

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL

FACULTY OF INFORMATICS

COMPUTER SCIENCE GRADUATE PROGRAM

# **Hybrid Synchrony Virtual Networks**

Rasha Hasan

Dissertation submitted to the Pontifical  
Catholic University of Rio Grande do Sul  
in partial fulfillment of the requirements for  
the degree of Ph. D. in Computer Science.

Advisor: Dr. Fernando Luís Dotti

Porto Alegre - Brazil  
January 2017



## Ficha Catalográfica

H344h Hasan, Rasha

Hybrid Synchrony Virtual Networks / Rasha Hasan . – 2017.

154 f.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Luís Dotti.

1. Distributed Systems. 2. Synchrony. 3. Virtual Networks. 4. Embedding. I. Dotti, Fernando Luís. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS  
com os dados fornecidos pelo(a) autor(a).





## TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "Hybrid Synchrony Virtual Networks" apresentada por Rasha Hasan como parte dos requisitos para obtenção do grau de Doutora em Ciência da Computação, aprovada em 16 de janeiro de 2017 pela Comissão Examinadora:

---

Prof. Dr. Fernando Luís Dotti  
PPGCC/PUCRS  
(Orientador)

---

Prof. Dr. Tiago Coelho Ferreto  
PPGCC/PUCRS

---

Prof. Dr. Luciano Paschoal Gaspar  
UFRGS

---

Prof. Dr. Osmar Marchi dos Santos  
UFSM

Homologada em...../...../....., conforme Ata No. .... pela Comissão Coordenadora.

---

Prof. Dr. Luiz Gustavo Leão Fernandes  
Coordenador.



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ





*I dedicate my work to Damascus  
I left you to see the world ...  
Then I discovered that, you are the world.*

*Rasha*



---

# Acknowledgment

I would like to thank my supervisor, Dr. Fernando Luís Dotti for the support and orientation he offered me all through my research.

I thank PUCRS staff, lecturers and colleagues for their help, cooperation, and smiles.

My mother and father, thank you for the education and care you gave me, since my early age till today. This PhD is yours. may Allah help me to match your expectations.

My beloved spiritual teacher, thank you for brightening my mind and heart. Your company is the biggest grace that Allah granted me.

Fernando, my beloved husband, thank you for all the support you offered me all through my PhD path.

My beloved daughters Aya and Nour. When I look at you I realize that you are my biggest achievement. Alhamdu li Allah. I am proud of you.

My sister Natasha, without you, this PhD might have become impossible. May Allah elevate your rank in heaven.

My sister Sacha, I missed sharing with you the ups and downs of my PhD path, just the way we shared all life together.

My brother Shadi, I missed a beautiful phase of your life when I traveled to follow my study. But I traveled carrying my great love to you.

My teachers Rajaa Kharsa, Kamal Jabr, Adib Manfoush, thank you for the orientation you offered me at the important phases of my educational life.

The first and last thanking is to You Allah. You always give me more than I deserve.



---

# Resumo

**N**as últimas três décadas de pesquisa em Sistemas Distribuídos (SDs), um aspecto central discutido é o de sincronia. Com um sistema assíncrono, não fazemos suposições sobre velocidades de execução de processos e / ou atrasos de entrega de mensagens; Com um sistema síncrono, fazemos suposições sobre esses parâmetros [Sch93b]. Sincronismo em SDs impacta diretamente a complexidade e funcionalidade de algoritmos tolerantes a falhas. Uma infra-estrutura síncrona contribui para o desenvolvimento de sistemas mais simples e fiáveis, mas tal infra-estrutura é muito cara e às vezes nem sequer viável de implementar. Uma infra-estrutura totalmente assíncrona é mais realista, mas alguns problemas foram mostrados como insolúveis em tal ambiente através do resultado de impossibilidade por Fischer, Lynch e Paterson [FLP85]. As limitações tanto em ambientes totalmente síncronos como totalmente assíncronos levaram ao desenvolvimento de sistemas distribuídos como sincronia parcial [CF99, Ver06].

Em um estudo de funcionalidade de sistemas distribuídos síncronos parciais e de propriedades de Redes Virtuais (RVs), descobrimos que existem vários desafios para este tipo de sistemas que podem ser resolvidos com RVs devido às propriedades que a virtualização traz. Por exemplo *a) partilha de recursos* fornecida por RVs permite diminuir o custo ao partilhar a parte síncrona da infra-estrutura física, *b) isolamento* fornecido por a natureza da RVs, isso pode beneficiar os SDs coexistentes na mesma infra-estrutura física que exigem certo nível de isolamento, *c) resiliência* garantido através do processo de alocação de recursos de Redes Virtuais, isso permite alocar recursos de reposição ao lado dos primários para redes virtuais que exigem garantias de disponibilidade, por exemplo, SDs tolerantes a falhas. Em nosso trabalho, argumentamos que as RVs e um adequado processo de alocação de recursos das RVs oferecem um ambiente adequado para executar aplicativos distribuídos com sincronia parcial. Isto levou à abstração de um novo tipo de RVs: *As Redes Virtuais com sincronia híbrida (RVSHs)*.

Nesta tese, apresentamos a idéia geral das Redes Virtuais com sincronia híbrida motivado pelos SDs com sincronia híbrida, e dividimos nosso trabalho em duas partes: *a) Espaço-RVSHs* propostos pelo SDs com sincronia híbrida em espaço, e *b) Tempo-RVSHs* propostos

pelo SDs com sincronia híbrida em tempo. No SDs com sincronia híbrida em espaço, a infra-estrutura é composta de subconjuntos de componentes síncronos e assíncronos, e cada um desses subconjuntos mantém seu status de sincronia através do tempo (i.e., os subconjuntos síncronos permanecem síncronos e os assíncronos permanecem assíncronos). No SDs com sincronia híbrida em tempo, a infra-estrutura é composta de subconjuntos de nós e laços que podem alternar seu status de sincronia através do tempo (i.e., os componentes se comportam de forma síncrona durante os intervalos de tempo e de forma assíncrona durante outros intervalos de tempo).

As principais contribuições desta tese são: *a)* caracterizam os RVSHs em seus dois tipos Espaço-RVSHs e Tempo-RVSHs para refletir tanto a natureza de sincronia em espaço e em tempo; *b)* propor uma estrutura adequada para o processo de alocação de recursos para ambos Espaço-RVSHs e Tempo-RVSHs, e *c)* fornecer uma avaliação dos modelos propostos para RVSHs.

**Palavras-chave :** Sistemas distribuídos, Sincronia, Redes virtuais, alocação de recursos.

---

# Abstract

In the last three decades of research in Distributed Systems (DSs), one core aspect discussed is the one of synchrony. With an asynchronous system, we make no assumptions about process execution speeds and/or message delivery delays; with a synchronous system, we do make assumptions about these parameters [Sch93b]. Synchrony in DSs impacts directly the complexity and functionality of fault-tolerant algorithms. Although a synchronous infrastructure contributes towards the development of simpler and reliable systems, yet such an infrastructure is too expensive and sometimes even not feasible to implement. On the other hand, a fully asynchronous infrastructure is more realistic, but some problems were shown to be unsolvable in such an environment through the impossibility result by Fischer, Lynch and Paterson [FLP85]. The limitations in both fully synchronous or fully asynchronous environments have led to the development of partial synchronous distributed systems [CF99, Ver06].

In a study of partial synchronous distributed systems functionality, and of Virtual Networks (VNs) properties, we found that there are several challenges for this kind of systems that can be solved with VNs due to the properties that virtualization brings. For example *a) resources sharing* provided by VNs allows decreasing the cost when sharing the synchronous portion of the physical infrastructure, *b) isolation* provided by the VNs nature can benefit the coexistent DSs on same physical infrastructure that demand certain level of isolation, *c) resilience* guaranteed through the Virtual Networks Embedding (VNE) process that allows allocating spare resources beside the primary ones for virtual networks that require availability guarantees, for example fault tolerant DSs. In our work, we argue that VNs and a suitable VN embedding process offer suitable environment for running distributed applications with partial synchrony. This has led to the abstraction of new type of VNs: *The Hybrid Synchrony Virtual Networks (HSVNs)*.

In this thesis, we introduce the general idea of Hybrid Synchrony Virtual Networks (HSVNs) motivated by the hybrid synchronous DSs, and we branch our work into two branches: *a) Space-HSVNs* addressed to spatial hybrid synchronous DSs, and *b) Time-HSVNs* addressed to the time hybrid synchronous DSs. In spatial hybrid synchronous DSs,

the hybrid synchronous physical infrastructure is composed of subsets of synchronous and asynchronous components, and each of these subsets maintains its synchrony status through time (i.e., synchronous subsets remain synchronous and asynchronous ones remain asynchronous). In time hybrid synchronous DSs, the hybrid synchronous physical infrastructure is composed of subsets of nodes and links that can alternate their synchrony status through time (i.e., the components behave synchronously during time intervals, and asynchronously during other time intervals).

The main contributions of this thesis are: *a)* characterize the HSVNs in its two types Space-HSVNs and Time-HSVNs to reflect both the synchrony space-variant and time-variant nature of DSs; *b)* propose a suitable embedding framework for both Space-HSVNs and Time-HSVNs, and *c)* provide an evaluation of the embedding models addressed to the HSVNs.

**Key words :** Distributed Systems, Synchrony, Virtual Networks, Embedding.



---

# List of Figures

2.1	Distributed system architecture . . . . .	35
2.2	Application topology with failure detector $\mathcal{P}$ [HMD13] . . . . .	39
2.3	Consensus algorithm-first round with P1 coordinator . . . . .	41
2.4	Failure detection algorithm in Cassandra: A cluster of three nodes, replication factor of 2, supposing node C goes down. Figure taken from [Hew11] . . . . .	43
2.5	WAS High-level architecture. Figure taken from [ea11] . . . . .	44
2.6	Chubby system structure. Figure taken from [Bur06] . . . . .	45
3.1	VN Management and Business Roles [SWP <sup>+</sup> 09] . . . . .	50
3.2	Virtual networks mapping . . . . .	51
3.3	Example of a virtual network in a backbone-star topology, where the four backbone nodes are connected into a complete graph [LT06] . . . . .	54
3.4	Topology remapping problem [LZW15] . . . . .	56
3.5	Comparison of Secure Virtual Node Ratio [WWG <sup>+</sup> 16] . . . . .	61
3.6	Assisting nodes and search regions [GKA <sup>+</sup> 16b] . . . . .	65
3.7	Illustrative figure for the sliding window technique . . . . .	66
3.8	Examples of generated slices of size 5 [IR11] . . . . .	67
4.1	Illustrates figure for Space-HSVNs embedding . . . . .	77
5.1	Embedding cost - Confidence interval . . . . .	86
5.2	Space-HSVNs with S-SN: the change of average value for embedding cost with the change of VNs synchrony demands . . . . .	87
5.3	Space-HSVNs with S-SN: the change of average value for embedding cost with the change of VNs size . . . . .	89
5.4	Embedding time - Confidence interval . . . . .	91

5.5	Space-HSVNs with SSN: the change of average value for embedding time with the change of VNs synchrony demands . . . . .	93
5.6	Space-HSVNs with S-SN: the change of average value for embedding time with the change of VNs size . . . . .	94
5.7	Group A - Embedding cost vs. embedding time . . . . .	95
5.8	Group B - Embedding cost vs. embedding time . . . . .	96
5.9	Embedding cost vs. embedding time for experiments B2, B3, and B4 . . . . .	97
5.10	Group C - Embedding cost vs. embedding time . . . . .	97
5.11	Unused physical nodes - Confidence interval . . . . .	99
5.12	Unused physical links - Confidence interval . . . . .	100
5.13	Group A - accumulative number of physical nodes vs. nodes load . . . . .	102
5.14	Group A - accumulative number of physical links vs. links load . . . . .	103
5.15	Group B - accumulative number of physical nodes vs. nodes load . . . . .	104
5.16	Group B - accumulative number of physical links vs. links load . . . . .	105
5.17	Group C - accumulative number of physical nodes vs. nodes load . . . . .	106
5.18	Group C - accumulative number of physical nodes vs. nodes load . . . . .	107
5.19	Percentage of synchronous nodes and links in SN for scenarios in set 1 for S-HSVN and C-HSVN . . . . .	109
5.20	CDF for nodes usage in experiment C3 . . . . .	111
5.21	CDF for links usage in experiment C3 . . . . .	111
5.22	Individual nodes load in experiment C3 . . . . .	112
5.23	Individual links load in experiment C3 . . . . .	113
5.24	Percentage of SN sync. resources: Comparing groups <i>set1</i> and <i>set2</i> . . . . .	114
5.25	Frequency of synchronous nodes vs. nodes connectivity - scenarios in set 2 . . . . .	115
5.26	Average connectivity of synchronous nodes for scenarios in set 2 . . . . .	116
5.27	Topology divergence of synchronous resources [HMOD14] . . . . .	116
6.1	Synchronous and asynchronous rounds for a virtual node or link . . . . .	121
6.2	SN synchronous frames and slots proposed in [ZQT <sup>+</sup> 11] . . . . .	121
6.3	Physical node or link synchrony frame during $T$ . . . . .	122
6.4	Block diagram for Time-HSVNs embedding phases . . . . .	123
6.5	Graph based example for Time-HSVNs embedding . . . . .	124

6.6	An illustrative scheme for the HSVNs optimization cycles . . . . .	130
7.1	Used bandwidth . . . . .	132
7.2	CDF for resource usage in experiment C2 . . . . .	134
7.3	Topology divergence of used physical resources . . . . .	135

---

# List of Tables

3.1	Taxonomy of works on virtual networks mapping . . . . .	71
4.1	List of variables definition for Space-HSVNs embedding model . . . . .	79
5.1	Space-HSVNs with S-SN scenarios parameters . . . . .	84
5.2	Space-HSVNs with S-SN: average of embedding cost . . . . .	87
5.3	Space-HSVNs with S-SN: average of embedding time . . . . .	93
5.4	Space-HSVNs with C-SN scenarios parameters set (set 1) . . . . .	108
5.5	Economy in SN synchrony resources between S-HSVN and C-HSVN- performed for scenarios set 1 . . . . .	110
5.6	Experiments parameters in set 2 . . . . .	114
6.1	List of variables definition for Space-HSVNs embedding model . . . . .	125
6.2	List of variables definition for Time-HSVNs micro mapping phase model . .	129
7.1	Experiments parameters . . . . .	132
7.2	Embedding time (in minutes) . . . . .	134
7.3	Number of mapped virtual links with different load and synchrony demands	136

---

# Abbreviations

**AN** Assisting node

**BRITE** Boston university Representative Internet Topology generator

**BW** Band Width

**CDF** Cumulative Distributed Function

**CPU** Central Processing Unit

**C-SN** Configurable Substrate Network

**DB** Data Base

**DiffServ** Differentiated Services

**DS** Distributed System

**DiffServ\_TE** Differentiated Services aware Traffic Engineering

**EDP** Edge Disjoint Path problem

**FLP** Fischer, Lynch and Paterson impossibility

## ABBREVIATIONS

---

**GB** Giga Byte

**Gbps** Giga bit per second

**GHZ** Giga Hertz

**GST** Global Stabilization Time

**HSDS** Hybrid Synchronous Distributed Systems

**HSVN** Hybrid Synchrony Virtual Network

**IP** Internet Protocol

**InP** Infrastructure Provider

**Mbps** Mega bit per second

**MILP** Mixed Integer Linear Program

**MIP** Mixed Integer Program

**MPLS** Multi Protocol Label Switching

**MVN** Multicast Virtual Network

**NP** Nondeterministic Polynomial

**NRF** Nodes Ranking and Filtering

## ABBREVIATIONS

---

**OF** Objective Function

**ORP** Opportunistic Redundancy Pooling

**OS** Operating System

**P2P** Peer-to-Peer

**QoS** Quality of Service

**RAM** Random Access Memory

**SLA** Service Level Agreement

**SP** Service Provider

**Space-HSVN** Spatial Hybrid Synchrony Virtual Network

**SN** Substrate Network

**S-SN** Settled Substrate Network

**TE** Traffic Engineering

**Time-HSVN** Timely Hybrid Synchrony Virtual Network

**UFP** Unsplittable Flow Problem

**VInf** Virtual Infrastructure

## ABBREVIATIONS

---

**VN** Virtual Network

**VNA** Virtual Network Assignment

**VNO** Virtual Network Operator

**VNP** Virtual Network Provider

**VoIP** Voice over Internet Protocol

**WAS** Windows Azure Storage

**ZIMPL** Zuse Institute Mathematical Programming Language



---

# Contents

<b>1</b>	<b>General Introduction</b>	<b>30</b>
<b>2</b>	<b>Work motivation: Hybrid synchrony in distributed systems</b>	<b>35</b>
2.1	Distributed systems . . . . .	35
2.2	Synchrony in distributed systems . . . . .	36
2.3	Hybrid synchrony in distributed systems . . . . .	37
2.3.1	Hybrid Synchrony in space . . . . .	37
2.3.2	Hybrid Synchrony in time . . . . .	39
2.4	Hybrid synchrony in practice . . . . .	41
2.4.1	Apache Cassandra . . . . .	42
2.4.2	Windows Azure . . . . .	43
2.4.3	Chubby lock service . . . . .	45
2.4.4	Observations . . . . .	46
2.5	Hybrid synchronous DSs and virtualization . . . . .	46
<b>3</b>	<b>Related Work: Network Virtualization</b>	<b>49</b>
3.1	Network Virtualization . . . . .	49
3.2	Virtual Networks properties . . . . .	50
3.3	Resources allocation in Virtual Networks . . . . .	51
3.4	Literature review on VNs Embedding . . . . .	53
3.4.1	Topology requirements . . . . .	53
3.4.2	BW and CPU requirements . . . . .	56
3.4.3	Security requirements . . . . .	59
3.4.4	Resilience requirements . . . . .	60
3.4.5	Delay requirements . . . . .	65

3.4.6	Miscellaneous requirements . . . . .	69
3.4.7	Synchrony requirements . . . . .	69
3.5	Review on the VNE difficulties . . . . .	71
<b>4</b>	<b>Space-HSVNs embedding</b>	<b>74</b>
4.1	Space-HSVNs: definition . . . . .	74
4.2	Work positioning in the literature . . . . .	74
4.3	The Substrate Network for space-HSVNs . . . . .	75
4.4	Graph based example for Space-HSVNs embedding . . . . .	76
4.5	Space-HSVNs embedding model on S-SN . . . . .	77
4.5.1	Variables definition . . . . .	77
4.5.2	Embedding model . . . . .	78
4.6	Space-HSVNs embedding model on C-SN . . . . .	81
4.6.1	Variables definition . . . . .	81
4.6.2	Embedding model . . . . .	82
<b>5</b>	<b>Performance Evaluation for Space-HSVN</b>	<b>83</b>
5.1	Space HSVNs over a settled SN . . . . .	83
5.1.1	Workload and tools . . . . .	83
5.1.2	Results . . . . .	85
5.1.2.1	Embedding cost . . . . .	86
5.1.2.2	Embedding time . . . . .	91
5.1.2.3	Embedding cost vs. embedding time . . . . .	95
5.1.2.4	Physical resources load . . . . .	98
5.2	Space HSVNs over a configurable SN . . . . .	106
5.2.1	Work load and tools . . . . .	107
5.2.2	Results . . . . .	109
5.2.2.1	Economy in the embedding cost . . . . .	109
5.2.2.2	Physical resources load . . . . .	110
5.2.2.3	Privilege of synchronous nodes . . . . .	114
5.2.2.4	Topological study . . . . .	115
<b>6</b>	<b>Time-HSVNs embedding</b>	<b>118</b>

6.1	Time-HSVNs: definition . . . . .	118
6.2	Work positioning in the literature . . . . .	119
6.3	Time-HSVNs characterization . . . . .	120
6.4	SN design . . . . .	120
6.5	Time-HSVNs Embedding . . . . .	122
6.5.1	The Macro Mapping Phase . . . . .	124
6.5.1.1	Variables definition . . . . .	124
6.5.1.2	The Macro mathematical model . . . . .	125
6.5.2	The Micro mapping phase . . . . .	127
6.5.2.1	Revision on the Cutting Stock Problem (CSP) . . . . .	127
6.5.2.2	The Micro mathematical model . . . . .	128
<b>7</b>	<b>Performance Evaluation for Time-HSVN</b>	<b>131</b>
7.1	Workload and tools . . . . .	131
7.2	Results . . . . .	132
7.2.1	Embedding cost . . . . .	132
7.2.2	Physical resources load . . . . .	133
7.2.3	Embedding time . . . . .	134
7.2.4	Topological study . . . . .	135
7.2.5	Micro mapping model efficiency . . . . .	135
<b>8</b>	<b>Conclusion</b>	<b>137</b>
8.1	The thesis contributions . . . . .	138
8.2	Achieved results . . . . .	139
8.2.1	Space-HSVNs over a settled SN (S-SN) . . . . .	139
8.2.2	Space-HSVNs over a configurable SN (C-SN) . . . . .	140
8.2.3	Time-HSVNs . . . . .	141
8.3	Work application . . . . .	142
8.4	Work generalization . . . . .	143
8.5	Work limitations . . . . .	143
8.6	Future work . . . . .	144
	<b>Bibliography</b>	<b>145</b>





A distributed System (DS) consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software that enables computers to coordinate their activities and to share the resources of the system hardware, software, and data [Sch93a]. Users of distributed systems should perceive a single, integrated computing facility even though it may be implemented by many computers in different locations. This is in contrast to a network, where the user is aware that there are several machines whose locations, storage replications, load balancing, and functionality are not transparent. Benefits of distributed systems include bridging geographic distances, improving performance and availability, maintaining autonomy, reducing cost, and allowing for interaction.

In the last three decades of research in Distributed Systems (DSs), one core aspect discussed is the one of synchrony. With an asynchronous system, we make no assumptions about process execution speeds and/or message delivery delays; with a synchronous system, we do make assumptions about these parameters [Sch93b]. In particular, in a synchronous system, the relative speeds of processes, as well the delays associated with communication channels are assumed to be bounded. The research on DSs has longly touched problems that base on the synchrony property of the system environment, for example the consensus problem [DLS88] where synchrony ensures progress of several distributed algorithms; another example is failure detection [Gar01].

Synchrony in DSs impacts directly the complexity and functionality of fault-tolerant algorithms. Although a synchronous infrastructure contributes towards the development of simpler and reliable systems; yet such an infrastructure is too expensive and sometimes even not feasible to implement. On the other hand; a fully asynchronous infrastructure is more realistic, but fundamental agreement problems showed to be unsolvable in such an environment, such as the impossibility result by Fischer, Lynch and Paterson [FLP85]. The limitations in both fully synchronous or fully asynchronous systems have led researchers to the development of partial synchronous distributed systems, which we call in this thesis, as well, by Hybrid Synchronous Distributed Systems.

There are some applications that may benefit from the hybrid synchronous assumptions, for example, Apache Cassandra, Windows Azure, and Chubby lock service. These applications do not force the use of hybrid synchronous environment, although they need, rather they run on asynchronous environments supported with algorithms and/or protocols (e.g., PAXOS protocol) to allow progress but without guarantees. If these application run on top of hybrid synchronous infrastructure then the progress will be guaranteed by the provisioning of elements designed to respect time upper bounds. The problem is that fully synchronous, or partially synchronous, environment is expensive to build, complex to configure, and difficult to control. This makes the infrastructure providers escape to asynchronous environments strengthened by algorithms and/or protocols with time-out specifications.

Virtual Networks (VNs) have attracted considerable attention in the last years, both as an experimental environment to evaluate new protocols, as well as a technology to be integrated in the current network architectures [CB09]. As can be seen in the literature, the diversity of applications pose different requirements on their supporting VNs, e.g., topology, security, and resilience requirements. In this context, the process of *resources allocation* (called as well *embedding* or *mapping*) is a key aspect that (i) defines how resources of a physical network (also called Substrate Network - SN) are used to support VNs, and (ii) assumes several variants according to the kinds of applications and respective VNs demands.

In a study of partial synchronous distributed systems functionality, and of Virtual Networks (VNs) properties, we found that there are several challenges for this kind of systems that can be solved with VNs due to the properties that virtualization brings, for example:

- Network virtualization is defined by the decoupling of the roles of the traditional Internet Service Providers (ISPs) into two independent entities [TT05]: infrastructure providers, who manage the physical infrastructure, and service providers, who create virtual networks by aggregating resources from multiple infrastructure providers and offer end-to-end services. The design of synchronous components in DSs requires fundamental handling mechanisms [RSK<sup>+</sup>00, RSK<sup>+</sup>01]. Considering a new architecture of the DSs based on VNs provides a new business model that allows sharing tasks: (i) the synchronous resources design is assigned to the SN provider, and (ii) the resources allocation process is assigned to the VN provider. Delegating the design of the synchronous resources to the SN provider and the embedding process to the VN provider may result in dividing the complexity into more tractable parts.
- Heterogeneity in the context of network virtualization comes mainly from two fronts [CB10]: first, heterogeneity of the underlying networking technologies (e.g., optical, wireless, and sensor); second, each end-to-end VN, created on top of that heterogeneous combination of underlying networks, can also be heterogeneous. Considering a new architecture of DSs based on VNs exhibits the system heterogeneity in the hybrid kind

of physical resources required to be synchronous and asynchronous, which allows using the VNs framework to manage hybrid environment.

- The building cost of synchronous resources in DSs is considerably high when compared to the asynchronous resources. VNs allow sharing the synchronous portion of the physical infrastructure between several applications, which will result in reducing the overall cost.
- The VNs Embedding process (VNE) allows allocating resources according to a pattern that guarantees resilience. In fault-tolerant DSs, system availability is a requirement. In other words, the occurrence of faults should be masked allowing the continuity of the system functioning. This can be tackled easily by adopting the VNs embedding framework, which allows allocating spare resources beside the primary ones for virtual networks that require availability guarantees.

In our work, we argue that VNs and a suitable VN embedding process offer suitable environment for running distributed applications with partial synchrony. This has led to the abstraction of new type of virtual networks that we name *The Hybrid Synchrony Virtual Networks (HSVNs)*. They are virtual networks that have subsets of nodes and links that obey time bounds for processing and communication. Although HSVNs can run on a fully synchronous SN, this decision would result in an excess of an unneeded cost, since even asynchronous virtual nodes and links will be mapped on synchronous physical ones. We argue that a hybrid synchronous SN, combined with a suitable embedding process, is capable to answer the synchrony requirements in an economic manner. Furthermore, revising the literature on the topic of VNs embedding, we have not found a mapping solution aware of the synchrony parameter so it can be adopted for the HSVN resources allocation, this motivated us to develop a mathematical model for the HSVNs embedding process.

### **Thesis contributions**

The main contributions of this thesis are:

1. Provide a state of the art for research on VNs embedding, classifying the works based on applications requirements.
2. Introduce the general idea of HSVNs motivated by the hybrid synchronous DSs. We branch our work into two paths:
  - Space-HSVNs: addressed for hybrid synchronous DSs in space, where the application synchronous and asynchronous components maintain their synchrony status during the system execution. With Space-HSVNs, we consider two types of Substrate Network (SN): (a) Settled Substrate network (S-SN), where physical nodes



and links are designed to behave synchronously or asynchronously independently of the VNs synchrony demands, and *(b)* Configurable Substrate Network (C-SN), where the physical nodes and links are configured to behave synchronously or asynchronously dependently on the VNs synchrony demands.

- Time-HSVNs: addressed for hybrid synchronous DSs in time, where the application components eventually alternate their synchrony status during the system execution.
3. Develop suitable embedding models for: *a)* Space-HSVNs over S-SN, *b)* Space-HSVNs over C-SN, and *c)* Time-HSVNs.
  4. Evaluate the performance of the proposed embedding models through simulating different scenarios.

We note down that the proposed embedding framework for Space-HSVNs is able to answer Time-HSVNs. But this would result in an excess of cost. Considering the synchrony time variant nature in Time-HSVNs would result in further sparing of the use of physical synchronous resources.

### Thesis organization

In Chapter 2, we start by building a background about some concepts with hybrid synchronous distributed systems, and at the end on Chapter 2, we point some limitations in hybrid synchronous DSs, and we propose Virtual Networks (VNs) as a possible solution due to properties guaranteed by the virtualization [CB10].

In Chapter 3, we provide a background about Virtual Networks (VNs), definition, properties, and we revise the literature on the topic of VN resources allocation, which is named as *mapping* or *embedding* problem in the literature as well. We classify the works on VNs embedding according to applications' constraints (e.g., topology, security, and resilience), and we note the absence of an embedding solution in the literature that considers the synchrony property in applications, which we need for our work. This gap led us to the development of an embedding framework that handles applications with hybrid synchrony constraints. Merging DSs with VNs has resulted in a new architecture for DSs with hybrid synchrony based on VNs, the new architecture we name: *the Hybrid Synchrony Virtual Networks* abbreviated to *HSVNs*. With the two types of hybrid synchrony (i.e., in Space and in Time), we branch our work into two branches detailed in Chapter 4 and Chapter 6.

In Chapter 4 we propose the Hybrid Synchronous Virtual Networks in Space, we name them the *Space-HSVNs*, we propose a suitable embedding model for two types of Substrate Network (SN): *i)* Settled SN and *ii)* configurable SN.

In Chapter 5 we evaluate the performance of the proposed embedding model of Space-HSVNs basically regarding the embedding cost, optimization time, and resources load. More-

---

over, we provide a study over the topology of the subnetworks composed of the used physical resources on the SN.

In Chapter 6 we propose the Hybrid Synchronous Virtual Networks in Time, we name them the *Time-HSVNs*, as well, we propose a suitable embedding model for the Time-HSVNs and in Chapter 7 we evaluate it basically for the same formerly mentioned metrics.

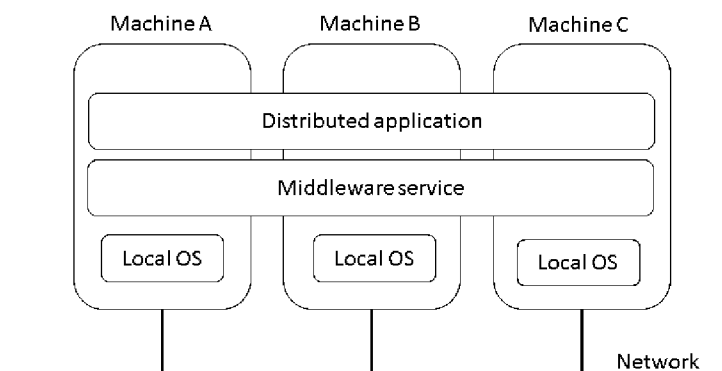
In Chapter 8 we conclude the work, and we highlight the work applications, generalization, limitations, and future steps.

## Work motivation: Hybrid synchrony in distributed systems

This chapter provides a background about distributed systems, highlighting mainly one of its important characteristics, the synchrony, and summarizes the motivation and the approaches of hybrid synchrony distributed systems from the literature.

### 2.1 Distributed systems

A *distributed system* consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility [Sch93a]. Figure 2.1 depicts simple diagram for distributed system architecture.



*Figure 2.1* — Distributed system architecture

Over the years, many interesting solutions based on distributed systems were proposed and a variety of applications emerged. For instance, by using of replication techniques data and services become accessible even in occurrence of failures. Overall performance and scalability can be increased through load balancing among replicas. Security and safety issues can also be handled by fault tolerant solutions using distributed processes. In short, distributed systems offer higher abstraction level to system developers and also might increase system availability, performance and security.

However, the development of distributed systems is not an easy task. Heterogeneity of system components, unpredictable environments and the concurrency inherent of this kind of systems are major challenges that must be addressed by distributed system developers.

## 2.2 Synchrony in distributed systems

The design of DSs is strongly dependent on the assumptions about the environment where they execute. For instance, different assumptions about process execution speeds and message delivery delays would require specific design decisions. In this sense, an important aspect to consider when developing a distributed system is the synchrony level offered by the underlying infrastructure. In an asynchronous system, no assumption about process execution speed and/or message delivery delays is made. Conversely, in a synchronous system, relative processing speed or the message delays are bounded [Sch93a].

The synchrony level impacts on the reliability and difficulty to build the system. Assuming that, underlying infrastructures behaving asynchronously showed to be realistic to a wide range of applications. Furthermore, it is the weakest model in terms of synchrony. That means an algorithm that works in an asynchronous model also works in other models with stronger synchrony assumptions. The opposite is not valid, *i.e.* an algorithm that works in a synchronous model is prone to incorrect behavior if timing constraints are violated. Although asynchronous models are very attractive, with them it is impossible to distinguish a crashed process from an arbitrarily slow process, in which some messages delivery are delayed [CT96]. As a consequence, many important problems of fault-tolerant computing are not solvable under the asynchronous assumption. For example, the FLP impossibility result by Fischer, Lynch and Paterson shows that consensus cannot be solved deterministically in asynchronous systems where at least one process may crash [FLP85].

By asserting that a system is synchronous, system developers can rely on the timely behavior of the components. This, in turn, enables one to employ simpler algorithms than those required to solve the same problem in an asynchronous system [Sch93a]. For instance, processes can perfectly distinguish faulty from slow processes. However, building synchronous systems requires infrastructures composed exclusively by timely components, which could be very expensive or even infeasible. Moreover, synchronous systems demand a priori knowledge on time bounds, which might not fit with dynamic systems.

The drawback of each of the two extremes (*i.e.*, the synchronous systems and the asynchronous systems) led to hybridize them both. This gave birth to new class of systems that is known in the literature as *partial synchrony systems* [CF99, Ver06].

We assume the existence of certain mechanisms that guarantee building physical network elements (nodes and links) that behave synchronously. These mechanisms can be related to

the type of physical materials used, or to the procedures followed for configuring them, such as admission control and Quality of Service policies. The exact mechanisms for building synchronous resources is out of the scope of our work, but we assume their existence. We refer the reader to [KR13, GB95, BFY<sup>+</sup>00, BBC<sup>+</sup>98] to learn more details about possible policies for building synchronous links, and [Liu00, Her04, LSS87] for configuring nodes with real-time tasks.

## 2.3 Hybrid synchrony in distributed systems

Hybrid models assume intermediate levels of synchrony, stronger than asynchronous and weaker than synchronous. Such systems are also called as partial synchronous. In this thesis, we will use the terms *hybrid synchrony* and *partial synchrony* to refer to the same concept. Hybrid synchrony was proposed as a solution for some drawbacks of the homogeneous synchrony models (i.e., fully synchronous or fully asynchronous). In the next two subsections, we introduce the two types of hybrid synchrony as in the literature: (a) hybrid synchrony in space, and (b) hybrid synchrony in time. We will define the hybrid synchrony in space and in time in the light of classical problems adopted from the literature.

### 2.3.1 Hybrid Synchrony in space

In [Ver06], Veríssimo presented the wormhole model, that exploits the space dimension to provide hybrid synchrony. This means that timely guarantees of system components may be different. For instance, one part of a system would behave synchronously, while other part would be fully asynchronous.

Once behaviors caused by faults and arbitrary delays are expected in the conventional infrastructures, hybrid models become a good option to improve the development of fault-tolerant applications. By enforcing small parts of the system to behave synchronously while other parts are asynchronous, stronger properties provided by synchronous parts can be used by the system as a whole. For this reason, hybrid systems overcome limitations of the homogeneous systems.

#### **Example - Building a perfect failure detector on spatial hybrid synchronous environment**

Failure detectors have attracted interest in the development of reliable DSs, since consensus and related problems (e.g., atomic broadcast [CT96]) can be solved with it. The failure detection approach can also be adapted to solve other relevant problems, such as predicate detection [GK00] and election [MIMF00].

Failure detectors are used to detect faulty processes in a group of processes, and they are

defined in terms of abstract properties, namely *accuracy* and *completeness*. Strong accuracy implies that no process is suspect before it crashes, while weak accuracy means that some correct process is never suspected. Strong completeness implies that eventually every process that crashes is permanently suspected by every correct process, while weak completeness means eventually every process that crashes is permanently suspect by some correct process. A failure detector that satisfies *strong accuracy* and *strong completeness* properties is a *perfect failure detector* ( $\mathcal{P}$ ) [CT96]. It means it never makes mistakes (suspects erroneously) and, eventually detects every crash.

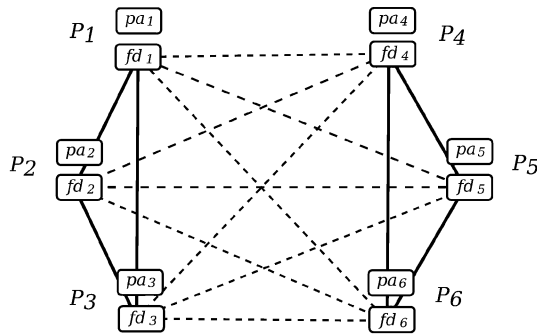
A perfect failure detector  $\mathcal{P}$  can be implemented on top of a fully synchronous environments. The problem is that implementing  $\mathcal{P}$  in fully synchronous environments depends on the existence of an underlying infrastructure with timely guarantees for all its components, which is too expensive and sometimes even infeasible. On the other hand, implementing a perfect failure detector on top of a fully asynchronous infrastructure is impossible.

Macêdo *et al.* [dAMG09] propose an implementation of a failure detector  $\mathcal{P}$  that runs on hybrid synchronous environments and provides both strong accuracy and strong completeness [CT96] making it a perfect failure detector. They assume the underlying system has synchronous processes, some channels behave synchronously and others asynchronously.

Basically, each module  $fd_i$  periodically asks to processes  $p_j$  if they are alive. Upon receiving a message “are you alive”, every correct process replies to the sender with a “I’m alive” message. Upon receiving the replying message,  $fd_i$  knows the process  $p_j$  is up. However, if a timeout expires, it means that no answer from  $p_j$  was received in the last  $\tau$  time units. If the channel connecting processes  $fd_i$  to  $fd_j$  is synchronous, then it is known that the process  $p_j$  has failed. Process  $p_j$  is added to the faulty list in  $p_i$ , and a notification informing the detection is sent to all other processes. Otherwise, if the channel is asynchronous, there is no way to detect if the process  $p_j$  has failed or the reply message is delayed.

We illustrate a failure detector  $\mathcal{P}$  running in a hybrid synchronous environment in Figure 2.2. It shows a hypothetical topology for an application composed by six processes. All processes are hosted in synchronous nodes, and they communicate with each other through payload channels ( $pa_i$ ). Further, a failure detector module  $fd_i$  is attached to each process  $P_i$ . Connection between failure detectors modules in a synchronous partition is done by synchronous channels (solid lines in the figure). Connection between  $fd$  modules in different partitions can be asynchronous (dotted lines). In order to improve legibility, payload channels  $pa_i$  were omitted in the figure. In this example, the payload channels should be represented by a complete graph connecting every pair of processes.

Although not all failure detectors are in the same synchronous partition, the  $\mathcal{P}$  implementation allows every application process to benefit from a perfect detection. Even in cases in which not all  $fd_i$  modules belong to a synchronous partition, it is possible to take advantage



**Figure 2.2** — Application topology with failure detector  $\mathcal{P}$  [HMD13]

of the existing synchrony, provided that some subgraphs are synchronous. In such cases, assumptions from weaker failure detectors (e.g.,  $\diamond\mathcal{P}$ ,  $\diamond\mathcal{S}$  [CT96]) would be ensured and still useful for the applications.

