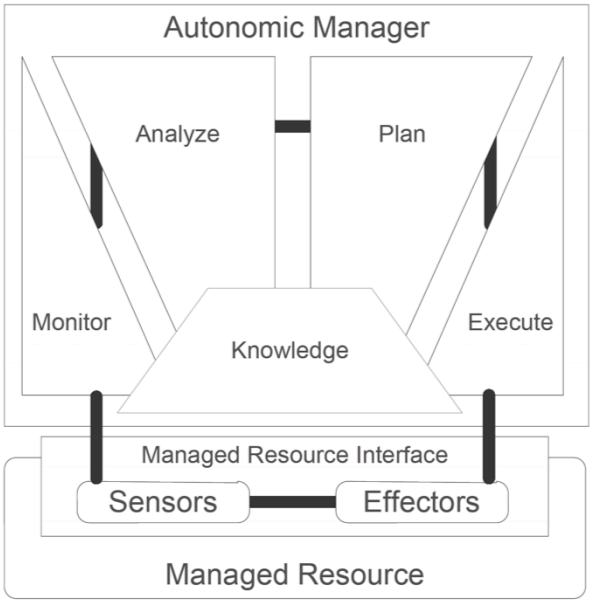
Self chop characteristics:

- **Self-configuration**
- **Self-healing**
- **Self-optimization**
- **Self-protection**