Another interesting aspect of the hybrid synchronous system is that application workload is totally independent of the failure detector modules. Application processes can communicate through asynchronous channels and still benefit from stronger properties provided by the failure detector service.

### 2.3.2 Hybrid Synchrony in time

Also named *timed asynchronous model* or, for short, the *timed model*. This model was first proposed by Cristian and Fetzer [CF99], where the system alternates between synchronous and asynchronous behavior. More specifically, according to [DLS88] partially synchronous systems can alternate between synchronous and asynchronous behavior, being hybrid in *time*. For each execution, there is a time after which the upper bound  $\delta$  is respected by the system. This time is called *Global Stabilization Time (GST)*. Since the upper bound cannot hold forever, it is accepted that it holds just for a limited time  $\Delta_s$ . In practical terms,  $\Delta_s$  is the time needed for consensus to make progress or to be reached. We call these *timely hybrid synchronous systems*.

Progress assumptions are similar to the *global stabilization* requirement of [DLS88] which postulates that eventually a system must *permanently* stabilize, in the sense that there must exist a time beyond which all messages and all non-crashed processes become timely. However, progress assumption only require that infinitely often there exists a majority set of processes that for a certain minimum amount of time are timely and can communicate with each other in a timely manner [Gar01]. In other words, algorithms for this model make progress when a system has just enough synchrony to make decisions.

**Example - solving the consensus problem in timely hybrid synchronous environment**

One of the fundamental problems in distributed systems are *agreement problems*. They occur when a set of processes must make a consistent decision. For example, if a database is replicated, all the processes have to agree whether or not to abort certain transactions. Many agreement problems have been condensed to one basic problem called *consensus*. In general, only system models where it is possible to solve consensus are really useful for fault tolerance [Gar01].

An algorithm that solves the consensus problem must guarantee three properties [Gar01]:

- Agreement: no two processes decide on two different values;
- Termination: every correct process eventually decides;
- Validity: the decided value must have been proposed by some process.

The consensus problem can be stated as follows [GdAMR07]: each process proposes a value, and has to decide a value, unless it crashes (termination), such that there is a single decided value (uniform agreement), and that value is a proposed value (validity).

The consensus problem is impossible to solve in asynchronous distributed systems prone to process crashes [GdAMR07]. This impossibility has motivated researchers to develop distributed computing models stronger than the asynchronous models, but weaker than the synchronous models, where the consensus problem can be solved.

Dwork et al. in [DLS88] proposed an algorithm for solving the consensus problem in timely hybrid synchronous environment. The algorithm consists of number of rounds that matches the number of the processes of which the system consists. During each round, one process becomes *coordinator*. Each round consists of four phases.

Consider an asynchronous system with three processes  $P_1$ ,  $P_2$  and  $P_3$  and at most one process may crash. At any time, each process has an *estimate* of the value it thinks will become the decision value.

In the algorithm first round, process  $P_1$  becomes the coordinator. Figure 2.3 illustrates the algorithm first round with  $P_1$  a coordinator. This round works through the following four phases:

- Phase 1: every process sends its actual estimate to the coordinator;
- Phase 2:  $P_1$  waits for the estimate of other processes, and then it chooses a new estimate for itself from the set of received estimates, including its own original estimate. The value with the highest frequency is the one chosen as the coordinator new estimate;
- Phase 3: every process, other than the coordinator, receives the new estimate and sends back a positive acknowledgment to the coordinator;



- Phase 4: if the coordinator receives at least one positive acknowledgment (since at maximum one process can crash in this system), then it can decide on the current estimate and tell the other processes to do the same.

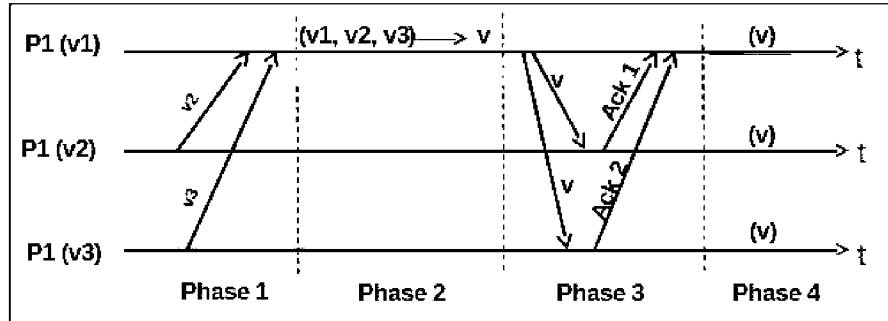


Figure 2.3 — Consensus algorithm—first round with P1 coordinator

The timely hybrid synchrony helps in guaranteeing the progress of the algorithm. For example, we notice that in phase 3, each process waits for the new estimate from the coordinator. If the coordinator crashed, then all the processes will wait infinitely. The asynchrony in this phase needs to be tackled by using some form of timeout to bound the waiting time. If a process times out, it sends a negative acknowledgment to the coordinator and switches to the next round of the algorithm when a new process becomes a coordinator. Consider that the coordinator had not crashed, then it will receive no positive acknowledgment in phase 4, but instead receives one or two negative ones, in this case, it also switches to the next round. This means the algorithm keeps safety even if progress is not assured. To assure progress, this algorithm has to consider the assumption of partial synchrony. In this specific case, it is enough that the subnetwork that consists of the three processes P1, P2, P3 and the two links (P1,P2) and (P1,P3) eventually turns to become synchronous. This will make the system behave within acceptable delays for enough time to make decision. This will guarantee the progress of the algorithm. Since the aforementioned subnetwork starts as asynchronous, then eventually turns to be synchronous, then back to asynchrony; then we are speaking about a system with hybrid synchrony in *time*.

## 2.4 Hybrid synchrony in practice

The research on DSs has longly touched problems that base on the hybrid synchrony property of the system environment, for example (as previously detailed) the consensus problem [DLS88] where a minimum level of synchrony ensures the progress of several distributed algorithms; another example is the perfect failure detector [dAMG09] built on a hybrid synchrony environment providing strong guarantees for accuracy and completeness [CT96] (i.e., *strong accuracy* does not assume processes will be erroneously suspected to be crashed; and

*strong completeness* assumes that at some time every failed process will be detected by every correct processes).

In fact, many applications benefit from consensus and failure detection as building blocks in the distributed algorithms. Bellow we mention some examples of such applications.

### 2.4.1 Apache Cassandra

Apache Cassandra is a massively scalable open source NoSQL database [Hew11]. This application was addressed for managing large amounts of data across multiple data centers and the cloud. Delivering continuous availability, scalability, and operational simplicity with fast response times.

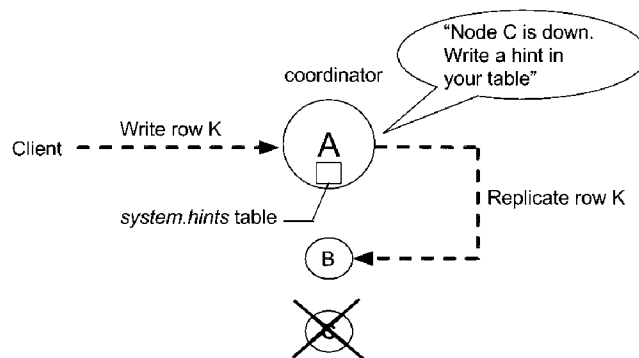
**Data distribution and replication consensus-** Apache Cassandra creates replicas of the data base (DB) stored by the client on different clusters of the data center. Any change on one replica of the DB should be transparent to the other replicas and eventually considered by all the replicas to have continuous availability. This agreement between the replicas is realized though consensus protocol *PAXOS* [Lam01] which is distinguished as a building block in the Cassandra's *Lightweight transactions* algorithms. Paxos consensus protocol allows Cassandra to support atomic, isolated, and durable transactions with eventual/tunable consistency that lets the user decide how strong or eventual they want each transaction's consistency to be.

**Failure detection and recovery-** Apache Cassandra considers a tunable failure detection method for locally determining from gossip state and history if another node in the system is up or down. Cassandra uses this information to avoid routing client requests to unreachable nodes whenever possible.

Node failures can result from various causes such as hardware failures and network outages. Nodes outage does not result in an automatically permanent removal of the node from the architecture because a node might have a possible recovery. Other nodes periodically try to re-establish contact with failed nodes to see if they are back up. To permanently change a node's membership in a cluster, administrators must explicitly add or remove nodes from a Cassandra cluster. When a node comes back online after an outage, it may have missed writes for the replica data it maintains. Once the failure detector marks a node as down, missed writes are stored by other replicas for a period of time (configurable upper bound) providing *Hinted Handoff* is enabled. If a node is down for longer than this upper bound, hints are no longer saved.

During a write operation, when *Hinted Handoff* is enabled and consistency can be met, the coordinator stores a hint about dead replicas in the local *system.hints* table. A hint indicates that a write needs to be replayed to one or more unavailable nodes. By default, hints are saved for three hours (by default, and can be configured) after a replica fails because

if the replica is down longer than that, it is likely permanently dead. After a node discovers from gossip that a node for which it holds hints has recovered, the node sends the data row corresponding to each hint to the target. Additionally, the node checks every ten minutes (configurable period) for any hints for writes that timed out during an outage too brief for the failure detector to notice through gossip. For example, in a cluster of three nodes, Figure 2.4, A (the coordinator), B, and C, each row is stored on two nodes in a key space having a replication factor of 2. Suppose node C goes down. The client writes row K to node A. The coordinator, replicates row K to node B, and writes the hint for downed node C to node A. When node C comes back up, node A reacts to the hint by forwarding the data to node C.



**Figure 2.4** — Failure detection algorithm in Cassandra: A cluster of three nodes, replication factor of 2, supposing node C goes down. Figure taken from [Hew11]

## 2.4.2 Windows Azure

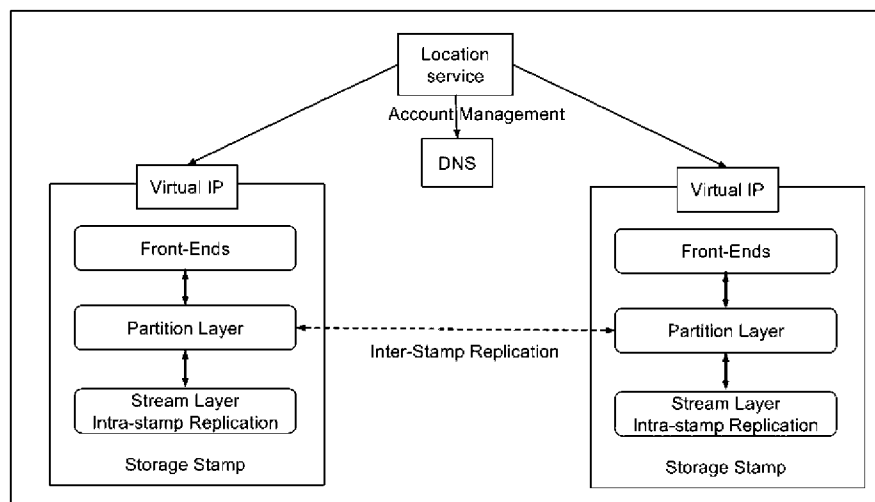
Windows Azure Storage (WAS) is a cloud storage system that provides customers the ability to store seemingly limitless amounts of data for any duration of time [ea11]. Differently from Cassandra; in WAS data is stored durably using both local and geographic replication to facilitate disaster recovery. WAS is used inside Microsoft for applications such as social networking search, serving video, music and game content, managing medical records, and more. In addition, there are thousands of customers outside Microsoft using WAS, and anyone can sign up over the Internet to use the system.

**Strong consistency-** Many customers want strong consistency [HW90] especially enterprise customers moving their line of business applications to the cloud. For this, WAS provides three properties that are claimed to be difficult to achieve at the same time [Bre00]: strong consistency, high availability, and partition tolerance.

The WAS production system consists of Storage Stamps, where a storage stamp is a cluster of  $N$  racks of storage nodes, and each rack is built out as a separate fault domain with redundant networking and power.

WAS has two replication engines, see Figure 2.5:

1. Intra-Stamp Replication: This system provides *synchronous* replication and is focused on making sure all the data written into a stamp is kept durable within that stamp. It keeps enough replicas of the data across different nodes in different fault domains to keep data durable within the stamp in the face of disk, node, and rack failures. Intra-stamp replication is done on the critical path of the customer's write requests.
2. Inter-Stamp Replication: This system provides *asynchronous* replication and is focused on replicating data across stamps. Inter-stamp replication is done in the background and is off the critical path of the customer's request. This replication is at the object level, where either the whole object is replicated or recent delta changes are replicated for a given account. Inter-stamp replication is used for (a) keeping a copy of an account's data in two locations for disaster recovery and (b) migrating an account's data between stamps.



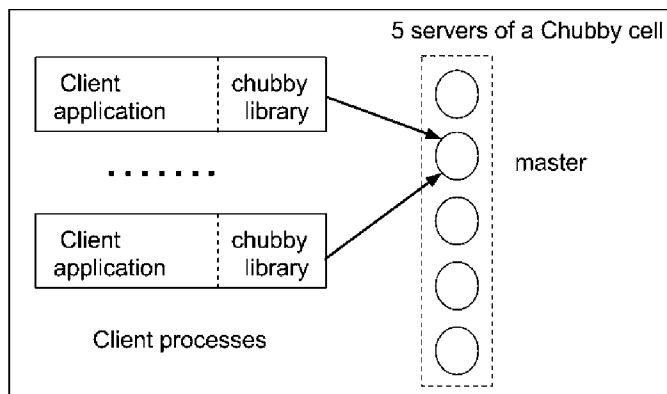
*Figure 2.5* — WAS High-level architecture. Figure taken from [ea11]

**Disaster Recovery-** WAS stores customer data across multiple data centers hundreds of miles apart from each other. This redundancy provides essential data recovery protection against disasters such as earthquakes, wild fires, tornadoes, nuclear reactor meltdown, etc. Intra-stamp replication provides durability against hardware failures, which occur frequently in large scale systems, whereas inter-stamp replication provides geo-redundancy against geo-disasters, which are rare. It is crucial to provide intra-stamp replication with low latency, since that is on the critical path of user requests; whereas the focus of inter-stamp replication is optimal use of network bandwidth between stamps while achieving an acceptable level of replication delay.

### 2.4.3 Chubby lock service

The purpose of the lock service is to allow its clients to synchronize their activities and to agree on basic information about their environment. Chubby lock service [Bur06] intends to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system, and in particular to deal with the problem of electing a leader [MMRT06] from among a set of otherwise equivalent servers. Primary goals include reliability, availability to a moderately large set of clients; whereas throughput and storage capacity were considered secondary. The primary goals attributes are easier to achieve when performance is less important. Because Chubby's database is small, it is possible to store many copies of it on-line (typically five replicas and a few backups), Figure 2.6. Full backups are taken multiple times per day, and via checksums of the database state, replicas are compared with one another every few hours. The weakening of the normal file system performance and storage requirements allows to serve tens of thousands of clients from a single Chubby master. By providing a central point where many clients can share information and co-ordinate activities, a class of problems faced by system developers was solved.

Chubby has become Google's primary internal name service; it is a common rendezvous mechanism for systems such as MapReduce [DG08]; the storage systems GFS and Bigtable use Chubby to elect a primary from redundant replicas.



*Figure 2.6* — Chubby system structure. Figure taken from [Bur06]

**Chubby Asynchronous Consensus** - Which describes the behavior of the vast majority of real networks, such as Ethernet or the Internet, that allow packets to be lost, delayed, and reordered. Asynchronous consensus is solved by the Paxos protocol [Lam01].

**Failure detection** - Chubby provides an event that allows clients to detect when a master fail-over has taken place. Chubby's default lease time is 12s and KeepAlives are exchanged every 7s.

### 2.4.4 Observations

Looking back at the aforementioned applications (i.e., Cassandra, WA, Chubby) we see that these applications do not force the use of hybrid synchronous environment, although they need to assure such behavior. These applications run on asynchronous environments supported with algorithms and/or protocols (e.g., PAXOS) to allow progress but not guaranteed. If these application run on top of hybrid synchronous infrastructure then the progress will be guaranteed by the provisioning of elements designed to respect time upper bounds. For example:

- In Cassandra: the failure detection algorithm needs to run on synchronous or hybrid synchronous subnetworks that communicate the cluster nodes, in order to guarantee delivering messages within the upper bound specified.
- In WA: The Intra-stamp replication protocols need to run on synchronous links that communicate the replicas on the same storage stamp together, while the Inter-stamp replication protocols need to run on asynchronous links that communicate the storage stamps together.
- In Chubby: The failure detection algorithm needs to run on synchronous environment to adjust perfectly the messages delay time.

The problem is that fully synchronous, or partially synchronous, environment is expensive to build, complex to configure, and difficult to control. This makes the infrastructure providers escape to asynchronous environments strengthened by algorithms and/or protocols with time-out specifications.

## 2.5 Hybrid synchronous DSs and virtualization

After studying the partial synchronous distributed systems, we noted down some of their constraints that pose difficulties for the service providers, for example, the high building cost of synchronous resources, the demand of DSs for isolation or resilience, and the complexity for realizing the hybrid synchronous DSs. In a research for relaxing these constraints; we investigated the space of Virtual Networks (VNs), and we found that virtual networks can offer a suitable environment for hosting hybrid synchronous distributed systems while optimizing a set of their constraints due to the properties that virtualization brings, for example:

1. The design of synchronous components in DSs requires fundamental handling mechanisms as detailed by [RSK<sup>+</sup>00]. The exact mechanisms for building synchronous resources is out of the scope of our work, but we assume their existence. Network virtualization is defined by the decoupling of the roles of the traditional Internet Service

- Providers (ISPs) into two independent entities [TT05]: infrastructure providers (InPs), who manage the physical infrastructure, and service providers (SPs), who create virtual networks by aggregating resources from multiple infrastructure providers and offer end-to-end services. Considering a new architecture of the DSs based on VNs provides a new business model that allows sharing tasks: *(i)* the synchronous resources design is assigned to the SN provider, and *(ii)* the resources allocation process is assigned to the VN provider. Delegating the design of the synchronous resources to the SN provider and the embedding process to the VN provider results in reducing the system complexity.
2. The building cost of synchronous resources in DSs is considerably high when compared to the asynchronous resources. The virtual networks environment allow providers to allocate some physical resources mutually among several clients as long as this multi use does not violate the performance expected by the client. Considering a new architecture for DSs based on VNs will allow sharing the synchronous portion of the physical infrastructure between several applications, which will result in reducing the overall cost for the service providers. It is predictable that sharing resources might affect the system performance [ABD<sup>+</sup>13], but we consider studying this effect is out of the scope of our work.
  3. The VNs Embedding process (VNE) [BHK12, HPN09] allows allocating resources flexibly respecting the constraints that serves the applications. For example, DSs resilience requirements that can be tackled during the VNE process aware of allocating backup resources or aware of live resources migration as will be detailed later in Chapter 3. Considering a new architecture for DSs built on top of VNs will allow adopting the VNE framework that will benefit the providers in supporting constrained applications in a flexible manner, benefiting from VNE constrained solutions in the literature.
  4. Heterogeneity in the context of network virtualization comes mainly from two fronts [CB10]: first, heterogeneity of the underlying networking technologies (e.g., optical, wireless, and sensor); second, each end-to-end VN, created on top of that heterogeneous combination of underlying networks, can also be heterogeneous. Considering a new architecture of DSs based on VNs exhibits the system heterogeneity in the hybrid kind of physical resources required to be synchronous and asynchronous. This allows using the VNs framework to manage hybrid environment. On the other hand, DSs providers will be able to implement freely network topology, routing protocol, QoS policies, independently of the coexisting VNs and independently of the the substrate network.

We argue that VNs and a suitable VN embedding process offer suitable environment for running distributed applications with partial synchrony. This has led to the abstraction of new type of virtual networks that we name *The Hybrid Synchrony Virtual Networks (HSVNs)* which we propose in our work.

## Summary

In this section, we provided a short background about distributed systems (DSs), and lightened an important aspect in the field, the synchrony. While asynchronous DSs support no time-bounds for processes execution and message delivery, the synchronous DSs provide time guarantees for them. Although fully synchronous DSs demand simpler algorithms, and can provide what asynchronous ones do, yet the undeniable problem of synchronous components (processes and channels) high cost led to the development of hybrid synchrony DSs.

Two branches of hybrid synchronous DSs are distinguished in the literature: *(i)* the hybrid synchronous in space, where subsets of the system components are synchronous while the others are asynchronous, and *(ii)* the hybrid synchronous in time, where the system components alternate between synchrony and asynchrony over time.

We provided examples of applications (i.e., Cassandra, WA, Chubby) that may benefit from the partial synchrony assumptions, as the progress will become guaranteed by the provisioning of elements designed to respect time upper bounds. The problem is that fully synchronous, or partially synchronous, environment is expensive to build, complex to configure, and difficult to control. This makes the infrastructure providers escape to asynchronous environments strengthened by algorithms and/or protocols with time-out specifications.

In a research for relaxing the ossifications of partial synchronous environment; we investigated the space of Virtual Networks (VNs), and we found that virtual networks can offer a suitable environment for hosting hybrid synchronous distributed systems while optimizing a set of their constraints due to the properties that virtualization brings.

In the next chapter, we define network virtualization, then we detail about the central problem with VNs, that is the problem of resource allocation (named also as mapping or embedding) from the literature.



---

## Related Work: Network Virtualization

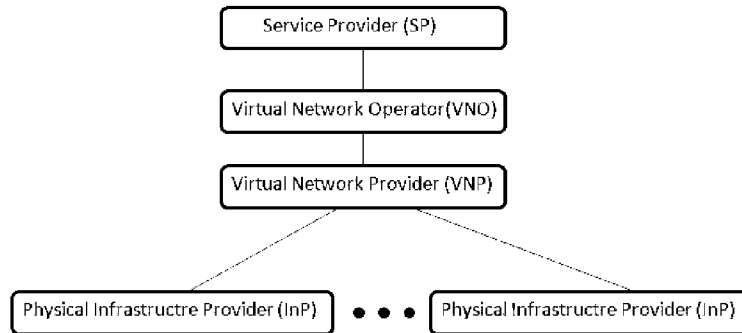
In this section, we start by defining the network virtualization, secondly we highlight the virtual networks properties as stated in the literature; thirdly we revise the literature on the topic of VNs resources allocation, which is named as *mapping* or *embedding* problem as well. We classify the works on VNs embedding according to the applications' constraints (e.g., topology, security, and resilience).

### 3.1 Network Virtualization

*Virtualization* is a technology introduced in 1973 [PG73], consists in using a single physical resource to host several virtual machines that share and access concurrently the actual hardware. The benefits of virtualization include reconfigurability, better resource utilization, mobility, isolation, and fault tolerance. Since similar benefits can be derived when virtualizing the network infrastructure, virtualization appeared as a solution to the architectural issues of the Internet, and this gave birth to the *Virtual Networks (VNs)*.

*Network virtualization* environments allow the coexistence of multiple VNs, each running certain applications on top of one shared physical infrastructure, the *substrate network (SN)*. Network virtualization is defined by the decoupling of the roles of the traditional Internet Service Provider (ISP) into two independent entities [CB10]: *infrastructure providers (InPs)*, who manage the physical infrastructure, and *service providers (SPs)*, who create virtual networks by aggregating resources from multiple infrastructure providers and create end-to-end services.

A business model for network virtualization was proposed by Schaffrath *et al.* [SWP<sup>+</sup>09], where the management and business roles of the service provider (SP) are separated. For this purpose, the SP was split into three entities as in Figure 3.1: *i*) the Virtual Network Provider (VNP), which assembles virtual resources from one or more InPs, *ii*) the Virtual Network Operator (VNO), which installs, manages, and operates the VN according to the needs of the SP, and *iii*) the Service Provider (SP), which is free of management and concentrates on business by using the VNs to offer customized services.



*Figure 3.1* — VN Management and Business Roles [SWP<sup>+</sup>09]

## 3.2 Virtual Networks properties

Network virtualization is a promising technology for overcoming Internet ossification [APST05], it was proposed to obtain certain properties and design goals [CB09]. For example, virtual networks allow:

1. **Revisitation:** which means that several virtual nodes and/or links belonging to same VN can be hosted on same physical node or link. This means resources sharing, which leads to a better use of the booked resources and thus reducing the cost;
2. **Coexistence:** several virtual networks of different service providers can coexist at the same time, over same infrastructure provider;
3. **Recursion:** named also nesting in VNs. It refers to the fact that, one virtual network belonging to a certain service provider, can take the place of a virtual infrastructure provider to another virtual network belonging to another service provider;
4. **Inheritance:** a given VN can inherit properties that exist in its parent, be it a physical infrastructure, or be it a virtual infrastructure;
5. **Flexibility:** each SP can implement freely network topology, routing protocol, QoS policies, independently of the coexisting VNs and independently of the the substrate network;
6. **Isolation:** network virtualization insures the isolation between the coexisting VNs, i.e., a fault existing in one VN does not propagate to any other coexisting VN. We should distinguish this case from the case when the fault exists in the underlying physical infrastructure. For example, if a physical node or link fails, the failure will surely impact all the virtual nodes and links mapped to it;
7. **Heterogeneity:** in network virtualization, the heterogeneity can be found on two levels, i) in the underlying infrastructure, which can be a mixture of several technologies, and ii)

in the VNs, where SPs can implement VNs different in their requirements (for instance, security, delay, jitter, ...).

### 3.3 Resources allocation in Virtual Networks

*Resource allocation* is a process formed by the InPs upon receipt of a request to establish a VN [BHK12]. Virtual networks can be constructed through a suitable deployment of the virtual routers and links on the SN resources [HPN09]. This process is also known as VN embedding or VN mapping. The problem of resource allocation has attracted considerable attention, and since the problem is considered to be an NP-hard problem [Kar72], it has been addressed in the literature also through optimization methodologies.

In this thesis, we will use three compatible terms referring to the same meaning: *mapping*, *embedding* and *resource allocation*.

Figure 3.2 shows an example for embedding two VNs on one SN, links bandwidth and nodes CPU values are presented for both virtual and substrate networks. This is a simple example, the only constraint is not to pass the physical nodes and links capacities for CPU and bandwidth respectively. The embedding output is detailed (written) in the bottom side of the figure. We notice that nodes  $c^1$  and  $c^2$  belonging to  $VN^1$  and  $VN^2$  respectively were mapped on the same physical node  $c$ . The physical link  $(f, s)$  participates in the two physical paths mapping virtual links,  $(f^2, e^2)$  and  $(c^2, e^2)$ .

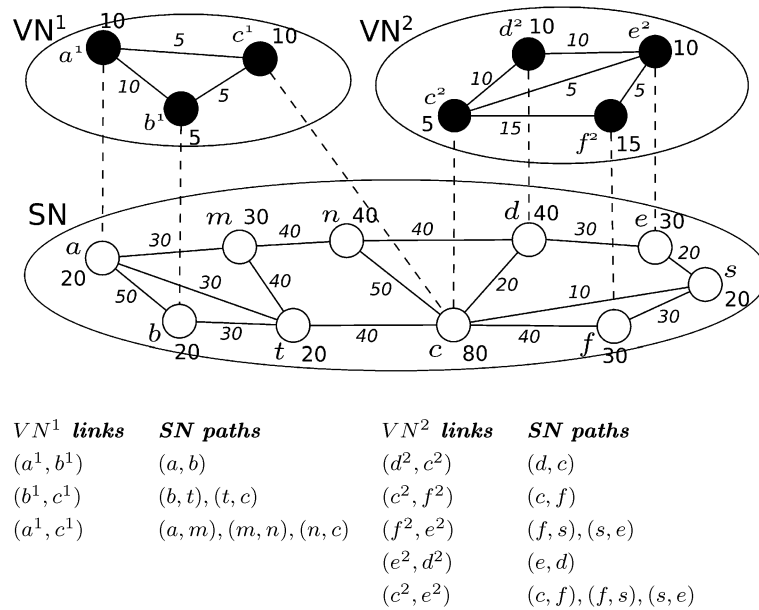


Figure 3.2 — Virtual networks mapping

#### Commanding the embedding process

In the virtual networks environment, the embedding process can be conducted by one

central entity, or by several ones that interact together to take decisions [BHK12]. The first approach is named *centralized* and the second *distributed*.

*Centralized approach:* where a single entity in the InP receives VN requests and performs resource allocation. The limitations for this approach are mainly: (i) this entity requires all the global knowledge and resources necessary for the allocation process, which is not always feasible; (ii) if this entity fails, so does the whole process, and  $c$  the communication between the central entity and the other nodes might cause a considerable overhead. Among the works that followed the centralized approach are [ZA06, LT06, CRB09].

*Distributed approach:* in which the resource allocation process is distributed over some or all of the physical nodes in the InP, where each node has local knowledge. The limitation in this approach lays in its complexity compared to the centralized approach, because the allocation process is coordinated through the communication and the cooperation protocols among the nodes. Among the works that followed this approach are [HLZ08, CSB10, HLZ<sup>+</sup>10].

### **Problem complexity**

When mapping VNs, several virtual nodes can be mapped on the same physical node, and several virtual links can be mapped to the same physical link. This mapping process should happen without passing the limits of the physical resources available, i.e., the sum of resources demanded by virtual nodes and/or links should not pass the capacity limits of resources offered by the physical nodes and/or links to which they are mapped.

In one basic version of the Unsplittable Flow Problem (*UFP*) [Kle98], we are given a graph  $G$ , an  $m$  pairs of vertices each associated with a non-negative demand  $\psi \leq 1$ , the problem negotiates the possibility of finding a path, within  $G$ , for each pair in a way that the accumulated demands on each path do not exceed 1, if it is possible then the demands are *realizable*.

VNs mapping problem turns to be similar to the *UFP*, that is a generalization of the well known Edge Disjoint Paths problem (*EDP*) [Kle98] which is a central problem in combinatorial optimization and algorithmic graph theory and is one of the Karp's original NP-complete problems [Kar72].

NP-complete problems are nondeterministic polynomial time problems [God04] [GJ79], if  $L$  is an NP-complete problem then there is no known efficient solution for it (i.e., no fast solution), and the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows. This is a reason why NP-complete problems are approached normally through optimization approaches, for example heuristics [MF04].

## 3.4 Literature review on VNs Embedding

By searching in the field of VNs mapping, we found that the process of virtual networks mapping comes with different flavours based on the application requirements, which impacts directly the mapping complexity and the kind of constraints that should be cared for. Here bellow we summarize several works that handle the problem of VNs embedding. Our state of the art classifies others works based on the VNs demands considered, in other words, the embedding constraints.

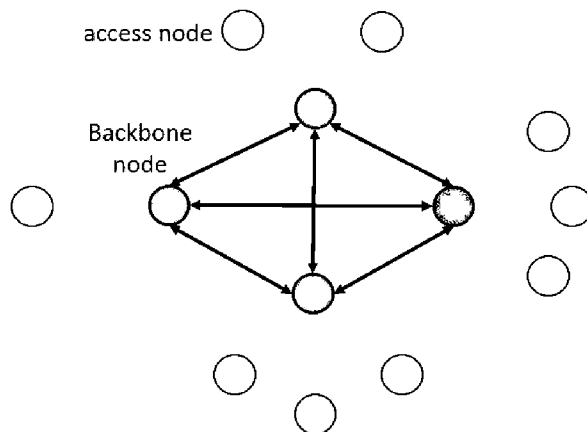
### 3.4.1 Topology requirements

When mapping VNs only with topology requirements; the point that matters is to preserve the nodes connectivity pattern. For example, let  $a$  and  $d$  be two virtual nodes on a given virtual network, connected together through a virtual link  $(a, d)$ . When mapping these two virtual nodes to two distinct physical nodes  $A$  and  $D$  respectively, then  $A$  and  $D$  should be connected together as well, be it direct connection through a physical link  $(A, D)$  or be it through an indirect connection through a physical path that starts with  $A$  and ends at  $D$ , like  $(A, B, C, D)$ .

One of the earliest works on resource allocation in network virtualization is [ZA06]. Only the VNs topology is considered; there are no constraints on virtual nodes (such as CPU and nodes location) and virtual links (such as bandwidth). Upon the arrival of a VN request, its topology is assigned to the substrate network to achieve low and balanced load on both substrate nodes and links. So, the objective when mapping is to reach load balance on the SN resources (i.e., physical nodes and links), where the load here can be defined as the amount of physical resources used by virtual nodes and virtual links in substrate nodes and substrate links, respectively. The authors in this work defined nodes stress and links stress for the physical resources, which became metrics adopted by other works later [Cui12]. Node stress measures the number of virtual nodes assigned to each physical node; and link stress measures the number of virtual links traversing a physical link. To ease the embedding process, it was proposed to divide the VNs into several smaller networks, then the embedding phase is conducted by a centralized entity with the objective of reducing the nodes stress or the links stress, depending on which is more critical. The work focuses on two versions of the VNs assignment problem: VN assignment without reconfiguration (VNA-I) and VN assignment with reconfiguration (VNA-II). For the VNA-I problem, where the VN assignment is fixed throughout the VN lifetime, the authors developed a basic scheme to achieve near optimal substrate node performance and used it as a building block for all other advanced algorithms. Subdividing heuristics and adaptive optimization strategies are then presented to further improve the performance. For the VNA-II problem, the authors

developed a selective VN reconfiguration scheme that prioritizes the reconfiguration for the most critical VNs. In doing so, the authors could achieve most performance benefits of the reconfiguration without excessively high cost. The results show that: *(i)* subdividing the VN topology is more significant when the topology is sparse; *(ii)* the advantage of the algorithms is greater when the substrate network is sparsely connected; and *(iii)* the algorithms can effectively avoid hot spots or congestion in the substrate network.

In [LT06], authors develop a VN cost-efficient mapping method able to handle VNs traffic pattern allowed by a general set of traffic constraints. The authors argue that the existing approaches, like simulated annealing or some similar local search techniques, that aim at cost-efficient VNs embedding, are not suitable. With such techniques, a given solution for the mapping is optimized by adding/removing links and/or nodes, then the links of the modified topology re-dimensioned, so that the cost can be evaluated. The authors find that this approach has two drawbacks: *(i)* excessive computing: with any modification to the current topology, the mapping cost is recomputed, and *(ii)* the huge space of candidate topologies makes it difficult to determine which of the large number of possible local modifications to choose from. To overcome these drawbacks, the authors of this paper propose a mapping approach that aims at finding the best topology in a family of backbone-star topologies. In a backbone-star topology, the nodes are designated as backbone nodes, or as access nodes. Each access node has a single edge connecting it to a backbone node, meaning that each backbone node is at the center of a *star* formed by its neighboring access nodes. Basically, the backbone nodes are connected in an arbitrary way, yet in this paper the authors consider particular topologies for the backbone nodes, such as a complete graph, a ring and a star. Fig 3.3 depicts a virtual network in a backbone-star topology, where the four backbone nodes are connected into a complete graph.



**Figure 3.3** — Example of a virtual network in a backbone-star topology, where the four backbone nodes are connected into a complete graph [LT06]

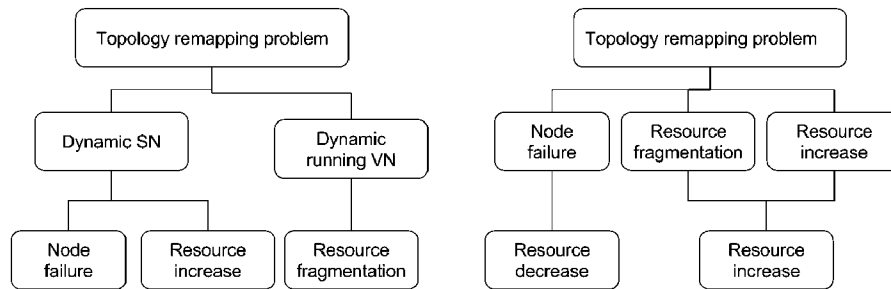
The outlines of the mapping method proposed can be summarized in five steps: *(i)* Select an initial mapping of backbone nodes onto the substrate, *(ii)* Connect access nodes to

backbone nodes, (iii) Compute shortest paths, (iv) Determine link capacities (This can be done using linear programming), and (v) Find best backbone node mapping. Main findings of this paper are: (i) as pairwise traffic constraints are relaxed, the least-cost backbone topology becomes increasingly tree-like. (ii) the quality of solutions improves as the traffic locality gets weaker.

In [YYRC08b], the authors lighten four main obstacles that make the VNs embedding problem a difficult one, they are: (i) virtual resources constraints (e.g., nodes CPU and links delay) where the more the constraints are, the more complex the mapping process gets; (ii) admission control on the SN level, since the SN has limited resources, then certain requests will be rejected or postponed to avoid violating the successfully mapped VNs; (iii) online requests, which implies that the VNs demands are not known in advance, and when a certain request arrives, the period of its remaining is not known as well; and (iv) the diverse topologies, where handling arbitrary topologies, while efficiently supporting the most common topologies, introduces an additional challenge for the embedding algorithm. The research precedent to this work has addressed these computational challenges by restricting the problem space in one or more dimensions to enable efficient heuristics, at the expense of limiting the practical applicability of the solutions. For example, the papers used to either solve an offline variant of the problem, consider only bandwidth constraints, or do not perform admission control. In this work, the authors advocate a different approach: rethinking the design of the substrate network to enable simpler embedding algorithms and more efficient use of resources, without restricting the problem space. In particular, they extend the virtual links embedding by: (i) allowing the substrate network to split a virtual link over multiple substrate paths, and (ii) employing path migration to periodically re-optimize the utilization of the substrate network. Flexible path splitting allows mapping the virtual links to the substrate in polynomial time, while making much more efficient use of substrate bandwidth and increasing robustness to substrate failures. This solves the ossification posed by the first three challenges listed above. To handle the fourth challenge, the work explores node-mapping algorithms that are customized to hub-and-spoke topologies of the VNs. Nodes with the most available resources are selected as hub nodes while the spoke nodes are mapped on substrate nodes based on the shortest paths to the nearest hub node. Simulation experiments show that, path splitting, path migration, and customized embedding algorithms enable a substrate network to satisfy much larger mix of virtual networks and makes the embedding problem computationally easier.

In [LZW15] Li e al. study the virtual networks embedding problem considering dynamic topologies, where *topology dynamicity*, according to the authors, comes from two fronts: (i) the substrate network topology change, which happens when the infrastructure provider may make some adjustments for more profits or some failure nodes that need to be corrected, and (ii) the birth and death of virtual networks, in other words, the new arriving virtual network

that need to be mapped and the expiration of VNs that release resources. Remapping the still existing (alive) VNs might result in better use of the physical resources and thus might lead to allowing the embedding of VNs that were rejected without the remapping process. For these two cases, the authors call them the dynamic substrate network and the dynamic running virtual networks. After researching the dynamic substrate network and running virtual network, the authors find that they can be divided into resource increase and deletion while they attribute the resource fragmentation to the resource increase, Figure 3.4. The authors propose a formal expression of the dynamic remapping problem, but they do not proceed to validating it.



**Figure 3.4** — Topology remapping problem [LZW15]

### 3.4.2 BW and CPU requirements

Among the attributes that were considered by most of the works on the field of VN mapping are: *(i)* the links bandwidth (BW), and *(ii)* the nodes processing power (CPU).

Trinh et al. [TEA11] propose to analyze the application of careful overbooking concept, that uses flexible levels of availability to provide Service Level Agreements (SLA) to users. Based on this framework, virtual network subscribers are provided with a service that is more suitable with their tolerability to utilize the soft-guaranteed bandwidths. So, the system will figure out the actual resources for customers to guaranteeing the quality of service even when the system is in the most congested time. This helps to save the cost of subscribers, and to increase the profitability of the provider. The virtual networks which do not need to have the exclusive service can be offered some other kinds of services whose quality are specified in three parameters: *(i)* probability of getting full availability, *(ii)* probability of getting limited availability, and *(iii)* reduction factor. So, each customer, who wants a service from infrastructure provider will be provided only one SLA proposal including the percentage of time to get full availability, the percentage of time to get limited availability and the bandwidth reduction factor from full availability to limited availability. The problem addressed was formulated in the shape of a Mixed Integer Program, whose objective is to minimize the cost, where the cost considered was a combination of both



bandwidth consumption and traffic routing cost. Practically, the optimization process pushes the virtual network demand through their best route for minimizing the cost of building all virtual networks. Evaluation results show that the cost saving ratio is of mean = 0.748 and variance = 0.0479, which is quite impressive based on reported numerical results. The infrastructure provider can give part of that obtainable saving to customers with limited availability, who in their turn will enjoy the reduction in the service price.

Botero et al. [BHFdM13] raised the importance of considering the capacity of the hidden hops while the embedding process, where hidden hops are the intermediate nodes on the physical paths that map the virtual links. The authors argue that, portion of the BW is consumed by the intermediate nodes, because they have to be configured to process and forward the packets passing through that virtual link, thus, it is important to consider the hidden hops while the VNs mapping phase. The work distinguishes between two types of the VNs: (i) VNs with specific demands where there can be bandwidth and CPU fixed demands for some virtual links and nodes, and (ii) VNs with no specific demands where nodes and links do not ask for any resources. The work models the problem as the shape of a Mixed Integer Program, with the objective function of maximizing the sum of the spare bandwidth and spare CPU in the substrate network. Then, they develop an algorithm that is based on the MIP. The algorithm is divided in two different steps: firstly, the algorithm maps the requests of each VN that explicitly ask their demands. In second place, the remaining resources are distributed equally among the remaining virtual nodes and links. The second step of the algorithm (i.e., to allocate the resources equally among the requests without demands) is divided in two parts; the *virtual node mapping*, this was treated in an easy manner by assuming that each virtual node is already mapped, and the *virtual link mapping*, this was treated based on Dijkstra algorithm for finding the shortest path connecting the end points. The proposed heuristic is mainly based on an approximated greedy algorithm proposed to solve the Unsplittable-Flow problem, the authors make few modification to adapt the algorithm to the mapping problem. This work does not handle evaluation for the proposed heuristic.

Hsu et al. in [HSWY12] consider nodes CPU and links BW as others, but for them BW is not only a link attribute, it is as well a node attribute, where node BW is the sum of links BW connected to it. The mapping approach they propose bases on path splitting and migration technique that aims at maximizing the number of coexisting VNs in a substrate network and increases the revenue of the Infrastructure Providers (InP). The algorithm proposed consists of three main blocks, they are: (i) the node mapping algorithm; ; (ii) the link mapping algorithm; and (iii) path migration. The *node mapping algorithm* working steps are: (i) choose the first virtual node with the largest degree, denoted  $X^v$  for example, where node degree is the number of the node neighbors (i.e., the number of links connected to the node), then the node BW required is calculated; (ii) choose the subset of SN nodes

that can satisfy the first chosen virtual node both for its CPU and BW, then the substrate node with the largest degree is chosen from the candidates, denoted  $X^s$  for example, to map  $X^v$ ; (iii) choose the next virtual node that has the largest number of neighboring nodes already mapped, denoted  $Y^v$  for example, and let  $T^s$  be the set of physical nodes that mapped  $Y^v$  neighbors; (iv) choose the physical nodes that can satisfy  $Y^v$  CPU and BW constraints, and for each node among the selected candidates, the algorithm calculates the shortest path between it and each physical node in  $T^s$ , and uses the maximum length of these computed shortest paths as the shortest distance between  $Y^v$  and  $T^s$ , and the node with shortest distance is selected to map  $Y^v$ ; (v) the steps (iii) through (v) are repeated till all the virtual nodes are selected. The second block of the proposed heuristic is the *links mapping algorithm*, and its working steps are summarized as the following: (i) sort the virtual links by BW requirements in decreasing sequence; (ii) processes the first virtual link in the ordered set (i.e., the link with the maximum BW requirement), let it be the link  $l^v$  with BW demand  $BW(l^v)$  connecting the two virtual nodes  $X^v$  and  $Y^v$ , where these virtual nodes are already mapped on the physical nodes  $X^s$  and  $Y^s$  for example; (iii) find the shortest physical path that connects  $X^s$  with  $Y^s$  and satisfies  $l^v$  bandwidth requirement. If such a path is found then  $l^v$  is mapped to it, otherwise, the *multiple path mapping* procedure is called. The main working steps of the aforementioned procedure are: (i) assign the virtual link that was unable to be mapped on one single physical path, let this link be denoted as  $l^v$  for example, with bandwidth requirement  $BW(l^v)$ ; (ii) split  $BW(l^v)$  in two,  $BW(l^v)/2$ ; (iii) search two physical shortest paths connecting  $X^s$  with  $Y^s$  with residual BW that allows mapping  $l^v$ , if such paths are found then  $l^v$  is mapped to them, otherwise step (ii) is repeated; (iv) maximum number of splitting is allowed, by passing it, the *path migration* algorithm is enabled, which forms the third part of the proposed heuristic. The *path migration* algorithm does not consider node-remapping for the VN requests (i.e., does not remap  $X^v$  and  $Y^v$ ), rather, it migrates a virtual link already mapped, to free more BW on the path under study and then uses the modified multiple-path algorithm to select one or more new substrate paths for the current virtual link. Simulation results indicate that the path splitting and migration offers better performance than existing mapping approaches.

Links BW and nodes CPU were among the attributes that were considered by most of the works on the field of VN mapping beside some other constraints, for example, Zhang et al. [ZWJY10] devise a mapping model for delay aware VNs, the model aims at minimizing both the bandwidth and CPU consumption on both physical links and nodes respectively (the work is more detailed in 3.4.5). Bay et al. [BOB<sup>+</sup>12a] addressed a security-aware VNs embedding model, which aims at minimizing the bandwidth consumption on physical links (more details about the work are given in 3.4.3).

### 3.4.3 Security requirements

Revising the literature, there were several works that devise a framework to provide secure virtual networks [CDRS07, HAM10]. These works study providing security to VNs after the phase of resource allocation.

However, some works consider security aspects while allocating VNs resources. The first who addressed security-aware VNs embedding were Bays et al. [BOB<sup>+</sup>12a]. In this work, the authors consider three levels of security, they are, in the increasing order of security: *(i) end-to-end security*, where the end points of the virtual links are mapped on physical routers that are able to encrypt and decrypt packets, *(ii) point-to-point security*, where the virtual links are mapped on physical paths composed of routers able to decrypt and encrypt the packets (i.e., be them edge routers of intermediate ones). So, at each point the packets are decrypted and encrypted again. In both aforementioned types, the encryption is on the level of the packet payload and header. And *(iii) non-overlapping networks*, which are networks that don't share the same substrate physical resources, i. e., they are networks mapped on distinct physical resources. Considering both optimal mapping and security to be equally important, the authors devise a mapping model in the shape of a Mixed Integer Program, that aims at minimizing the bandwidth consumption, while considering levels of security as detailed above. The main findings of this paper are: *(i)* bandwidth usage grows proportionally to the number of virtual network requests; *(ii)* raising resource limits on each request also causes a growth in bandwidth consumption. However, the effect is notably less significant than the previous factor; *(iii)* the proposed method avoids loading the physical resources; *(iv)* there is a trade-off between running time and optimality, in other words, finding the optimal solution results requests long computational time, that reached 24h in some scenarios. In order to investigate the impact of considering security requirements during the VNs embedding phase, the authors envisage a scenario in which all security related constraints are disabled and all the security requirements in the VNs are removed. It was stated that the disconsideration of security assessments results in reducing the bandwidth consumption. The authors refer this overhead to the following reasons: *(i)* the routers that provide encryption/decryption protocols forms subset of the overall available routers, thus resulting in a more constrained solution space; *(ii)* the non-overlapping requirement forces the allocation algorithm to select detour paths in the substrate network, which results in higher resource consumption. These reasons also indicate that, in the best case scenario, the bandwidth consumption considering security-related constraints will be as good as without considering them. This can be an evidence that minimizing bandwidth consumption is a suitable objective function for the MIP developed for VNs mapping with security-related constraints.

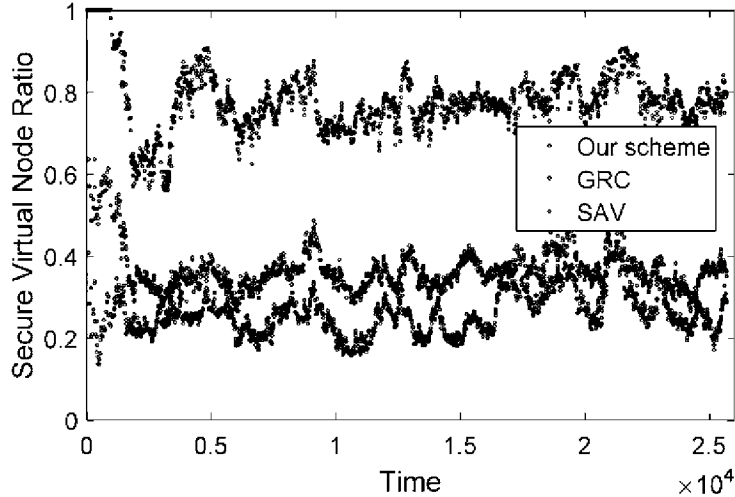
In [WWG<sup>+</sup>16] the authors treat the problem of information leak possible between virtual nodes that share the same physical node. This happens because two virtual nodes

can communicate without being monitored or controlled by their underlying system through a covert channel. The authors propose a method to mitigate the risk of covert channel attacks in network virtualization. The proposed method was divided into two categories: *heavyweight* methods for the SN and *lightweight* methods for the VN. Heavyweight methods imply a modification in the underlying physical infrastructure by inserting additional components resulting in performance overhead and additional cost, whereas lightweight methods focus on the resources allocation process with no need to physical components change. The scenario is as the following: a victim virtual node connected to an adversary virtual node through a covert channel, and all other virtual nodes are regarded as bystanders for the first two nodes. The victim node, the adversary, and the bystanders are virtual nodes mapped on one physical node. Two novel attributes are associated to each virtual node: an *error rate* and an *expected threshold*. A bystander virtual node causes an error rate to the covert channels between any other two virtual nodes (i.e., the victim and the adversary) in the same substrate node. The error rate of each virtual node is considered a parameter that expresses the noise effect of the bystanders on the transmission quality of a covert channel. A victim virtual node leaks information via a covert channel. Higher noise on the covert channel means less information leak. The victim node is considered secured if the accumulated error rate of the bystanders on the covert channel is no less than the expected threshold of the victim node. The VNs embedding problem is formulated as an optimization problem with novel constraints that are the risk-tolerant coexistence constraints which consider proper embedding of the virtual nodes to mitigate information leak. The proposed scheme was compared to other two schemes suggested in the literature: 1) *SAV* [LCXX15] which embeds a VN satisfying security demands; and 2) *GRC* [GWZL14] which aims to maximize the resource utilization without security consideration. Simulation results show that the proposed scheme in [WWG<sup>+</sup>16] improves the percentage of secure virtual nodes by 40% in comparison with [LCXX15, GWZL14], Figure 3.5 depicts a comparison of Secure Virtual Node Ratio in the three works.

### 3.4.4 Resilience requirements

Failures in VNs are mainly related to physical nodes and links. The main two approaches for minimizing the VNs failures are [Oli13]: (i) backup resources, and (ii) live reconfiguration and migration.

**Backup resources** - the idea is that certain physical resources are duplicated, so that one copy of these resources is considered primary, and the other one (i.e., the backup copy) as secondary. If the primary resources fail; the backup ones take their place. The simplest algorithms proposed for nodes backup preserve one single node as backup for each critical physical node, thus, providing immunity against one failure only. A better approach is



**Figure 3.5** — Comparison of Secure Virtual Node Ratio [WWG<sup>+</sup>16]

to request  $K$  shared redundant virtual nodes [YAQS11]. This allows any critical node of a protected VN to sustain up to  $K$  consecutive node disruptions. The authors model the mapping problem in a Mixed Integer Linear Program (MILP) and propose efficient heuristics based on the MILP formulations. The work compares the performance through evaluating the redundancy ratio performance metric, which is the ratio of the total backup resource cost to the total working resource cost. The redundancy ratio performance metric is studied in three scenarios: (i) both cross and backup share (labeled as *share*); (ii) only backup share (labeled as *bshare*); and (iii) no share (labeled as *noshare*). As well, the work compares the performance of the 1-redundant solution and  $k$ -redundant solution. This is done considering the following performance parameters: (i) node cost ratio (i.e., the ratio of the node redundancy cost); (ii) link cost ratio (i.e., the ratio of the link redundancy cost); and (iii) total cost ratio (i.e., the ratio of the total redundancy cost). Simulation results show that the proposed backup and cross share strategies have a significant impact in conserving backup resources and improving resource utilization. Furthermore, under majority of the circumstances, the  $K$ -redundant solution is more efficient than the 1-redundant solution especially when communication costs are higher than the node computing costs.

In [RB13], the problem of links backup is handled through a detours preallocation mechanism that tries to protect the connectivity of the endpoints of each physical link by routing its traffic to neighboring links, while restoration paths are used to protect an entire virtual link through links duplication. To improve the efficiency in resource utilization, restoration paths can be shared by different virtual links. In this work, the authors distinguish between two types of failures: (i) failures at the physical layer; and (ii) failures at the logical layer. Logical failures affect the logical layer only, in contrast, physical failures affect both the physical and logical layers. The main contributions of this paper are: (i) propose a survivability mechanisms to the VNs embedding phase using efficient restoration and protection policies;

(*ii*) add service level agreement (SLA) assurance to the VNs embedding phase by prioritizing the restoration of failed virtual links while minimizing the failure effect and maximizing the InP business profit; (*iii*) formulate the problem stated in the shape of a linear program and develop heuristic that bases on it; and (*iv*) introduce path-flow based optimization formulations for the different recovery and protection policies. The evaluation results show that the proposed solution outperforms the baseline solution in InP business profit, acceptance ratio, bandwidth efficiency, and response time.

In [CLW<sup>+</sup>10], another links backup mechanism is used, bases on an on-demand embedding of restoration paths. This work focuses on the case of one single link failure, assuming that the probability of two links failing at the same time is small. The authors present an embedding heuristic that pays simultaneous attention to two missions: (*i*) assuring a cost-effective usage of physical resources through an intelligent bandwidth sharing technique, and (*ii*) protecting VN services against network failures. To determine the restoration paths for virtual links, we present a state independent and path-based path selection scheme. The selection of the restoration paths is not dependent on any deterministic failure scenario and the interrupted virtual links are switched from their primary substrate paths to pre-reserved restoration paths when there occurs any substrate link failure. The restoration paths are used only when a failure occurs, and are used temporarily till the failure is fixed, thus the bandwidth reserved for the restoration paths is lower than the one reserved for the primary paths. Furthermore, the restoration paths share the same bandwidth reserved, as a way to reduce more the bandwidth consumption. Evaluation results show that the proposed algorithm outperforms the common algorithms both in terms of network resource usage and effectiveness of economic revenue over cost. Particularly, the proposed algorithm reduces the additional restoration bandwidth by over 35% comparing to the traditional algorithms.

In [YWK10], the authors introduce the concept of fault tolerance at the virtualization layer. The benefits of this technique are: (*i*) various levels of reliability can be customized over the same physical infrastructure, and (*ii*) no need for specialized fault tolerance servers. This is achieved through an opportunistic redundancy pooling mechanism (ORP), where backup resources are pooled and shared across multiple virtual infrastructures, and intelligently embedded in the physical infrastructure. The level of reliability is limited to the number of failed resources, where this last one should not pass the number of the redundant resources. For example, by achieving an  $n:k$  redundancy architecture,  $k$  redundant resources can be backups for any of the  $n$  primary resources, and share the backups across multiple virtual infrastructures ( VInfs). Another contribution of this paper is a method to statically allocate physical resources to the primary and redundant VInfs simultaneously, considering the output of the ORP mechanism. The embedding problem is formulated in this work in the shape of a Mixed Integer Program with the objective of minimizing the amount of resources used for a VInf. To evaluate their approach, the authors considered three scenarios in their

simulations for allocating resources: (i) with redundancy pooling and redundant bandwidth reduction (this scenario is labeled *share*), (ii) without redundancy pooling and redundant bandwidth reduction (this scenario is labeled *noshare*), and (iii) a system with VInfs that do not have reliability requirement, i.e., zero redundancy (labeled *nonr*), this last scenario is considered as baseline of comparison. The main results of this work are: (i) *noshare* has the least acceptance rate and VInf occupancy, and more backup nodes per VInf than *share*; (ii) CPU usage per VInf is slightly higher in *noshare* than *share*; (iii) the redundant nodes in *share* consume less resources than that in *noshare* (despite admitting more VInfs); and (iv) the bandwidth usage per VInf is actually smaller for *share* than *noshare*.

The drawbacks of the backup approach (be it for nodes or links) are mainly twofolds: (i) it is costly, because the backup resources are reserved independently of the failure occurrence; and (ii) the backup resources might become wasted if no failure occurs (i.e., the backup resources are used only when a failure occurs).

**Live reconfiguration and migration-** The idea is to recalculate physical resources and reallocate virtual nodes and/or links whenever a failure occurs. This approach is considered cheaper compared to the backup approach. Researchers in this track consider one of two procedures: (i) a pro-active procedure that bases on migrating virtual nodes and/or links that are subject to failures (or that are more likely to fail) prior to failure occurrence, and (ii) reactive procedure that bases on executing the recalculation and migration after the failure occurs. The first procedure avoids service interruption, while the second one does not, and researchers in this case (i.e., resources migration after failures) compete in minimizing the time of service interruption.

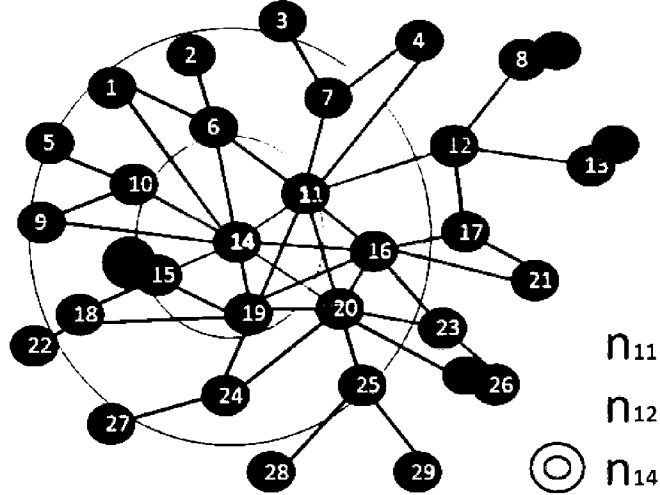
Among the works that consider the former approach we mention [HLZ<sup>+</sup>10]. In this work, Houidi et al. argue that it is important, while the VNs mapping process, to consider the dynamic changes that are induced by the VNs and/or the SN, like: (i) topology changes, (ii) resources restriction because of the already mapped VNs, and (iii) resource failures, and *resource degradation*. In this regard, an adaptive virtual resource provisioning is needed in order to preserve virtual networks, allocated initially on demand, in response to a virtual network creation request (i.e., service level agreement). More precisely, the authors detail about the adaptive resources allocation in two cases; (**case 1-updated virtual demands**), when the VN user asks for new requirements, like in the VN topology (due to VNs expansion) and/or in the service requirements. In this case, an *adaptive matching algorithm* is required to identify new physical resources as candidates to handle the updated virtual requirements. The matching algorithm starts by searching candidates in the proximity of the virtual components (i.e., virtual nodes and links) that were subject to the change. So, the first candidates nominated are necessary parents, children or siblings of prior matching. this choice avoids returning to the dendrome root [BHK12] unless needed. **Case 2-resources failure or degradation**, when physical resources allocated to previously instantiated VNs

suffer from failure or from performance degradation. In this case, the InP maintains the VNs topologies by searching for new resources to replace the affected ones. The authors propose an adaptive VNs embedding algorithm only for the case of resources failures (which is the second case among the two stated above) since it is the most important case. Three types of failures are discussed in this paper: *(i) virtual node failure*, when a certain virtual node fails, it is needed to either re-instantiate another virtual node in the same substrate node or, if not possible, to call for other substrate nodes. This implies reallocation of the virtual links associated to the affected virtual node; *(ii) substrate node failure*, when a certain physical node hosting multiple virtual nodes fails, in this case, every virtual node hosted on the crashed physical node should be reallocated, together with its associated links; and *(iii) substrate link failure or degradation*, when a physical link fails (interrupts) or gets congested or overloaded, then a new physical link or path should be reallocated. Simulation results show that, the proposed fault-tolerant embedding algorithm can react quickly and efficiently to resources failures.

In [GKA<sup>+</sup>16b], Ghaleb et al. deal with the problem of multiple link/node substrate failures that impact a multicast virtual network (MVN) in which link recovery is not feasible and node migration is mandatory (scenarios when backup is not enough/good). This work introduces a recovery algorithm that aims to minimize service downtime while satisfying the QoS requirements of the VNs. The proposed approach is based on the following three pillars: *i)* Minimizing the search region by using the intermediate (assisting) nodes (ANs) that interconnects the failed MVN's distribution tree to find a backup node. *ii)* Performing nodes ranking and filtering algorithm (NRF); to start the search from nodes that most likely give faster recovery and minimize the search region further (ranking), and remove nodes that does not provide any solution (filtering). *iii)* Performing a shortest-path search from each node that enables finding more than one candidate backups (hosts for the failed VMs) without repeating the same search. Figure 3.6 illustrates a multicast network with one source (S) and three terminals ( $t_1, t_2, t_3$ ). The three terminals are mapped to the nodes  $n_8, n_{13}, n_{26}$  respectively. A failure is assumed to hit node  $n_8$  or link  $(n_{12}, n_8)$  disconnecting  $n_8$  from the multicast network. The AN set is therefore formed from  $n_{11}, n_{12}, n_{14}, n_{20}$  and  $n_{23}$ . The search for a backup terminal can start from any node in the ANs list. However, the search region for each AN is different, and is determined by the minimum and maximum distances in terms of delay units allowed, while respecting the original delay variation between the terminal before the failure occurrence. Simulation results show that the proposed restoration algorithm is highly scalable and achieves a high restoration rate with a much faster restoration time even with high load and large number of concurrent failures.

The same authors of [GKA<sup>+</sup>16b] had a former work [GKA<sup>+</sup>16a] where they considered a reactive approach for link recovery with end-delay and delay-variation constraints. However, the approach addressed a single link failure that isolates single or multiple nodes in which





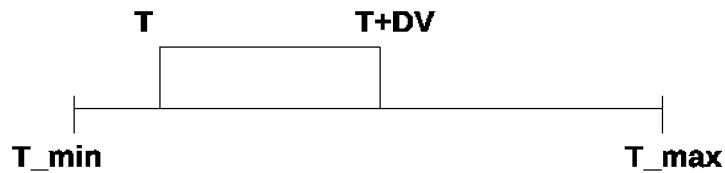
**Figure 3.6** — Assisting nodes and search regions [GKA<sup>+</sup>16b]

alternative paths can be constructed to the affected nodes. The recent work [GKA<sup>+</sup>16b] complements [GKA<sup>+</sup>16a] by providing a comprehensive failure recovery framework, considering repairing the failed MVNs reactively while maintaining the end-delay and delay-variation requirements in a failure-prone data center network.

### 3.4.5 Delay requirements

Zhang *et al.* in [ZWJY10] address the problem of mapping virtual multi-cast networks, which indicates the existence of one sender and several receivers. An application of this kind of networks is video gaming, video conferencing and similar real-time applications. In this kind of applications, packets are supposed to be received at the destination within specific delay, and the delay difference of packets reception at multiple destinations should be minimal. The authors formulate the mapping problem in the shape of MIP which is composed of: (i) an objective function that aims at minimizing the use of the total physical resources needed, and (ii) three mapping constraint, the first  $C1$  to assure not passing the physical resources capacity (i.e., nodes CPU and links BW), the second  $C2$  is to bound the messages delay between each pair sender-receiver, and the third  $C3$  is to bound the messages delay variation between each pair of the receivers. Hereafter, the authors propose a heuristic algorithm based on the MIP for mapping virtual multi-cast service-oriented networks subject to delay and delay variation. The algorithm working steps can be summarized by the following steps: (i) receive a request  $R$  denoted by six parameters, the message sender, the list of the message receivers, the node CPU, the link BW, the maximum delay allowed for each pair sender-receiver, and the maximum delay variation allowed between each pair of the receivers; (ii) calculate the  $k$  shortest paths between the sender and each receiver, this assures meeting constraints  $C1$  and  $C2$ , the value of  $k$  can be defined according to the networks

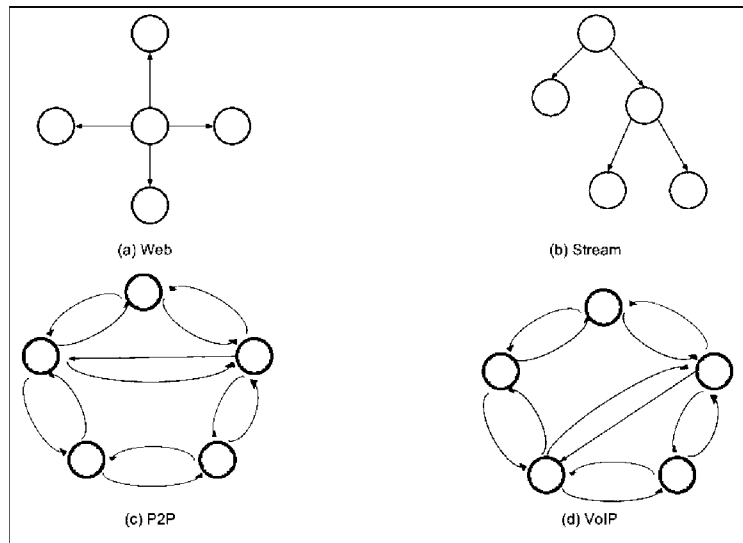
sizes (the VNs and the SN); *(iii)* apply sliding window technique to filter the solutions nominated in the previous step and keep only the ones that meet constraint  $C3$  too (the sliding window technique will be detailed next); finally *(iv)* optimize the obtained solution through comparing the solutions previously obtained (which meet the three constraints  $C1$ ,  $C2$ ,  $C3$ ) and choose the best solution that minimizes the objective function. We explain the sliding window technique through an example to improve legibility. Consider a multi-cast network consists of one sender  $S$  and three destinations  $D1$ ,  $D2$  and  $D3$ . The  $k$  shortest paths are computed for each pair  $S - D1$ ,  $S - D2$  and  $S - D3$ . The complete set of computed paths, denoted  $P$ , are sorted in an increasing order on an axis, see Figure 3.7, where  $T_{min}$  denotes the minimum value in  $P$  and  $T_{max}$  denotes the maximum value in  $P$ . A window with width  $DV$  is placed on the axis, where  $DV$  is the value of the delay variation allowed and adjusted in constraint  $C3$ , the lower edge of the window is  $T$  and the upper edge is  $T + DV$ . The initial value of  $T$  is  $T_{min}$ , then it is increased gradually with a unit step which causes the window to slide on the axis till the moment when the window upper edge meets  $T_{max}$ . The paths that fall within the sliding window are the ones that meet constraint  $C3$ . The problem will be transferred then to find a subnetwork that connects  $S$  with all the destinations from the selected options gained by the sliding window.



**Figure 3.7** — Illustrative figure for the sliding window technique

Inführ *et al.* [IR11] studied the mapping problem of virtual networks with the following demands: *(i)* the common properties of bandwidth, supplied by the SN links and demanded by the VN links; *(ii)* the common properties of CPU, supplied by the SN nodes and demanded by the virtual nodes; *(iii)* communication delay, that each SN link can transfer messages with a maximum delay, and each virtual link demands maximum delay for messages transfer; *(iv)* routing capacity, that the physical routers cannot rout the full BW connected to them, and *(v)* location constraints, where some/all virtual routers have possible placements constraints on the SN which limits the set of the candidates of the physical routers able to map them. The goal when mapping the VNs is to satisfy all the posed constraints while minimizing the cost of the physical components subsets chosen to map the VNs demands. For generating the benchmark-instances, the authors used real network topologies to model the physical networks instead of relying on random graphs, and they used different classes of virtual networks to model possible use-cases. Four different categories were used to represent cases in which VNs have different sets of requirements regarding BW, delay, and nodes CPU: *(i)* web slice for low BW requirements, short delays, and no specific CPU requirements, *(ii)*

*stream slice* for medium to high BW requirements, no delay bounds, and 3 processing units per routed bandwidth, (iii) *Peer-to-Peer (P2P) slice* for medium BW and CPU requirements, and no delay bounds, and (iv) *VoIP (Voice over Internet Protocol) slice* for medium BW and delay requirements, and high CPU requirements. Different topologies were used to simulate the VNs slices mentioned above, for example, the web slices were modeled by a star graph, where the central node and the leaf nodes represent the web server and the customers respectively. In the stream slice, the network was modeled with a random tree graph, where the tree root is the video source, the leaf nodes are the customers and the intermediate leaf nodes between the root and the leaf nodes split the stream and forward only the stream related to the channel being watched by the related customer. In the P2P slice and the VoIP slice, the network structure was generated by the `small_world_iterator` of the boost graph library. An example of the generated topologies in the four slices case is illustrated in Figure 3.8, the example is set for networks of 5 nodes. The mapping problem was formulated in the shape of a Mixed Integer Program and the MIP was solved using CPLEX [bmc14]. Basic findings of this work are: (i) finding the optimal solution for more than 74% instances was reachable in less than one hour; (ii) the biggest influence on the instance hardness was the topology map chosen to create the instance; and (iii) large problem instances were not harder to solve than smaller instances.



**Figure 3.8** — Examples of generated slices of size 5 [IR11]

In [GH16], Ghazisaeedi, et al. propose a novel energy-efficient embedding method, named *EnergyMap*, that maps heterogeneous MapReduce-based virtual networks onto a heterogeneous data center network, that also controls the incast queuing delay caused by incast traffic, where *MapReduce* is a cloud computing paradigm that is widely deployed in many data centers. According to the authors, the incast problem in Virtual Data Centers (VDCs) is different from the incast problem in non-virtualized (traditional) data centers. During a VDC embedding process, a specific amount of bandwidth capacity is allocated to each

virtual link in substrate paths. The traffic flows in the allocated paths are limited to their assigned bandwidth capacity. Hence, the incast problem might happen when the allocated substrate paths become congested, resulting in a longer queuing delay in the virtual link. According to the authors, this problem is significantly different from the case when incast happens only in a single bottleneck physical link of a non-virtualized data centers. EnergyMap method aims at finding a mapping for every VN such that the data center network's total energy consumption (by physical servers, physical switches/routers, and physical links) is minimized. the method also controls the incast queuing delay according to a given maximum tolerable queuing delay for a virtual link. The embedding approach allows embedding computation-based virtual nodes on multiple physical servers as they may need parallel processing, whereas other kind of virtual nodes are mapped on one single physical switch/router. In order to control the introduced incast queuing delay, it is required to find the end-to-end queuing delay for incast traffic pattern in the substrate path allocated to every virtual link, be it a link that terminates at a splitted or unsplit virtual node. The authors adopt the assumption that the physical nodes do not block the traffic, thus the reason for traffic delay would happen on the links because of limited bandwidth which are defined by the link bandwidth capacity. The end-to-end incast queuing delay in the substrate path could be calculated by knowing the amount of allocated traffic capacity to every virtual link that is mapped to a physical link composing this path. The incast queuing delay is influenced by: *i)* the choice for mapping the end virtual nodes, since the amount of bandwidth allocated to the virtual links are proportional to the assigned capacity of its end virtual nodes. And *ii)* the allocated substrate path to each of the virtual links adjacent to the end virtual node.

In order to control the introduced incast queuing delay, it is required to find the end-to-end queuing delay for incast traffic pattern in the substrate path allocated to every virtual link that terminates at a splitted and mapped reducer virtual node: - Most of today's switches/routers are internally non- blocking. Therefore, traffic can only be blocked by limited bandwidths of output ports which are defined earlier by the link bandwidth capacity. - We model the queue of an allocated bandwidth capacity to a virtual link in a substrate link by M/M/1 queue. - According to Jackson Networks theorem and because we do not split the generated traffic of an allocated virtual node, the end-to-end incast queuing delay in the substrate path could be calculated by knowing the amount of allocated traffic capacity to the virtual link in each physical link over the substrate path. - Since the amount of bandwidth we allocate to the virtual links are proportional to the assigned capacity of its end virtual nodes, the way we split reducer virtual nodes impacts the incast queuing delay. - Besides, the substrate node which we map the splitted reducer virtual node onto, and accordingly the allocated substrate path to each of the adjacent virtual links, also may influence the incast queuing delay. Clearly, this limits the level of freedom regarding energy-efficient embedding of the VNs, and may affect the energy saving rate. Following the above discussion, we can .

Detailed formulation is omitted due to the size limitation. The formulated MIDCP is a type of VN embedding problem. Simulation results show that the proposed approach *i)* saves energy more than any existing VNs energy-efficient embedding method that do not allow virtual node splitting; *ii)* controls the incast queueing delay; and *iii)* illustrates the influence of controlling the incast queueing delay on energy saving rates.

### 3.4.6 Miscellaneous requirements

Besides the above mentioned applications requirements, some works were concerned with:

**Routing requirements-** for example, [TEA11] addresses the problem of VNs embedding considering different SLA (the work details are given in 3.4.2). The authors propose a model in the shape of MIP, whose objective function aims at minimizing both the BW consumption and the routing cost. Another work [IR11] considers the routing capacity of the SN nodes, i.e., each physical node is considered with a maximum value of routing traffic, and the traffic passing through it should not pass this value (more details of this work can be found in 3.4.5).

**Location requirements-** where for certain virtual routers, subsets of the physical nodes only are candidates for the embedding. For example, virtual router  $X$  cannot be mapped on any of the SN nodes, rather, it can be mapped on one of the nodes of the subset  $X$  that is composed of 10 routers out of the SN nodes. This happens when clients request virtual networks to provide connectivity between two or more defined geographical locations. Among the works that considered location constraints is [BOB<sup>+</sup>12a] (more detailed in 3.4.3).

**Delay variation requirements -** Virtual multicast networks are expected to support many real-time applications, such as video-conferencing, distributed database replication, and online games. These applications require that packets are received by the destinations within a specified delay bounds, and the delay difference of packet receipts at multiple destinations should be minimal. An example of a system that is attentive to delay variation is a multi-cast system studied by [ZWJY10]. This system consists of one sender and several receivers, where a maximum delay variation is allowed between the times of receiving the sent message by each pair of receivers. Detailed about this work can be found in 3.4.5.

### 3.4.7 Synchrony requirements

In our work, we propose and argue that virtual networks and the virtual networks embedding process offer both abstractions and techniques to support applications with Hybrid Synchrony demands, as detailed in Section 2.5. To the authors best knowledge, this is undiscussed in the VN field and, as detailed in Chapter 2, of paramount importance to host a prominent class of distributed systems. This has led to the abstraction of new type of VNs,

we name it **The Hybrid Synchrony Virtual Networks**, abbreviated to **HSVNs**.

By revising the literature on the topic of VNs embedding; we note the absence of embedding solutions in the literature that consider the synchrony property in applications, which we need for our work to attend DSs with hybrid synchrony. This gap led us to the development of an embedding framework that handles applications with hybrid synchrony constraints.

### **What are HSVNs**

They are virtual networks that have subsets of nodes and links that obey time bounds for processing and communication. This abstraction put us to meet three main aspects associated: (i) the design of SN suitable for HSVNs, since VNs inherit properties that only exist in the underlying infrastructure, and (ii) suitable efficient embedding process for the HSVN.

Although HSVN can run on fully synchronous SN, this decision would have to pay the excess in an unneeded cost, since even asynchronous virtual nodes and links will be mapped on synchronous physical ones. We argue that hybrid synchronous SN, combined with a suitable embedding, is capable to answer the synchrony requirements in an economic manner. Hybrid synchronous SNs have two classes of nodes: (i) *synchronous nodes* with functioning time guarantees, achieved through the implementation of periodical real-time tasks, and (ii) *asynchronous nodes* that have no timely guarantees. Analogously, two classes of physical links are available: (i) *synchronous links* that have time-bounded messages transmission delay, achieved through the implementation of QoS policies and admission control, and (ii) *asynchronous links* that have no timely guarantee.

Two types of HSVNs can be distinguished, inspired by the two types of hybrid DSs (see 2.3.1 and 2.3.2):

1. Space-HSVNs: where the virtual networks are composed of synchronous and asynchronous components, where both types of components maintain their synchrony status during the system functionality. The Space-HSVNs will be discussed in details in a separated chapter, that is Chapter 4.
2. Time-HSVNs: where the virtual networks are composed of subsets of nodes and links that change their synchrony status over time (i.e., synchronous resources become asynchronous and vice versa). The Time-HSVNs will be discussed in details in a separated chapter, that is Chapter 6.