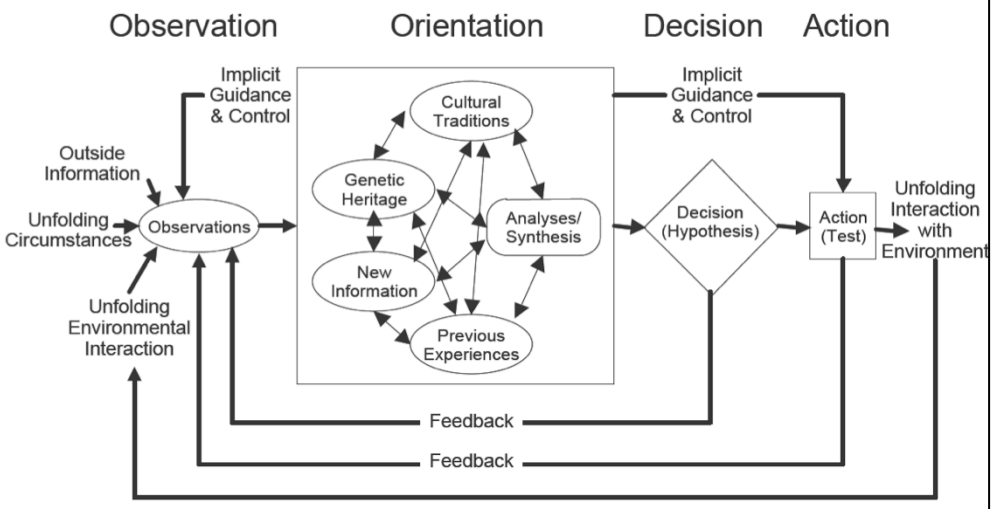
Control Loops



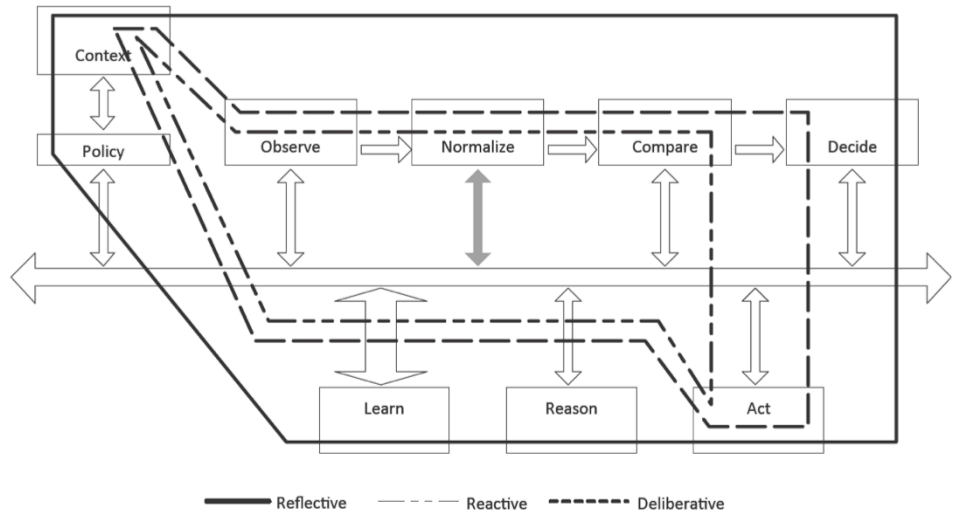
MAPE control loop



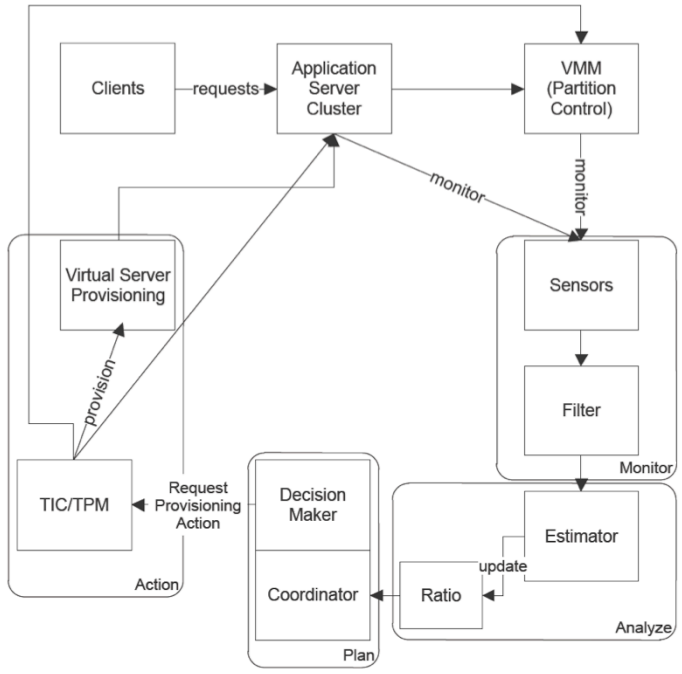
OODA control loop



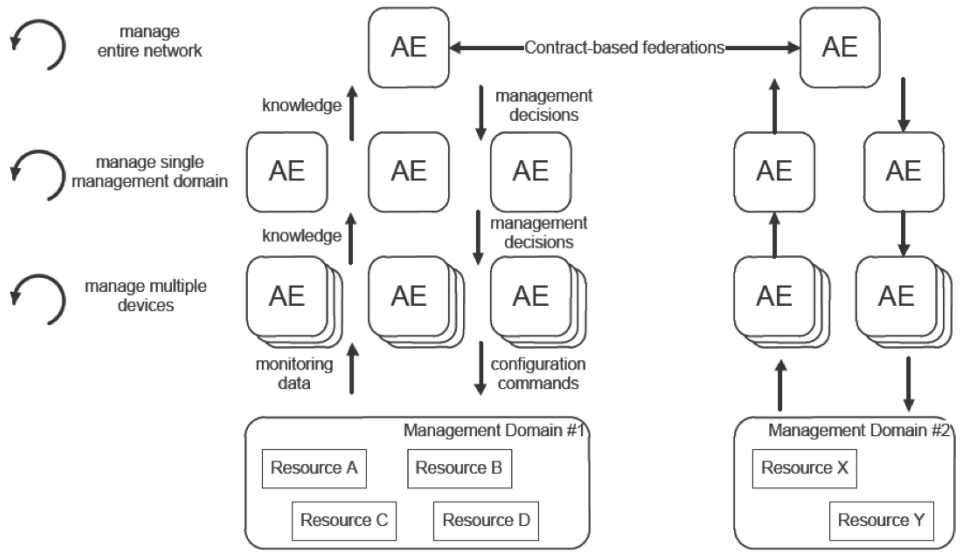
FOCALE control loop



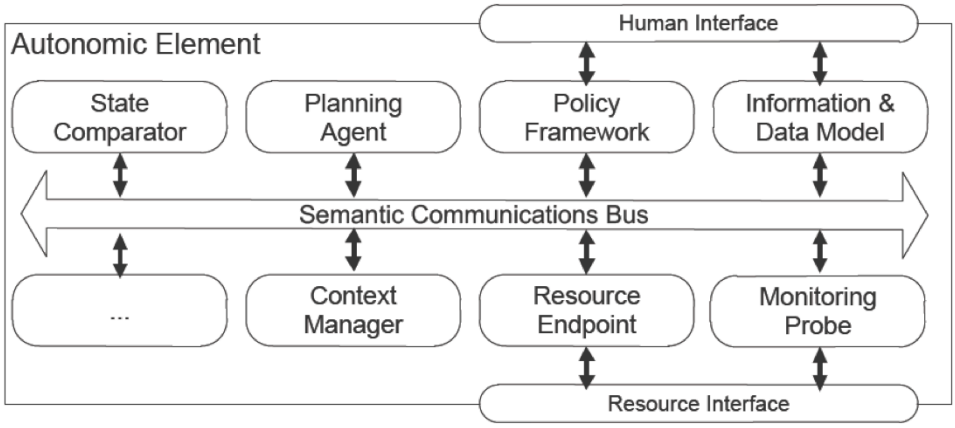
Control loop example



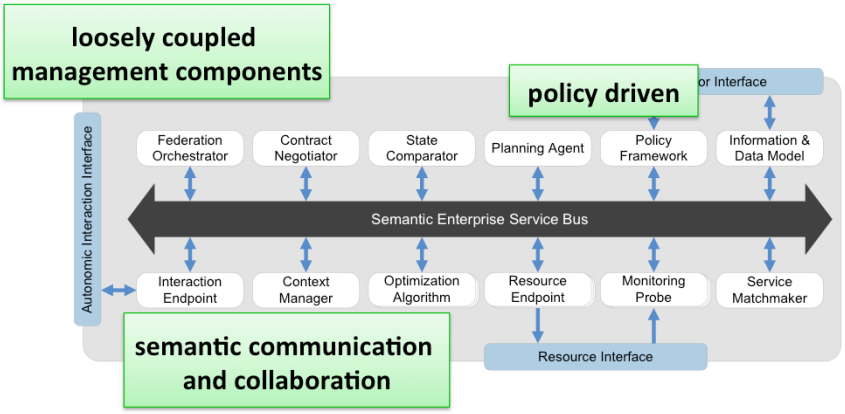
Architecture for autonomic systems



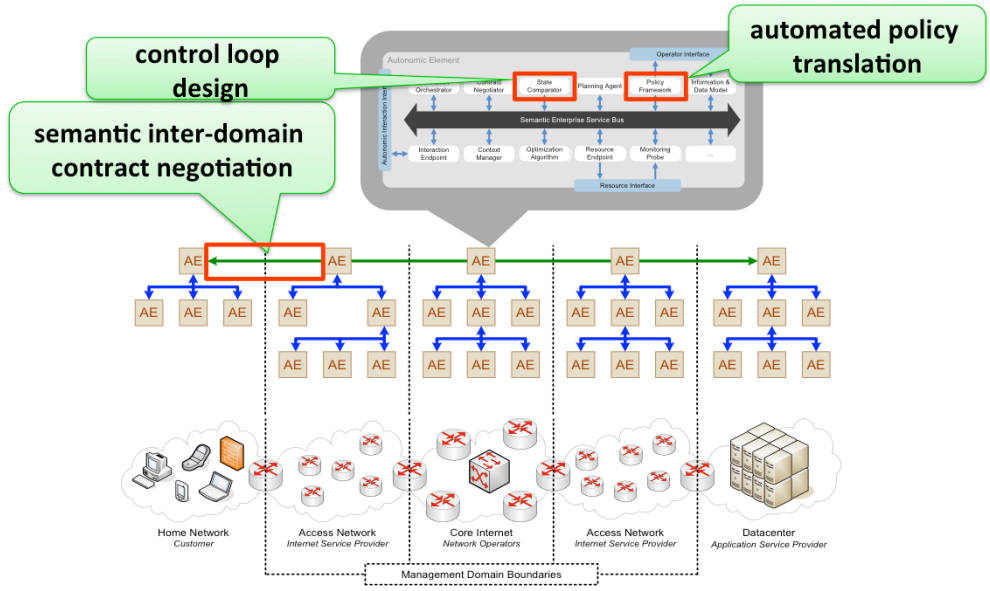
Architecture for autonomic systems



Architecture for autonomic systems



Future directions



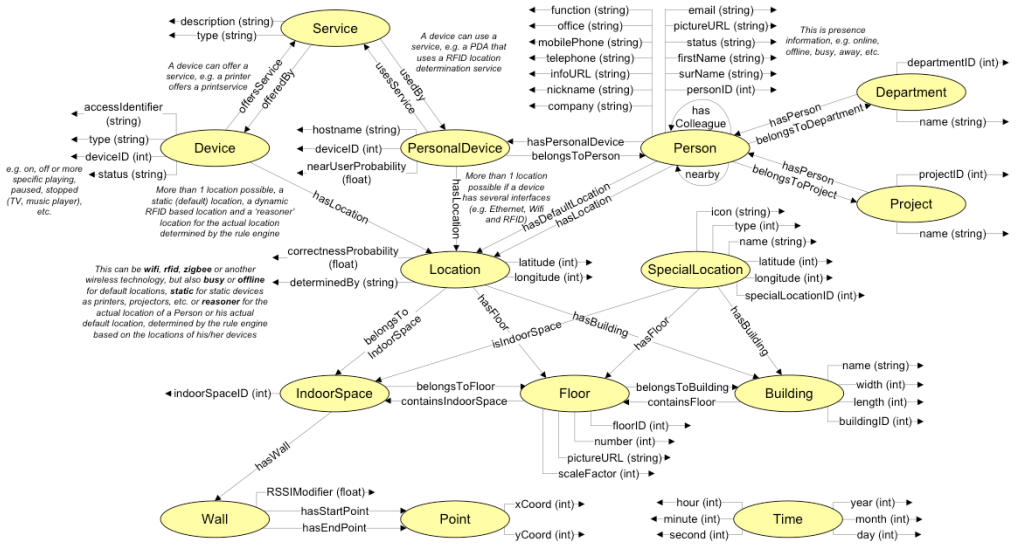
Knowledge representation

To augment the value of the available data, semantic context information should be attached to it.

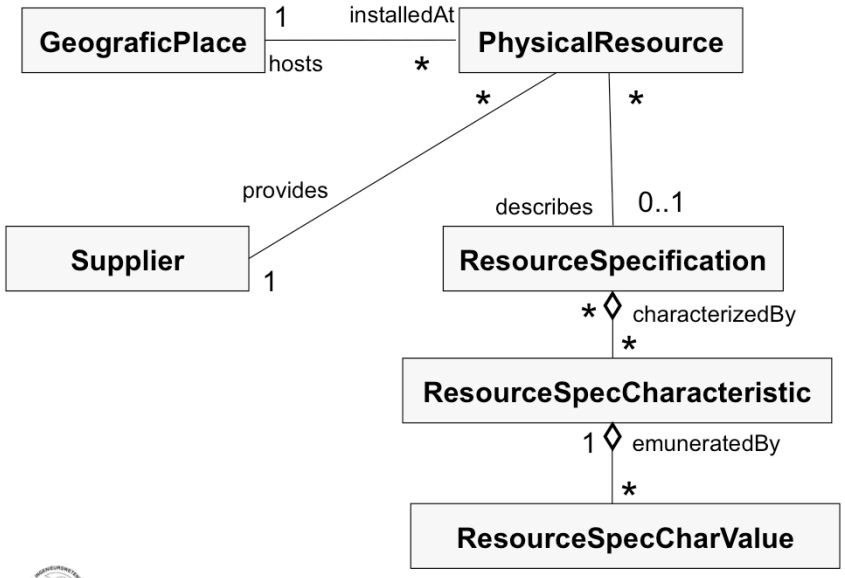
1	*Data* at its basic level
2	*Information* adding context to the data
3	*Knowledge* understanding how to use the information
4	*Wisdom* understanding when to use the knowledge

1	The wheel impact measured is TOO HIGH
2	The measurement was taken at PLACE A and TIME T
3	Impact TOO HIGH at A and T ➔ Take the train out of service
4	If remedial action is already planned ➔ Ignore further warnings

Ontology-based knowledge representation



UML knowledge representation



Ontology-based knowledge representation

Advantage: reasoning!

- Giraffes only eat leaves
- Leaves are parts of trees, which are plants
- Plants and parts of plants are disjoint from animals and parts of animals
- Vegetarians only eat things which are not animals or parts of animals

=> Giraffes are Vegetarians

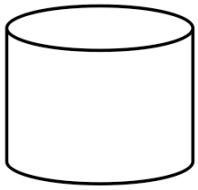
Ontology-based knowledge representation

Web Ontology Language (OWL)

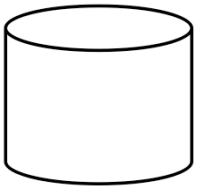
Syntax	*Serialization*
Functional-Style Syntax	`ClassAssertion( :Person :Mary )`
RDF/XML Syntax	`<Person rdf:about=‘‘Mary’’/>`
Turtle Syntax	`:Mary rdf:type :Person .`
Manchester Syntax	`Individual: Mary` `Types: Person`

OWL/XML Syntax	`<ClassAssertion>`
	`<Class IRI=‘‘Person’’/>`
	`<NamedIndividual IRI=‘‘Mary’’/>`
	`</ClassAssertion>`

Data Integration Problem



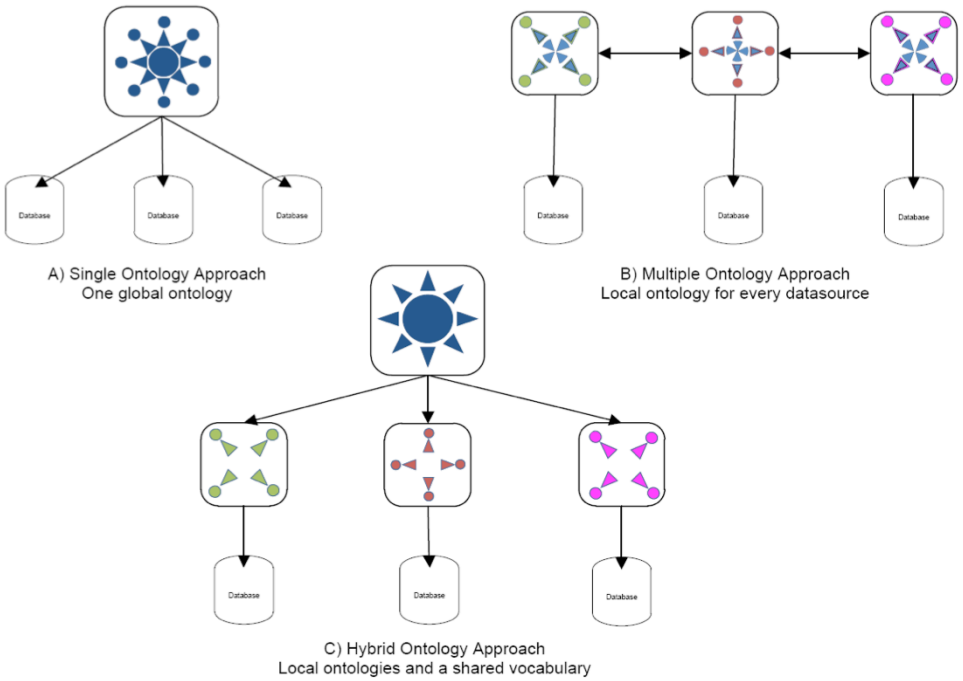
Data store 1



Data store 2

Structural heterogeneity
Syntax heterogeneity
Implementation heterogeneity
Semantic heterogeneity
- **Synonyms**
- **Homonyms**
- **Classification**

Ontology usage



Ontology construction

1. Specification: the purpose and the scope of the ontology are identified.

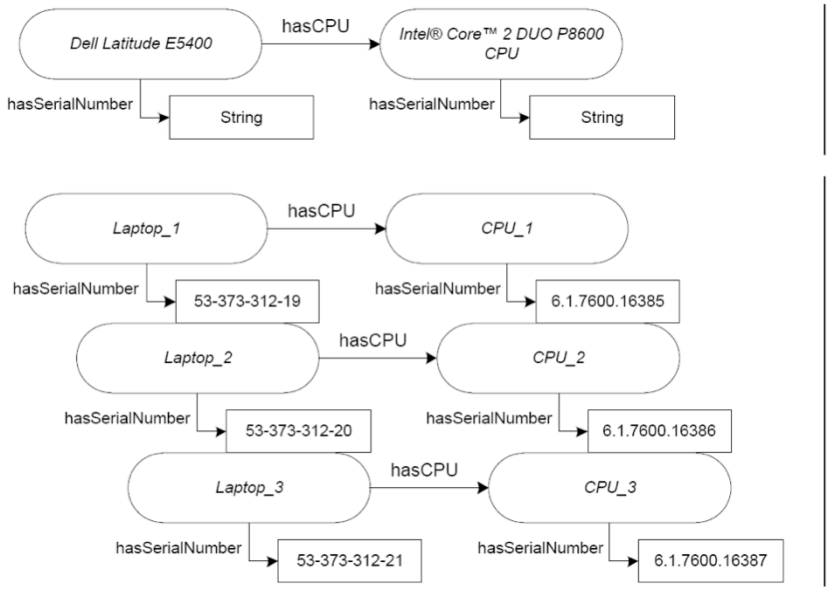
2. Conceptualization: a conceptual model of the ontology is constructed. It consists of the different concepts, relations and properties that can occur in the domain.

3. Formalization: the conceptual model is translated into a formal model for example by adding axioms that restrict the possible interpretations of the model.

4. Implementation: the formal model is implemented in a knowledge representation language, for example OWL.

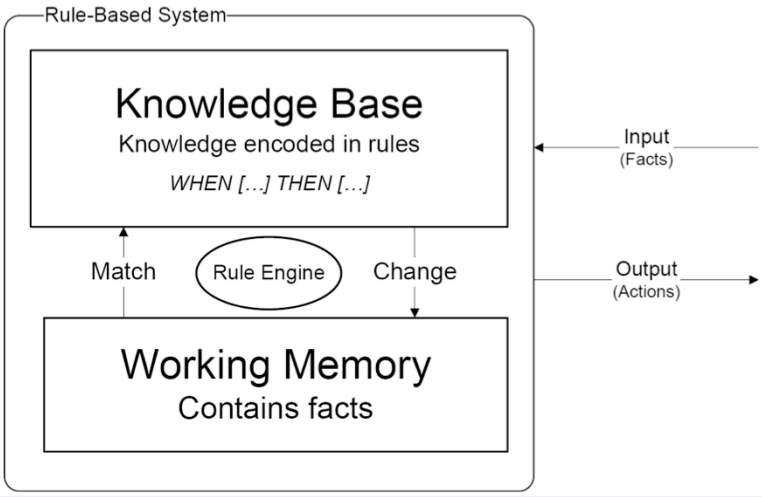
5. Maintenance: the implemented ontology has to be constantly evaluated, updated and corrected. To update an ontology, the previous steps can be used.

Reasoning



DL reasoning : Description Logics reasoning

Rule engines



Benefits:

- **Make it possible to alter and extend the provided functionality at runtime.**
- **Rules are declarative**
- **They are also more easily interpretable by domain experts**

Study of emerging topics

- Questionnaire with 24 participants (industry+academia)

topic	Industry	Academia
Autonomic and self management	**53.8%**	**36.4%**

- 3 main scientific conferences (NOMS – IM – CNSM)

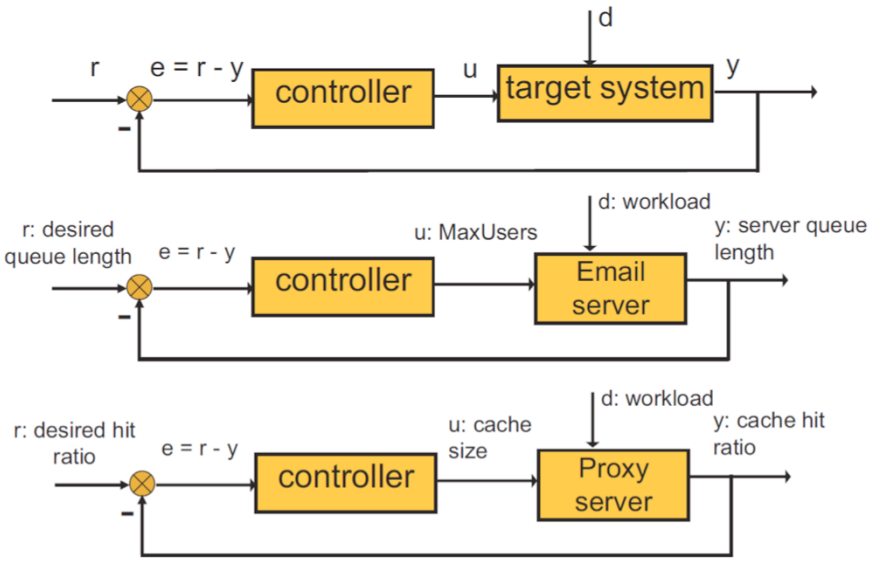
topic	NOMS	IM	CNSM
Autonomic and self management	**11.8%**	**16.8%**	**20.7%**

Autonomic and self management

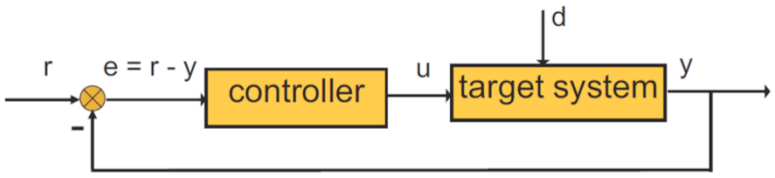
Challenge	Time frame	Context
Academia		
Automatic trust management	2 years	industry–academia
Automated translation of business objectives	3 years	industry–academia
Automation of management tasks	3–5 years	industry
Fully automated systems management	10 years	industry–academia
Autonomic management of virt. infrastr.	5 years	all
Automated management in clouds and big data	3–7 years	all
Industry		
Autonomic and policy-based mgmt of SDN	2 years	industry-standardization
Automated management of virtual networks	4 years	all
Highly dynamic self-organizing LTE networks	2–3 years	industry-academia
Distributed self-organising wirel. netw. mgmt	2–3 years	all
Embedding Autonomic features in telco netw.	3–6 years	industry-standardization
Remediation actions in automated mgmt	1-2 years	all

Control theory

Feedback control:



Feedback control



Feedback: Compare the actual result with the desired result, and take actions based on the difference.

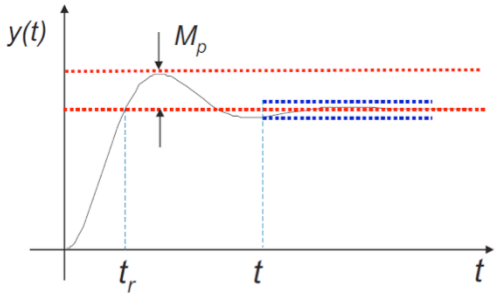
- closed-loop
- acts only where there is observed error
- robust to model errors
- reduce the effect of external disturbance
- can stabilize an unstable system
- risk for instability

Goals of control system

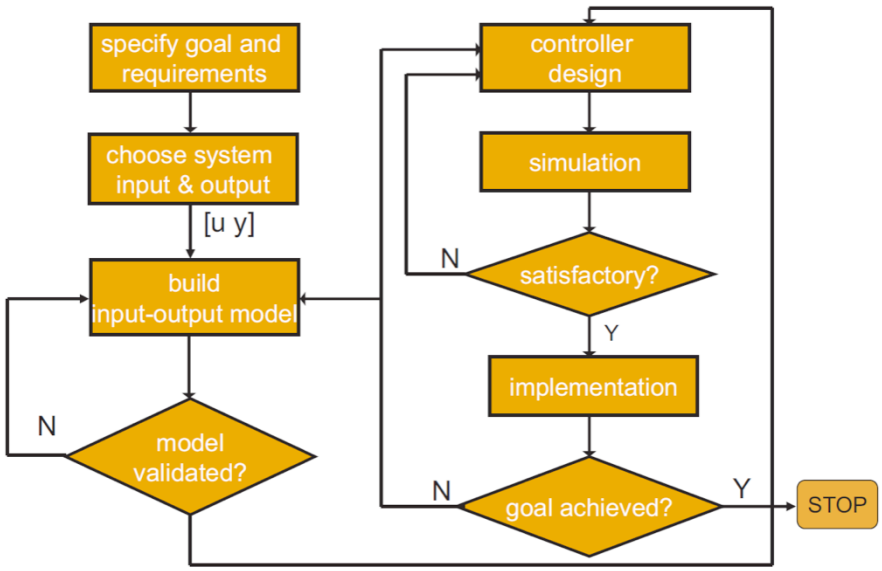
- Regulatory control
  - Keep output close to a constant target
    - thermostat, server utilization control
- Tracking control
  - Keep output around a moving target
    - robot movement, radar, adjust TCP window to network bandwidth
- Optimization
  - Minimize/maximize output
    - minimize response time, maximize throughput, minimize power
  - Online optimization algorithms may be used