**Table 3.1** — Taxonomy of works on virtual networks mapping

Main constraint	work
<b>Topology requirements</b>	<ul style="list-style-type: none"> <li>- Zhu et al., 2006, Algorithms for assigning substrate network resources to virtual network components [ZA06]</li> <li>- Lu et al., 2006, Efficient mapping of virtual networks onto a shared substrate [LT06]</li> <li>- Yu et al., 2008, Rethinking virtual network embedding: Substrate support for path splitting and migration [YYRC08b]</li> <li>- Li et al., 2015, The study of Dynamic Topology Remapping in Virtual Network Embedding [LZW15]</li> </ul>
<b>BW and CPU constraints</b>	<ul style="list-style-type: none"> <li>- Trinh et al., 2011, Quality of service using careful overbooking for optimal virtual network resource allocation [TEA11]</li> <li>- Botero et al., 2013 Optimal mapping of virtual networks with hidden hops [BHFdM13]</li> <li>- Hsu et al., 2012, Virtual Network Mapping Through Path Splitting and Migration [HSWY12]</li> </ul>
<b>Security requirements</b>	<ul style="list-style-type: none"> <li>- Bays et al., 2012, Security-aware Optimal Resource Allocation for Virtual Network Embedding [BOB<sup>+</sup>12a]</li> <li>- Wang et al., 2016, Secure virtual network embedding to mitigate the risk of covert channel attacks [WWG<sup>+</sup>16]</li> </ul>
<b>Resilience requirements</b>	<ul style="list-style-type: none"> <li>- Yu et al., 2011, Cost efficient design of survivable virtual infrastructure to recover from facility node failures [YAQS11]</li> <li>- Rahman et al., 2013, SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization [RB13]</li> <li>- Chen et al., 2010, Resilient virtual network service provision in network virtualization environments [CLW<sup>+</sup>10]</li> <li>- Yeow et al., 2010, Designing and embedding reliable virtual infrastructures [YWK10]</li> <li>- Houidi et al., 2010, Adaptive virtual network provisioning [HLZ<sup>+</sup>10]</li> <li>- Ghaleb et al., 2016, Surviving Multiple Failures in Multicast Virtual Networks with Virtual Machines Migration [GKA<sup>+</sup>16b]</li> </ul>
<b>Delay requirements</b>	<ul style="list-style-type: none"> <li>- Zhang et al., 2010, Mapping multicast service-oriented virtual networks with delay and delay variation constraints [ZWJY10]</li> <li>- Infuhr et al., 2011 Introducing the virtual network mapping problem with delay, routing and location constraints [IR11]</li> <li>- Ghazisaeedi et al., 2016, EnergyMap: Energy-efficient embedding of MapReduce-based virtual networks and controlling incast queuing delay [GH16]</li> </ul>
<b>Routing requirements</b>	<ul style="list-style-type: none"> <li>- Trinh et al., 2011, Quality of service using careful overbooking for optimal virtual network resource allocation [TEA11]</li> <li>- Infuhr et al., 2011, Introducing the virtual network mapping problem with delay, routing and location constraints [IR11]</li> </ul>
<b>Location requirements</b>	<ul style="list-style-type: none"> <li>- Bays et al., 2012, Security-aware Optimal Resource Allocation for Virtual Network Embedding [BOB<sup>+</sup>12a]</li> </ul>
<b>Delay variation requirements</b>	<ul style="list-style-type: none"> <li>- Zhang et al., 2010, Mapping multicast service-oriented virtual networks with delay and delay variation constraints [ZWJY10]</li> </ul>

### 3.5 Review on the VNE difficulties

After surveying existing work on VNs resource allocation in the literature, we identify the following major difficulties (not limited to them):

- problem complexity: which scales with the increment of the problem size, be it with the increment in the VNS and/or the SN size, or be it the constraints that need to

be considered while the mapping process. For example, mapping virtual networks with topology constraints alone is less complex than mapping VNs with topology, CPU and BW constraints. The problem complexity affects directly the computational time, making the process of finding the optimal solution for VNs mapping demands long time. For example, in [BOB<sup>+</sup>12a], mapping some VNs demanded time in the order of 24 hours. This situation motivated the development of heuristics for VNs mapping, for example [ZA06]. Even though heuristics speed up the embedding computation time, but this is with the cost of not reaching the optimal solution. Yet, for some applications, a quick semi-optimal VNs mapping is more preferable than a slow optimal one;

- online VNs mapping: this is when the VNs are born, remain for certain time, then after they are dead. This cycle demands allocating physical resources for certain time interval, then after, free these resources once the VNs are dead, in order to be able to use them again. The complexity of this process increases in case it involved remapping the alive already mapped VNs according to the currently available resources, which is changeable with the flow on the VNs arrival online;
- dynamic VNs demands: this is when the VNs *demands* change over time (and not the VNs *number*, which is the previous item). In this case, the initial VNs mapping should be done considering long-terms characteristics (mapping in worst case), later on, the initial mapping will be adapted to shorter-terms based on the current requirements. This adaptive mapping is done in order to minimize the mapping cost, because the initial mapping (mapping in worst case) might result in reserving resources that become not used hereafter with the change of the VNs demands. Possible way to achieve this adaptive mapping is to apply the principles of the feed-back control theory.

## Summary

In this chapter we reviewed the literature on the topic of networks virtualization: the VNs definition, VNs properties, and the VNs embedding problem (VNE). We classified the works on the VNE field based on the application constraints/requirements, for example, topology, security, and resilience constraints.

By revising the literature on the topic of VNE, we note the absence of embedding solutions in the literature that consider the synchrony property in applications, which is of paramount importance to host a prominent class of distributed systems, the hybrid synchronous DSs, as formerly detailed in Chapter 2. This gap led us to the development of an embedding framework that handles applications with hybrid synchrony constraints, as will be detailed in the next two chapters.



---

In Chapter 4, we will address the Space-HSVNs, they are virtual networks with hybrid synchrony in space, addressed to HSDS in space, and we will propose an embedding model to handle the resources allocation problem in this kind of VNs. Analogously, in Chapter 6, we will address the Time-HSVNs, they are virtual networks with hybrid synchrony in time, addressed to HSDS in time, and we will propose an embedding framework to handle the resources allocation problem in this kind of VNs.

# 4 Space-HSVNs embedding

In this chapter we define the Hybrid Synchronous Virtual Networks in Space (Space-HSVNs), we propose a suitable embedding model for two types of Substrate Network (SN): *i*) Settled SN (S-SN) and *ii*) configurable SN (C-SN). We start this chapter by defining the Space-HSVNs, then we locate our work among others in the literature, finally we propose an embedding model for Space-HSVNs on both S-SN and C-SN.

## 4.1 Space-HSVNs: definition

Formerly in this thesis, Section 2.3.1, we referred to distributed systems that demand synchrony in space during the system life, in other words, they demand certain elements (i.e., nodes and links) to behave synchronously while others may behave asynchronously during the system life. We assume that this kind of systems are supported by virtual networks that reflect the space synchrony nature. We name this type of VNs the Space Hybrid Synchronous Virtual Networks, abbreviated to Space-HSVNs.

The Space-HSVNs are virtual networks that carry all the common properties of VNs [CB10], but in addition, they have subsets of nodes and links that obey time bounds for processing and communication. Both types of components maintain their synchrony status during the system functionality, i.e., the synchronous nodes and links remain synchronous during the system execution time, and the asynchronous nodes and links remain asynchronous (provide no synchrony guarantees). The Space-HSVNs are addressed to host distributed systems with hybrid synchrony in space.

## 4.2 Work positioning in the literature

Revising the literature in the topic of VNs mapping, we find that our work is nearer to those who were concerned with delay constraints, see Section 3.4.5. For example, Zhang *et al.* [ZWJY10] propose a heuristic algorithm for mapping virtual multi-cast service-oriented networks subject to delay and delay variation. They consider SNs composed of links with maximum delay. Their work benefits real-time and interactive applications, where packets

are supposed to be received at the destination within specific time bounds, and the delay difference of packets reception at multiple destinations should be minimal, besides the aim of minimizing the mapping cost, where the cost is defined as the sum of total substrate resources (e.g. CPU and BW) allocated to the multi-cast network. A sliding window method is proposed to construct a set of feasible paths and solve the mapping problem based on the feasible paths with the goal of minimizing the cost and load balancing, where the cost is defined as the sum of total substrate resources (e.g. CPU and BW) allocated to the multi-cast network.

Inführ *et al.* [IR11] addressed the VNs mapping problem with delay constraints besides routing and location constraints. The SN considered is composed of links with maximum delay, and nodes that have maximum routing capacity and location constraints. Four different categories were used to represent cases in which VNs have different sets of requirements regarding BW, delay, and nodes CPU: (1) *web slice* for low BW requirements, short delays, and no specific CPU requirements, (2) *stream slice* for medium to high BW requirements, no delay bounds, and 3 processing units per routed bandwidth, (3) *P2P slice* for medium BW and CPU requirements and no delay bounds, and (4) *VoIP slice* for medium BW and delay requirements, and high CPU requirements.

The study we present about space-HSVNs is distinct from the aforementioned works in the following aspects: (1) we consider the delay constraints (or time bounds) on both links and nodes, not only links, since in the considered class of DSs some links should have time guarantees in delivering the messages, and some nodes should be performing real-time tasks; (2) a physical path is considered synchronous not only when its links are synchronous, rather the path's intermediate nodes should be all synchronous as well, since they play role in the routing process, impacting the source-destination delay; (3) the mapping model we propose aims at optimizing the usage of the synchronous resources whose building cost is high comparatively. For example, some VNs slices adopted in [IR11] had no delay requirements, yet the SN considered had no distinction in kind of resources, which results in an unneeded cost, and (4) unlike other works, the SN we consider is hybrid in its components synchrony. Some nodes and links have time bounds and others do not, which is suitable for DSs applications that have hybrid synchronous requirements.

### 4.3 The Substrate Network for space-HSVNs

To map VNs with hybrid synchrony requirements, the Substrate Network (SN) should be hybrid synchronous as well. The reason is that, the VNs cannot reach any functionality unless exists in the underlying infrastructure, from where they inherit it [CB09].

We assume the existence of certain mechanisms that guarantee building physical network

elements (nodes and links) that behave synchronously. These mechanisms can be related to the type of physical materials used, or to the procedures followed for configuring them, such as admission control and Quality of Service (QoS) policies. Studying the exact mechanisms for building synchronous resources is out of the scope of our work.

We distinguish between two types of SNs that provide hybrid synchrony:

**Settled Substrate Network (S-SN):** has two classes of nodes; (i) *synchronous nodes* with functioning time guarantees, achieved through the implementation of periodical real-time tasks, and (ii) *asynchronous-nodes* that have no time-bounded guarantees. Analogously, two classes of physical links are available: (i) *synchronous links* that have time-bounded messages transmission delay, achieved through the implementation of Quality of Service (QoS) policies and (ii) *asynchronous links* that have no time-bounded guarantee. In a settled SN, the physical resources synchrony status is static, predefined, independently of the virtual networks demands. In the embedding model detailed later, the synchrony state of a S-SN components is an input parameter of the model.

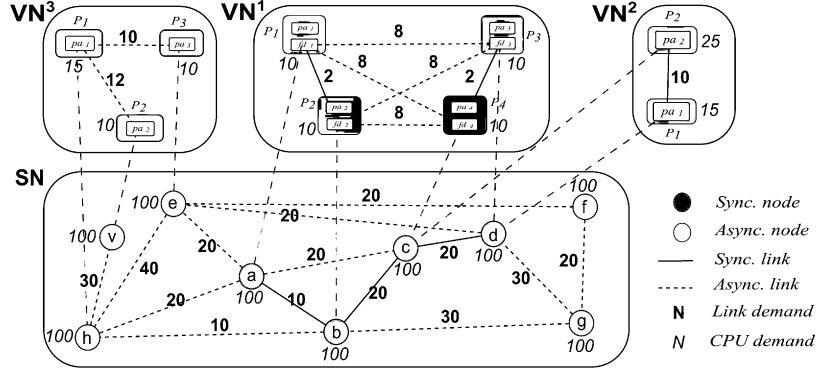
**Configurable Substrate Network (C-SN):** where the SN components (i.e., nodes and links) have no synchrony orientation initially, and both can receive configuration process that turns them to become synchronous or asynchronous. In a configurable SN, the physical resources synchrony status is dynamic, dependent of the virtual networks demands. In this case, we consider the synchrony demands on the HSVNs as an input for the embedding model. With this information, the embedding model determines a sufficient number of physical synchronous components, and their location on the SN, so that the VNs requirements are satisfied.

We don't claim that one type of a SN (i.e., S-SN or C-SN) is more important than the other type, because both are of equal necessity. In fact, the infrastructure provider is the player who determines which embedding model is to be adopted, based on the SN type he built (i.e., settled or configurable).

## 4.4 Graph based example for Space-HSVNs embedding

After characterizing both the Space-HSVNs and the SN suitable to support the DSs with hybrid synchrony in space, we drive our effort to the resources allocation problem. How to map the Space-HSVNs on both (i) a settled substrate network (S-SN) and on (ii) a configurable substrate network (C-SN)? The rest of this chapter will be about proposing a suitable embedding model for both cases. Yet, at this stage of the thesis we present a simple graph-based example to see Space-HSVNs mapped on top of a hybrid synchronous SN in space (be it S-SN or C-SN). We will just present the illustrative graph, to allow the reader to grasp the concept, and we will not enter now in details about the way for solving the

embedding problem.



$VN^1$ links	$SN$ path	$VN^2$ links	$SN$ path	$VN^3$ links	$SN$ path
$(fd_1, fd_2)$	$(a, b)$	$(pa_1, pa_2)$	$(d, c)$	$(pa_1, pa_2)$	$(h, v)$
$(fd_1, fd_3)$	$(a, e), (e, d)$			$(pa_1, pa_3)$	$(h, e)$
$(fd_1, fd_4)$	$(a, c)$				
$(fd_2, fd_3)$	$(b, g), (g, d)$				
$(fd_2, fd_4)$	$(b, h), (h, a), (a, c)$				
$(fd_3, fd_4)$	$(d, c)$				

Figure 4.1 — Illustrates figure for Space-HSVNs embedding

## 4.5 Space-HSVNs embedding model on S-SN

Bellow we introduce the variables definition, followed by the mathematical embedding model.

### 4.5.1 Variables definition

The substrate network is represented by an undirected graph  $G(N, L)$ , composed of a finite set of physical nodes  $N$  connected through a finite set of physical links  $L : NXN$ . The set  $N$  is given by  $N_s \cup N_a$ , where  $N_s$  and  $N_a$  contain all the synchronous and asynchronous SN nodes, respectively. Similarly,  $L$  is given by  $L_s \cup L_a$ .

The virtual network number  $k$  is given by  $VN^k$ , belonging to the finite set of virtual networks  $VN$ .  $VN^k$  will be presented by an undirected graph  $G^k(N^k, L^k)$ , where  $N^k = N_s^k \cup N_a^k$  and  $L^k = L_s^k \cup L_a^k$ .

We consider that there is a cost  $c(i, j)$  for one unit of traffic going through the physical link  $(i, j) \in L$ . Analogously,  $c(i)$  is the cost for processing one unit of traffic in node  $i \in N$ .  $c(i, j)$  and  $c(i)$  are of higher value if the link and node were synchronous. Synchronous physical resources are more costly than the asynchronous ones. We turn this trade off in our

work such a way that we consider the cost of passing one unit of traffic in a synchronous link and the cost of processing it in a synchronous node is ten times more costly than doing that on asynchronous node and link.

A binary function  $sync(i)$  expresses the SN nodes synchrony:  $sync(i) = 1$  if  $i \in N_s$ , otherwise  $sync(i) = 0$  (i.e.,  $i \in N_a$ ). Similarly,  $sync(i, j)$  expresses the SN links synchrony. Functions  $sync(i^k)$  and  $sync(i^k, j^k)$  indicate the virtual nodes and links synchrony respectively ( $i^k \in N^k$  and  $(i^k, j^k) \in L^k$ ).

Besides synchrony, two other attributes are considered for the SN and VN elements: nodes  $CPU$ , and links bandwidth ( $BW$ ). The syntax for those attributes on the SN and VN respectively are:  $cpu(i)$ ,  $bw(i, j)$ ,  $cpu(i^k)$ , and  $bw(i^k, j^k)$ .

Finally, we define the output variables for our mathematical model. The values of these variables illustrate the allocation of the physical resources to the virtual demands, they are: a binary function  $\sigma(i^k, i)$  that expresses whether node  $i \in N$  maps node  $i^k \in N^k$ , and a binary function  $\rho(i^k, j^k, i, j)$  that expresses whether the physical link  $(i, j) \in L$  is part of the path that maps the virtual link  $(i^k, j^k) \in L^k$ .

Table 4.1 provides a list of variables definition for Space-HSVNs embedding model.

## 4.5.2 Embedding model

We formulate the space-HSVNs mapping problem in the shape of an Integer Program (IP) [Sie02], consists of two blocks: the objective function and the embedding constraints as detailed bellow.

**Mapping objective: minimize**

$$\begin{aligned} & \sum_{\forall VN^k \in VN} \sum_{\forall (i^k) \in N^k} \sum_{\forall (i) \in N} (\sigma(i^k, i) \cdot c(i) \cdot cpu(i^k)) \\ & + \sum_{\forall VN^k \in VN} \sum_{\forall (i^k, j^k) \in L^k} \sum_{\forall (i, j) \in L} (\rho(i^k, j^k, i, j) \\ & \cdot c(i, j) \cdot bw(i^k, j^k)) \end{aligned} \quad (4.1)$$

The Objective Function (O.F.) we consider is inspired from [ZWJY10], which is to minimize the total resources used (e.g., BW and CPU). We modify the O.F. with the goal of minimizing the use of synchronous resources besides the BW and CPU. For this purpose,  $c(i)$  and  $c(i, j)$  are inserted as in Equation (4.1). The mapping constraints (detailed next) will allow mapping the synchronous virtual demands on top of synchronous physical resources, and asynchronous virtual demands on top of either synchronous or asynchronous physical resources prioritizing asynchronous physical resources which is guaranteed by this O.F. In other words, mapping asynchronous virtual demands on top of synchronous physical resources will be considered as the last resource invested only before rejecting the demand.

**Mapping constraints-**

**Table 4.1** — List of variables definition for Space-HSVNs embedding model

Variables group	symbol	description
<b>Substrate Network</b>	$G(N, L)$	undirected graph representing the SN
	$N$	the set of physical nodes
	$N_s$	the set of physical synchronous node
	$N_a$	the set of physical asynchronous node
	$L : NXN$	the set of physical links
	$L_s$	the set of physical synchronous links
	$L_a$	the set of physical asynchronous links
	$i$	a notation for a physical node $i \in N$
	$(i, j)$	a notation for a physical link $(i, j) \in L$
	$sync(i)$	the synchrony of physical node $i$ $sync(i) = 1$ if $i \in N_s$ $sync(i) = 0$ if $i \in N_a$
	$sync(i, j)$	the synchrony of physical link $(i, j)$ $sync(i, j) = 1$ if $(i, j) \in L_s$ $sync(i, j) = 0$ if $(i, j) \in L_a$
	$cpu(i)$	the CPU of physical node $i$
	$bw(i, j)$	the bandwidth of physical link $(i, j)$
	$c(i)$	the cost for processing one unit of traffic in node $i \in N$
$c(i, j)$	the cost for one unit of traffic going through the physical link $(i, j) \in L$	
<b>Virtual Networks</b>	$VN$	the set of all virtual networks
	$k$	the number of a virtual network that belongs to $VN$
	$VN^k$	The virtual network number $k$
	$G^k(N^k, L^k)$	undirected graph representing $VN^k$
	$N^k$	the set of virtual nodes of $VN^k$
	$N_s^k$	the set of virtual synchronous nodes
	$N_a^k$	the set of virtual asynchronous nodes
	$L^k : N^k X N^k$	the set of virtual links of $VN^k$
	$L_s^k$	the set of virtual synchronous links
	$L_a^k$	the set of virtual asynchronous links
	$i^k$	a notation for a virtual node $i^k \in N^k$
	$(i^k, j^k)$	a notation for a virtual link $(i^k, j^k) \in L^k$
	$sync(i^k)$	the synchrony of virtual node $i^k$ $sync(i^k) = 1$ if $i^k \in N_s^k$ $sync(i^k) = 0$ if $i^k \in N_a^k$
	$sync(i^k, j^k)$	the synchrony of virtual link $(i^k, j^k)$ $sync(i^k, j^k) = 1$ if $(i^k, j^k) \in L_s^k$ $sync(i^k, j^k) = 0$ if $(i^k, j^k) \in L_a^k$
	$cpu(i^k)$	the CPU of virtual node $i^k$
	$bw(i^k, j^k)$	the bandwidth of virtual link $(i^k, j^k)$
<b>Output variables</b>	$\sigma(i^k, i)$	$\sigma(i^k, i) = 1$ ; node $i \in N$ maps node $i^k \in N^k$ $\sigma(i^k, i) = 0$ ; otherwise
	$\rho(i^k, j^k, i, j)$	$\rho(i^k, j^k, i, j) = 1$ ; $(i, j) \in L$ is part of the path mapping $(i^k, j^k) \in L^k$ $\rho(i^k, j^k, i, j) = 0$ ; otherwise

- *Capacity constraints:*

for every  $(i, j) \in L$

$$\sum_{\forall VN^k \in VN} \sum_{\forall (i^k, j^k) \in L^k} \rho(i^k, j^k, i, j) \cdot bw(i^k, j^k) \leq bw(i, j) \quad (4.2)$$

for every  $i \in N$

$$\sum_{\forall VN^k \in VN} \sum_{\forall i^k \in N^k} \sigma(i^k, i) \cdot cpu(i^k) \leq cpu(i) \quad (4.3)$$

- *Nodes mapping constraints:*

for every  $VN^k \in VN, i^k \in N^k$

$$\sum_{\forall i \in N} \sigma(i^k, i) = 1 \quad (4.4)$$

for every  $VN^k \in VN, i \in N$

$$\sum_{\forall i^k \in N^k} \sigma(i^k, i) \leq 1 \quad (4.5)$$

- *Links mapping constraint:*

for every  $VN^k \in VN, (i^k, j^k) \in L^k, i \in N$

$$\sum_{\forall j \in N} \rho(i^k, j^k, i, j) - \sum_{\forall j \in N} \rho(i^k, j^k, j, i) = \sigma(i^k, i) - \sigma(j^k, i) \quad (4.6)$$

- *Nodes synchrony constraints:*

for every  $VN^k \in VN, i^k \in N^k, i \in N$

$$sync(i^k) \cdot \sigma(i^k, i) \leq sync(i) \quad (4.7)$$

- *Links synchrony constraints:*

for every  $VN^k \in VN, (i^k, j^k) \in L^k, (i, j) \in L$

$$sync(i^k, j^k) \cdot \rho(i^k, j^k, i, j) \leq sync(i, j); \quad (4.8)$$

for every  $VN^k \in VN, (i^k, j^k) \in L^k, (i, j) \in L$

$$sync(i^k, j^k) \cdot \rho(i^k, j^k, i, j) \leq sync(i) * sync(j); \quad (4.9)$$

The capacity constraint (4.2) assures that the total bandwidth of the virtual links, mapped on paths that include a certain physical link, does not exceed the bandwidth capacity of this physical link. Similarly, constraint (4.3) represents the equivalent restriction regarding nodes *CPU*.

The node mapping constraint (4.4) assures that each virtual node is mapped, and only once, on a physical node. Without this constraint, and since the O.F. aims at minimizing cost, then the optimizer might choose not to map any node, which is against the goal.

Constraint (4.5) assures that virtual nodes belonging to the same *VN* are not mapped on the same physical node. This is to achieve load balancing besides improving the reliability, since the unavailability of a SN node will impact, at most, one node on a given *VN*. This procedure minimizes the number of virtual nodes prone to failure by a physical node failure.



For any virtual link  $(a, b)$ , the links mapping constraint (4.6), adopted in [BOB<sup>+</sup>12a] and [ZWJY10], assures the creation of a valid physical path. Because the right side of the equation will be 1 and -1 for  $a$  and  $b$  respectively, meaning  $a$  will have an outgoing arc and  $b$  an ingoing one. For all other nodes on the SN, the right side of the equation will be zero, thus the concatenation of arcs will form a valid path.

The nodes synchrony constraint (4.7) assures that synchronous virtual nodes are mapped only on synchronous SN nodes, whereas asynchronous virtual nodes are allowed to be mapped on synchronous or asynchronous SN nodes. This is acceptable because the synchronous SN nodes supply what the asynchronous ones do, but the reverse is not valid.

Similarly, the links synchrony constraint is presented in (4.8). Note that the allocation of synchronous physical resources for asynchronous virtual demands is done only if there are no other possible options (physical asynchronous resources got exhausted). This is achieved via minimizing the O.F. Finally, constraint (4.9) guarantees that when the intermediate physical nodes on the synchronous physical path should be also synchronous. This is because these nodes play role in the routing process, thus impacting the source-destination delay. After solving the mathematical model, each virtual node is mapped to one physical node, and each virtual link is mapped to one physical path at maximum, where a physical path can be a unique physical link or a concatenation of physical links.

## 4.6 Space-HSVNs embedding model on C-SN

Similarly to S-SN, we introduce the variables definition, followed by the embedding model.

### 4.6.1 Variables definition

The variables considered are the same detailed for the S-SN (see subsection 4.5.1), but the SN synchrony is no more an input parameter to the embedding model, it is defined by the model output. In other words,  $sync(i)$  and  $sync(i, j)$  are output variables of the embedding model, besides  $\sigma(i^k, i)$  and  $\rho(i^k, j^k, i, j)$ . So, after solving the mathematical model, each virtual node will be mapped on a physical node, and the synchrony of this physical node will be determined, and each virtual link will be mapped on a physical path, and the synchrony of the physical links that form this physical path will be determined. Determining the synchrony status of the physical nodes/links can be achieved, for example, by switching on/off a configuration technique able to configure the nodes/links as synchronous/asynchronous.

## 4.6.2 Embedding model

The mapping constraints are the same ones considered for a S-SN (see subsection 4.5.1), whereas the objective function is different. As aforementioned, the synchrony demands on the HSVNs are considered as an input for the embedding process. With this information, the embedding process determines a sufficient number of synchronous components, and their location on the SN, so that the VNs requirements are satisfied. The mapping objective aims at minimizing the synchronous resources made available by the SN, which is expressed mathematically by the O.F. in (4.10).

**C-SN Objective: minimize**

$$\sum_{\forall VN^k \in VN} \sum_{\forall (i^k, j^k) \in L^k} \sum_{\forall (i, j) \in L} (\rho(i^k, j^k, i, j) \cdot bw(i^k, j^k)) + \sum_{\forall i \in N} sync(i) + \sum_{\forall (i, j) \in L} sync(i, j) \quad (4.10)$$

## Summary

In this Chapter we defined the Space-HSVNs. They are new kind of VNs that have synchronous subsets of nodes and links that obey time bounds for processing and communication. Both types of components maintain their synchrony status during the system functionality. The Space-HSVNs are addressed to host distributed systems with hybrid synchrony in space.

The SN that supports Space-HSVNs is hybrid synchronous. We distinguish between two types of SNs that provide hybrid synchrony: *i*) Settled Substrate Network (S-SN) where the physical resources synchrony status is static, predefined, independently of the virtual networks demands. And *ii*) Configurable Substrate Network (C-SN): where the SN components (i.e., nodes and links) have no synchrony orientation initially, and both can receive configuration process that turns them to become synchronous or asynchronous. In a configurable SN, the physical resources synchrony status is dynamic, dependent of the virtual networks demands.

We treat the embedding problem for Space-HSVNs embedding with the two cases: *i*) Space-HSVNs on S-SN, and *ii*) Space-HSVNs on C-SN. The mapping problem was formulated in the shape of an Integer Program for the two aforementioned cases.

In the next Chapter we evaluate the two proposed model for Space-HSVNs embedding.

---

# Performance Evaluation for Space-HSVN

In this section, we run experiments that allow investigating the performance of the proposed embedding approach of Space-HSVNs over both a settled SN (S-SN) and a configurable SN (C-SN).

## 5.1 Space HSVNs over a settled SN

The parameters considered for the analysis of our model are: (i) the embedding cost; (ii) the physical resources load; and (iii) the embedding time.

### 5.1.1 Workload and tools

The experiments scenarios were designed as a full factorial [Jai91], exploring all possible combinations between the network parameters. Such choice of experiments was done by other works like [BOB<sup>+</sup>12a]. Similar to [YYRC08a, BOB<sup>+</sup>12a], physical and virtual networks were randomly generated. For this we used the BRITE tool (Boston university Representative Internet Topology generator) [MLMJ14] with Waxman model [Wax88]. We implemented the mathematical model with the ZIMPL language (Zuse Institute Mathematical Programming Language) [Koc04], and we used the CPLEX optimizer [bmc14] to solve the Integer Program (IP), running on a DELL XPS 8500, processor Intel(R), Core(TM) i5-330P, CPU 3.10 GHZ (Giga Hertz), Random Access Memory (RAM) 8.00 GB (Giga Byte), and operating system (OS) Windows 8, 64 bits. Table 5.1 details the parameters of the twelve scenarios we considered, divided into four groups: group A, B, and C.

In all the scenarios, the substrate network size was fixed to 25 nodes. Initially all CPUs (Central Processing Unit) of SN nodes are free, and links Band Width (BW) is uniformly distributed between 1-3 Gbps (Giga bit per second). 33% and 34% of the physical nodes and links respectively were fixed to be synchronous on the SN.

We ran scenarios divided into three groups, A, B and C, with VNs total size of 12, 24, and 36 nodes in each group respectively. The VNs were generated with 3, 4, or 5 nodes each, with CPU demands 10%, 15%, or 25% of the SN nodes CPU capacity, and BW demands

**Table 5.1** — Space-HSVNs with S-SN scenarios parameters

scenario	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
<b>VN size</b>	12 nodes				24 nodes				36 nodes			
<b>VN sync.</b>	0%	30%	60%	100%	0%	30%	60%	100%	0%	30%	60%	100%
<b>each VN size</b>	3,4,5 nodes											
<b>VNs BW</b>	uniformly distributed: 100Mbps-1Gbps											
<b>VNs CPU</b>	10,15,25 % of SN nodes CPU											
<b>SN size</b>	25 nodes											
<b>SN sync.</b>	33% of the nodes and 34% of the links											
<b>SN BW</b>	uniformly distributed: 1 Gbps-3 Gbps											
<b>SN CPU</b>	nodes fully free initially											

uniformly distributed between 100 Mbps (Mega bit per second) and 1 Gbps.

The SN and VNs size was chosen of small scale to allow solving the embedding model during a reasonable time, allowing us to evaluate the model performance.

VNs synchrony demands were increasing from scenario 1 to scenario 4 of each group:

- VNs of 0% synchrony demands in scenario 1 of each group (i.e., experiments A1, B1, C1);
- VNs of 30% synchrony demands in scenario 2 of each group (i.e., experiments A2, B2, C2);
- VNs of 60% synchrony demands in scenario 3 of each group (i.e., experiments A3, B3, C3);
- VNs of 100% synchrony demands in scenario 4 of each group (i.e., experiments A4, B4, C4).

To perform a suitable analysis, each of the twelve scenarios (A1, ..., A4, B1, ..., B4, C1, ..., C4) of Table 5.1 had its confidence level and interval calculated. Each scenario was repeated 10 times, randomly generating both the SN and the VNs. For example, scenario A1 was repeated ten times (A10, A11, A12, ... A19), scenario A2 was repeated ten times (A20, A21, A22, ... A29). The same logic was applied to scenarios A3, A4, B1, B2, B3, B4, C1, C2, C3, and C4.

Within the same group (A, B, and C), the SNs and VNs generated were the same, but the VNs synchrony was different. For example, experiments A10, A11, ... A19 are with distinct SN and VNs, whereas A10, A20, A30, and A40 are with the same SN and VNs topologies and attributes except for the synchrony demands which are 0%, 30%, 60% and 100% in counterparts experiments respectively.

Experiments that reached an optimization gap that is less than 1% were interrupted. Similar decision was taken by other researchers, for example in [BOB<sup>+</sup>12a] researchers stopped the experiments that took more than 24 hours. For some applications, it might be that

reaching a semi-optimal embedding solution within a quick optimization time is more interesting than reaching an optimal embedding solution during too long time. The kind of application and/or the clients requirements are the players for determining whether to stop the optimization process or not. In our work, the majority of experiments reached the optimal solution, and we interrupted the optimization process for few experiments when the optimization gap became less than 1%.

With ten readings for every scenario in Table 5.1, we achieved a total of 120 experiments. During this chapter, we will use the term “scenario” referring to the twelve categories that are in Table 5.1, and we will use the term “experiment” referring to the 120 experiments that we ran (A10, A11, ..... C49).

A 90% confidence interval was then calculated for every scenario of the twelve in Table 5.1. Equation 5.1 quoted from [LK00] was used to calculate a  $100(1 - \alpha)$  percent of confidence interval

$$\bar{X}(n) \mp (t_{n-1,1-\alpha/2})(\sqrt{(S^2(n))/n}) \quad (5.1)$$

Where:

- $\bar{X}(n)$  is the average reading for  $n$  observations;
- $(1 - \alpha)$  is the area of a density function for the standard normal distribution, limited between the two critical points;
- $t_{n-1,1-\alpha/2}$  is the upper  $1 - \alpha/2$  critical point for the  $t$  distribution with  $n - 1$  observations. These critical points are given in Table T.1 of the Appendix at the back of the book [LK00]. In our study, for 10 observations and with a 90% confidence interval, the value of the critical point  $t_{n-1,1-\alpha/2}$  is equal to 1.83;
- $S^2(n)$  is the sample variance calculated by Equation 5.2

$$S^2(n) = \frac{\sum [X_i - \bar{X}(n)]^2}{n - 1} \quad (5.2)$$

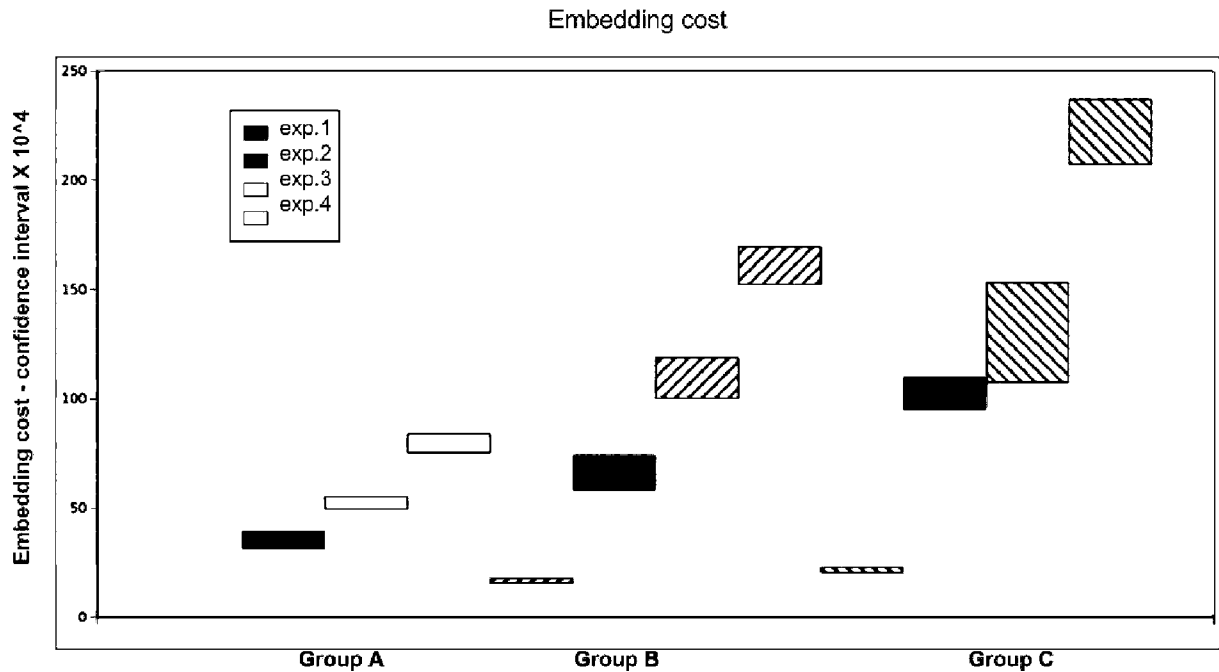
### 5.1.2 Results

The parameters considered for the analysis of our model are: (i) the embedding cost; (ii) the embedding time; and (iii) the physical resources load.

### 5.1.2.1 Embedding cost

The mapping cost, is a combination of the CPU and BW used. It is expressed by the model objective function, Equation (4.1).

Figure 5.1 depicts the interval of mapping cost for each of the twelve scenarios performed with 90% confidence level. We note down the following main observations:



*Figure 5.1* — Embedding cost - Confidence interval

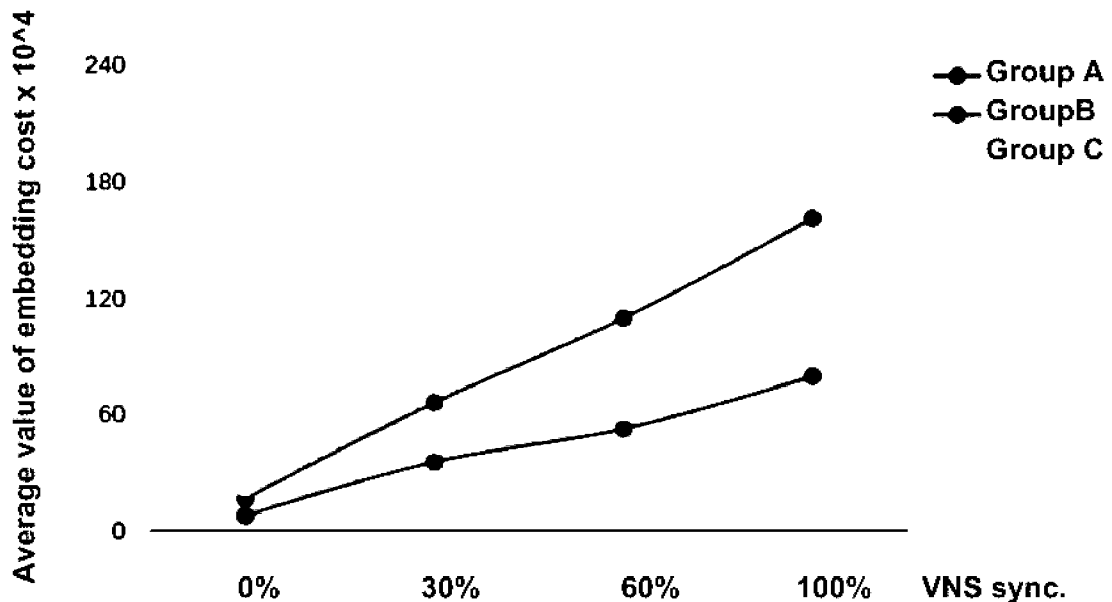
(i) We do not notice overlaps neither in the intervals of the four scenarios within same group (e.g., scenarios A1, A2, A3, and A4), nor within the counterpart scenarios in the groups (e.g., scenarios A2, B2, and C2). This means that the confidence and interval levels chosen differentiate well the scenarios and allow to draw conclusions.

(ii) Within each group, an increment of the VNs synchrony requests results in an increment in the mapping cost, for example, the cost interval for scenario C1 is  $[20 \times 10^4, 23 \times 10^4]$ , for C2 is  $[95 \times 10^4, 110 \times 10^4]$ , for C3 is  $[107 \times 10^4, 153 \times 10^4]$  and for C4 is  $[207 \times 10^4, 237 \times 10^4]$ . This behavior is because the increase in the synchrony demands leads to the increase in the synchronous physical resources needed to map these demands. And since the cost of the synchronous physical resources is higher in comparison with the asynchronous resources, then the total embedding cost will increase. To define more accurately the increment pattern of the embedding cost with the increment in VNs synchrony demands, we calculate the average value of embedding cost for every scenario, see Table 5.2.