Controller design criteria

Desired Properties	Controller Design Criteria
Stability	Poles within the unit-circle
Reference tracking	Zero steady-state error
Fast response	Small rise time (t_r) and settling time (t_s)
Bounded peak	Small overshoot (M_p)

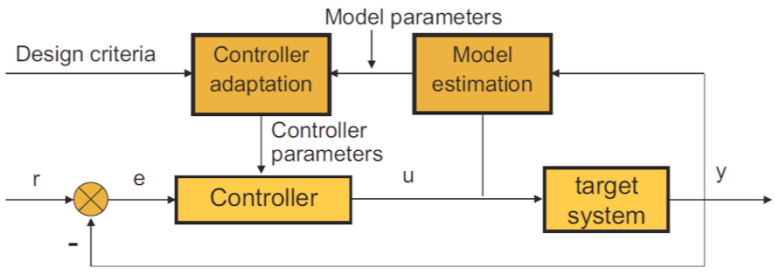


Control system design methodology



Advantages of CT-based approach

- System transients are covered, not in queuing theory
- Correlation between different metrics can be taken into account
- Well studied controllers exist:
  - PID controllers (Proportional, Integrative and Derivative)
  - LQR (Linear Quadratic Regulator)
- Tradeoff between responsiveness and stability can be pursued
- Non-linear and time-varying behavior can be handled:



UNIVERSITEIT GENT

47

Performance Modeling

Predict behaviour of system

- Response times
- Throughput

Goal:

- Determine number of required threads
- Number of required servers
- Where to instantiate server processes ?

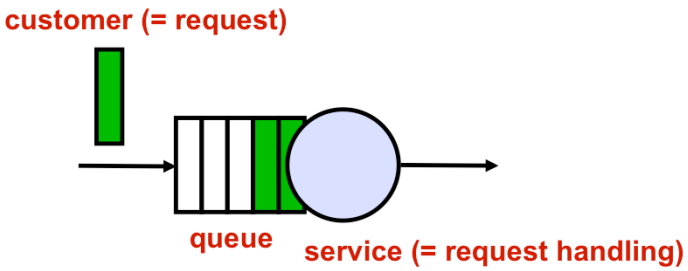
Most important metrics: T_{wait} and T_{proc}

Need for Queuing system theory

Queuing System (1)

Customers (= requests) arrive at random times to obtain service

Service time (= processing delay) is the time to fulfill a request



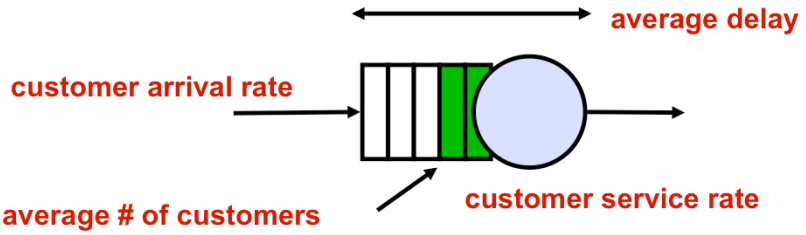
Queuing System (2)

Assume that we already know:

- Customer arrival rate
- Customer service rate

We want to know:

- Average number of customers in the system
- Average delay per customer



Little's Theorem

N = Average number of customers

λ = Arrival rate

T = Average customer time

$$N = \lambda T$$

Holds for almost every queuing system that reaches a steady-state

Express the natural idea that crowded systems (large N) are associated with long customer delays (large T) and reversely

Poisson Process

A stochastic process *A(t)* (*t* > 0, *A(t)* >=0) is said to be a Poisson process with rate λ if

1. *A(t)* is a counting process that represents the total number of arrivals in [0, *t*]
2. The numbers of arrivals that occur in disjoint intervals are independent
3. The number of arrivals in any [*t*, *t* + τ] is Poisson distributed with parameter $\lambda\tau$

$$P\{A(t+\tau)-A(t)=n\}=e^{-\lambda\tau}\frac{(\lambda\tau)^n}{n!},\quad n=0,1,\ldots$$

Properties of Poisson Process (1)

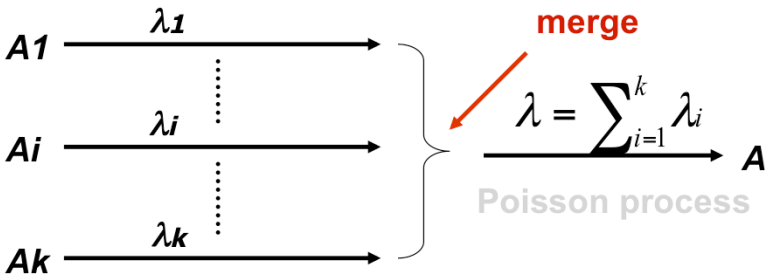
Interarrival times τ_n are independent and exponentially distributed with parameter λ

$$P\{\tau_n \leq s\} = 1 - e^{-\lambda s}, \quad s \geq 0$$

The mean and variance of interarrival times τ_n are $1/\lambda$ and $1/\lambda^2$, respectively

Properties of Poisson Process (2)

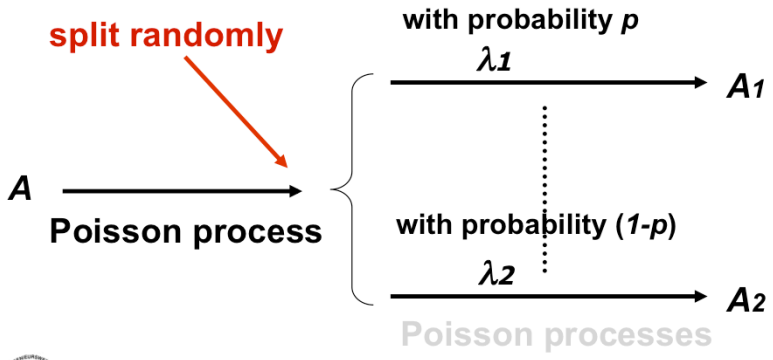
If two or more independent Poisson processes *A1, ..., Ak* are merged into a single process $A = A1 + A2 + ... + Ak$, the process *A* is Poisson with a rate equal to the sum of the rates of its components



independent Poisson processes

Properties of Poisson Process (3)

If a Poisson process *A* is split into two other processes A_1 and A_2 by randomly assigning each arrival to A_1 or A_2, processes A_1 and A_2 are Poisson

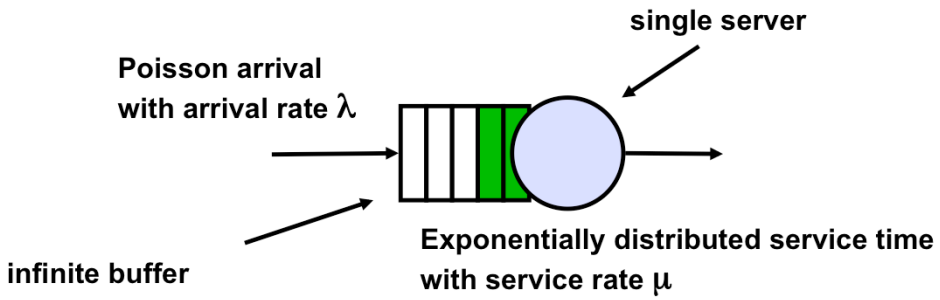


M/M/1 Queuing System

A single queue with a single server

Customers arrive according to a Poisson process with rate λ

The probability distribution of the service time is exponential with mean $1/\mu$