Figure 5.2 depicts the average values for the embedding cost versus the VNs synchrony demands. We note down the following observations:

**Table 5.2** — Space-HSVNs with S-SN: average of embedding cost

Expe.	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
Ave. cost x 10 <sup>4</sup>	7.8	35.5	52.5	79.6	16.6	66	109.5	160.9	21.8	102.5	130	222.2

**Figure 5.2** — Space-HSVNs with S-SN: the change of average value for embedding cost with the change of VNs synchrony demands

- In the first three scenarios of each group, the VNs synchrony increases linearly from 0% in exp.1, to 30% in exp.2, and to 60% in exp.3. In group A, the embedding cost of these scenarios increases  $27.7 \times 10^4$  from A1 to A2, and increases  $17 \times 10^4$  from A2 to A3. Analogously, in group C, the embedding cost increases  $80.7 \times 10^4$  from C1 to C2, and increases  $27.5 \times 10^4$  from C2 to C3. So, with a linear increment in VNs synchrony demands, the increment in the cost was nonlinear. A possible interpretation for the non-linear increment is that, the increase in the synchrony demands implies an increase in the number of the synchronous virtual nodes and links that need to be mapped, but the links mapping is not 1:1 mapping since a link can be mapped on a physical path composed of several links. So, a linear increase in the VNs synchrony demands leads to a quicker increase in the number of resources allocated, thus, in the mapping cost.
- In groups A and C, we notice that the increase in the cost from scenario 1 to scenario 2 is bigger than the increase in cost from scenario 2 to scenario 3. In group A, the cost increased  $27.7 \times 10^4$  from A1 to A2, which is more than the increment value  $17 \times 10^4$  from A2 to A3. Also, in group C, the embedding cost increases  $80.7 \times 10^4$  from C1 to C2, more than the increment value  $27.5 \times 10^4$  from C2 to C3. So, we say that, a transition from a fully asynchronous VNs (i.e., exp.1) to hybrid synchronous VNs (i.e., exp.2) results in a

bigger increment in the embedding cost than the transition from hybrid asynchronous VNs (i.e., exp.2) to another hybrid synchronous VNs with higher synchrony demands (i.e., exp.3). The reason is that the change from fully asynchronous VNs to hybrid VNs will imply start reserving synchronous resources on the SN, whose cost is bigger when compared to the asynchronous resources, and this would make the increment in the embedding cost remarkable.

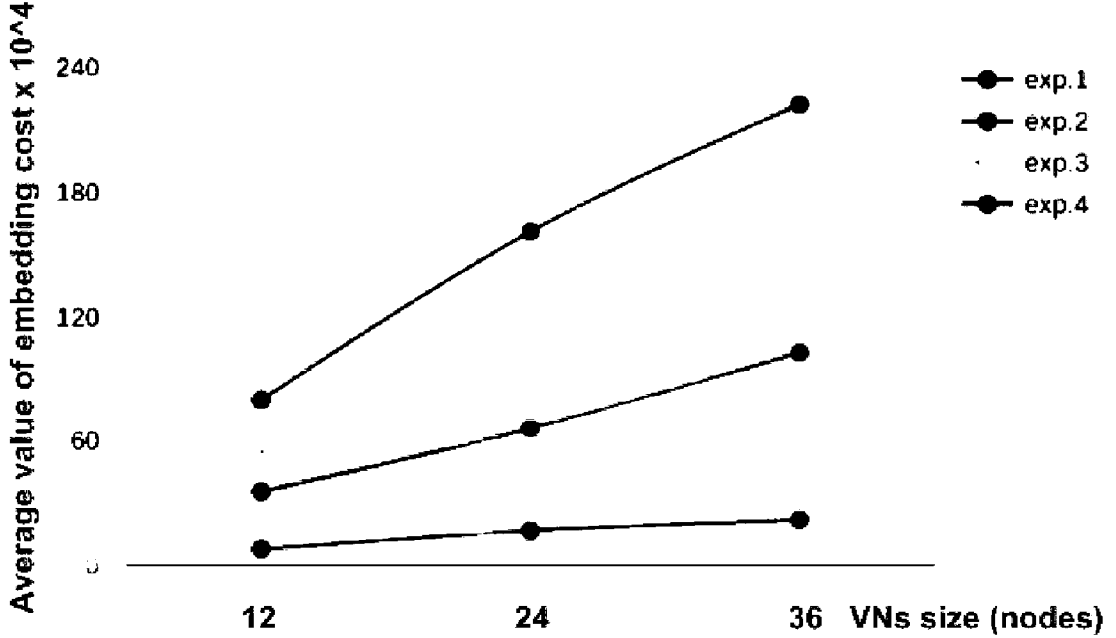
- Comparing the curve of *group A* and of *group C* in Figure 5.2, we notice that curve group A is with smoother increment than the curve of group C that has higher jumps from one scenario to another, and between C3 and C4 we notice a slope up. A possible reason for this behavior is the comparative size of the VNs and the SN of each group. We remember that the mapping model maps the asynchronous virtual demands on either synchronous or asynchronous physical resources, prioritizing the asynchronous ones as long as they are available. In group A, where the VNs were of 12 nodes, smaller than the SN, there were available asynchronous physical resources for the asynchronous virtual demands, so, there won't be a need for reserving synchronous SN resources for the asynchronous VNs demands. Moreover, in group A, the virtual links can be mapped on short physical paths (possibly one physical link) due to the resources availability. This would make the cost curve of group A increases smoothly. Whereas in group C, the VNs are with 36 nodes, bigger than the SN, this would load the physical resources more, leading to (i) the unavailability of asynchronous physical resources, pushing to the case of mapping asynchronous virtual demands over synchronous physical resources, and (ii) virtual links will be mapped on longer physical paths (making routes through available SN links). These two reasons would make the cost curve of group C increases sharply.
- In group B, the embedding cost increases  $49.4 \times 10^4$  from B1 to B2, and increases  $43.5 \times 10^4$  from B2 to B3, nearly the same values. So, with a linear increment in VNs synchrony demands, the increment in the cost was linear (or very near to be linear). It calls the attention that both the SN and the VNs in group B were of proximate size, i.e., SN of 25 nodes and VNs of 24 nodes.
- The three curves representing groups A, B, and C do not cross together, because at a fixed VNs synchrony demands (i.e., 0% , 30% , 60% ,and 100%), the embedding cost will be bigger for the VNs of bigger size, making the curve of group A the lowest and of curve C the highest.

We continue our observations about Figure 5.1.

(iii) By comparing the counterpart scenarios of the three groups, we notice that, with an increment in the VNs size from 12 nodes to 24 then 36 in groups A, B, and C respectively, there is an increment in the mapping cost. For example, see the intervals that represent



the cost for scenarios A3, B3, and C3:  $[50 \times 10^4, 55 \times 10^4]$ ,  $[100 \times 10^4, 118 \times 10^4]$ ,  $[101 \times 10^4, 153 \times 10^4]$  respectively. To define more accurately the increment pattern of the embedding cost with the increment in VNs synchrony demands, we draw Figure 5.3 that depicts the average values for the embedding cost (taken from Table 5.2) versus the VNs size. We note down the following observations:



*Figure 5.3* — Space-HSVNs with S-SN: the change of average value for embedding cost with the change of VNs size

1. In scenarios 1, 3, and 4, the VNs size increases linearly from 12 nodes in group A, to 24 nodes in group B, to 36 nodes in group C, while the embedding cost increases non-linearly. For example, the embedding cost increases  $8.8 \times 10^4$  from A1 to B1, and increases  $5.2 \times 10^4$  from B1 to C1. Analogously, the embedding cost increases  $57 \times 10^4$  from A3 to B3, and increases  $20.5 \times 10^4$  from B3 to C3. Also, the embedding cost increases  $81.3 \times 10^4$  from A4 to B4, and increases  $61.31 \times 10^4$  from B4 to C4. A possible interpretation for the non-linear increment is that, the increase in the VNs size implies an increase in the number of virtual nodes and links that need to be mapped, but the links mapping is not 1:1 mapping since a link can be mapped on a physical path composed of several links. So, a linear increase in the VNs size leads to a quicker increase in the number of resources allocated, thus, in the mapping cost.
2. In scenario 2, the embedding cost increases  $30.5 \times 10^4$  from A2 to B2, and increases  $36.5 \times 10^4$  from B2 to C2, nearly the same values. So, in scenario 2 of every group, with a linear increment in VNs size, the increment in the cost was linear (or very near to be linear). It calls the attention that, in scenario 2, the SN and VNs are with proximate synchrony percentage (i.e., 33% for the SN and 30% for the VNs).

We continue our observations about Figure 5.1.

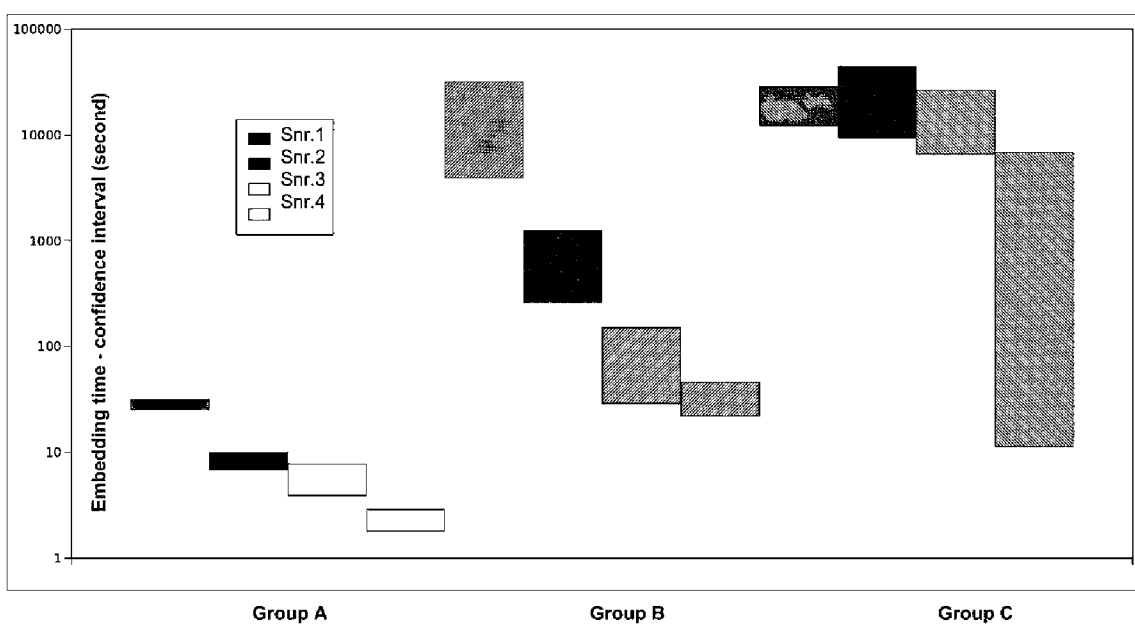
(iv) It is true that hybrid synchronous VNs can be mapped on a fully synchronous SN (since synchronous physical resources provide what asynchronous resources do), but in our work we argue that to map hybrid synchronous VNs in an economic manner we need (1) a hybrid synchronous SN, and (2) a suitable embedding model that considers the hybrid synchronous nature of both SN and VNs (which we proposed formerly). In our experiments, if the SN was fully synchronous, then synchronous physical nodes and links will be assigned to the virtual nodes and links independently of their synchrony requirements, which will result in an un-needed cost, in other words, in an uneconomic embedding solution. In scenario 4 of every group, the VNs were fully synchronous and the SN was hybrid synchronous. In case our SN was fully synchronous, then the embedding cost for mapping the VNs in scenarios 1, 2, and 3 will be approximately the embedding cost illustrated in scenario 4, because in scenario 4 all the VNs nodes and links were embedded on the synchronous subnetwork of the hybrid SN. Scenario 4 of each group will be the base for cost comparison since it can simulate the case where all the SN nodes and links provide time bounds. For example, the SN considered by Zhang *et al.* in [ZWJY10] can be viewed as a fully synchronous SN since all its resources provide time bounds (whether the time bounds were high or low). Comparing the three first scenarios within each group with the fourth one, we can say that our proposal leads to mapping hybrid synchronous VNs in an economic way. That is, the SN can be used in an optimized way to allocate these demands, the synchronous demands are mapped on synchronous physical resources, and the asynchronous demands are mapped on synchronous or asynchronous physical resources prioritizing the asynchronous ones as long as they are available. For example, scenarios C1, C2, and C3 depict the mapping of a hybrid VN with 0%, 30% and 60% synchrony demands respectively on a hybrid SN with 33% synchronous resources. The mapping cost for these three scenarios were  $[20 \times 10^4, 23 \times 10^4]$  for C1,  $[95 \times 10^4, 110 \times 10^4]$  for C2, and  $[107 \times 10^4, 153 \times 10^4]$  for C3. Whereas mapping the same VNs demands of C1, C2, and C3 on a fully synchronous SN, scenario C4, is subject to an extra un-needed cost  $[207 \times 10^4, 237 \times 10^4]$ . The cost in scenario 4 of each group is independent of the VNs synchrony demands. Our conclusion is that hybrid VNs do not need fully synchronous SN, rather a hybrid SN with suitable mapping is enough to allocate the needed demands, and spares resources for future demands. The question that may rise at this phase is: what is the minimum subnetwork of the SN that need to be synchronous to map certain given VNs? The answer for this question led us to propose *configurable SN* detailed in Section 4.6 and will be evaluated later during this Chapter.

**Conclusions on the embedding cost:** We summarize the aforementioned observations about the embedding cost for HSVNs over S-SN as the following: (i) The proposed embedding model considers the hybrid synchronous nature of VNs, mapping them onto a hybrid synchronous SN in an economic manner, trying to spare the synchronous physical

resources whose building cost is expensive when compared to the asynchronous ones. *(ii)* With the increase in the problem size (be it through the increase in the VNs size or through the increase in the synchrony demands), the mapping cost increases. *(iii)* A linear increase in the VNs size and/or synchrony demands leads to a nonlinear increment in the embedding cost. *(iv)* When the VNs and the SN are with approximate size and/or synchrony percentage, the linear increase in one of these parameters leads to a linear increment in the embedding cost. With the set of experiments we ran, we could not distract reasons behind this last behavior (i.e., the behavior in item *(iv)*), and investigating this point remains for our future works.

### 5.1.2.2 Embedding time

The embedding time is the time needed by CPLEX to find an optimal solution for the embedding problem. Figure 5.4 depicts the mapping time on a logarithmic scale for each of the twelve scenarios performed with 90% confidence interval. We note down the following main observations:



*Figure 5.4* — Embedding time - Confidence interval

*(i)* We notice some overlaps in the embedding time intervals for some scenarios, for example in group C there are overlaps observed between the three experiments C1, C2 and C3. It is possible that repeating every scenario for 10 times resulted in a wide confidence interval for the embedding time, and to have a narrower interval we needed to run every scenario for a bigger number of times. Yet with the available set, we try to derive some useful observations over the embedding time.

*(ii)* The confidence interval for the embedding time for scenario B3 had a negative part

[-153 , 915]. This is possible numerically, but not physically since we are expressing time. Looking closely at the 10 experiments for scenario B3, we notice that the experiment B33 demanded an embedding time 2989.49 (sec.) which was much bigger than the other 9 readings that ranged between 21 and 356 seconds. We consider the time value in B33 as a noise that might distort conclusions, so we disconsider this reading, and calculate the confidence intervals for B3 to become [29 , 152] seconds.

(iii) Within the same group, the embedding time decreases from scenario 1 to scenario 4. For example, in group A: the embedding time interval was [25.07 - 32.07] seconds in A1, [6.87 , 10.11] in A2, [3.91 , 7.85] in A3, and [1.77 , 2.93] in A4. Similarly, the embedding time decreases in groups B and C from scenario 1 to scenario 4. The reason for this is that the VNs synchrony increases from 0% in scenario 1, to 30% in scenario 2, to 60% in scenario 3, and to 100% in scenario 4. In other words, the VNs asynchrony decreases from 100% in scenario 1, to 70% in scenario 2, to 40% in scenario 3, and to 0% in scenario 4. Remembering that the SN adopted for our experiments is with 33% synchronous and 67% asynchronous resources (the asynchronous physical subnetwork is bigger than the synchronous one), and remembering that our developed embedding model prioritizes mapping the asynchronous demands over asynchronous resources, then during the optimization process the space of possible mapping solutions will decrease from scenario 1 to scenario 4 resulting in decreasing the embedding time. In scenario 1, the solutions space is the asynchronous physical subnetwork which is a 67% of the SN. In scenario 4, the solutions space is the synchronous physical subnetwork which is only 33% of the SN. Experiments 2 and 3 go gradually between the two extremities. So, as the VNs synchrony increases the solutions space decreases, thus the model variables decrease, and thus the time needed to find the embedding solution decreases as well. It is important to mention that if the SN synchrony was chosen to be for example 33% asynchronous resources and 67% synchronous resources, then the observation about embedding time will be reversed: the experiments with more synchrony demands will need more embedding time as the solution space will be wider. This observation drives the attention to the impact of the SN synchrony not only on the embedding cost, as formerly detailed, but as well on the embedding time.

(iv) By comparing the counterpart scenarios between the groups, we notice that the embedding time increases with the increment in the VNs size. For example, the increment in the VNs size from 12 nodes in scenario A2 to 24 nodes in scenario B2, and to 36 nodes in scenario C2 results in increasing the embedding time as the following: [6.87 , 10.11] in A2, [259.63 , 1257.41] in B2, and [9399 , 44230] in C2. The reason is that, bigger VNs means more nodes and links to be mapped, this generates more variables in the embedding model, and thus demand more optimization time for finding the embedding solution.

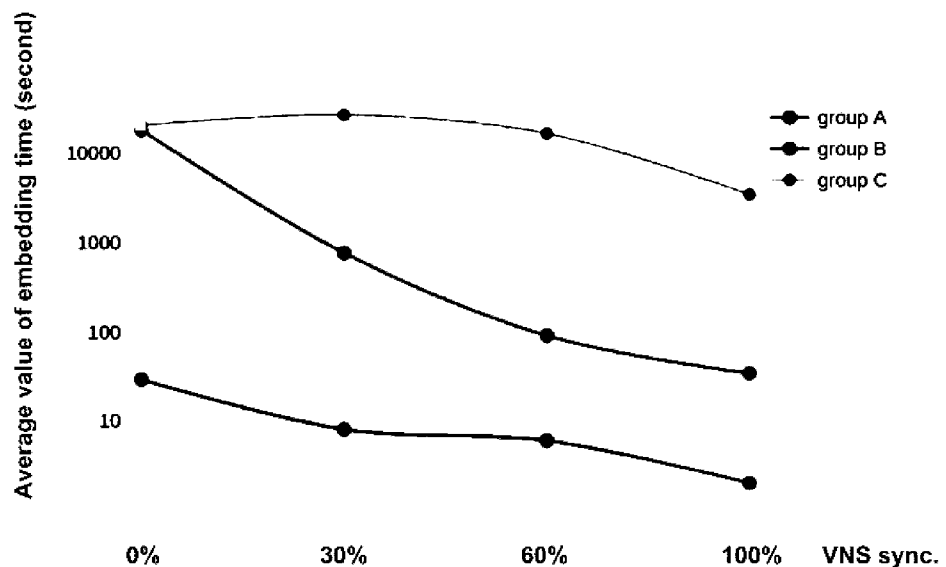
To define more accurately the change pattern of the embedding time with the increment in VNs synchrony demands and size, we distract the average value (rounded values to the

neares ones) of embedding cost for every scenario, see Table 5.3.

**Table 5.3** — Space-HSVNs with S-SN: average of embedding time

Expe.	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
Ave. time (sec)	29	8	6	2	18088	759	90	34	20429	26815	16585	3437

Figure 5.5 depicts the average values for the embedding time versus the VNs synchrony demands, and Figure 5.6 depicts the average values for the embedding time versus the VNs size. The figures are depicted on a logarithmic scale to improve legibility.



**Figure 5.5** — Space-HSVNs with SSN: the change of average value for embedding time with the change of VNs synchrony demands

From Figure 5.5 and Figure 5.6 we note down the following observations:

1. With a linear increment of VNs size, Figure 5.6, the curves representing the computational time change in a semi-linear or hyperbolic manner on a logarithmic scale. This means that on a linear scale these curves would be more sloppy. The reason for this behavior is that, a small change in the embedding model input (i.e., virtual nodes and links number) results in a big change in the model variables that need to be computed. For example, for a SN of 25 nodes, one more virtual node in the model input would result in 25 more variables to be computed by the embedding model ( $\sigma(i^k, i)$ ).

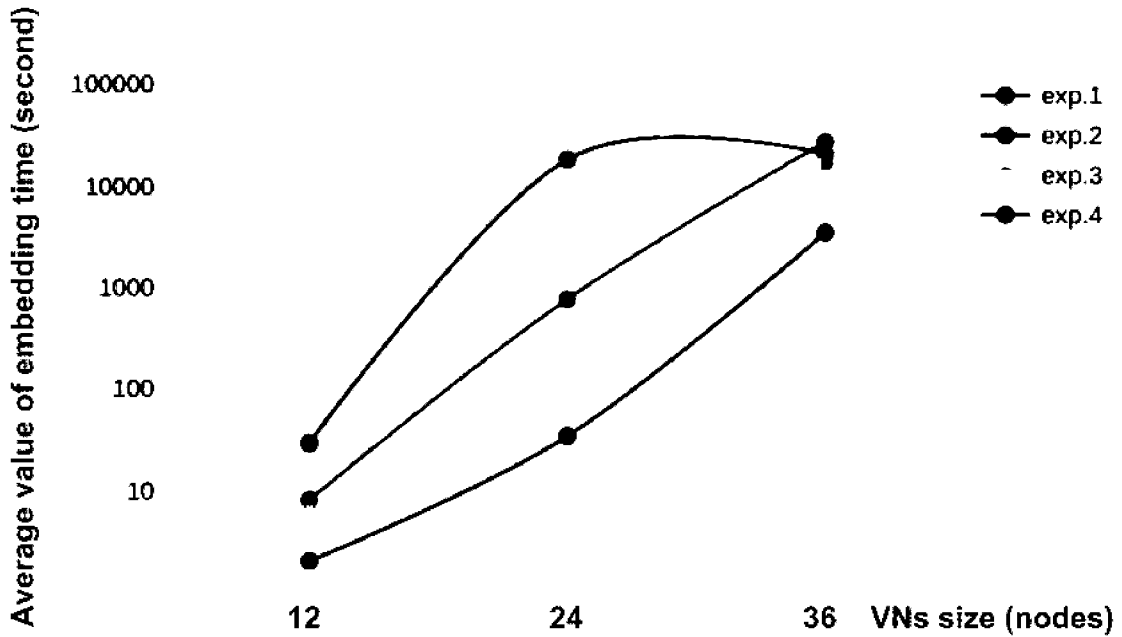


Figure 5.6 — Space-HSVNs with S-SN: the change of average value for embedding time with the change of VNs size

2. The optimization time for the scenarios in group A are within small range (27 second). The reason is that in group A the VNs size is small, and the VNs synchrony difference will not result in a remarkable difference in the computational time. Notice that A2 and A3 are nearly with the same computational time.
3. The optimization time for scenarios C1, C2, and C3 is nearly the same. The reason is that scenarios of group C are with big VNs size, most of them demanded much optimization time, and most of them were interrupted at a gap that is less than 1%.
4. The optimization time for scenarios B1 and C1 is nearly the same. These scenarios are with (i) VNs size that is equal or bigger than the SN, and (ii) both with no synchrony requirements, so the asynchronous physical subnetwork is the space for solutions. For the two aforementioned reasons, these two scenarios demanded much optimization time, and most of them were interrupted at a gap that is less than 1%.

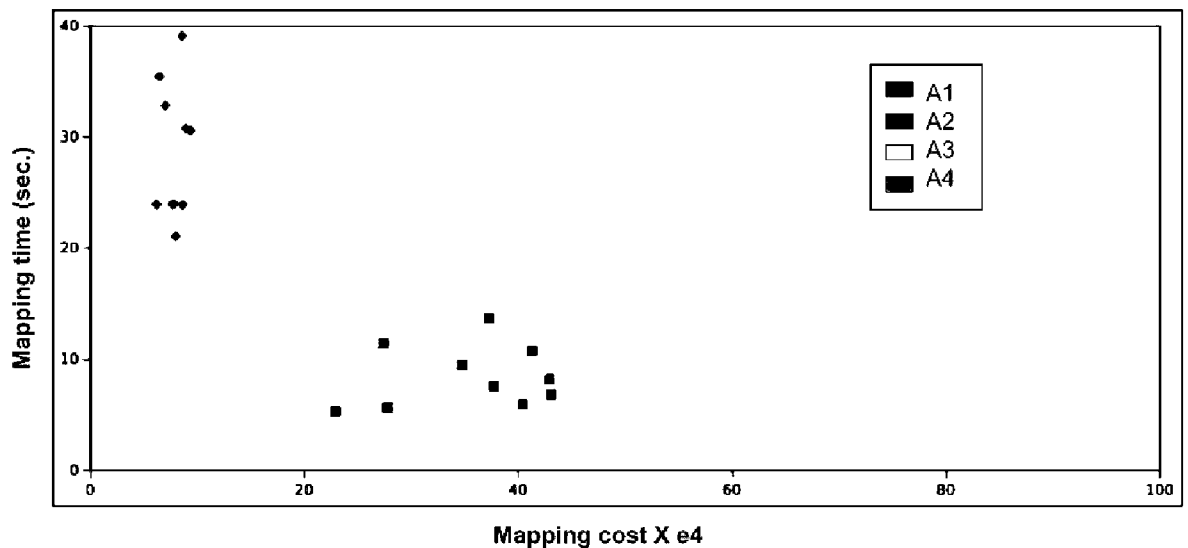
**Conclusions on the embedding time:** The computational time is proportional to the number of the model variables, and this last one changes with the problem size as the following: (i) when the VNs size increases, the model variables number increases, and thus the computational time increases, and (ii) when the VNs synchrony demands increases, the solution space becomes increasingly within the synchronous physical subnetwork, which is in our case a small portion of the SN, this means that the model variables will become less, and thus the computational time to find the variables values would become less.

### 5.1.2.3 Embedding cost vs. embedding time

Previously we studied the embedding cost and the embedding time separately. In this subsection we study the relation between both.

Figure 5.7 depicts the mapping cost versus time for all the 40 experiments in Group A: 10 samples for A1 + 10 samples for A2 + 10 samples for A3 + 10 samples for A4. The dots corresponding to the same subgroup form a zone in the figure. We notice that:

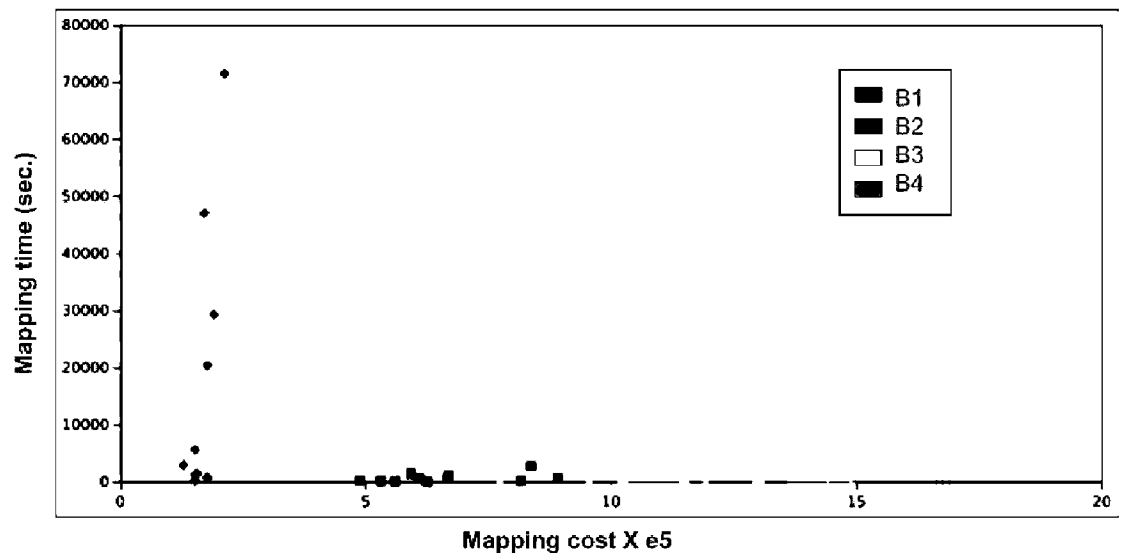
1. The four zones are clearly separated, no overlapping zones.
2. The samples are arranged on a hyperbolic curve as a superior envelope of the zones, this means that for VNs of same size (experiments of Group A are all with VNs of 12 nodes) the increment in VNs synchrony demands results in: (1) increasing the embedding cost (axis X), and (2) decreasing the embedding time (axis Y). These two results agree with our previous observations.
3. We notice that the time values for experiments A2 and A3 are within the same range. This agrees with our previous observations.



*Figure 5.7* — Group A - Embedding cost vs. embedding time

Similarly, Figure 5.8 depicts the mapping cost versus time for all the 40 experiments in Group B. Figure 5.9 is the same as Figure 5.8 with a zoom in to experiments B2, B3, and B4. We notice that:

1. The zones show only once an overlap on the horizontal scale (i.e., the embedding cost), that is between B2 and B3. This unique overlap disappeared when calculating the cost confidence interval for scenarios B2 and B3, see Figure 5.1.
2. All the four zones overlap on the vertical scale (i.e., embedding time). these many overlaps made it visible even after calculating the time confidence interval, see Figure 5.4.
3. The computational time for one trace within B3 subgroup is out of the B3 zone, this trace refers to experiment B33, and is the one that was disconsidered when calculating the time confidence interval for scenario B3.
4. We notice again the hyperbolic pattern as a superior envelope of the region where the samples are distributed, this means that for VNs of same size (experiments of Group B are all with VNs of 24 nodes) the increment in VNs synchrony demands results in: (1) increasing the embedding cost (axis X), and (2) decreasing the embedding time (axis Y). These two results agree with what was formerly detailed.



*Figure 5.8* — Group B - Embedding cost vs. embedding time

In the same way, Figure 5.10 depicts the mapping cost versus time for all the 40 experiments in Group C. We notice that:



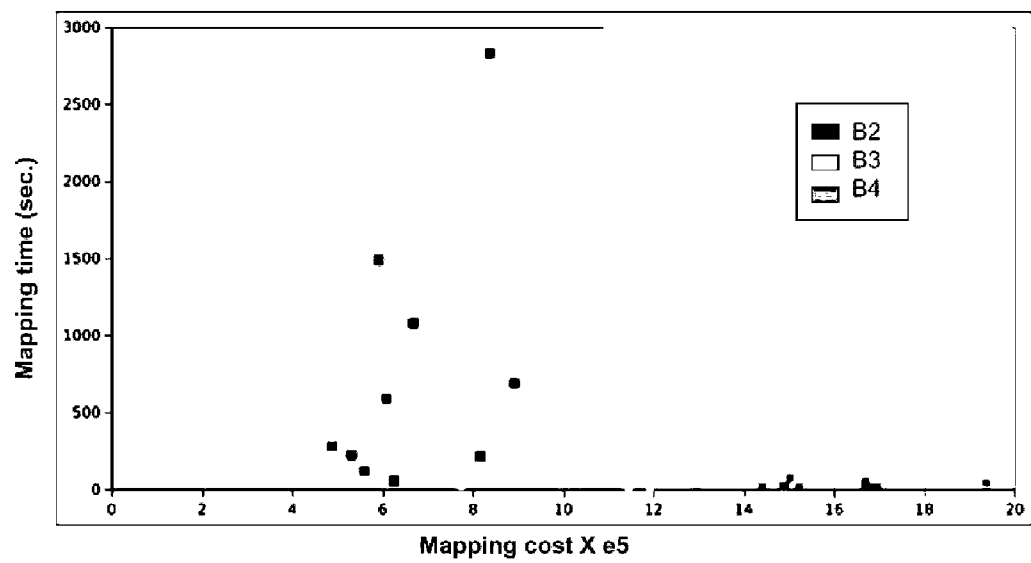


Figure 5.9 — Embedding cost vs. embedding time for experiments B2, B3, and B4

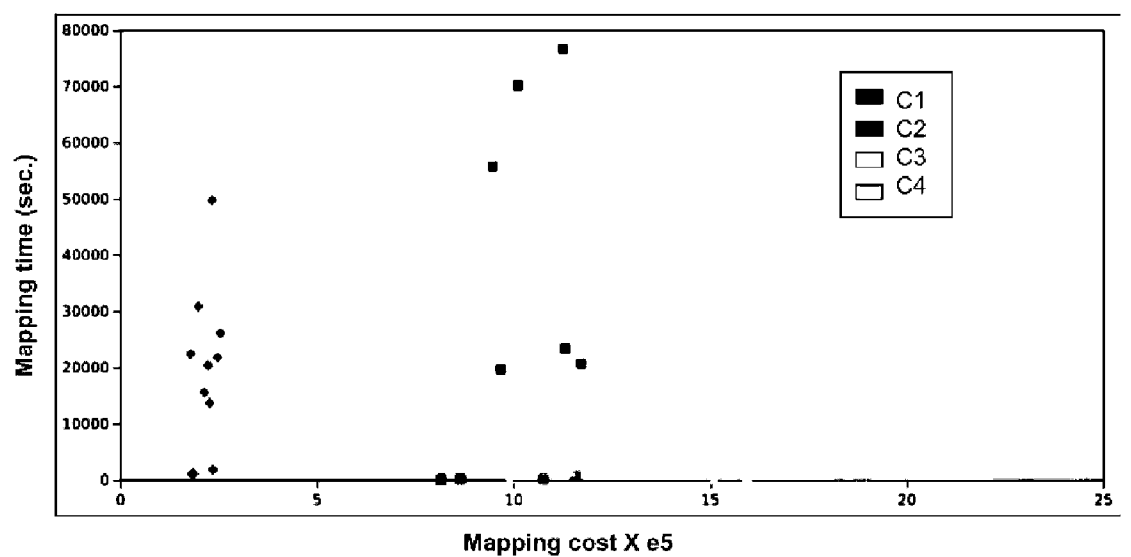


Figure 5.10 — Group C - Embedding cost vs. embedding time

1. C2 and C3 zones show overlaps on the horizontal scale (i.e., the embedding cost), for example, C3 contains C2 on the horizontal axis (i.e., the cost axis). This overlap that appears in Figure 5.10 does not appear after calculating the cost confidence interval, see Figure 5.1.
2. C1, C2, and C3 zones show overlaps on the vertical scale (i.e., the embedding time). This overlap that appears in Figure 5.10 appears as well after calculating the time confidence interval, see Figure 5.4, and the time average values of these scenarios are of approximate values, see Figure 5.6.
3. All the four zones show overlaps on the vertical scale (i.e., embedding time), these many overlaps made it visible even after calculating the time confidence interval, see Figure 5.4. We referred this behavior previously to the interruption of these experiments after reaching a gap less than 1%.

#### 5.1.2.4 Physical resources load

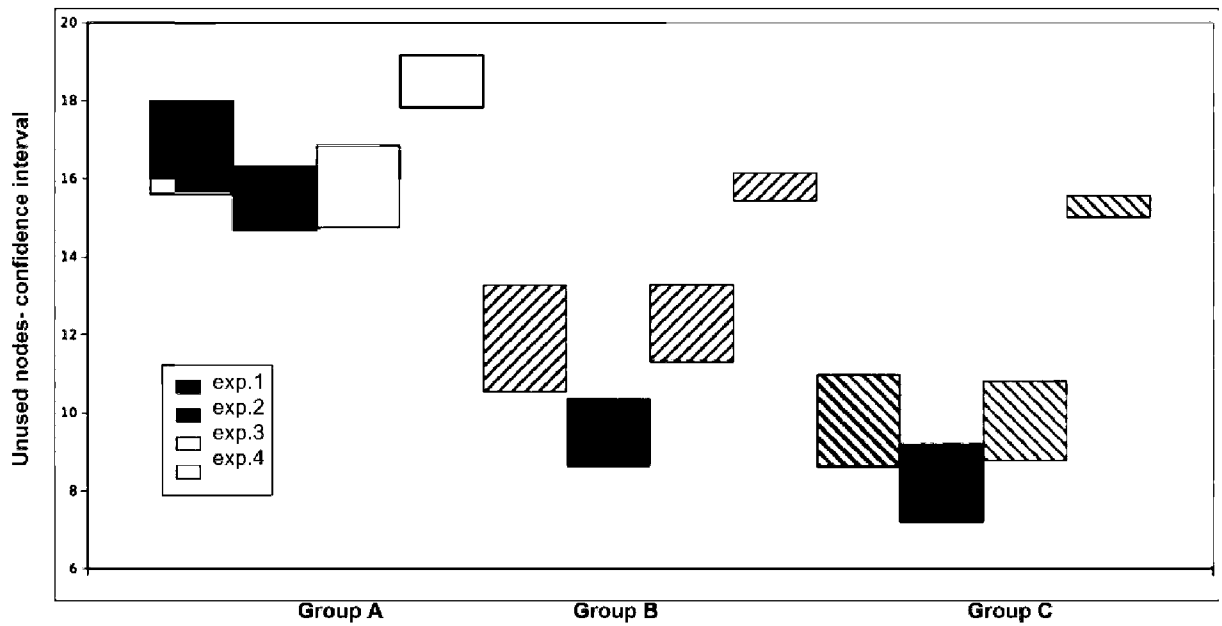
##### A. Free resources:

Figure 5.11 and Figure 5.12 depict the confidence interval for unused nodes and links respectively for all scenarios. We note down the following observations:

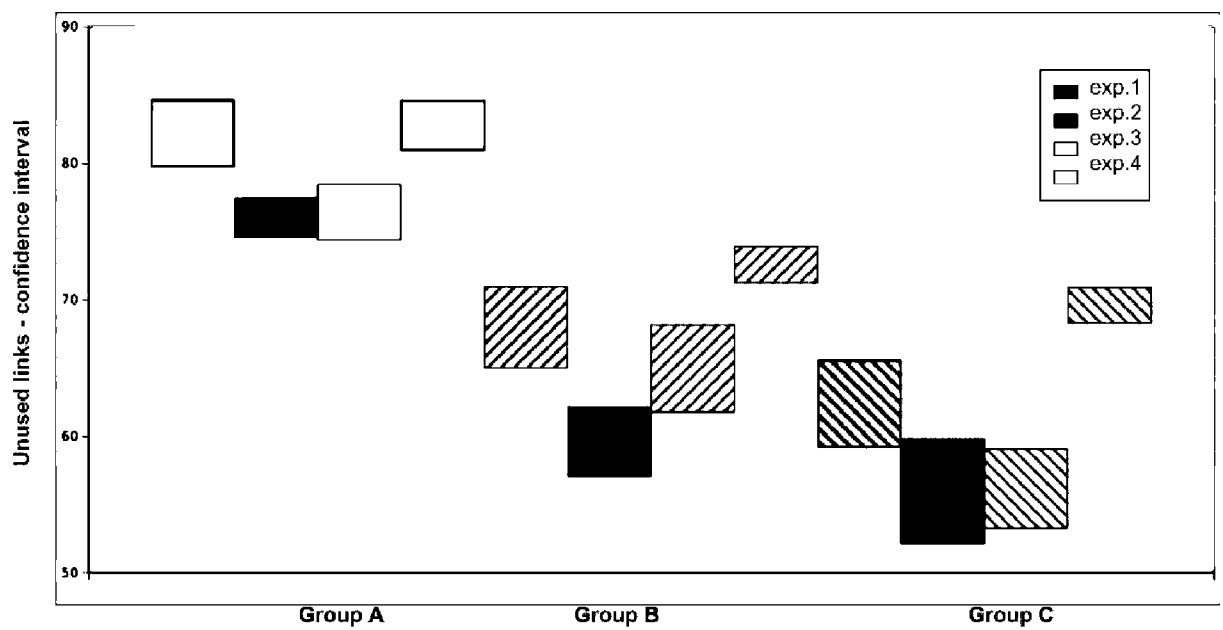
1. Within each group, the highest interval was the one of scenario 4. This scenario represents the VNs with full synchrony demands. The physical resources that mapped these demands were the resources of the synchronous portion of the SN, that is the 33% of the SN. Whereas the remaining 67% physical resources (i.e., the asynchronous portion of the SN) were unused. In all of the remaining scenarios, the unused resources will be less due to the existence of asynchronous virtual demands in the other scenarios.
2. Within each group, the second highest interval is the one of scenario 1. This scenario represents the VNs with full asynchrony demands. With our model that aims at sparing the use of synchronous physical resources, the synchronous portion of the SN will be avoided, resulting in increasing the number of unused resources. But still, the free resources in scenario 1 remain less than the free resources in scenario 4 because the avoided portion of the SN in these two scenarios are the asynchronous and the synchronous portion of the SN respectively, that form 67% and 33% of our SN respectively.
3. By comparing scenario 2 and scenario 3 within each group, we notice that the unused resources in scenario 3 are more (except for the free links in C2 and C3 where the two intervals overlap). Scenario 3 contains VNs with more synchronous demands than in scenario 2. This makes VNs in scenario 3 that are mapped on the small synchronous portion of our SN more than the VNs that are mapped on this synchronous portion

in scenario 2. So, the free asynchronous resources in scenario 3 are more than the free asynchronous resources in scenario 2. Thus, the total unused resources in scenario 3 are more.