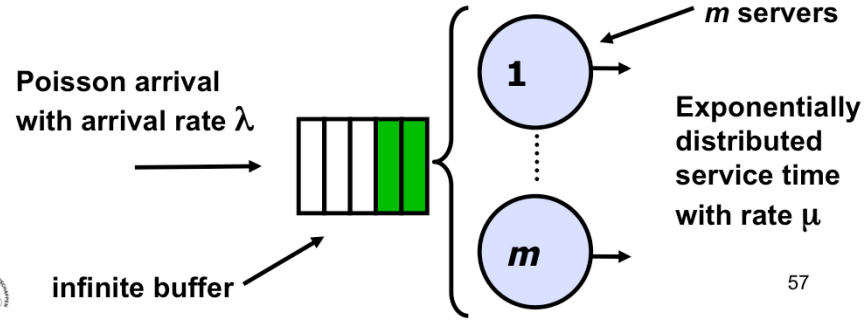


M/M/m Queuing System

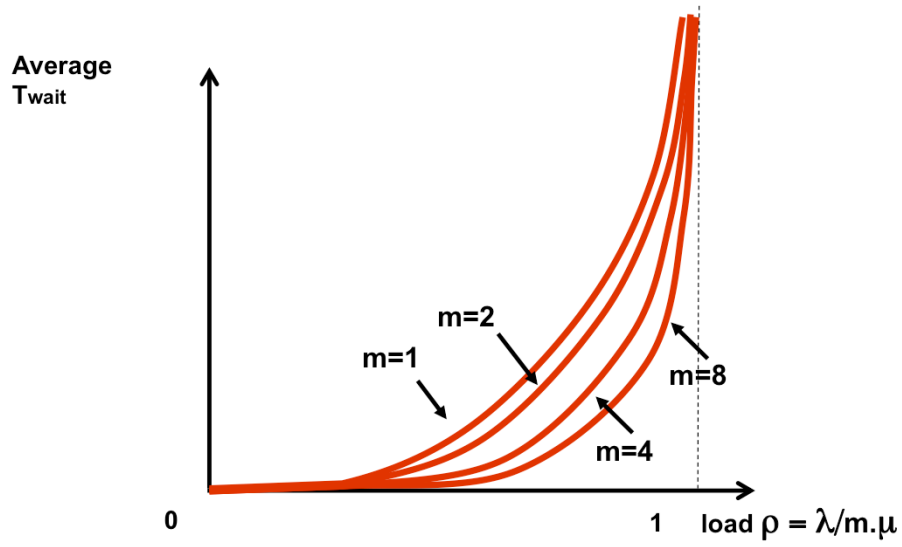
A single queue with *m* servers

Customers arrive according to a Poisson process with rate λ

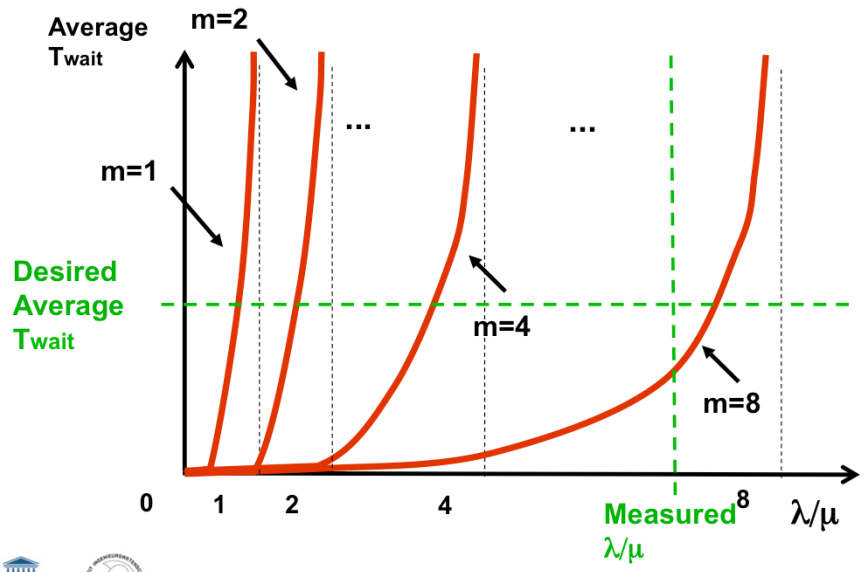
The probability distribution of the service time is exponential with mean $1/\mu$



T_{wait} calculation



Calculation of required threads



Dimensioning

Determine number of required threads

Number of required servers

Evaluation of clustering modes

- i.e. how to best group process on the workstations

Resource Allocation in Virtualized environments

Structure:

6.1 Virtualized Networks

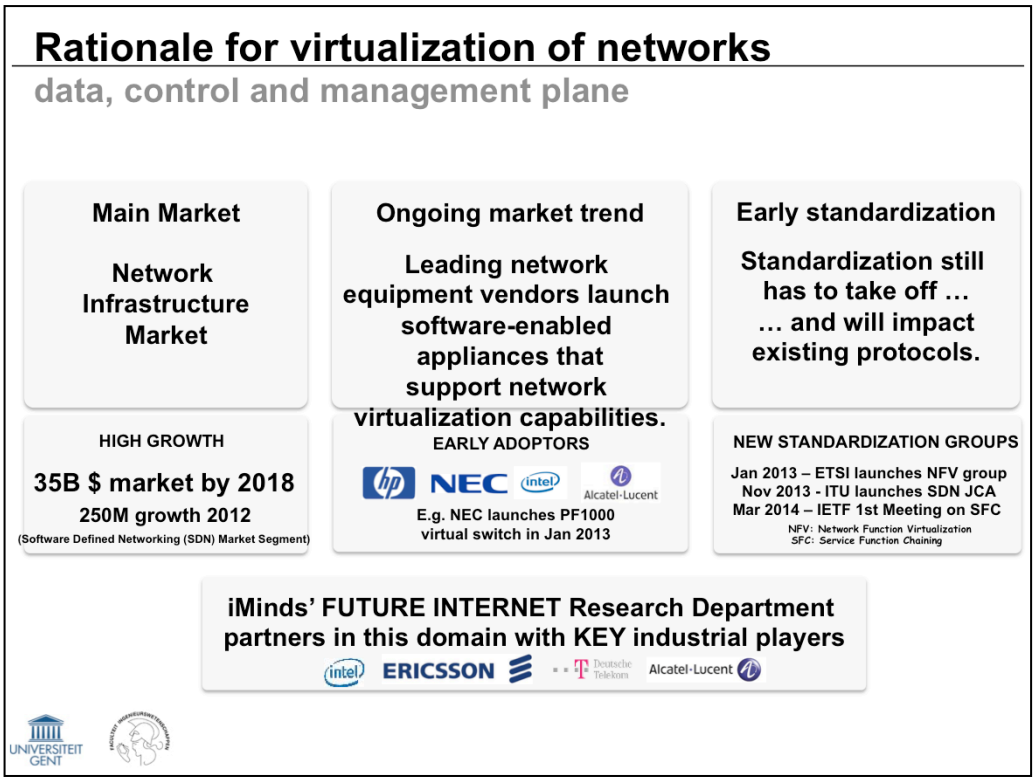
6.2 Software Defined Networks

6.3 Network Virtualization

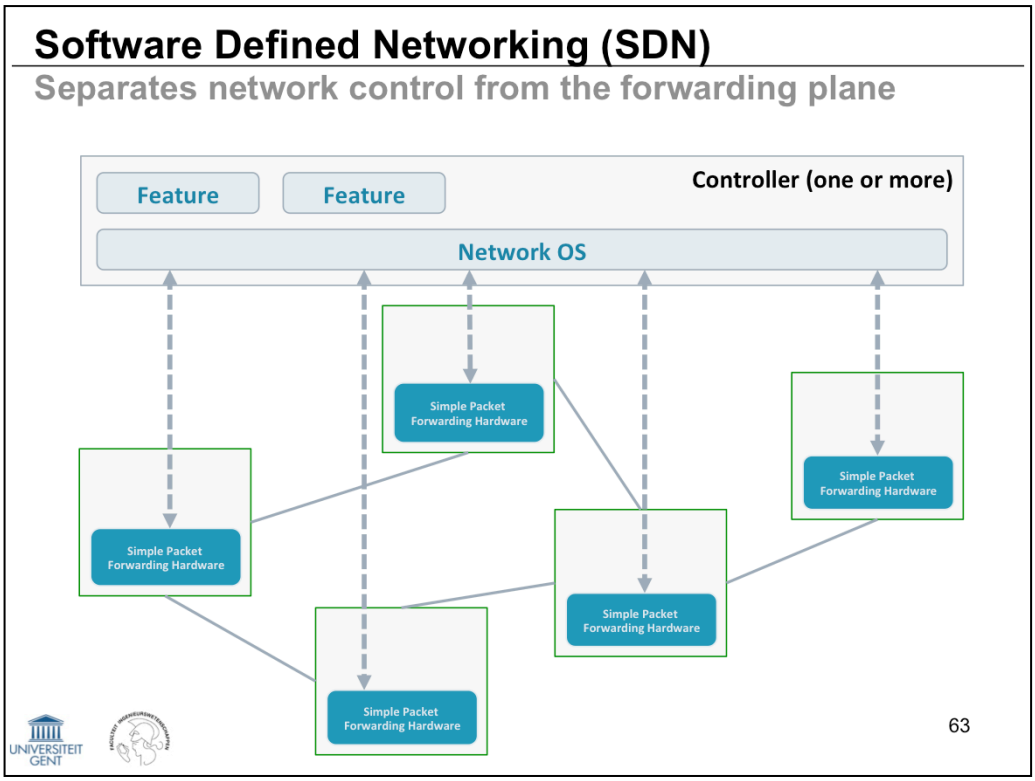
6.4 Network Function Virtualization

6.5 Service Function Chains

6.6 Challenges



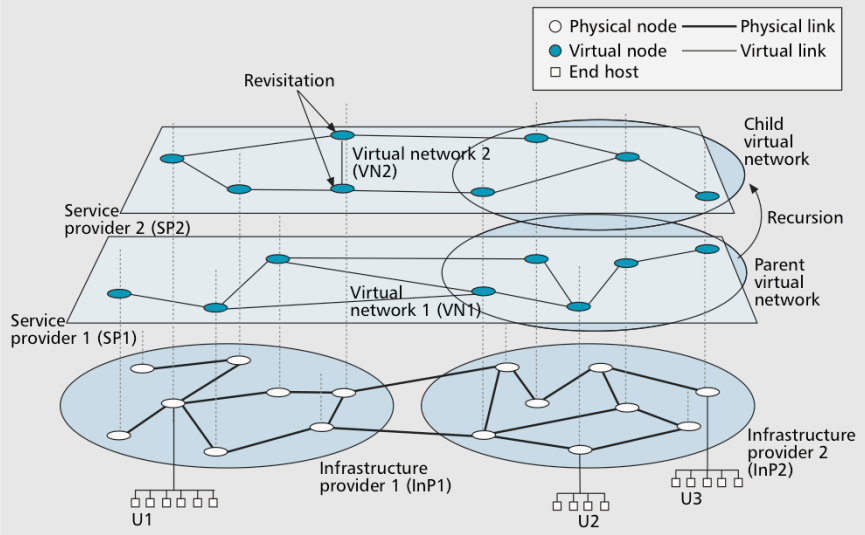
- SDN and NFV are growing markets, that are expected to reach a multi-billion dollar market share by 2018
- All the big equipment vendors have jumped on the SDN/NFV wagon, including HP, NEC, Intel, Alcatel-Lucent
- Many standardization efforts have been launched in 2013/2014
- iMinds is involved in several SDN/NFV projects with the big players in this field



SDN is a concept that splits up the network's control and forwarding planes. In traditional routers and switches, the control logic as well as the actual forwarding behavior are co-located on the same physical device. This results in expensive hardware, and limited flexibility. In SDN, the control logical is lifted out of switches and placed into a logically centralized controller entity. This controller entity contains network applications that adapt the forwarding tables of the SDN-enabled switches, based on policies and current network activity.