4. By comparing the counterpart experiments in the three groups, we notice that the number of free resources decreases with the increment in the VNs size. The reason is that bigger VNs will demand more physical resources to map them. Thus, the free resources will become less.



*Figure 5.11* — Unused physical nodes - Confidence interval



*Figure 5.12* — Unused physical links - Confidence interval

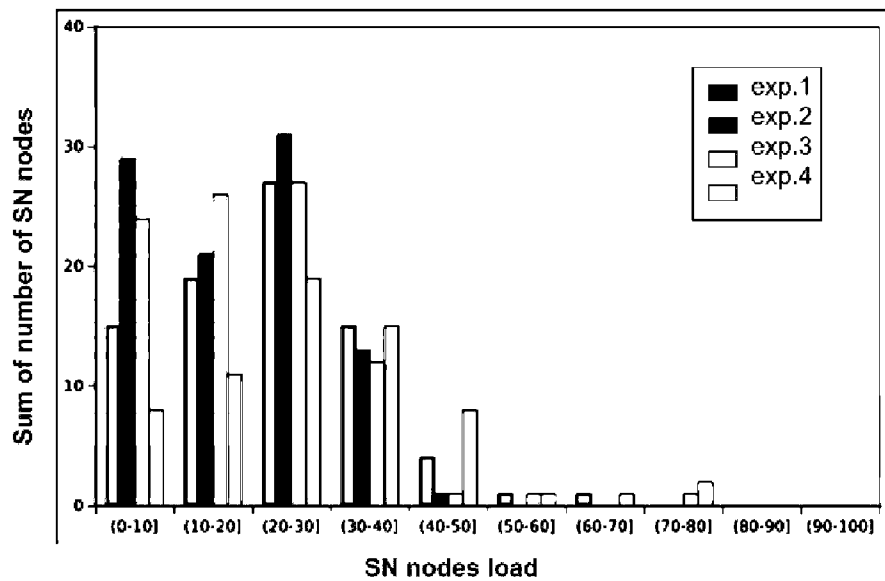
### B. Used resources:

The following graphs will illustrate the physical resources load (i.e., nodes and links) for the twelve scenarios in Table 5.1. We divide the resources load into ten intervals, and at each interval we depict the number of physical resources (nodes and links separately) that are loaded in correspondence with that interval. For example, the first interval shows the number of physical nodes and links that are with a load between (0%-10%] of their nominal capacity (resources at 0% load are excluded at this interval, and resources at 10% load are included). The second interval shows the number of physical resources that are with a load between (10%-20%] of their nominal capacity (resources at 10% load are excluded at this interval, and resources at 20% load are included). The last interval depicts the number of physical resources that are loaded with (90%-100%] of their nominal capacity.

We are interested in investigating the distribution pattern of the physical resources load. Figure 5.13 illustrates the accumulative number of physical nodes with their load in the four scenarios of group A. We see that the horizontal axis is divided into ten intervals, representing the physical nodes actual load in comparison with their nominal capacity (a percentage). In the first interval, we see four columns that represent the four scenarios A1, A2, A3, and A4. For example, in scenario A1, we accumulate the number of resources with a load (0%-10%] through the ten experiments that represent scenario A1 (i.e., experiments A10, A11, ... A19). And the accumulative value can be read on the vertical axis of Figure 5.13.

From Figure 5.13 we note that:

1. The distribution pattern of the accumulative number of nodes is a gaussian-like curve. Samples are more frequent on 0% to 30% categories, with an overall dominance on 20% to 30% range. And categories over 50% are negligibly populated. And no nodes are fully loaded. This means that the embedding model tends towards distributing the load over the physical nodes. True that the embedding model aims at minimizing the use of physical resources (nodes and links), but we have not inserted constraints to make the model use the allocated synchronous resources till exhaustion. The load distribution is an important characteristic because it impacts directly the mapping ratio (i.e., the number of accepted VNs on a given SN). Heavily or fully loaded nodes make these nodes reject mapping, and this would influence the number of accepted VNs.
2. Few resources appear at load (70%-80%] in scenarios 3 and 4. These two scenarios are with more synchronous demands than the other scenarios. With a SN of 33% synchronous resources, increasing synchronous demands will more likely start increasing the load of synchronous resources, which appear in the Figure.
3. At low interval (from 0% to 30%) we notice that the scenario with the smallest ac-



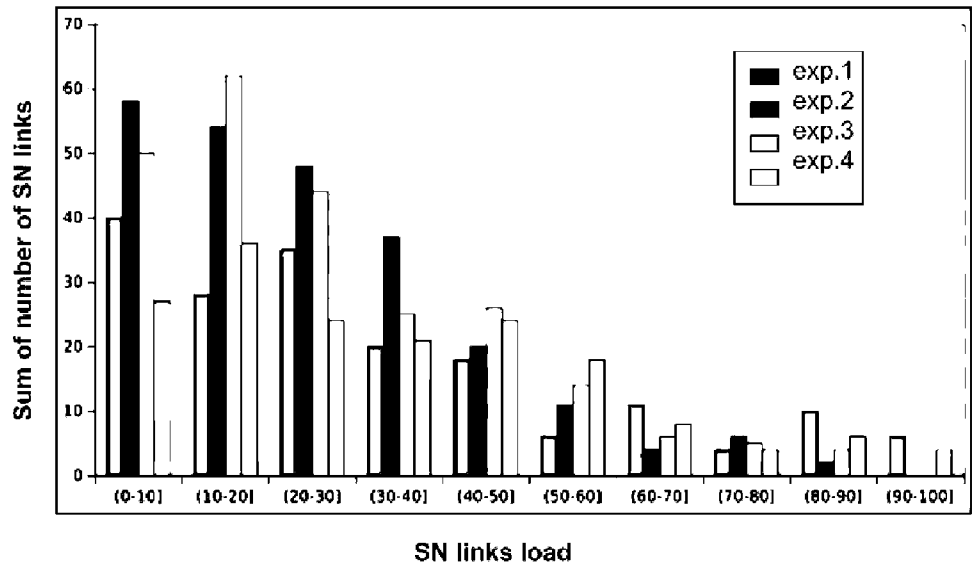
*Figure 5.13* — Group A - accumulative number of physical nodes vs. nodes load

cumulative number of nodes is scenario 4. This scenario represents the VNs with the highest synchrony demands, which will be concentratedly mapped on the small portion of synchronous physical resources (33% of the SN), leading to more heavily loaded nodes and less lightly loaded nodes in comparison with the other scenarios. If we chose a SN that is with 33% asynchronous resources, then scenario 1 will appear with the smallest accumulative number of nodes at low load intervals.

Figure 5.14 depicts the accumulative number of loaded links in scenarios of group A. We drive similar observations noted regarding the nodes:

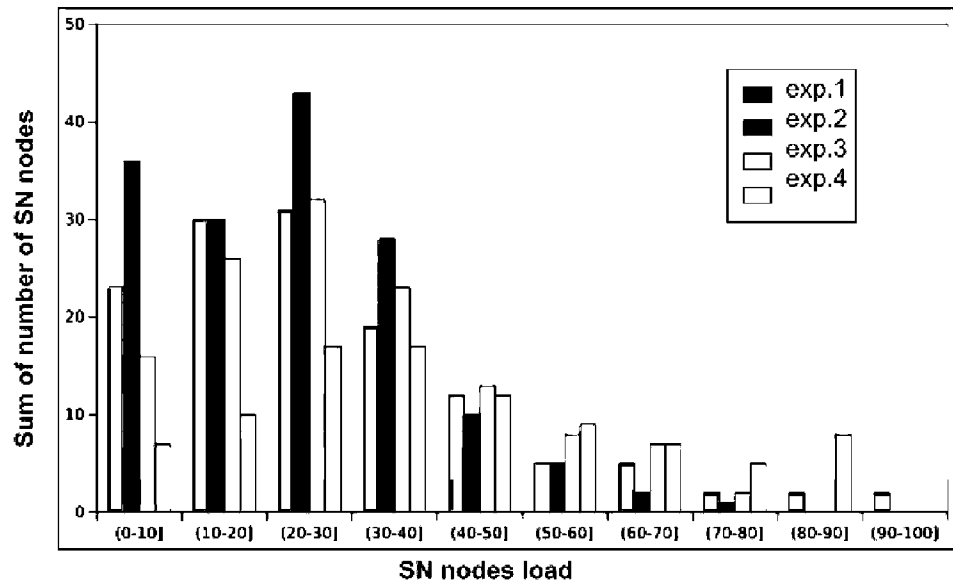
1. The distribution pattern of the accumulative number of links is a gaussian-like curve.
2. Samples are more frequent at low load intervals with picks on 10% to 20%.
3. Few number of links appear at high load.
4. with the increment in the VNs synchrony demands (from scenario 1 to scenario 4), the number of heavily loaded links increases and the number of lightly loaded links decreases.

Analogously, Figure 5.15 and Figure 5.16 illustrate the accumulative number of used nodes and links, respectively, in scenarios of group B.



*Figure 5.14* — Group A - accumulative number of physical links vs. links load

1. The distribution patten of the accumulative number of resources (nodes and links) is a gaussian-like curve.
2. The accumulative number of resources at law load intervals is bigger than at high load intervals. This was previously noticed as well in scenarios A. The reason is as detailed above regarding the model tendency towards load distribution, avoiding exhausting resources.
3. Nodes distribution concentrates on up to 40%, more specifically on 20% to 30% category. Links distribution concentrates more on up to 40% usage, with a pick on 30% to 40% category. In comparison with scenarios of group A, we notice that the pick values in scenarios B are shifted upwards on the horizontal axis, this means that with the increment in VNs size (from 12 nodes in group A to 24 nodes in group B) the accumulative number of resources (nodes and links) is increasing at high load intervals and is decreasing at law load intervals. A possible interpretation for this behavior is that bigger VNs brings more virtual nodes and links that need to be mapped, this demands more physical resources, and thus will lead to loading these physical resources more.
4. Comparing the four scenarios of group B together (i.e., B1, B2, B3, and B4), we notice that scenario B2 appears with the highest value of accumulative number of resources at law load intervals. We try to explain this behavior as the following. Scenarios B1



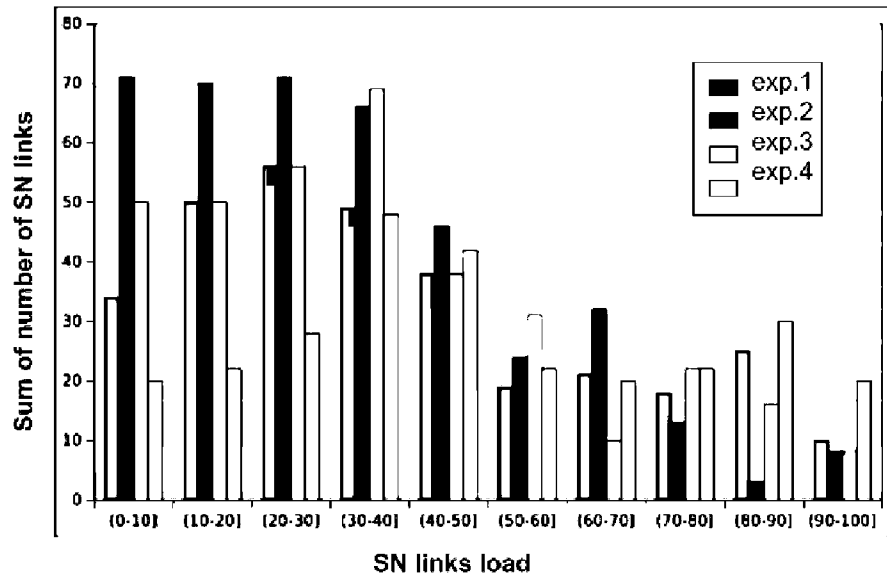
*Figure 5.15* — Group B - accumulative number of physical nodes vs. nodes load

and B4 are with VNs of fully asynchronous and fully synchronous demands respectively, that will be mapped concentratedly on the asynchronous and synchronous portion of the SN, respectively, resulting in more heavily loaded resources in comparison with B2. In B3, VNs are with hybrid synchrony requirements, 40% asynchrony requirements that will be mapped basically on the asynchronous portion of the SN and 60% synchronous requirements that will be mapped concentratedly on the synchronous physical resources (that is only 33% on the SN) resulting in more heavily loaded resources in comparison with B2 (where VNs of 30% synchrony demands will be mapped on the 33% synchronous portion of the SN). The SN and VNs in scenario B2 are with approximate size (VNs with 24 nodes and a SN with 25 nodes) and synchrony (VNs with 30% synchronous demands and a SN with 33% synchronous resources), so 30% synchronous demands will be mapped on the synchronous 33% part of the SN, and 60% asynchronous demands will be mapped basically on the asynchronous 67% part of the SN, this provides wider space of possible embedding solutions in comparison with scenarios B1, B3, and B4, and thus leads to choosing a solution that does not need to exhaust resources.

Figure 5.17 and Figure 5.18 illustrate the accumulative number of used nodes and links, respectively, in scenarios of group C.

From the last two figures, we note down similar observations to those derived for group





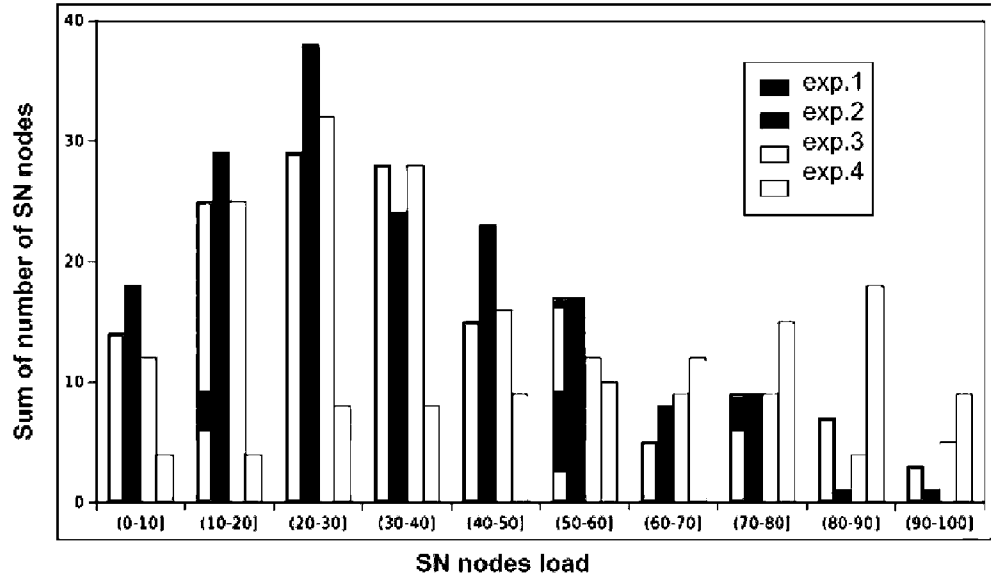
*Figure 5.16* — Group B - accumulative number of physical links vs. links load

A and B. We note down:

1. The distribution patten of the accumulative number of resources is a gaussian-like curve.
2. The accumulative number of resources at law load intervals remains bigger than it is at high load intervals due to the model tendency towards load distribution. But for scenario C4 it is the opposite way (i.e., increment of accumulative number of resources at high load), the reason is that scenario C4 represents fully synchronous VNs at big size in comparison with the SN size, and they will be mapped concentratedly on the small synchronous portion of the SN resulting in increasing the load of these resources.
3. Comparing scenarios C with scenarios of A and B, we notice that in group C resources concentrate on a wider range, up to 60%. And we notice the fully loaded resources increment. The reason is that bigger VNs demands more resources, and thus loads resources more.

### Conclusions on the resources load:

The number of free resources is a function of VNs size and synchrony demands as the following: (i) the number of free resources decreases with the increment in the VNs size, because bigger VNs will demand more resources to map them, and (ii) the number of free



*Figure 5.17* — Group C - accumulative number of physical nodes vs. nodes load

resources in homogeneous VNs is more than it is in hybrid synchronous VNs because they avoid entirely the use of the physical subnetwork that does not match their synchrony nature.

Regarding the used physical resources, we notice that: (i) The model tends towards load distribution over the physical resources, that is why the number of resources at low load is bigger than the number of resources at high load. This is an important feature in the embedding model because few heavily, or fully, loaded resources would increase the number of accepted VNs on a given SN. (ii) The pattern of the accumulative number of used resources is a gaussian-like distribution pattern, its pick is at low load value for small VNs, and it shifts to higher load values with the increment in the VNs size and synchrony demands.

## 5.2 Space HSVNs over a configurable SN

The aspects considered during the analysis of our model are: i) economy in the embedding cost; ii) physical resources load; iii) the kind of physical resources chosen to be synchronous, and iv) the topology of the subnetwork composed of synchronous resources on the SN.

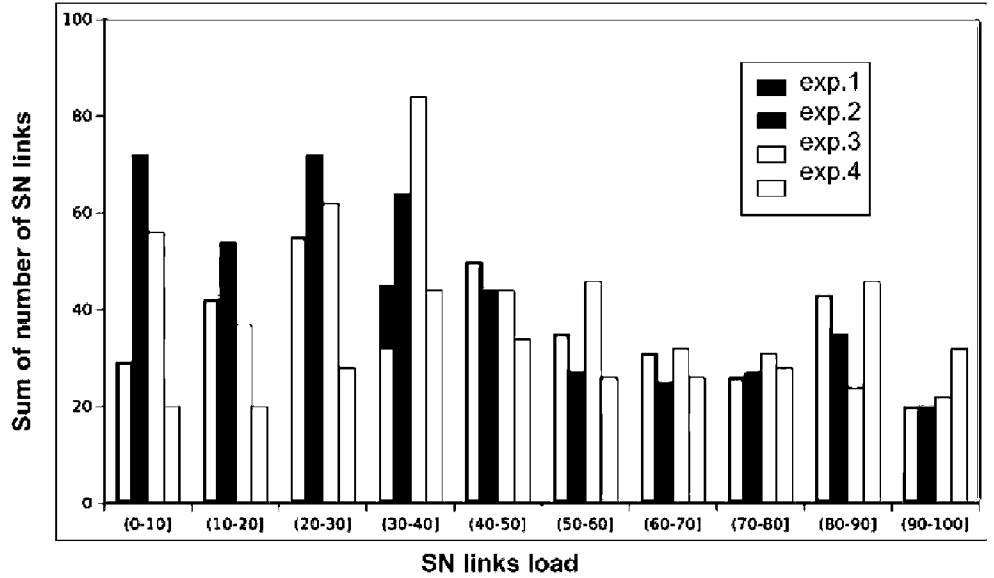


Figure 5.18 — Group C - accumulative number of physical nodes vs. nodes load

### 5.2.1 Work load and tools

Similarly to HSVNs over S-SN, physical and virtual networks were randomly generated. For this we used BRITE [MLMJ14] tool with Waxman [Wax88] model. Waxman algorithm has been used by some researchers [GH16] to generate random virtual network topologies, and in our work we use it. Waxman generates random network topologies based on two parameters: alpha and beta. As the first parameter grows, the probability of having an edge between any nodes in the topology is increased. As the second parameter grows there is a larger ratio of long edges to short edges. In [GH16] the researchers chose alpha and beta to be 0.4 and 0.2, and in our work we choose approximate values: 0.15 and 0.2. We implemented the mathematical model with ZIMPL language [Koc04] and used CPLEX [bmc14] to solve the IP, running on a computer with 6 cores Intel Xeon processor, 2x2.66 GHz, 32GB of RAM memory, and MAC operating system.

Most of the experiments took a considerable time to reach the optimal solution, the reason is that the IP we propose has four output variables, which leads to considerable set of variables based on the problem size under analysis. Besides, two of those variables (the mapping variables) are based on the value of the other two variables (the synchrony variables) found by the solver, this makes the optimization process long and consumes exponentially the machine memory. For these reasons, we decided to stop the solver after finding a solution,

even if the solution is not optimal. This might match some realistic scenarios, in which the client might prefer a semi-optimal solution in an acceptable computational time, rather than an optimal solution in too long computational time. Thus, during the discussion of results the reader should consider that an optimal solution would perform even better in terms of synchronous resource sharing.

In all the following experiments, the SN size was fixed in 25 nodes. Initially all CPUs of SN nodes are free, and links BW is uniformly distributed between 1-3 Gbps. We start reporting twelve experiments divided into three groups, A, B and C, with VNs total size of 10, 20, and 30 nodes respectively. We refer to these scenarios together as *set 1* to facilitate reference. The VNs were generated with 3, 4, or 5 nodes each, and CPU demands 10%, 15%, or 25% of the SN nodes CPU capacity, and BW demands uniformly distributed between 100 Mbps and 1 Gbps.

In scenarios 1, 2, 3 and 4 of each group, the VNs synchrony demand varies between 0%, 30%, 60%, and 100%. The parameters for each experiment in the set 1 are described in Table 5.4.

**Table 5.4** — Space-HSVNs with C-SN scenarios parameters set (set 1)

Expe.	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
<b>VN size</b>	10 nodes				20 nodes				30 nodes			
<b>VNs sync.</b>	0%.	30%	60%	100%	0%	30%	60%	100%	0%	30%	60%	100%
<b>SN size</b>	25 nodes											
<b>SN BW</b>	uniformly distributed: 1 Gbps-3 Gbps											
<b>SN CPU</b>	nodes fully free initially											
<b>each VN size</b>	3,4,5 nodes											
<b>VNs BW</b>	uniformly distributed: 100Mbps-1Gbps											
<b>VNs CPU</b>	10,15,25 % of SN nodes CPU											

Earlier in this section (paragraph 5.1.2.1), we argued that for mapping virtual networks with hybrid synchrony demands over a settled SN, it is a waste of resources to use a fully synchronous SN. Rather, using a hybrid synchronous SN, together with an economic mapping process, reduces the cost. The mapping model proposed was aware of sparing the physical synchronous resources, and using them only when needed.

With a configurable SN (C-SN), we proposed a new approach for mapping HSVNs, with the goal of minimizing the mapping cost even more. We do not reserve, in advance, the synchronous resources on the SN. Rather, the SN synchrony is defined as an output of the IP. Thus, the physical resources chosen to be synchronous are subject to the VNs demands, and they change if the VNs demands change (*i.e.*, physical synchronous resources are different in each scenario in Table 5.4. The model proposed minimizes the number of physical synchronous resources to the limit enough for a given set of VNs.

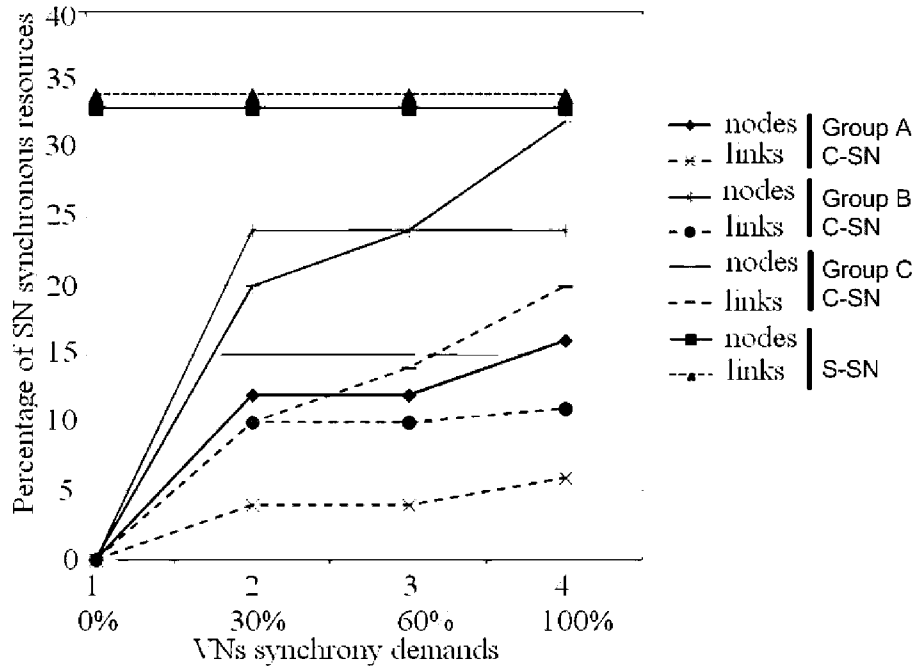
## 5.2.2 Results

The parameters considered for our analysis are: i) economy in the embedding cost; ii) physical resources load; iii) the kind of physical resources chosen to be synchronous, and iv) the topology of the subnetwork composed of synchronous resources on the SN.

### 5.2.2.1 Economy in the embedding cost

We consider the scenarios of *set 1* to be mapped in two cases: (i) on a settled SN with a predefined synchronous resources (33% synchronous nodes and 34% synchronous links), and (ii) a configurable SN with no predefined synchronous resources. We compare the number of used physical synchronous resources in either case. Figure 5.19 depicts the embedding cost for either case.

The continuous lines and the hashed lines illustrate the synchronous nodes and links percentage respectively. We notice that in the case of a S-SN the synchronous resources are fixed, 33% and 34% of the physical nodes and links respectively are predefined to be synchronous. In the case of a C-SN, we notice that the synchrony of the SN changes with the VNs synchrony demands.



**Figure 5.19** — Percentage of synchronous nodes and links in SN for scenarios in set 1 for S-HSVN and C-HSVN

By comparing the case of C-SN with the S-SN, we notice that a C-SN allows a clear reduction in the amount of SN synchronous resources needed to map the virtual components. For example, in scenario A4, 16% of synchronous nodes and 6% of synchronous links were enough to map the demand. In comparison with the case of S-SN, this means economizing

52% and 83% of SN synchronous nodes and links, respectively. Obviously, higher gains are observed in scenarios A1, B1, and C1, where no synchronous resources are required. Since in the S-SN approach some physical components must be synchronous independently of VNs demands, with a C-SN the improvement on synchronous resources reservation is of 100%. The reduction rate in the SN synchronous resources for the complete set of scenarios can be found in Table 5.5.

**Table 5.5** — Economy in SN synchrony resources between S-HSVN and C-HSVN-performed for scenarios set 1

SN sync resources	economy in SN sync nodes (%): economy in SN sync links (%)			
	1	2	3	4
Scenario				
Group A	100 : 100	64 : 88	64 : 88	52 : 83
Group B	100 : 100	27 : 71	27 : 71	27 : 68
Group C	100 : 100	39 : 71	27 : 59	3 : 41

In Figure 5.19, we note some observations: *i)* for each group, synchrony demands on links and nodes grow compatibly. *ii)* in some scenarios, even when VNs synchrony demands increases, the number of SN synchronous resources does not necessarily increase (*e.g.* scenarios B2, B3, and B4). This can be explained by resources sharing provided by VNs mapping. Several virtual components can be mapped on the same physical resources, as long as this does not violate the constraints of CPU and BW capacities for nodes and links, constraints 4.2 and 4.3, respectively. *iii)* the economy of links is more significant than of the nodes (see Table 5.5), this can be explained by the fact that virtual links can be mapped on physical paths which can be composed of one physical link or by several physical links. Thus, sparing one physical path reflects on sparing several physical links.

### 5.2.2.2 Physical resources load

We study the physical resources load for both S-SN and C-SN. We issue this study for one scenario only. We chose scenario C3, because within the experiments set, it is one with VNs of big size and high synchrony demands. In Figure 5.20 and Figure 5.21, we can read the Cumulative Distributed Function (CDF) for nodes load and links load respectively, for both S-HSVN and C-HSVN in scenario C3. We note that *(i)* with S-HSVN, 36% of the SN nodes and 64% of the links had 0 load (*i.e.*, not used for the mapping), whereas with a C-HSVN, this number raises up to 60% for the nodes and 74% for the links. *(ii)* with a S-HSVN, the maximum nodes load reached was 65%, whereas with C-HSVN, nodes reached higher load, where 4% of the SN nodes had full charge (*i.e.*, with 100% CPU load). Similar observation can be seen regarding the links.

Comparing the S-HSVN and C-HSVN used for mapping the same demands, we note that,

with C-HSVN, there are more free resources (i.e., unused nodes and links), but the drawback is that, the used resources are more loaded. In other words, with S-HSVN, the load is better balanced (i.e., more resources used with less load each).

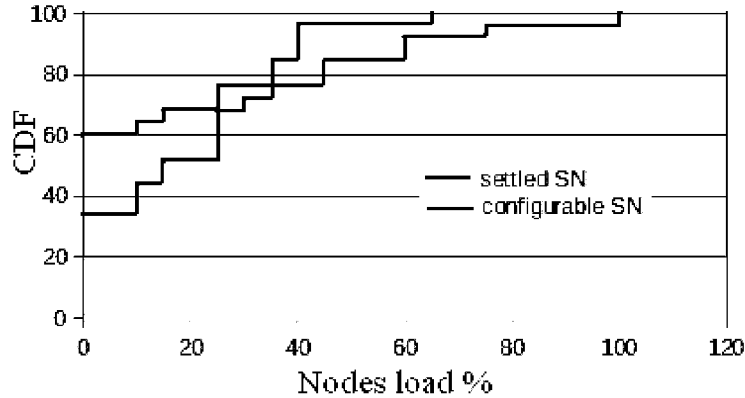


Figure 5.20 — CDF for nodes usage in experiment C3

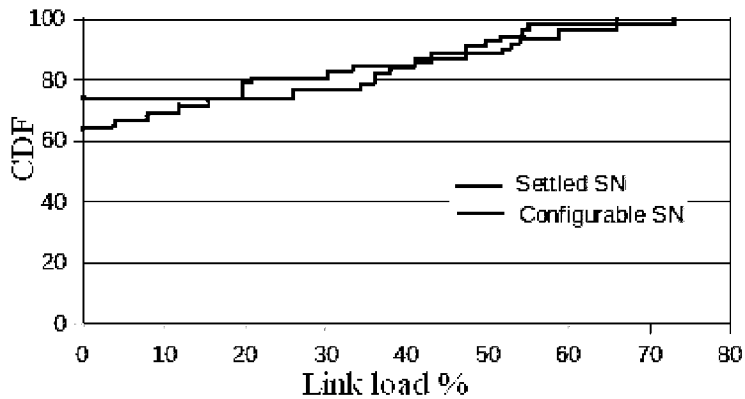
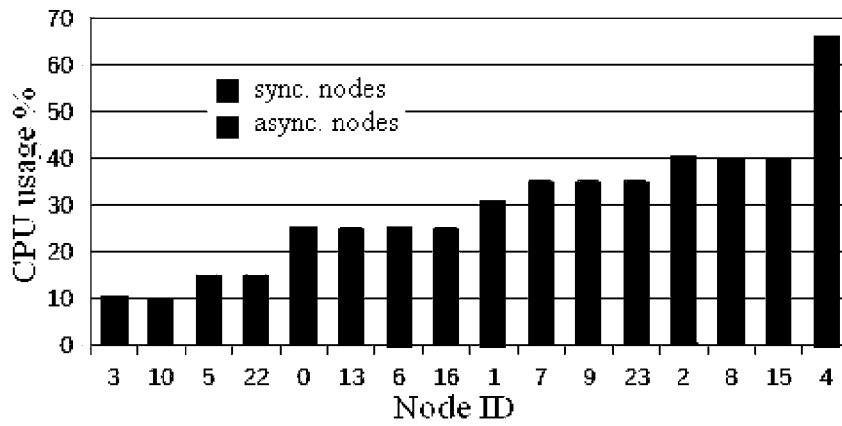


Figure 5.21 — CDF for links usage in experiment C3

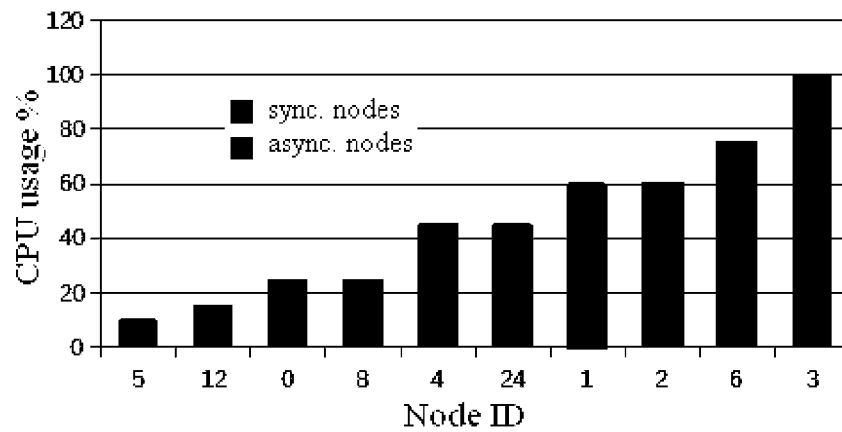
Next, we investigate the type of physical resources (the type in sense of synchrony properties) that are highly loaded in both S-HSVN and C-HSVN. For this purpose, the individual nodes and links load is depicted in Figure 5.22 and Figure 5.23 respectively, ordered in increasing order, illustrating as well the synchrony status of each physical node and link. We notice that, (i) with C-HSVN, the resources (i.e., nodes and links) with high load are the synchronous ones, whereas the resources with low load are the asynchronous ones, which is the reversed case in S-HSVN, and (ii) the load balance is clearer with S-HSVN through the fact that the synchronous and asynchronous resources are not concentrated on one side of the figure, rather they are used alternatively.

We conclude that, for mapping the same HSVN demands, it is true that the C-HSVN spares more the number of the synchronous resources used compared to the S-HSVN, but, the draw back is that, the C-HSVN charges the synchronous resources more, which is the expensive subset of resources.

Highlighting the advantages and drawbacks of S-HSVN and C-HSVN is not supposed to lead us to conclude which among both is more important, because both are of equal necessity. In fact, the infrastructure provider is the player who determines which HSVN model is to be adopted, based on the SN type he built (i.e., settled or configurable).



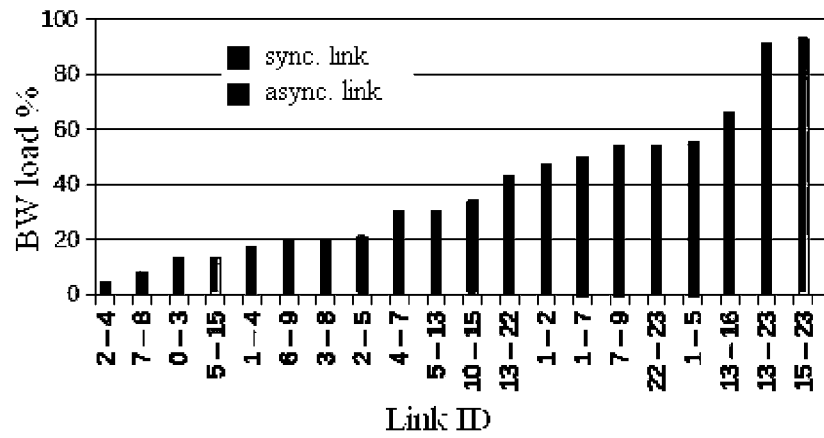
(a) settled SN %



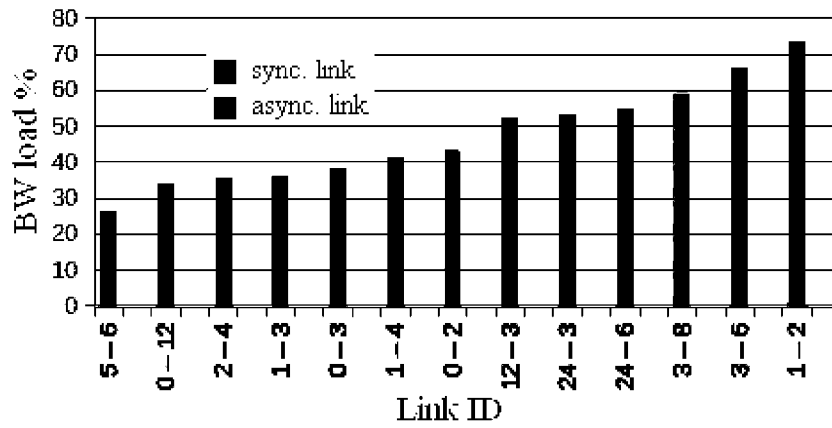
(b) configurable SN %

**Figure 5.22** — Individual nodes load in experiment C3





(a) settled SN %



(b) configurable SN %

Figure 5.23 — Individual links load in experiment C3

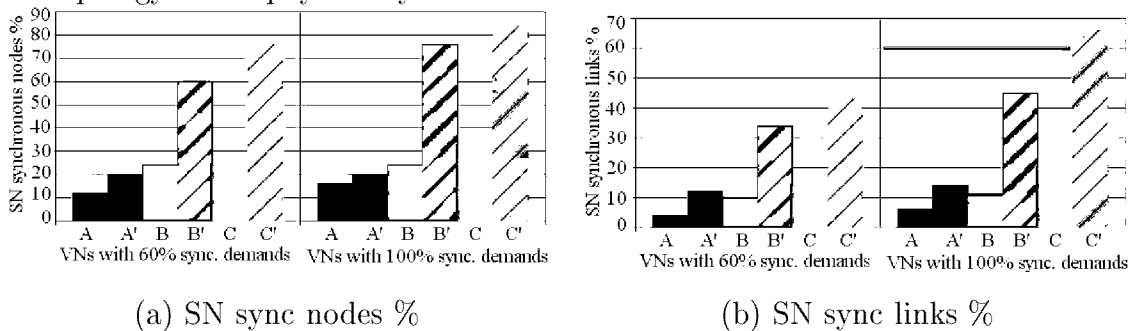
### 5.2.2.3 Privilege of synchronous nodes

To perform this kind of evaluation, we increased the problem size under analysis. We increase: *a)* the size of each virtual network which became 5 nodes each, *b)* the CPU demands that reached up to 50% of the physical nodes capacity, and *c)* the total size of VNs by adding Group D' with VNs total size of 40 nodes. The new set of scenarios are named set 2, and their parameters are shown in Table 5.6. Note that the groups in Table 5.6 are with prim sign (e.g., A', B', C', and D') to distinguish them from the groups in set 1 in Table 5.4.

**Table 5.6** — Experiments parameters in set 2

Expe.	A'1	A'2	A'3	A'4	A'5	A'6	B'1	B'2	B'3	B'4	B'5	B'6
VN size	10 nodes						20 nodes					
Expe.	C'1	C'2	C'3	C'4	C'5	C'6	D'1	D'2	D'3	D'4	D'5	D'6
VN size	30 nodes						40 nodes					
VNs sync.	0%.	20%	40%	60%	80%	100%	0%.	20%	40%	60%	80%	100%
SN size	25 nodes											
SN BW	uniformly distributed: 1 Gbps-3 Gbps											
SN CPU	nodes fully free initially											
each VN size	fixed to 5 nodes											
VNs BW	uniformly distributed: 100Mbps-1Gbps											
VNs CPU	10,20,30,40,50% of SN nodes CPU											

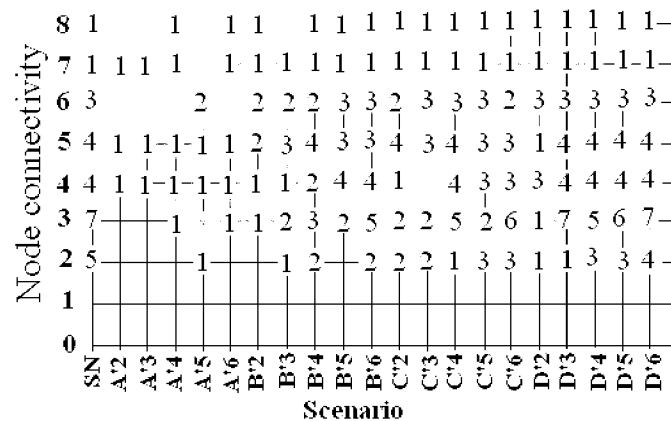
By increasing the problem size (scenarios in set 2), we push the model to allocate more physical resources (in comparison with set 1), and this allows us to perform our study on the physical resources chosen. Figures 5.24(a) and 5.24(b) depict a comparison in the SN synchronous resources needed in the counterparts scenarios in set 1 and set 2, at similar VNs synchrony demands, i.e. 60% and 100%. The figures illustrate that, the increment in the problem size has pushed the model to define larger set of synchronous physical nodes and links. And this justifies our choice of scenarios in Table 5.6 to conduct the study on the kind and topology of the physical synchronous resources.



**Figure 5.24** — Percentage of SN sync. resources: Comparing groups *set1* and *set2*

The SN described in Table 5.6 has nodes that vary in their connectivity degree between [2-8]. See Figure 5.25, its vertical axis indicates the connectivity degree, and its horizontal axis starts with the SN (as base of comparison, as we will see), and continues with each scenario performed in set 2. On the left column of Figure 5.25, we can see the number of

nodes available on the SN at each connectivity degree in the range [2-8]. For example, on the figure we can read that the SN has 1 node with connectivity 8, 1 node with connectivity 7, 3 nodes with connectivity 6, and so on. The rest of the figure illustrates the total number of physical synchronous nodes for each scenario in set 2, at each connectivity degree [2-8]. For example, in scenario B6, we note that the model has allocated 19 synchronous nodes, in a way that 1 was with connectivity 8, 1 with connectivity 7, 3 with connectivity 6, and so on. In general, we notice that the model tends toward choosing the physical nodes with high connectivity degree to be synchronous. For example, in scenario C5, the model chose all the physical nodes with connectivity [6-8] on SN to be synchronous (compare the numbers at scenario C5 with the base numbers at SN). A possible interpretation for this behavior is that, nodes with high connectivity degree allow multi-use of the same node, since on one hand, it is connected to a high number of neighbor nodes which fulfills topology constraints, and on the other hand, nodes with high connectivity have high bandwidth sum (*i.e.* the sum of BW capacity of all the physical links connected to it) which fulfills BW constraint. Since the embedding model of HSVNs over C-SN aims at minimizing the number of the synchronous resources (assured by the model objective function Equation (4.10)), then such nodes are chosen.

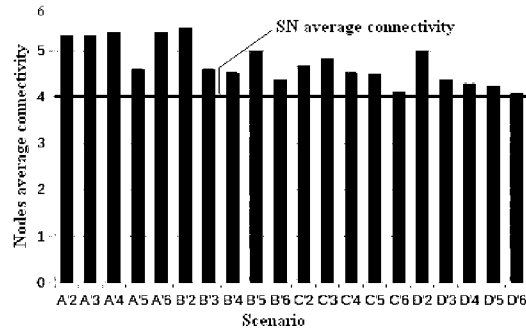


**Figure 5.25** — Frequency of synchronous nodes vs. nodes connectivity - scenarios in set 2

The average of the synchronous nodes connectivity chosen by the model is shown in Figure 5.26, the chart is plotted for each scenario executed. It is clear that, at each scenario, the connectivity average has exceeded the SN average connectivity, which reflects the model tendency towards choosing the nodes with high connectivity degree to be synchronous.

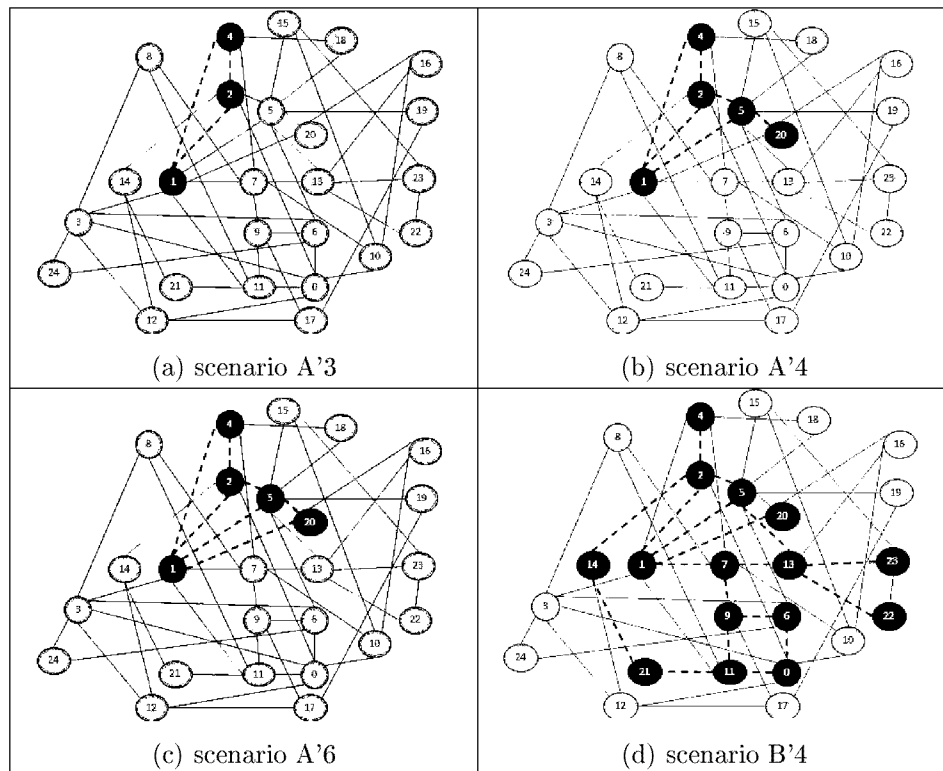
#### 5.2.2.4 Topological study

Next, we evaluate the topology of synchronous resources in scenarios set 2 (Table 5.6). Figure 5.27 depicts the topology of the synchronous resources in scenario A'3, A'4, A'6, and B'4 as an example. Synchronous nodes are plotted as red circles, and synchronous links as hashed



**Figure 5.26** — Average connectivity of synchronous nodes for scenarios in set 2

lines. We notice that the synchronous resources gather on one subnetwork, no synchronous islands are observed. Such a gathering allows multi-use of same synchronous nodes and links to answer given VNs, which fits with the model goal in minimizing the number of synchronous resources. We noticed that the topology of the synchronous subnetwork starts by a ring topology for small problem size, and tends towards becoming mesh topology with the increment in the problem size. This observation confirms the previous one regarding the synchronous nodes chosen, being the ones with high connectivity degree.



**Figure 5.27** — Topology divergence of synchronous resources [HMOD14]

We relate our observations in (5.2.2.2), (5.2.2.3) and (5.2.2.4) with a recent article by Luizelli et al. [LBB<sup>+</sup>16]. In this article, the authors provide consistent insights on how a physical network topology affects virtual network embedding quality. They note that substrate network topologies that are intrinsically more connected tend to reject a lower number of virtual requests, and consequently, tend to incur comparatively higher resource utilization.

In our work with Space-HSVNs over C-SN, the embedding model aims at minimizing the number of synchronous resources used, thus the physical resources chosen to be synchronous were: *i)* highly loaded; *ii)* nodes with high connectivity degree, *iii)* forming together a physical subnetwork that reveal high connectivity.

## Summary

In this chapter we evaluate the embedding models of Space-HSVNs on a settled SN (S-SN) and a configurable SN (C-SN). The main results show that:

### (1) With Space-HSVNs on S-SN:

- The proposed embedding model considers the hybrid synchronous nature of VNs, mapping them onto a hybrid synchronous SN in an economic manner, trying to spare the synchronous physical resources whose building cost is expensive when compared to the asynchronous ones
- The computational time is proportional to the number of the model variables, and this last one changes with the VNs size and synchrony demands.
- The model tends towards load distribution over the physical resources. This is an important feature in the embedding model because few heavily, or fully, loaded resources would increase the number of accepted VNs on a given SN.

### (2) With Space-HSVNs on C-SN:

- Adopting a C-SN allows a substantial spare of synchronous resources compared to the case of S-SN.
- With C-SN, there are more free resources than in S-SN, but the drawback is that, the used resources with C-SN are more loaded than with S-SN.
- The model tends towards choosing the physical nodes with high connectivity degree to be synchronous in order to minimize the number of synchronous nodes used. And the synchronous resources configured on the SN tend towards gathering in a mesh topology.

After presenting the Space HSVNs, we now address the Time-HSVNs in the next chapter. We note down that the proposed embedding framework for Space-HSVNs is able to answer Time-HSVNs. But this would result in an excess of cost. Considering the synchrony time variant nature in Time-HSVNs would result in further sparing of the use of physical synchronous resources.

**D**uring our research, we argued that Virtual Networks (VNs) and a suitable VN embedding process offer suitable environment for running distributed applications with partial synchrony. This has led to the abstraction of new type of virtual networks that we name *The Hybrid Synchrony Virtual Networks (HSVNs)*. They are virtual networks that have subsets of nodes and links that obey time bounds for processing and communication. Our previous contributions treated the *Space-HSVNs* considering physical resources hybrid in space, i.e. resources behave either synchronously or asynchronously during all the time. The *Space-HSVNs* are addressed to the DSs of hybrid synchrony in *space*, which we call *space hybrid synchronous systems*.

In this chapter we discuss hybrid synchrony virtual networks for the “time” dimension. We regard the time HSVNs abstractions and techniques as being a refinement of the space HSVNs, since it further defines repeating time windows of synchrony while allowing the resource to behave asynchronously for a period of time (as will be detailed through this chapter). The timely hybrid model leads to the possibility of further sparing synchronous resources, if compared to the space model, as will be presented in the model performance evaluation (Chapter 7).

In this chapter we *(i)* define the assumptions and abstractions needed to characterize both Substrate Networks (SNs) and Virtual Networks (VNs) suitable for Time-HSVNs, *(ii)* locate our work among others in the literature, and *(iii)* develop an embedding model for Time-HSVNs that answers the timely synchronous nature of the system and aware of sparing synchronous resources which are relatively expensive.

## 6.1 Time-HSVNs: definition

The timed-asynchronous model assumes that the system alternate between synchronous and asynchronous behavior. More specifically, according to [DLS88] partially synchronous systems can alternate between synchronous and asynchronous behavior, being hybrid in *time*. For each execution, there is a time after which the upper bound  $\delta$  is respected by the system. This time is called *Global Stabilization Time (GST)*. Since the upper bound cannot

hold forever, it is accepted that it holds just for a limited time  $\Delta_s$ . In practical terms,  $\Delta_s$  is the time needed for consensus to make progress or to be reached. We call these *timely hybrid synchronous systems*.

Actually, both models (i.e., hybrid models in *space* and in *time*) are not completely excludent. If a resource has no ability to behave synchronously, then it offers no time guarantees at all. However if a resource is able to behave synchronously, the space model defines that it is always synchronous while the time model defines that it behaves eventually synchronous as described above.

From the point of view of resources utilization, we can regard the time model as being a refinement of the space model, since it further defines repeated windows of synchrony while allowing the resources to behave asynchronously for a period of time. It is possible to support timely hybrid synchronous systems with Space-HSVNs, adopting the assumption and embedding model proposed for Space-HSVNs, but this choice would result in wasting synchronous resources because of reserving them for virtual demands that would behave synchronously only eventually (i.e., only during time windows). With the goal of sparing synchronous resources, we propose new type of HSVNs, that is the Time-HSVNs suitable for the timely hybrid synchronous nature of certain DSs.

## 6.2 Work positioning in the literature

The time-variant nature of networks has attracted considerable attention in the literature. Xie et al. [XDH12] observed that during the networking intensive phases of applications, collision and competition occurs for the network resources, resulting in making the applications running time unpredictable. The uncertainty in execution time further translates into unpredictable cost, as tenants need to pay for the reserved virtual machines for the entire duration of their jobs. Xie et al. [XDH12] propose the design of the first network abstraction (to the authors' best knowledge), TICV (Time Interleaved Virtual Clusters), that captures the time-varying nature of cloud applications, and they propose a systematic profiling-based methodology for making the abstraction practical and readily usable in today's data center networks. The network abstractions that [XDH12] consider are similar to those proposed in [GLW<sup>+</sup>10, BKR11], but the last two works overlook the real time variant nature of resource requirements and simply assume that the customer will specify them somehow.

Zhang et al. conducted a series of research that considers the bandwidth (BW) variant nature of VNs during the process of resources provisioning, [ZQT<sup>+</sup>11, ZQWL12, ZQW<sup>+</sup>14]. The authors modeled the time-variant nature of the VNs demands as the combination of a *basic* sub-requirement, which exists all through the VNs lifetime, and a *variable* sub-requirement, which exists with a probability. For the basic sub-flows, fixed bandwidth is

allocated (traditional BW sharing). But for the variable sub-flows, the authors consider a specific design of the SN; they assume that the time is partitioned into frames of equal length, and each frame is further divided into slots of equal length. The authors develop two first-fit algorithms that map the variable sub-flows to the time slots on the SN. The first algorithm does not consider the inter-flows collision per slot, whereas the second algorithm is aware of it. The inter-flows collision per slot is calculated based on the probability of occurrence of the variable sub-flows.

Our work considers the synchrony time-variant nature of virtual networks, addressed for timely synchronous distributed systems. The problem raised here (i.e., time-HSVNs embedding) could be solved using the models proposed in our previous works [HMD13, HMOD14, DODH15] by overlooking the synchrony timely-variant nature of VNs, but this would result in reserving synchronous resources permanently for demands that require synchrony only during time windows. In this chapter, we argue that, adopting suitable abstractions and techniques, together with a suitable embedding model, increases the resources usage efficiency.

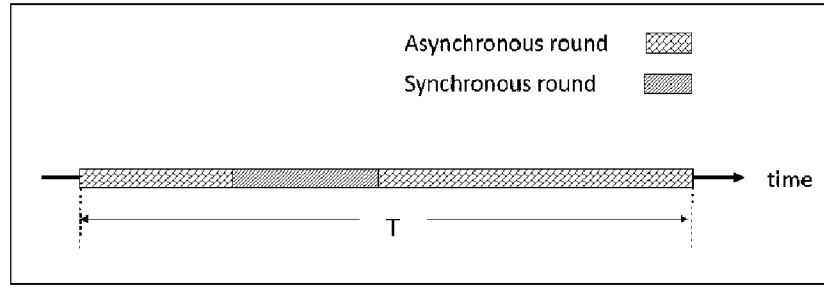
### 6.3 Time-HSVNs characterization

Time-HSVNs carry the common features of typical VNs [CB10], but in addition, they need to be further characterized to allow them to reflect the timely synchrony nature. We consider that the synchrony demands of each VN has a cyclic pattern with the cycle  $T$  time units. During  $T$ , each virtual node and link demands synchrony once, for a certain period of time. The time windows when the virtual element is provided synchrony is named *the synchronous round*, and the time windows when it is not provided synchrony is named *the asynchronous round*, see Figure 6.1. We assume that the client is able to define the synchronous round he needs within  $T$ , and he is able to express it to the virtual network provider. The client needs to be provided synchrony, eventually within  $T$ , during the specified time duration he expresses, without caring for *when* it will be provided within  $T$ . The VNs cyclic pattern makes them reflect the nature of timely synchronous DSs, which repeatedly demand eventual synchrony during the system life.

### 6.4 SN design

Previously in our work, we stated that the exact way of designing the physical synchronous resources is out of the scope of our work. Yet at this phase of our work (i.e., Time-HSVNs), the synchrony time variant nature of HSVNs implied taking care of the SN possible design, because the proposed design of the SN for Time-HSVNs is considered for the embedding

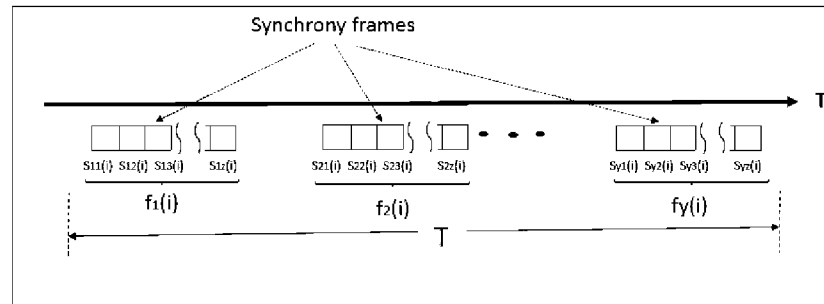




**Figure 6.1** — Synchronous and asynchronous rounds for a virtual node or link

model detailed later. In our previous works with Space-HSVNs, we distinguished between two types of resources; synchronous and asynchronous, where both types maintain their synchrony status during the system life. In the current step of our work, Time-HSVNs demand a refinement of the Space-HSVNs abstractions and techniques to suit better the new view of synchrony (i.e., periodic eventual synchrony).

In [ZQT<sup>+</sup>11], Zhang et al. propose a bandwidth sharing technique that allocates bandwidth (BW) in accordance with VNs traffic fluctuation as detailed in the related works (Section 6.2). The authors consider specific design of the SN as the following: the time is partitioned into frames of equal length, and each frame is further divided into slots of equal length, see Figure 6.2. The authors develop an algorithm that maps the variable sub-flows to the time slots on the SN in a way that the sub-flows mapped to the same slot do not violate the BW capacity, neither exceed a collision threshold allowed, where the sub-flows collision is calculated based on the probability of their occurrence. The proposed methodology allows an opportunistic bandwidth sharing between the sub-flows.



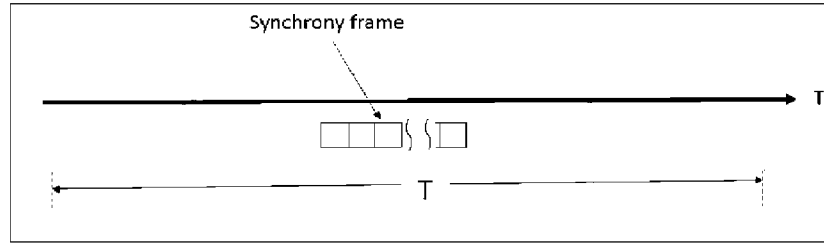
**Figure 6.2** — SN synchronous frames and slots proposed in [ZQT<sup>+</sup>11]

In our work, we inspire a suitable SN design from the work of Zhang et al. [ZQT<sup>+</sup>11] after adapting it in what matches our problem:

1. The virtual flows are of fixed BW demand during time (not opportunistic demands), thus, we disconsider collision probability.
2. the HSVNs demand synchrony once during  $T$ , see 6.3, so, we need only one time window during  $T$  that applies Zhang et al. technique. We name this time window *synchronous*

frame, see Figure 6.3. The synchronous frame is further partitioned into time slots of equal size, we name them *synchronous slots*.

3. the virtual demands mapped to a synchronous slot should not violate the physical BW capacity to eliminate competition and assure synchrony. The length of the synchronous frame and the number of time slots within a frame is related to the VNs number and demands. We assume that each virtual node and link do not demand synchrony slots that exceed the number of slots per synchronous frame.



**Figure 6.3** — Physical node or link synchrony frame during  $T$

## 6.5 Time-HSVNs Embedding

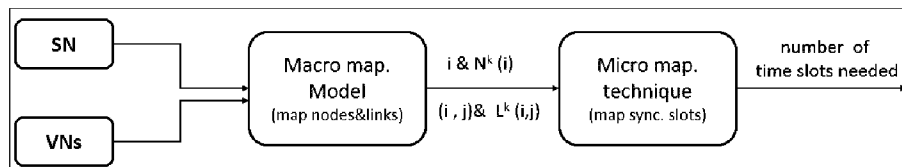
The virtualization architecture we adopt is the one proposed by Schaffrath et al. [SWP<sup>+</sup>09]. We assume that the virtual network provider (VNP) has complete information about: *i*) the SN topology and its attributes (nodes Central Processing Unit (CPU), links bandwidth (BW), and synchronous slots number and length), and *ii*) the virtual networks topology and demands (nodes CPU, links BW, synchrony demands). The VNP receives the synchrony demands in term of time period, and translates it into number of synchronous slots of the SN slots. We deal with the case of off-line VNs embedding. The time-HSVNs will be provided synchrony during the synchronous frame. Out of the synchronous frame, additional asynchronous demands can be mapped and competition can occur. This does not pose any problem for demands that do not expect synchrony.

The Time-HSVNs embedding problem can be stated as the following: *How to map the virtual synchronous slots to the physical synchronous slots, with the objective of minimizing the mapping cost represented by the used BW?*

The approach we followed for solving the Time-HSVNs was to benefit from our previous works on Space-HSVNs [HMD13, HMOD14, DODH15], by refining the proposed model for Space-HSVNs to a new version that expresses the synchronous slots. Further, we enhanced the achieved solution to allow more VNs to be mapped on the same SN. So, the Time-HSVNs mapping would go through two phases. We explain these two phases briefly in this section and we detail them more later through this chapter.

1. the macro mapping phase: This phase maps the virtual elements (nodes and links) to the physical resources that do respect all the embedding constraints. This phase leads to a mapping solution, that considers minimizing the physical bandwidth consumption. At the end of this phase, each virtual node and link will be mapped to a physical node and path that can support them. The macro mapping phase model is achieved by refining the Space-HSVNs mapping model to express synchronous slots.
2. the micro mapping phase: This phase increases the efficiency of the solution achieved in the macro mapping phase, by allowing embedding possible future VNs demands on the same given SN. This phase is performed individually for each physical node and link used in the macro mapping phase. The micro mapping phase maps the virtual synchronous demands to the physical synchronous slots. For solving the micro mapping phase, we adopted an off-the-shelf problem from the literature due to its similarity, that is the Cutting Stock problem (CSP).

Figure 6.4 depicts a block diagram for the Time-HSVNs embedding phases. The first phase is the macro mapping phase. Its inputs are the SN and VNs together with their attributes. This phase maps each virtual node and link on a physical node and path that answer all the embedding constraints. The second phase is the micro mapping phase. Its inputs are every physical node  $i$  with the set of virtual nodes it mapped  $N^k(i)$ , and every physical link  $(i, j)$  with the set of virtual links it mapped  $L^k(i, j)$ . The micro mapping phase is performed individually for every physical node and link. This phase maps the synchronous demands on the synchronous slots of the physical node or link. The output of the micro mapping phase will be indicating the minimum number of synchronous slots enough to map the synchronous demands of the virtual elements. For example, for the input  $i$  and  $N^k(i)$ , the micro mapping phase will tell us what is the minimum number of the synchronous slots of the node  $i$  that are enough to map all the synchronous demands of all the virtual nodes in  $N^k(i)$ .



**Figure 6.4** — Block diagram for Time-HSVNs embedding phases

Figure 6.5 illustrates a graph based example for Time-HSVNs embedding. The macro mapping phase is applied on two VNs and a SN, and the micro mapping phase is applied on the physical node  $i$  and the two virtual nodes it mapped:  $b^1$  and  $d^2$ .

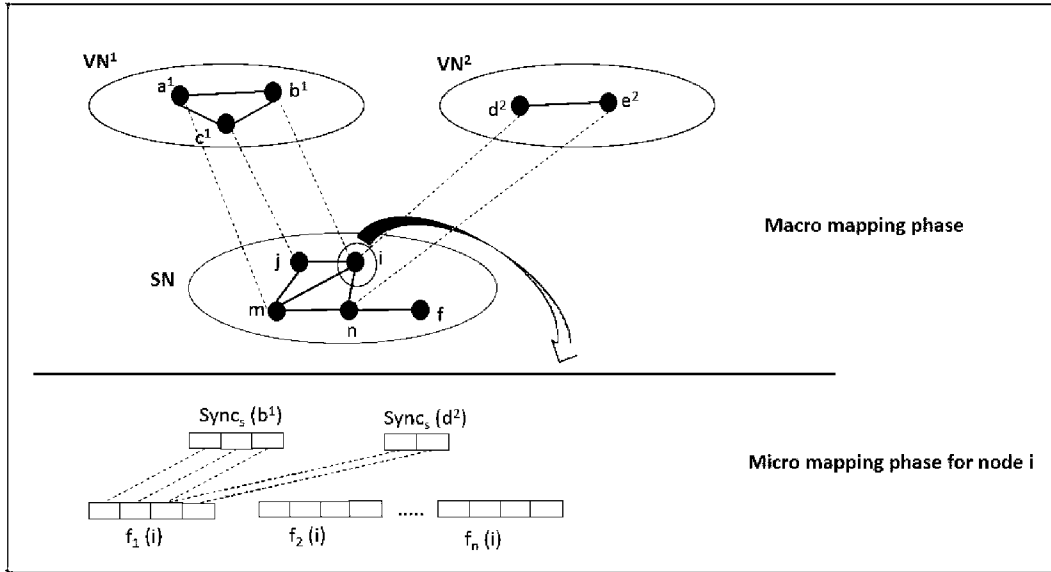


Figure 6.5 — Graph based example for Time-HSVNs embedding

### 6.5.1 The Macro Mapping Phase

The inputs of this phase are: (i) the SN topology and attributes, and (ii) the VNs topologies and demands. And the output of this phase will be assigning each virtual node to one physical node and each virtual link to a physical path, where a path can be composed of one link or more. At this phase, the problem turns to be: *how to map the VNs on top of the SN with the least physical bandwidth consumption possible*. We formulate the macro mapping problem in the shape of a Integer Program (IP).

#### 6.5.1.1 Variables definition

The SN is represented by an undirected graph  $G(N, L)$ , composed of a finite set of physical nodes  $N$  and links  $L : NXN$ . Analogously, each virtual network  $VN^k$  belonging to the set of virtual networks  $VN$  will be presented by an undirected graph  $G^k(N^k, L^k)$ . The number of synchronous slots provided by the physical node  $i$  and physical link  $(i, j)$  are  $sync(i)$  and  $sync(i, j)$ . Analogously,  $sync(i^k)$  and  $sync(i^k, j^k)$  are the number of synchronous slots demanded by the virtual node  $i^k$  and link  $(i^k, j^k)$ . Besides synchrony, two other attributes are considered for the SN and VN elements: nodes  $CPU$ , and links bandwidth ( $BW$ ). The syntax for those attributes on the SN and VN respectively are:  $cpu(i)$ ,  $bw(i, j)$ ,  $cpu(i^k)$ , and  $bw(i^k, j^k)$ .

Finally, we define the model output variables, they are: a binary function  $\sigma(i^k, i)$  that expresses whether node  $i \in N$  maps node  $i^k \in N^k$ , and a binary function  $\rho(i^k, j^k, i, j)$  that expresses whether link  $(i, k)$  is part of the physical path that maps the virtual link  $(i^k, j^k)$ . After solving the macro mapping model, each physical node  $i$  is mapping a set of virtual nodes  $N^k(i)$ , and each physical link  $(i, j)$  on the SN is mapping a set of virtual links  $L^k(i, j)$ .

Table 6.1 provides a list of variables definition for Time-HSVNs embedding model.

**Table 6.1** — List of variables definition for Space-HSVNs embedding model

Variables group	symbol	description
<b>Substrate Network</b>	$G(N, L)$	undirected graph representing the SN
	$N$	the set of physical nodes
	$L : NXN$	the set of physical links
	$i$	a notation for a physical node $i \in N$
	$(i, j)$	a notation for a physical link $(i, j) \in L$
	$sync(i)$	the number of synchronous slots provided by $i$
	$sync(i, j)$	the number of synchronous slots provided by $(i, j)$
	$cpu(i)$	the CPU of physical node $i$
<b>Virtual Networks</b>	$VN$	the set of all virtual networks
	$k$	the number of a virtual network that belongs to $VN$
	$VN^k$	The virtual network number $k$
	$G^k(N^k, L^k)$	undirected graph representing $VN^k$
	$N^k$	the set of virtual nodes
	$L^k : N^k X N^k$	the set of virtual links
	$i^k$	a notation for a virtual node $i^k \in N^k$
	$(i^k, j^k)$	a notation for a virtual link $(i^k, j^k) \in L^k$
	$sync(i^k)$	the number of synchronous slots demanded by $i^k$
	$sync(i^k, j^k)$	the number of synchronous slots demanded by $(i^k, j^k)$
	$cpu(i^k)$	the CPU of virtual node $i^k$
	$bw(i^k, j^k)$	the bandwidth of virtual link $(i^k, j^k)$
<b>Output variables</b>	$\sigma(i^k, i)$	$\sigma(i^k, i) = 1$ ; node $i \in N$ maps node $i^k \in N^k$ $\sigma(i^k, i) = 0$ ; otherwise
	$\rho(i^k, j^k, i, j)$	$\rho(i^k, j^k, i, j) = 1$ ; $(i, j) \in L$ is part of the path mapping $(i^k, j^k) \in L^k$ $\rho(i^k, j^k, i, j) = 0$ ; otherwise

### 6.5.1.2 The Macro mathematical model

It is formulated in the shape of an IP as bellow:

**Mapping objective-** The Objective Function (6.1), we consider is inspired from our work on Space-HSVNs, which is to minimize the total bandwidth used.

**Objective: minimize**

$$\sum_{\forall VN^k \in VN} \sum_{\forall (i^k, j^k) \in L^k} \rho(i^k, j^k, i, j) \cdot sync(i^k, j^k) \cdot bw(i^k, j^k); \quad (6.1)$$

**Mapping constraints-**

- *Capacity constraints:*

for every  $(i, j) \in L$  and every  $(i^k, j^k) \in L^k$

$$\rho(i^k, j^k, i, j) \cdot bw(i^k, j^k) \leq bw(i, j) \quad (6.2)$$

for every  $i \in N$  and every  $i^k \in N^k$

$$\sigma(i^k, i) \cdot cpu(i^k) \leq cpu(i) \quad (6.3)$$

- *Nodes mapping constraints:*

for every  $VN^k \in VN$ ,  $i^k \in N^k$

$$\sum_{\forall i \in N} \sigma(i^k, i) = 1 \quad (6.4)$$

for every  $VN^k \in VN$ ,  $i \in N$

$$\sum_{\forall i^k \in N^k} \sigma(i^k, i) \leq 1 \quad (6.5)$$

- *Links mapping constraint:*

for every  $VN^k \in VN$ ,  $(i^k, j^k) \in L^k$ ,  $i \in N$

$$\sum_{\forall j \in N} \rho(i^k, j^k, i, j) - \sum_{\forall j \in N} \rho(i^k, j^k, j, i) = \sigma(i^k, i) - \sigma(j^k, i) \quad (6.6)$$

- *Nodes synchrony constraints:*

for every  $i \in N$

$$\sum_{\forall VN^k \in VN} \sum_{\forall (i^k) \in N^k} \sigma(i^k, i) \cdot \text{sync}(i^k) \leq \text{sync}(i) \quad (6.7)$$

- *Links synchrony constraints:*

for every  $(i, j) \in L$

$$\sum_{\forall VN^k \in VN} \sum_{\forall (i^k, j^k) \in L^k} \rho(i^k, j^k, i, j) \cdot \text{sync}(i^k, j^k) \leq \text{sync}(i, j) \quad (6.8)$$

The capacity constraint (6.2) assures that the bandwidth of every virtual link does not exceed the bandwidth of the physical link mapping it. Similarly, constraint (6.3) represents the equivalent restriction regarding node *CPU*.

The node mapping constraint (6.4) assures that each virtual node is mapped once on a physical node.

Constraint (6.5) assures that virtual nodes belonging to the same  $VN$  are not mapped on the same physical node. This is to achieve load balancing besides improving the reliability. This procedure minimizes the number of virtual nodes prone to failure by a physical node failure.

For any virtual link  $(a, b)$ , the links mapping constraint (6.6), adopted from [ZWJY10], assures the creation of a valid physical path.

When mapping a set of virtual nodes  $N^k(i)$  on one physical node  $i$ ; the nodes synchrony constraint (6.7) assures that the number of virtual slots mapped on  $i$  do not exceed the number of synchrony slots provided by  $i$ . This constraint considers the worst case, when each virtual slot requires a complete synchrony slot alone without sharing. Similarly, constraint (6.8) represents the equivalent restriction regarding links synchrony.

## 6.5.2 The Micro mapping phase

This phase increases the efficiency of the solution achieved in the macro phase, by allowing embedding possible future VNs demands on the same SN. This is achieved by scheduling the synchronous demands efficiently within the synchronous frame. By viewing the problem at this stage as an optimization problem, the problem turns to be: *how to schedule the virtual demands within a synchronous frame minimizing the number of synchronous slots used*. Revising the literature, we found a very similar problem that is *the Cutting Stock Problem (CSP)* [HS91], which is one of the NP-hard problems cited by KARP [Kar72], and from the cutting stock problem we inspired the solution of the timely HSVNs micro mapping problem.

### 6.5.2.1 Revision on the Cutting Stock Problem (CSP)

In operations research, the cutting-stock problem is the problem of cutting standard-sized pieces of stock material (e.g., paper rolls or sheet metal) into pieces of specified sizes while minimizing material wasted. We explain the CSP through an example: A factory that produces rolls of  $W$  (c.m.) width, received a demand from a client in four sorts as the following:

- sort 1:  $b_1$  units of width  $0.5 * W$  (c.m.)
- sort 2:  $b_2$  units of width  $0.3 * W$  (c.m.)
- sort 3:  $b_3$  units of width  $0.3 * W$  (c.m.)
- sort 4:  $b_4$  units of width  $0.2 * W$  (c.m.)

For such a demand of four sorts, a roll of  $W$  (c.m.) can be cut into two units of sort 1 (of width  $0.5 * W$  (c.m.)), for example, or to three units of sort 3 (of width  $0.3 * W$  (c.m.)), or to one unit of sort 3 (of width  $0.3 * W$  (c.m.)) and three units of sort 4 (of width  $0.2 * W$  (c.m.)) ..., etc. The possible combinations of cutting patterns of a roll unit of  $W$  (c.m.) to fulfill the aforementioned demand can be tabulated using the *Delayed Column Generation* method, detailed in [HS91]. Every pattern  $j$  of these patterns tells that, a roll or  $W$  (c.m.) can be cut into  $a_{1j}$  units of demand sort 1 +  $a_{2j}$  units of demand sort 2 +  $a_{3j}$  units of demand sort 3 +  $a_{4j}$  units of demand sort 4. So, in pattern  $j$ , we will need  $X_j$  rolls of  $W$  (c.m.) width to generate  $a_{ij}$  units for each sort in the demand. The solution aims at minimizing the roll units of width  $W$  (c.m.) that are needed to fulfill the demand.

By considering *SORT* is the set of sorts in a given demand, and *PATTERNS* the set of possible cutting pattern, the *CSP* can be formulated in the shape of a linear program as the following:

**Objective: minimize**

$$\sum_{\forall j \in PATTERNS} (X_j) \quad (6.9)$$

**Cutting constraint**

for every  $i \in SORT$

$$\sum_{\forall j \in PATTERNS} (a_{ij} \cdot X_j) \geq b_i \quad (6.10)$$

### 6.5.2.2 The Micro mathematical model

Translating the CSP elements into the micro mapping problem:

- the *specified sized pieces* are the synchronous slots demanded by the VNs;
- the *standard-sized pieces of stock material* is the synchronous slot of physical node/link;
- a *pattern* in the CSP is the set of demands accepted within a stock, and in our problem a *pattern* will be the set of virtual slots accepted within the physical slot ;
- the objective of the CSP is to *minimizing the stock waste* and in our problem it would be to minimize the number of synchronous slots used within the physical synchronous frame.

We express the micro mapping model for one physical link, but it goes similarly for physical nodes.

Consider one physical link  $(i, j)$  with a synchronous frame of  $sync(i, j)$  slots, that maps a set of virtual links  $L^k(i, j)$ , where every virtual link has two attributes: the number of synchronous slots demanded  $sync(i^k, j^k)$  and the capacity of BW demanded  $bw(i^k, j^k)$ . Within each physical slot, the virtual synchronous slots that can be mapped to it form a pattern  $X_j$  [HS91]. The patterns are formed based on the BW of the virtual demands compared with the physical link BW. These patterns are the input to the micro model. The micro model aims at minimizing the number of synchronous slots used within the synchronous frame which is achieved by minimizing the total number of patterns (Equation (6.11)). Assuming  $a_{ij}$  is the number of times order  $i$  appears in pattern  $j$  [HS91]; constraint (6.12) assures providing every virtual link with a number of synchronous slots that is at minimum equal to the number of synchronous slots demanded  $sync(i^k, j^k)$ . The output of the micro mapping model will be telling which are the used patterns, and how many of each pattern is needed. Table 6.2 stands for variables definition.

**Objective: minimize**

$$\sum_{\forall j \in PATTERNS} (X_j) \quad (6.11)$$



**Table 6.2** — List of variables definition for Time-HSVNs micro mapping phase model

var.	symbol	description
Input var.	$PATTERNS$	set of possible patterns
	$L^k(i, j)$	set of virtual links mapped to $(i, j)$
	$j$	a given pattern in set PATTERNS
	$a_{ij}$	number times order $i$ appears in pattern $j$
	$i$	virtual link in $L^k(i, j)$
	$sync(i^k, j^k)$ $sync(i^k, j^k)$	sync. slots demanded by $(i^k, j^k)$
Output variables	$X_j$	number of units of pattern $j$

**Cutting constraint**

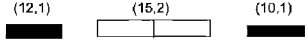
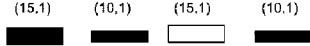




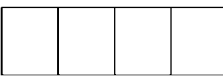

for every  $(i^k, j^k) \in L^k(i, j)$

$$\sum_{\forall j \in PATTERNS} (a_{ij} \cdot X_j) \geq sync(i^k, j^k) \quad (6.12)$$

After performing the micro mapping phase, the SN is updated (used BW and CPU is subtracted from the SN nodes and links capacity), and the macro mapping phase can run again, allowing more VNs to be mapped on the same SN. The combination on the macro phase, the micro phase, and the SN updating we name *an optimization cycle* (Opt\_cyc). Figure 6.6 illustrates two optimization cycles for mapping groups of virtual links on one physical link. The physical link updating happens either by reducing its capacity, or reducing the number of synchronous slots it supports. Either way, there will be a waste in the physical bandwidth, we refer to the bandwidth waste resulted by reducing the physical capacity  $W_h$ , and by reducing the synchronous slots  $W_v$ . The updating approach chosen is the one with the least waste (the smaller value between  $W_h$  and  $W_v$  is marked with a star \* in the figure). This updating methodology we follow is because the macro mapping phase (at the beginning of each optimization cycle) considers physical slots of equal capacity and fully empty.