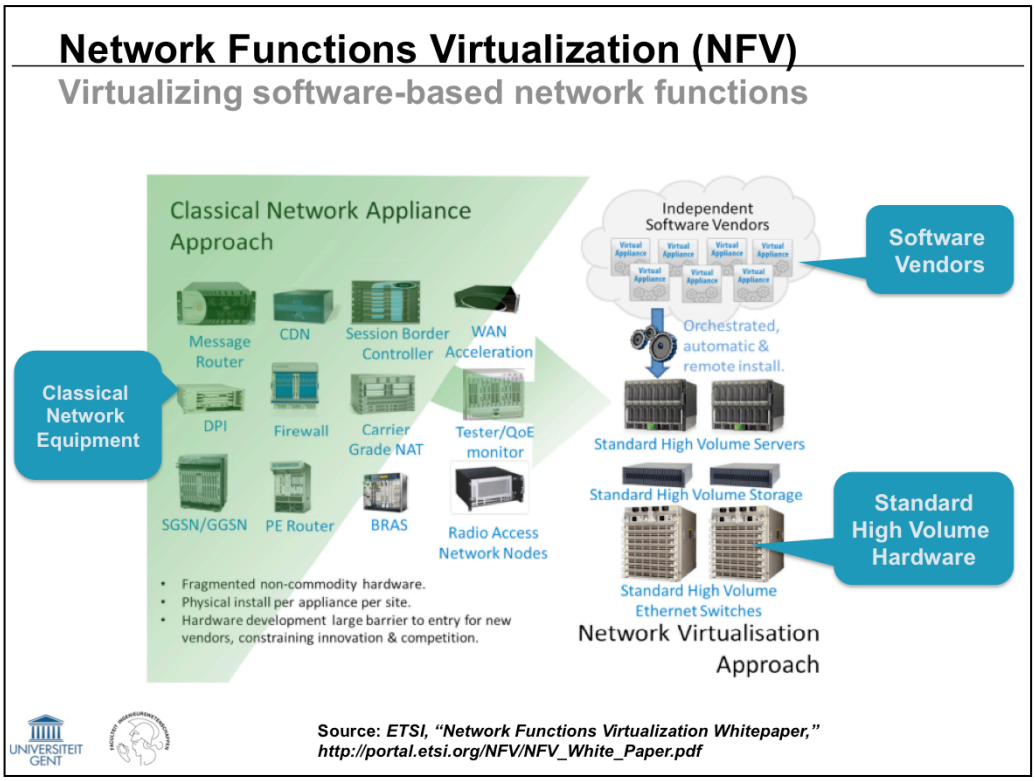
Network virtualization (NV)

Bringing virtualization concepts to the network



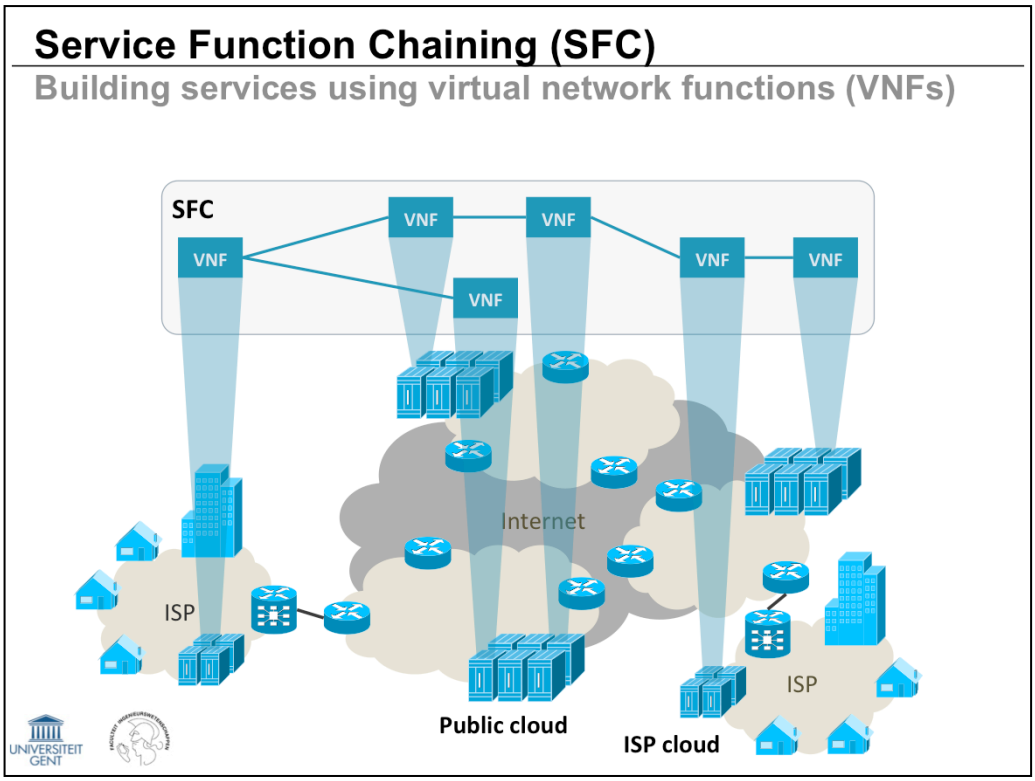
Source: *N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," IEEE Communications Magazine, 47(7):20–26, 2009.*

Network Virtualization is a technique that allows the logical network topology to be separated from the actual physical infrastructure. In essence, virtual networks are built up out of virtual switches and routers (of which one or more are linked to a single physical switch or router) and virtual links (which are mapped onto a path of physical links). The advantage of this approach is that multiple virtual networks can exist independently on top of the same physical infrastructure. This provides improved separation of control, security, easier configuration and management, and the ability to dynamically scale provisioned resources (similar to cloud computing).



Network Functions Virtualization (NFV) is a concept that applies Cloud Computing and SDN concepts to carrier networks. Classical hardware network appliances and equipment are replaced by software-based functionality. This allows them to be deployed on generic cloud-like datacenter hardware. In NFV, services are composed out of fundamental building blocks, called network functions. These functions are connected into a graph, which is referred to as a Service Function Chain or a Network Forwarding Graph.

NFV has several advantages, such as replacing expensive specialized hardware with cheap generic servers, increasing flexibility by supporting dynamic resource scaling, and faster service time to market.



As stated, in NFV a service is composed out of (virtual) network functions. The VNFs are subsequently deployed in one or multiple cloud datacenters, which can be located across multiple administrative domains. The SFC graph edges represent logical links between VNFs, and requirements (e.g., bandwidth, delay) could be attached to them. These can be enforced using network virtualization techniques or by way of SDN.

Advantages of SDN, NFV, and SFC

Network Operator

- Replace dedicated hardware with generic hardware and software-based functions
- Maximize resource utilization and optimize energy usage
- Faster and easier deployment, configuration, and updating of network functions
- Support for the Network-as-a-Service business model

Service Provider

- Dynamically scale network, computing and storage resources based on service requirements
- More easily facilitate inter-domain Quality of Service
- Reduced time to market for services
- Flexibly manage service deployment based on traffic patterns and user mobility

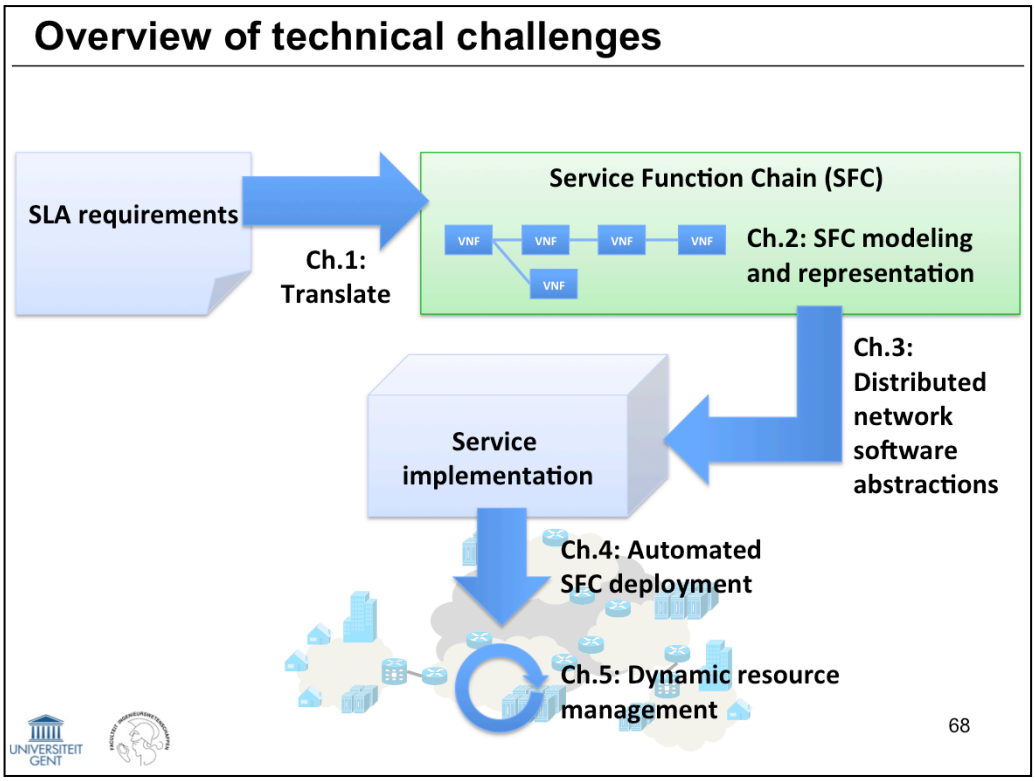
67

Network operator:

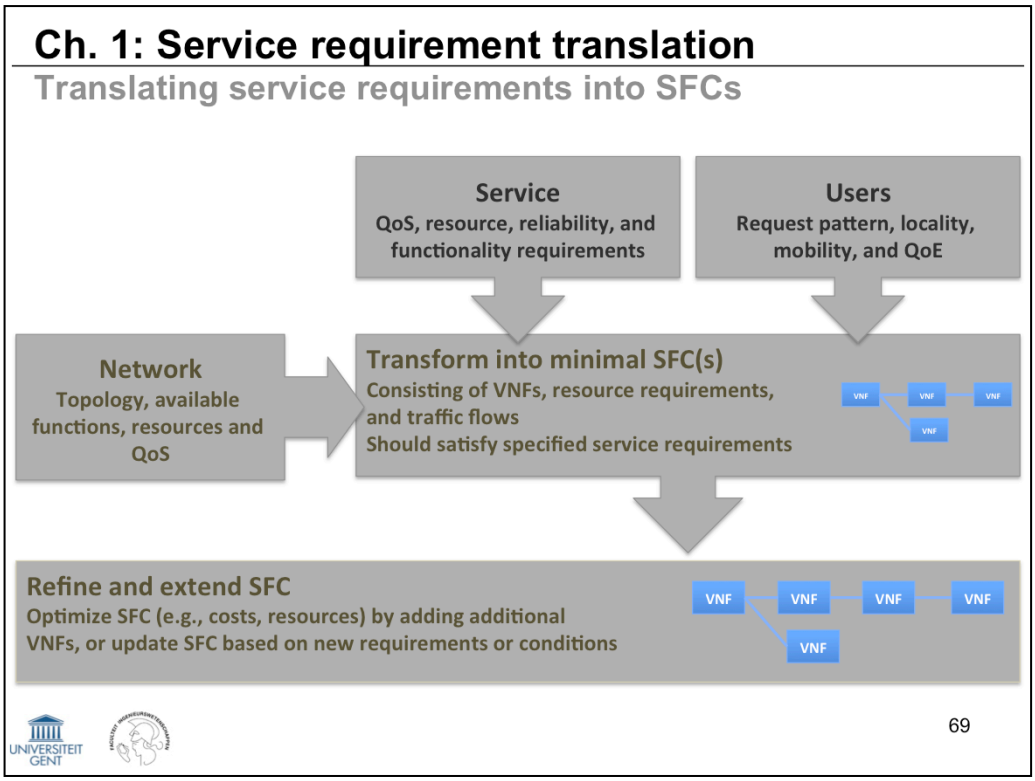
- "Replace dedicated hardware with generic hardware and software-based functions" -> SDN (forwarding functions) and NFV
- "Maximize resource utilization and optimize energy usage" -> SDN and NFV, but this was also promised by PCE, GMPLS,.. and was never achieved.
- " Faster and easier deployment, configuration, and updating of network functions" -> NFV, SDN required for SFC
- "NaaS" -> SDN

Service provider:

- " Dynamically scale network, computing and storage resources based on service requirements" -> NFV, Cloud Computing
- " Guarantee Quality of Service across network domains" -> fluffy... kan eigenlijk met ATM en MPLS... zou misschien meer de klemtoon leggen dat het met nu makkelijk zou kunnen met goedkopere "SDN" hardware
- " Reduced time to market for services" -> NFV en SFC
- " Flexibly manage service deployment based on traffic patterns and user mobility" -> NFV, SDN->SFC



This slide shows the five main NFV related research challenges that we identified, and illustrates how they are linked together. The first challenge is related to translating a set of service requirements into a service chain template. The second challenge is concerned with modelling and representing SFCs. Languages and models are needed to represent functions, their relationships, their resource requirements, and the flow of data through the SFC. Third, abstractions are needed for programming network services on a high level in order to link the SFC template to the actual deployment. Fourth, VNFs need to be deployed within datacenters and resources need to be provisioned in the network and within the datacenter in order to satisfy service requirements. Finally, there is a need for dynamically managing the resources associated with the SFC deployment, to adapt to network changes and problems.



In NFV, there are several approaches towards specifying SFCs. For example, SFCs could be created using abstract high-level network programming languages. An alternative is the automatic generation of SFC templates based on service requirements and user patterns. In essence, based on the requirements of a service, the functions and their connection could be identified. The user patterns could subsequently be used to determine resource provisioning and locality of functions. Finally, network characteristics might come into play, as they determine what kinds of SFCs can actually be deployed on the infrastructure. Note that a set of requirements may be translated into multiple alternative SFCs, and some of those may not be feasible for deployment.

The challenge is related to the more general problem of automated software requirements refinement, a sub-area of requirements engineering [1]. Requirements refinement is concerned with translating high-level customer business requirements into concrete software component specifications. Although some progress has been made in this area, existing solutions are limited to semi-automated tools for assisting humans in performing refinement tasks, rather than fully automated algorithms. In the area of network and service management, requirements refinement principles have been applied to the policy refinement [2] problem, which aims to translate high-level business policies into low-level device configurations. In line with

Ch. 1: Service requirement translation

Main open research challenges

How to perform the translation with limited information about the environment
(e.g., inter-domain SFCs)

Select suitable subset of SFCs out of alternatives

How to determine compatibility of SFC with underlying network

How to represent VNFs in terms of their effect on service traffic

70

There are several open issues that arise when solving this challenge:

- When an SFC transfers multiple domains, the algorithm will only have limited information about the underlying infrastructure
- Many alternative SFCs may exist that satisfy the set of requirements, as shown for the use case in slide 27. The algorithm will need to select the most suitable one, but many metrics exist to determine this (e.g., cost efficiency, ease of deployment, resource optimization, etc.)
- The theoretically most optimal SFC may not be deployable in the actual infrastructure. As such, interaction is needed between the algorithm and the underlying network to determine what is feasible and what is not
- In order to be able to perform this translation automatically, the capabilities and requirements of SFCs need to be semantically modeled. The algorithm needs to know the effect of VNFs on the traffic flows they process

Ch. 2: SFC modeling and representation

Modeling individual Network Functions
- Right abstraction level – atomic components vs. composition
- Distinguish control vs. data plane components

Mechanisms to describe composition (chains/graphs) of Network Functions
- Syntactic frameworks (e.g. static descriptions)
- Semantic frameworks (e.g. ontology-based descriptions)

Providing (static/a priori) guarantees in the description of Service Function Chains
- Correctness
- QoS/Resiliency

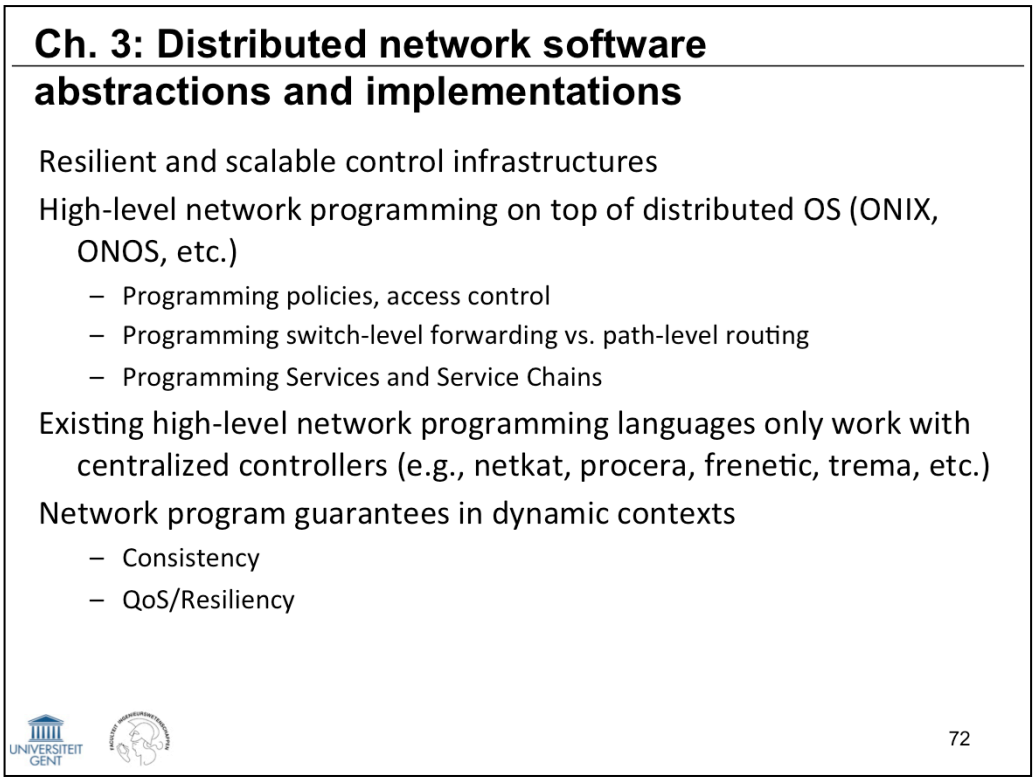
71

Flexibility in the control and provisioning of services built from virtualized network functions is envisioned through the introduction of API's and programmability at different levels. However, programmability at the level of services, service function chains, as well as individual network functions remains an open challenge. Individual network functions can be considered as packet-processing components similar to Click components, [Kohler2009] or as generic as images of virtual machines or light-weight containers containing any potential network application. The interconnection of these components into a Service Chain or Service Graph, can be modeled through static syntactic representations (e.g., XML/JSON-description of the graph), or via semantic frameworks (e.g., using ontologies).

The following publications can be used as initial steps towards specifying NFs and SFCs.

References:

1. Ter Beek, Maurice H., Antonio Bucchiarone, and Stefania Gnesi. "Formal methods for service composition." *Annals of Mathematics, Computing & Teleinformatics* 1.5 (2007): 1-10.
2. Koslovski, Guilherme Piegas, Pascale Vicat-Blanc Primet, and Andrea

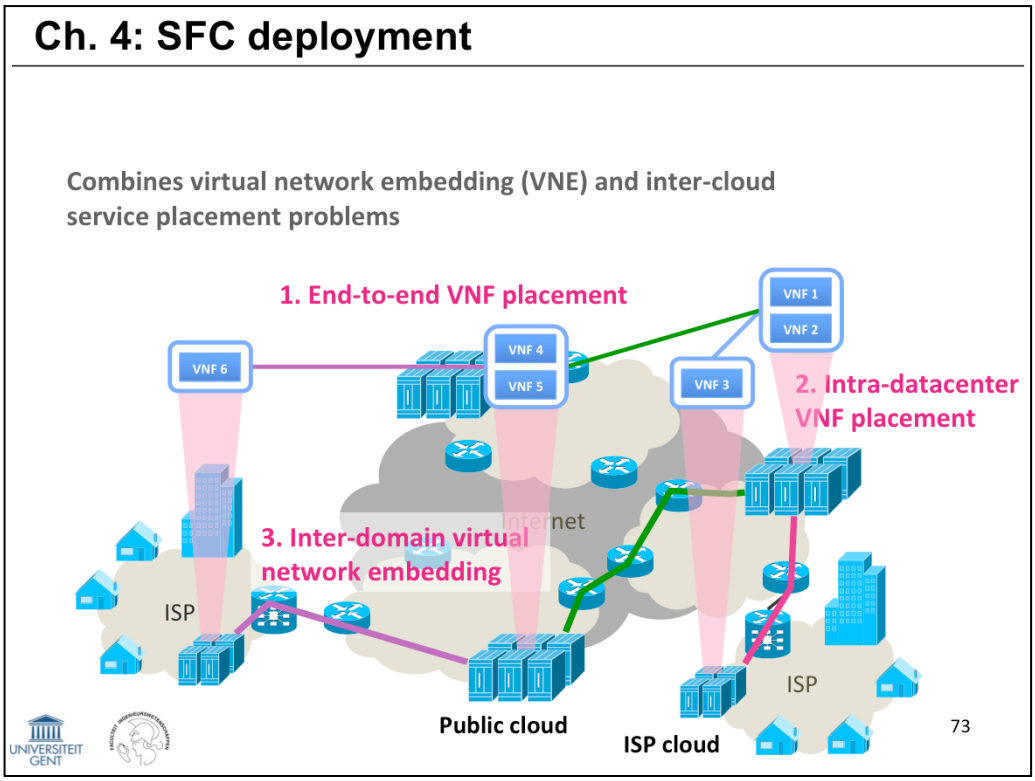


Flexible API's and languages will introduce tremendous flexibility in the description of new services. However, in order to support the control processes for setting up these services, a truly resilient and scalable control platform and/or network operating system will be needed in order to be able to handle thousands of service requests in a day on a network infrastructure consisting of more than a million of devices (see earlier slides). Early SDN control frameworks were mainly physically centralized, more recent platforms such as ONOS enable network programmability on a physically distributed (logically centralized) set of SDN controllers. The concepts of SFC and NFV put even more stress on these control frameworks, as not only network, but also cloud resources need to be managed and controlled/ orchestrated.

Initial proposals to cope with these challenges are proposed below.

References:

1. ONOS, http://www.slideshare.net/umeshkrishnaswamy/open-network-operating-system
2. Canini, Marco, et al. "STN: A Robust and Distributed SDN Control Plane." *Open Networking Summit 2014*. 2014.
3. Bezahaf, Mehdi, Abdul Alim, and Laurent Mathy. "FlowOS: a flow-based



The NFV resource allocation problem aims to map Service Function Chains (SFCs) unto the underlying physical infrastructure. This process consists of three (interconnected) steps:

1. **End-to-end Virtual Network Function (VNF) placement:** Map VNFs unto one or multiple cloud datacenters.
2. **Intra-datacenter VNF placement:** Select a suitable datacenter server for each VNF (possibly from multiple SFCs) and allocate the necessary computing, memory, and storage resources to them.
3. **Intern-domain SFC edge embedding:** Map SFC edges unto physical paths through the network that interconnect the datacenters on which SFCs are placed

These steps are related to two problems addressed in state of the art literature; (inter-) cloud application placement (steps 1 and 2) [1], and virtual network embedding (VNE) (steps 1 and 3) [2]. Independently, both of these problems are known to be computationally hard (i.e., NP-complete). The related NFV problem is even more complex due to the following reasons:

1. In NFV, VNF placement and VNE are interconnected. The choices made when solving either one sub-problem affects the other, possibly leading to

Ch. 4: SFC deployment

Main open research challenges

Distributed inter-domain VNE with limited information exchange has not been studied much

Interaction between VNE and inter-datacenter VNF placement remains largely unexplored

Problem not solved at "Internet-scale", i.e. over 40.000 autonomous systems, and many thousands of public cloud data centers

Other open issues

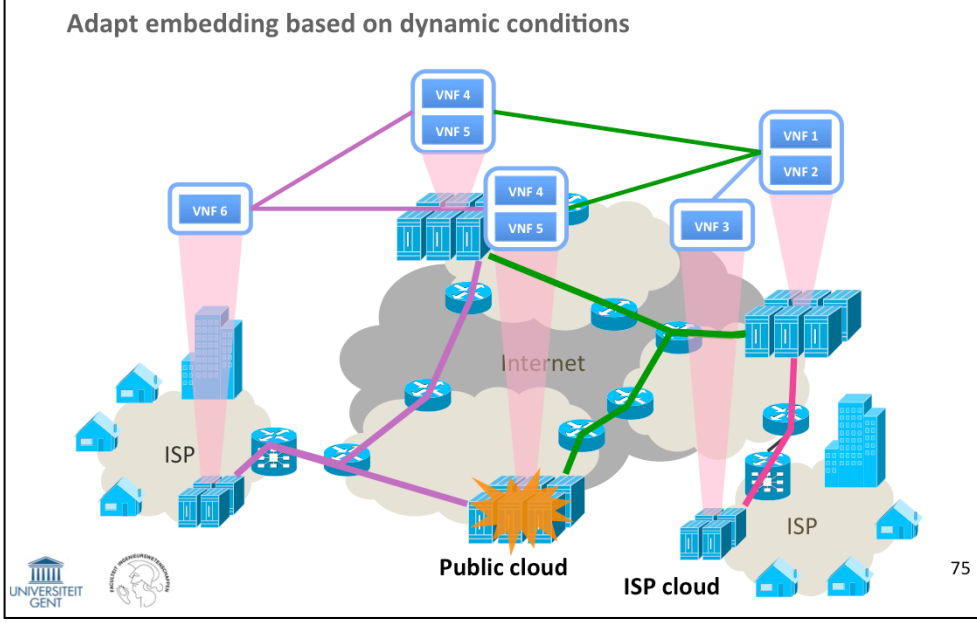
- Resiliency and survivability of inter-domain embedding
- Energy efficient embedding

74

Recent surveys of the state of the art related to this problem have exposed several open research issues:

1. Service providers may which to embed SFCs across multiple administrative domains, for reasons of efficiency, cost-effectiveness, or resilience. The problem of deploying SFCs across domains in a scalable manner, while maintaining autonomy and independence of individual infrastructure providers is challenging and remains largely unaddressed [3].
2. The VNE problem has received a lot of attention in literature. However, most existing algorithms focus on the solving the static and centralized problem. There have only been very few studies that tackle the VNE problem in a distributed manner (e.g., for scalability reasons or due to inter-domain concerns) when conditions dynamically change (e.g., due to failures, network dynamics, changing service requirements, or user mobility) [2].
3. An essential aspect of meeting SLA goals is to be able to predict performance of services, systems, and networks based on load and allocated resources. Currently, this remains an unsolved problem both for computing [1] as well as networking [3]. Current operating systems do not provide real-time guarantees, which, in combination with the added virtualization layer, significantly complicates performance prediction [1]. Moreover, there is an inherit, but unstudied, conflict between maximizing resource utilization and providing guaranteed performance [1][3].
4. Energy consumption is a big concern in cloud data center environments. As

Ch. 5: Dynamic resource management



The statically deployed SFCs may need to be redeployed over time. Such redeployments may mean placing functions in another datacenter, adapting the amount of provisioned resources, or using entirely different network domains. Dynamics that may trigger redeployment include:

- Changes in the network environment
- Resource shortage in the network or data center
- Changes in service requirements
- Changes in user locality due to mobility
- Etc.

Ch. 5: Dynamic resource management

Main open research challenges

Inter-domain dynamic virtual resource management problem remains unexplored

Real-time SFC redeployment requires efficient and scalable coordination between data center and network providers

Redeployment on several levels

- Intra-domain adaptation to quickly resolve problems locally
- End-to-end QoS monitoring to trigger service provider driven inter-domain SFC redeployment
- Interaction between these different processes

76

Existing solutions for the service placement problem across multiple (edge) cloud data centers has not considered the dynamic case, where demand and systems conditions (e.g., resource price, available network resources) change over time. Moreover, the problem of how to efficiently monitor, control, and manage such a set of distributed services has not been studied (e.g., to detect the origin of faults across domains).

References:

1. Fischer, A.; Botero, J.F.; Till Beck, M.; de Meer, H.; Hesselbach, X., *"Virtual Network Embedding: A Survey,"* IEEE Communications Surveys & Tutorials, vol.15, no.4, pp.1888–1906, 2013.

2. Bari, M.F.; Boutaba, R.; Esteves, R.; Granville, L.Z.; Podlesny, M.; Rabbani, M.G.; Qi Zhang; Zhani, M.F., *"Data Center Network Virtualization: A Survey,"* IEEE Communications Surveys & Tutorials, vol.15, no.2, pp.909–928, 2013.

Thank you!

77