**Summary**

In this section we define the Time HSVNs as a kind of VNs that has subsets of nodes and links that eventually behave synchronously. From the point of view of resources utilization, we can regard the time model as being a refinement of the space model, since it further defines repeated windows of synchrony while allowing the resources to behave asynchronously for a period of time.

	1st optimization cycle	2nd optimization cycle
VNs demands (bw,slots)	(12,1)      (15,2)      (10,1) 	(15,1)    (10,1)    (15,1)    (10,1) 
macro map. phase	 SN(100,4)	 SN(73,4)
micro map. phase		
SN updated	 Wh=56 * Wv=148 SN(73,4)	 Wh=150 Wv=23 * SN(73,3)

**Figure 6.6** — An illustrative scheme for the HSVNs optimization cycles

It is possible to support timely hybrid synchronous systems with Space-HSVNs, adopting the assumption and embedding model proposed for Space-HSVNs, but this choice would result in wasting synchronous resources because of reserving them for virtual demands that would behave synchronously only eventually

From the literature, we inspire a suitable design for the SN that will support Time-HSVNs. The assumed SN will have nodes and links that can provide synchrony during pre-defined time windows that we name synchronous frames. The VNs synchrony demands are supposed to be answered only during these frames.

We develop an embedding framework for the Time-HSVNs that consists of two phases:

1. the macro mapping phase: This phase maps the virtual elements (nodes and links) to the physical resources that do respect all the embedding constraints. The macro mapping phase model is achieved by refining the Space-HSVNs mapping model to express synchronous slots.
2. the micro mapping phase: This phase maps the virtual synchronous demands to the physical synchronous slots. This phase increases the efficiency of the solution achieved in the macro mapping phase, by allowing embedding possible future VNs demands on the same given SN. For solving the micro mapping phase, we adopted a problem from the literature due to its similarity, that is the Cutting Stock problem (CSP).

In the next Chapter we evaluate proposed model for Time-HSVNs embedding.

# Performance Evaluation for Time-HSVN

The Time-HSVNs abstractions we considered together with the Time-HSVNs embedding model are supposed to lead to further sparing of synchronous resources, if compared to the space model. We run preliminary experiments that allow investigating the performance of the proposed embedding approach of Time-HSVNs. The aspects considered during the analysis of our model are: *(i)* the embedding cost; *(ii)* the physical resources load; *(iii)* the optimization time; *(iv)* the topology of the physical sub-network composed of the used resources; and *(v)* the micro mapping phase efficiency.

## 7.1 Workload and tools

Experiments were designed as a full factorial [Jai91], exploring all possible combinations between the networks parameters. Such choice of experiments was done by other works like [BOB<sup>+</sup>12a]. Similar to [YYRC08a, BOB<sup>+</sup>12a], physical and virtual networks were randomly generated. For this we used BRITE tool (Boston university Representative Internet Topology gEnerator) [MLMJ14] with Waxman model [Wax88]. We implemented the mathematical model with ZIMPL language (Zuse Institute Mathematical Programming Language) [Koc04], both for the macro and micro mapping phases, and we used CPLEX [bmc14] to solve the Integer Program (IP), running on a computer Intel HM75, Core i3-3217U 1.80 GHz (Giga Hertz), cash 3 MB (Mega Byte), Random Access Memory (RAM) of 2 GB (Giga Byte), DDR3 and operating system Xubuntu 14.04.

In all the following experiments, the substrate network size was fixed to 15 nodes. Initially all CPUs (Central Processing Unit) of SN nodes are free, and links Band Width (BW) is uniformly distributed between 1-3 Gbps (Giga bit per second). We ran twelve experiments divided into three groups, A, B and C, with VNs total size of 10, 20, and 30 nodes in each group respectively.

The SN and VNs size was chosen of small scale to allow solving the embedding model during a reasonable time, allowing us to evaluate the model performance.

The VNs were generated with 3, 4, or 5 nodes each, and CPU demands 10%, 15%, or 25% of the SN nodes CPU capacity, and BW demands uniformly distributed between 100

Mbps (Mega bit per second) and 1 Gbps. VN nodes demand one synchronous slot per  $T$ . In scenarios 1, 2, 3 and 4 of each group, the virtual links synchronous slots demanded varies between 1, 2, 3, and 4 slots. The SN provides periodically, each  $T = 20$  seconds, a synchronous frame of 4 seconds length, divided into four equal time slots. Table 7.1 details the experiments parameters.

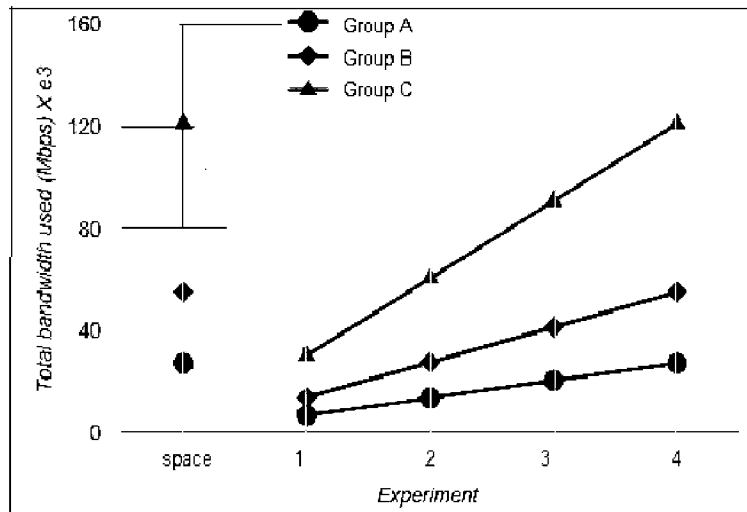
**Table 7.1** — Experiments parameters

Expe.	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
VN size	10 nodes				20 nodes				30 nodes			
Virtual links sync. slots	1	2	3	4	1	2	3	4	1	2	3	4
Virtual nodes sync. each VN size	1 slot per $T$ 3,4,5 nodes											
VNs BW	uniformly distributed: 100Mbps-1Gbps											
VNs CPU	10,15,25 % of SN nodes CPU											
SN size	15 nodes											
SN BW	uniformly distributed: 1 Gbps-3 Gbps											
SN CPU	nodes fully free initially											

## 7.2 Results

### 7.2.1 Embedding cost

The mapping cost is represented by the used BW, which is the model objective function (Equation (4.1)). We evaluate our results by comparing them with the BW used in case the experiments in Table 7.1 were mapped using the space model and the SN previously addressed in 4.6, see Figure 7.1.



**Figure 7.1** — Used bandwidth

With the time model, the used BW indicated is the one consumed within the synchrony frame. To allow just comparison, we calculate the BW used in the space model during time

window equal to the frame length. We note down the following main observations:

1. Within one experiments group, the used BW is proportional to the number of synchronous slots demanded by the VNs. For example, in scenario A2 the BW used is 13.624 Mbps/frame, and it is 20.436 in scenario A3, the proportion between the two figures is the proportion of the number of synchronous slots demanded in each scenario  $2/3$ . Similarly, the used BW in B3 and B4 is 41.37 (Mbps) and 55.16 (Mbps), their proportion is  $3/4$ . This means that the used BW is subject to the VNs synchrony demands in time, i.e., the more synchronous slots are; the higher used BW is.
2. By comparing the counterparts experiments of the three groups (e.g., A2, B2, and C2) we notice that the BW used increases. This is due to the VNs size increment, which tends naturally to reserve more resources. For example, the used BW with the aforementioned three experiments is 13.624, 27.58, and 60.52 (Mbps/frame) respectively. The reason is that bigger VNs demands more physical resources to map them, and thus more BW.
3. Within each experiment's group, the BW used with the space model is equal to the maximum BW used within the group (i.e., experiment 4 of each group). This is because the space model does not recognize the VNs synchrony slots demanded. For example, mapping the VNs in group B with the space model needs BW of 55,16 (Mbps/frame), which is the BW needed when mapping VNs in scenario B4 with the time model.
4. The time model is more efficient as it spares more resources. This is because the time model reserves resources proportionally to the synchrony demands in time, whereas the space model does not. For example, mapping the VNs in scenario B3 with the time model spares 33,33% of the resources needed when mapping the same set of VNs with the space model. And the spared ratio increases when the synchronous demands within  $T$  decreases, e.g. in B2, the time model spares 100% of the resources, and in B1 spares 300%.

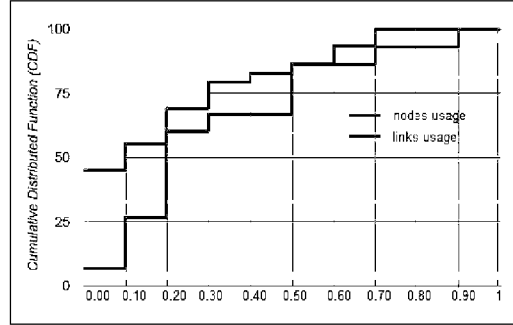
### 7.2.2 Physical resources load

This study is useful when done on a scenario that can possibly load the SN. We chose scenario C2 (VNs of big size). Figure 7.2 depicts the Cumulative Distributed Function (CDF) for physical nodes and links in experiment C2.

We note that 80% of the SN nodes had a load that varies between 10% and 60%, only 13.33% had a load between 60% and 80%, and no nodes were highly loaded more than 80%. And regarding the physical links, we note that, 41.37% of the SN links had a load that ranges

between 10% and 60%, and only 14% of the physical links had a load that exceeded 60%, and no links were fully loaded.

The SN resources seem to have load distribution which is good, since concentrating the load in certain elements will result in congestion, leading to block mapping certain VNs in the future. This is achieved because, the proposed model does not push the mapping process to exhaust the used physical resources before allocating new ones, rather, all resources are given the same chance to be chosen, as long as they allow mapping on the shortest path.



*Figure 7.2* — CDF for resource usage in experiment C2

### 7.2.3 Embedding time

The third parameter evaluated is the mapping time. The optimization process reached its end with scenarios A and B. Whereas in scenarios C, the optimization was terminated with optimization gap less than 2%. We took this decision when the optimization progressed slowly without much gain. For example, in scenario C3, it took 30 minutes to reach a solution with 4.46% gap, then another 33 minutes to reach another solution with 1.86% gap. In realistic scenarios, the client might prefer a semi-optimal solution in a short computational time, than an optimal solution after long time.

Table 7.2 illustrates the optimization time for embedding the scenarios of Table7.1 both with the space and the time models.

*Table 7.2* — Embedding time (in minutes)

Group.	space	exp.1	exp.2	exp.3	exp. 4
A	0.13	0.07	0.08	0.09	0.19
B	0.75	1.46	1.16	5.31	18.13
C	8	15.09	46.55	65.95	38.89

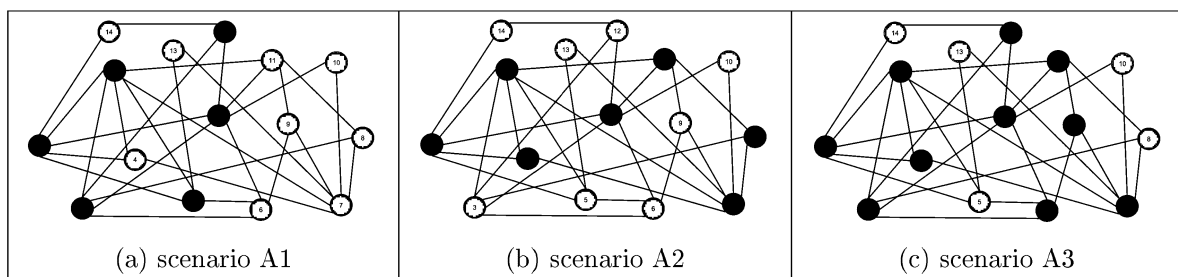
From Table 7.2 we notice that:

1. Most of the scenarios demanded optimization time that is less than 20 minutes, which is an acceptable computational time.

2. Solving the embedding model for Time-HSVNs took longer time than Space-HSVNs model.
3. For a given scenario, the difference in optimization time between Time-HSVNs and Space-HSVNs models increases with the increment of the problem size (i.e., VNs size and synchronous slots demanded) which increases the number of variables that need to be solved by the optimization process.

### 7.2.4 Topological study

In this subsection we study the topology of the physical subnetwork composed of the physical used elements (i.e., nodes and links). Figure 7.3 illustrates this topology for scenarios A1, A2, and A3. The topologies under study are the ones in red. We notice that, even though all these scenarios with the same VNs size (10 nodes), yet the model tends towards reserving more physical elements with the increment of the synchronous slots demanded by the VNs. For example, in scenario A1 the physical subnetwork under study is composed of 6 nodes, whereas in scenario A3 it is composed of 10 nodes. Previously, we noted down that the model tends towards distributing the BW load on the physical resources, Figure 7.2. Now we add that the model tends also towards distributing the synchrony load as well. So, when the VNs increase their synchrony demands (i.e., number of synchrony slots), the model tends towards reserving new elements. This behavior avoids congesting the synchronous frames of the used elements, allowing mapping new arriving VNs. Because the time HSVNs will be blocked or by exhausting the SN CPU and BW, or by exhausting the SN synchronous slots.



*Figure 7.3* — Topology divergence of used physical resources

### 7.2.5 Micro mapping model efficiency

The micro mapping model efficiency will be represented by the number of VNs accepted. The goal of this study is to define the parameters that affect the micro mapping efficiency.

We run this study on the case of one physical link and an endless queue of virtual links attended in order. We consider five experiment groups with different load range, Table 7.3. We run three experiments per group, with different synchrony demands. In

each experiment several `opt_cyc` are performed till the physical link is exhausted and no demands are accepted.

**Table 7.3** — Number of mapped virtual links with different load and synchrony demands

Scenario	VNs load/SN capacity	$sync(i^k, j^k)=1,2,3$ slots	$sync(i^k, j^k)=1,2$ slots	$sync(i^k, j^k)=1$ slot
<b>K</b>	(0-20] %	9	13	16
<b>L</b>	(20-40] %	6	8	12
<b>M</b>	(40-60] %	5	6	8
<b>N</b>	(60-80] %	1	2	4
<b>O</b>	(80-100] %	1	2	4

Our main observations:

1. The efficiency decreases when the virtual links load increases. For example, the efficiency in group K was 9, 13, and 16 whereas in group L it was 6, 8, and 12.
2. The efficiency increases when the maximum number of synchronous slots demanded decreases. For example, in group K, when the maximum number of synchronous slots demanded decreased from 3 to 1, the model efficiency increased from 9 to 16.
3. The micro model efficiency is the same in group N and O, the reason is that, both groups are with high virtual links load, this does not allow slots sharing between virtual demands, and the mapping solution achieved in the macro mapping phase cannot be optimized further with the micro mapping phase.

## Summary

In this chapter we evaluate the embedding models of Time-HSVNs. Simulation results show that the proposed embedding framework answers the synchrony time-variant demands efficiently (spares synchronous resources), distributes the load over the physical resources (nodes and links), and has an acceptable computation time for reaching the embedding solution. In addition, topological study of the subnetworks composed of the used resources on the SN showed that, the embedding model is aware of the synchronous demands variation, and the resulting subnetwork scatters more on top of the SN when the synchronous demands increase. Further study of the micro-mapping phase showed that its efficiency is a function of the VNs load and synchrony.



In this thesis, we provided a background about distributed systems (DSs), and lightened an important aspect in the field, the synchrony. While asynchronous DSs support no time-bounds for processes execution and message delivery, the synchronous DSs provide time guarantees for them. Although fully synchronous DSs demand simpler algorithms, and can provide what asynchronous ones do, yet the undeniable problem of synchronous components (processes and channels) high cost led to the development of hybrid synchrony DSs.

Two branches of hybrid synchronous DSs are distinguished in the literature: *(i)* the hybrid synchronous in space, where subsets of the system components are synchronous while the others are asynchronous, and *(ii)* the hybrid synchronous in time, where the system components alternate between synchrony and asynchrony over time.

We provided examples of applications (i.e., Apache Cassandra, Windows Azure, Chubby) that may benefit from the partial synchrony assumptions, as the progress will become guaranteed by the provisioning of elements designed to respect time upper bounds. The problem is that fully synchronous, or partially synchronous, environment is expensive to build, complex to configure, and difficult to control. This makes the infrastructure providers escape to asynchronous environments strengthened by algorithms and/or protocols with time-out specifications.

In a research for relaxing the ossifications of partial synchronous environment; we investigated the space of Virtual Networks (VNs), and we found that virtual networks can offer a suitable environment for hosting hybrid synchronous distributed systems while optimizing a set of their constraints due to the properties that virtualization brings.

By revising the literature on the topic of VNE, we note the absence of embedding solutions in the literature that consider the synchrony property in applications, which is of paramount importance to host a prominent class of distributed systems, the hybrid synchronous DSs. This gap led us to the development of an embedding framework that handles applications with hybrid synchrony constraints.

In our work, we propose and argue that virtual networks and the virtual networks embedding process offer both abstractions and techniques to support applications with Hybrid Synchrony demands. To our best knowledge, this is undiscussed in the VN field and is of paramount importance to host a prominent class of distributed systems. This has led to the abstraction of new type of VNs, we name it **The Hybrid Synchrony Virtual Networks**, abbreviated to **HSVNs**.

### What are HSVNs

They are virtual networks that have subsets of nodes and links that obey time bounds for processing and communication. Although HSVN can run on fully synchronous SN, this decision would have to pay the excess in an unneeded cost, since even asynchronous virtual nodes and links will be mapped on synchronous physical ones. We argue that hybrid synchronous SN, combined with a suitable embedding, is capable to answer the synchrony requirements in an economic manner. Hybrid synchronous SNs have two classes of nodes: (i) *synchronous nodes* with functioning time guarantees, achieved through the implementation of periodical real-time tasks, and (ii) *asynchronous nodes* that have no timely guarantees. Analogously, two classes of physical links are available: (i) *synchronous links* that have time-bounded messages transmission delay, achieved through the implementation of QoS policies and admission control, and (ii) *asynchronous links* that have no timely guarantee.

Two types of HSVNs can be distinguished, inspired by the two types of hybrid DS:

1. Space-HSVNs: where the virtual networks are composed of synchronous and asynchronous components, where both types of components maintain their synchrony status during the system functionality.
2. Time-HSVNs: where the virtual networks are composed of subsets of nodes and links that change their synchrony status over time (i.e., synchronous resources become asynchronous and vice versa).

By revising the literature on the topic of VNs embedding; we note the absence of embedding solutions in the literature that consider the synchrony property in applications, which we need for our work to attend DSs with hybrid synchrony. This gap led us to the development of an embedding framework that handles applications with hybrid synchrony constraints.

## 8.1 The thesis contributions

The main contributions of this thesis are:

- Provide a literature review on the topic of virtual networks embedding, classifying the works based on the embedding constraints.
- Address applications that may benefit from hybrid synchrony feature (namely: Cassandra, WA, and Chubby) and provide a review about each.
- Propose the use of VNs for hybrid synchronous applications, which results in the Hybrid Synchronous Virtual Networks (HSVNs).
- Define the Space-HSVNs and propose a suitable embedding model to handle the resources allocation problem.
- Define the Time-HSVNs and propose a suitable embedding model to handle the resources allocation problem.

## 8.2 Achieved results

The main results achieved with simulations are:

### 8.2.1 Space-HSVNs over a settled SN (S-SN)

- **Embedding cost:** *(i)* The proposed embedding model considers the hybrid synchronous nature of VNs, mapping them onto a hybrid synchronous SN in an economic manner, trying to spare the synchronous physical resources whose building cost is expensive when compared to the asynchronous ones. *(ii)* With the increase in the problem size (be it through the increase in the VNs size or through the increase in the synchrony demands), the mapping cost increases. *(iii)* A linear increase in the VNs size and/or synchrony demands leads to a nonlinear increment in the embedding cost.
- **Embedding time:** The computational time is proportional to the number of the model variables, and this last one changes with the problem size as the following: *(i)* when the VNs size increases, the model variables number increases, and thus the computational time increases, and *(ii)* when the VNs synchrony demands increases, the solution space becomes increasingly within the synchronous physical subnetwork, which is in our case a small portion of the SN, this means that the model variables used in the solution will become less, and thus the computational time to find the variables values would become less.
- **Free resources:** The number of free resources is a function of VNs size and synchrony demands as the following: *(i)* the number of free resources decreases with the increment in the VNs size, because bigger VNs will demand more resources to map them, and

(ii) the number of free resources in homogeneous VNs is more than it is in hybrid synchronous VNs because they avoid entirely the use of the physical subnetwork that does not match their synchrony nature.

- **Resources load:** Regarding the used physical resources, we notice that: (i) The model tends towards load distribution over the physical resources, that is why the number of resources at low load is bigger than the number of resources at high load. This is an important feature in the embedding model because few heavily, or fully, loaded resources would increase the number of accepted VNs on a given SN. (ii) The pattern of the accumulative number of used resources is a gaussian-like distribution pattern, its pick is at low load value for small VNs, and it shifts to higher load values with the increment in the VNs size and synchrony demands.

### 8.2.2 Space-HSVNs over a configurable SN (C-SN)

- **Embedding cost:** i) By comparing the case of C-SN with the S-SN, we notice that a C-SN allows a clear reduction in the amount of SN synchronous resources needed to map the virtual components. and ii) The economy of links is more significant than of the nodes which can be explained by the fact that virtual links can be mapped on physical paths that can be composed of one physical link or by several physical links. Thus, sparing one physical path reflects on sparing several physical links.
- **Resources load:** i) Comparing the S-HSVN and C-HSVN used for mapping the same demands, we note that, with C-HSVN, there are more free resources (i.e., unused nodes and links), but the drawback is that, the used resources are more loaded. In other words, with S-HSVN, the load is better balanced (i.e., more resources used with less load each). ii) With C-HSVN, the resources (i.e., nodes and links) with high load are the synchronous ones, whereas the resources with low load are the asynchronous ones, which is the reversed case in S-HSVN. So, the draw back is that, the C-HSVN charges the synchronous resources more, which is the expensive subset of resources.
- **Synchronous nodes privilege:** i) We notice that the model tends toward choosing the physical nodes with high connectivity degree to be synchronous. A possible interpretation for this behavior is that, nodes with high connectivity degree allow multi-use of the same node, since on one hand, it is connected to a high number of neighbor nodes which fulfills topology constraints, and on the other hand, nodes with high connectivity have high bandwidth sum (i.e. the sum of BW capacity of all the physical links connected to it) which fulfills BW constraint. Since the embedding model of HSVNs over C-SN aims at minimizing the number of the synchronous resources, then such nodes are chosen.

- **The topology of synchronous physical resources:** We noticed that the topology of the synchronous subnetwork starts by a ring topology for small problem size, and tends towards becoming mesh topology with the increment in the problem size. This observation confirms the previous one regarding the synchronous nodes chosen, being the ones with high connectivity degree.

Highlighting the advantages and drawbacks of S-HSVN and C-HSVN is not supposed to lead us to conclude which among both is more important, because both are of equal necessity. In fact, the infrastructure provider is the player who determines which HSVN model is to be adopted, based on the SN type he built (i.e., settled or configurable).

### 8.2.3 Time-HSVNs

- **Embedding cost:** It is possible to support timely hybrid synchronous systems with Space\_HSVNs, adopting the assumption and embedding model proposed for Space\_HSVNs, but this choice would result in wasting synchronous resources because of reserving them for virtual demands that would behave synchronously only eventually (i.e., only during time windows). The time model is more efficient than the Space model as it spares more resources. This is because the time model reserves resources proportionally to the synchrony demands in time, whereas the space model does not.
- **Resources load:** The SN resources seem to have load distribution which is good, since concentrating the load in certain elements will result in congestion, leading to block mapping certain VNs in the future. This is achieved because, the proposed model does not push the mapping process to exhaust the used physical resources before allocating new ones, rather, all resources are given the same chance to be chosen, as long as they allow mapping on the shortest path.
- **Embedding time:** *i)* The time model demands more time than the space model. And *ii)* For a given scenario, the difference between time and space models increases with the increment of the problem size (i.e., VNs size and synchronous slots demanded) which increases the number of variables that need to be solved by the optimization process.
- **The topology of synchronous physical resources:** The model tends towards distributing the synchrony load. So, when the VNs increase their synchrony demands (i.e., number of synchrony slots), the model tends towards reserving new elements. This behavior avoids congesting the synchronous frames of the used elements, allowing mapping new arriving VNs. Because the time HSVNs will be blocked or by exhausting the SN CPU and BW, or by exhausting the SN synchronous slots.

- **Micro mapping model efficiency:** The micro mapping model efficiency was represented by the number of VNs accepted. *i)* The micro model efficiency decreases when the virtual links load increases. *ii)* The efficiency increases when the maximum number of synchronous slots demanded decreases. *iii)* The micro model efficiency decreases when the virtual elements are with high CPU or BW demands, because this does not allow slots sharing between virtual demands, and the mapping solution achieved in the macro mapping phase cannot be optimized further with the micro mapping phase.

We note down that the proposed embedding framework for Space-HSVNs is able to answer Time-HSVNs. But this would result in an excess of cost. Considering the synchrony time variant nature in Time-HSVNs would result in further sparing of the use of physical synchronous resources.

### 8.3 Work application

In real life, there are some widely used applications that may benefit from our work. For example:

1. Apache Cassandra: a massively scalable open source NoSQL database.
2. Windows Azure Storage (WAS): a cloud storage system that provides customers the ability to store seemingly limitless amounts of data for any duration of time
3. Chubby lock service: The purpose of the lock service is to allow its clients to synchronize their activities and to agree on basic information about their environment.

These applications do not force the use of hybrid synchronous environment, although they need. These applications run on asynchronous environments supported with algorithms and/or protocols (e.g., PAXOS) where safety is assured but not progress. If these applications run on top of hybrid synchronous infrastructure then progress can be guaranteed by the provisioning of elements designed to respect time upper bounds. For example:

- In Cassandra: the failure detection algorithm needs to run on synchronous or hybrid synchronous subnetworks that communicate the cluster nodes, in order to guarantee delivering messages within the upper bound specified.
- In WA: The Intra-stamp replication protocols need to run on synchronous links that communicate the replicas on the same storage stamp together, while the Inter-stamp replication protocols may run on asynchronous links that communicate the storage stamps together.

- In Chubby: The failure detection algorithm needs to run on synchronous environment to adjust perfectly the messages delay time.

The problem is that fully synchronous, or partially synchronous, environment is expensive to build, complex to configure, and difficult to control. This makes the infrastructure providers escape to asynchronous environments strengthened by algorithms and/or protocols with time-out specifications. These applications may benefit from the partial synchrony assumptions that we propose in our work, as the progress will become guaranteed by the provisioning of elements designed to respect time upper bounds.

## 8.4 Work generalization

In our work, we motivated the abstraction of the HSVNs through the existence of certain class of distributed systems, namely the fault tolerant distributed systems, that can benefit from the hybrid synchrony. In fact, the abstraction of HSVN can be seen from a broader angle, we express this by generalizing the HSVN idea in three dimensions:

- Although we have dedicated enough efforts to illustrate perfect failure detectors and the consensus problem, a wider set of applications benefit from hybrid synchrony. For instance, general purpose applications would communicate mainly through asynchronous channels and still rely on timely execution triggers. Thus, certain actions would be executed in a timely fashion (e.g., check pointing [EAWJ02], election [MIMF00], or any round-based agreement).
- The hybrid SN we are proposing, combined with our embedding model, can host not only hybrid synchrony applications, but also homogeneous ones (fully synchronous or fully asynchronous).
- While in this step of our work we are concerned with synchrony, we envisage that similar models may, in the future, be used to denote other kinds of specific functionalities expected from the resources. such as subsets of nodes and links with special security or resilience features.

## 8.5 Work limitations

- With HSVNs, we consider that the synchrony pattern of each virtual node and link is independent of each other, which is not the case in real applications, e.g, the failure detector.

- For evaluating the performance of the embedding model for Time-HSVNs, we need to run statistical study, similar to that done for Space-HSVNs. This would provide higher confidence to the observations noted. But, due to time constraints we could not perform this statistical study.
- The assumptions and techniques we propose for HSVNs embedding are valid for offline mapping only, and not online mapping. For example, with Time-HSVNs we assumed that the client is able to define in advance the synchronous round he needs within  $T$ , and this may not be the case that a client can always do.
- In order to tackle the problem of resources allocation, we needed to characterize the HSVNs to reflect the time synchrony behavior, and for this we considered that each node and link would express synchrony in a cyclic pattern. This assumption might not be always valid for as the clients might not need synchrony in a cyclic pattern.

## 8.6 Future work

- With HSVNs, we consider that the synchrony pattern of each virtual node and link is independent of each other. In future works, we intend to consider cases when VNs elements synchrony is mutually dependent, to reflect better the nature of the time hybrid synchronous DSs.
- In a recent article by Luizelli et al. [LBB<sup>+</sup>16], the authors provide consistent insights on how a physical network topology affects virtual network embedding quality. They note that substrate network topologies that are intrinsically more connected tend to reject a lower number of virtual requests. In our future work we intend to better study the impact of the physical synchronous subnetworks topology on the SN and the HSVNs topologies on the embedding quality, basically in terms of embedding cost, time, and resources load.
- With Time-HSVNs, we scheduled the synchrony demands on the slots assuming that the synchrony demands exist together (i.e., at the same time), which might not be the case. This assumption represents the worst case. In our future work we intend to consider more realistic scenarios, when the synchrony demands might exist with a probability.



---

# Bibliography

- [AAC<sup>+</sup>03] I. F. Akyildiz, T. Anjali, L. Chen, J. C. Oliveira, C. Scoglio, A. Sciuto, J. A. Smith, and G. Uhl. A new traffic engineering manager for diffserv/mps networks: design and implementation on an ip qos testbed. *Journal of computer communications*, 26:388–403, 2003.
- [ABD<sup>+</sup>13] A. Abel, F. Benz, J. Doerfert, B. Dorr, S. Hahn, F. Hauptenthal, M. Jacobs, A. H. Moin, J. Reineke, B. Schommer, and R. Wilhelm. Impact of resource sharing on performance and performance prediction: a survey. In : *24th international conference on concurrency theory (CONCUR)*, pages 25–43. Springer, 2013.
- [ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Journal of IEEE transactions on dependability and secure computing*, 1(1), 2004.
- [APST05] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer journal*, 38(4):34–41, 2005.
- [BBC<sup>+</sup>98] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, December 1998.
- [Bea96] J. E. Beasley. *Advances in linear and integer programming*. Oxford science, 1996.
- [BFH<sup>+</sup>10] M. Bienkowski, A. Feldmann, D. Hurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer. Competitive analysis for service migration in vnets. In : *ACM SIGCOMM workshop on virtual infrastructure systems and architectures (VISA'10)*, 2010.
- [BFY<sup>+</sup>00] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A framework for integrated services operation over diffserv networks. RFC 2998, November 2000.

- [BHFdM13] J. F. Botero, X. Hesselbach, A. Fischer, and H. de Meer. Optimal mapping of virtual networks with hidden hops. *Journal of telecommunications system*, 52(3), 2013.
- [BHK12] A. Belbekkouche, M. M. Hasan, and A. Karmouch. Resource discovery and allocation in network virtualization. *Journal of IEEE communication surveys and tutorials*, 14(4):1114–1128, 2012.
- [Bir12] K. P. Birman. *Guide to reliable distributed systems*, chapter 15, pages 457–470. Springer, 2012.
- [BKR11] H. Ballani, T. Karagiannis, and A. I. T. Rowstron. Towards predictable data-center networks. In : *11th ACM SIGCOMM conference*, pages 242–253, 2011.
- [bmc14] International business machines corporation. Ibm ilog cplex optimizer. [online]. available: <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer>, October 2014.
- [BOB<sup>+</sup>12a] L. R. Bays, R. R. Oliveira, L. S. Buriol, M. P. Barcellos, and L. P. Gasparry. Security-aware optimal resource allocation for virtual network embedding. In : *8th international conference on network and service management (CNSM)*, Las Vegas, Nevada, USA, 2012.
- [BOB<sup>+</sup>12b] L. R. Bays, R. R. Oliveira, L. S. Buriol, M. P. Barcellos, and L. P. Gasparry. Um modelo para mapeamento ótimo de redes virtuais com requisitos de segurança. In : *12th brazilian symposium on information and computer science (SBSEG 2012)*, 2012.
- [Bre00] Eric A. Brewer. Towards robust distributed systems (invited talk). In : *principles of distributed computing*, 2000.
- [BRW12] R. Bless, M. Rohricht, and C. Werle. Authenticated quality-of-service signaling for virtual networks. *Journal of communications*, 7(1):17–27, 2012.
- [Bur06] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In : *7th symposium on operating system design and implementation*, pages 335–350, 2006.
- [CB09] N. M. Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges. *Journal of IEEE communications magazine*, 47(7):20–26, July 2009.
- [CB10] N. M. Chowdhury and R. Boutaba. A survey of network virtualization. *Journal of computer networks*, 54(5):862–876, 2010.

- [CDRS07] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. In : *14th ACM conference on computer and communications security*, New York, NY, USA, 2007.
- [CF99] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *Journal of IEEE Transactions on parallel and distributed systems*, 10(6):642–657, 1999.
- [CLW<sup>+</sup>10] Y. Chen, J. Li, T. Wo, C. Hu, and W. Liu. Resilient virtual network service provision in network virtualization environments. In : *IEEE 16th international conference on parallel and distributed systems*, 2010.
- [CN02] B. E. Carpenter and K. Nichols. Differentiated services in the internet. *Journal of proceedings of the IEEE*, 90(9), 2002.
- [CRB09] N. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In : *IEEE INFOCOM09*, Rio de Janeiro, Brazil, 2009.
- [CSB10] N. Chowdhury, F. Samuel, and R. Boutaba. Polivine: Policy-based virtual network embedding accross multiple domains. In : *ACM SIGCOMM workshop on virtual infrastructure systems and architectures (VISA '10)*, 2010.
- [CT96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
- [Cui12] Hongyan Cui. A new algorithm of virtual network embedding based on minimum node stress and adjacent principle. In : *globecom workshops (GC Wkshps)*, pages 792–796, Anaheim, CA, 2012.
- [dAMG09] R.J. de Araujo Macedo and S. Gorender. Perfect failure detection in the partitioned synchronous distributed system model. In : *ARES '09 international conference on availability, reliability and security*, pages 273–280, March 2009.
- [DG08] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Journal of ACM communications*, 51(1):107–113, 2008.
- [DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM (JACM)*, 35(2):288–323, 1988.
- [DODH15] R. R. De Oliveira, F. B. Dotti, and R. Hasan. Heurísticas para mapeamento de redes virtuais de sincronia híbrida. In : *33rd simpósio brasileiro de redes de computadores e sistemas distribuídos (SBRC)*, pages 291–304, 2015.

- [ea11] Brad Calder et al. Windows azure storage: a highly available cloud storage service with strong consistency. In : *23rd ACM symposium on operating systems principles (SOSP11)*, pages 143–157. ACM, 2011.
- [EAWJ02] Elmootazbellah Nabil Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B Johnson. A survey of rollback-recovery protocols in message-passing systems. *Journal of ACM computing surveys (CSUR)*, 34(3):375–408, 2002.
- [Emm97] Wolfgang Emmerich. Distributed system principles. Technical report, Department of computer science - university college London, 1997.
- [FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
- [Gar01] F. C. Gartner. A gentle introduction to failure detectors and related subjects. Technical report, Darmstadt university of technology, department of computer science, 2001.
- [GB95] T. M. Ghazalie and T. P. Bake. Aperiodic servers in a deadline scheduling environment. *Real-time systems*, 9(1):31–67, 1995.
- [GdAMR07] S. Gorender, R.J. de Araujo Macedo, and M. Raynal. An adaptive programming model for fault-tolerant distributed computing. *Journal of IEEE transactions on dependable and secure computing*, 4(1):18–31, 2007.
- [GGS01] G. C. Goodwin, S. F. Graebe, and M. E. Salgado. *Control system design*. Prentice Hall, 2001.
- [GH16] E. Ghazisaeedi and C. Huang. Energymap: energy-efficient embedding of mapreduce-based virtual networks and controlling incast queuing delay. In : *8th IEEE international conference on communication software and networks (ICCSN)*, pages 698–702, 2016.
- [GJ79] M. Garey and D. Johnson. *Computer and contractability: a guide to the theory of NP-completeness*. Freeman, New York, 1979.
- [GK00] Felix C Gartner and Sven Kloppenburg. Consistent detection of global predicates under a weak fault assumption. In : *19th IEEE symposium on reliable distributed systems*. IEEE, 2000.
- [GKA<sup>+</sup>16a] A. M. Ghaleb, T. Khalifa, S. Ayoubi, K. B. Shaban, and C. Assi. Surviving link failures in multicast vn embedded applications. In : *IEEE/IFIP network operations and management symposium*, pages 645–651, 2016.

- [GKA<sup>+</sup>16b] A. M. Ghaleb, T. Khalifa, S. Ayoubi, K. B. Shaban, and C. Assi. Surviving multiple failures in multicast virtual networks with virtual machines migration. *Journal of IEEE transactions on network and service management*, 13(4), 2016.
- [GLW<sup>+</sup>10] C. Guo, G. Lu, H. J. Wang, S. Yang, and C. Kong. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In : *ACM CoNEXT conference*, 2010.
- [God04] Wayne Goddard. *Introduction to algorithms*. Clemson university, 2004.
- [GWZL14] L. Gong, Y. Wen, Z. Zhu, and T. Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In : *IEEE conference on computer communications (INFOCOM)*, pages 1–9, Toronto, Canada, 2014. IEEE.
- [HAM10] D. Huang, S. Ata, and D. Medh. Establishing secure virtual trust routing and provisioning domains for future internet. In : *IEEE global telecommunications conference (GLOBECOM)*, 2010.
- [Her04] Kopetz Hermann. *Real time systems : design principles for distributed embedded applications*, page 338. Springer, 2004.
- [Hew11] Eben Hewitt. *Cassandra: the definitive guide*. Oreilly media, 2011.
- [HLAZ11] I. Houidi, W. Louati, W. B. Ameer, and D. Zeglache. Virtual network provisioning across multiple substrate networks. *Journal of computer networks*, 55(4):1011–1023, 2011.
- [HLZ08] I. Houidi, W. Louati, and D. Zeglache. A distributed virtual network mapping algorithm. In : *IEEE international conference on communications (ICC08)*, 2008.
- [HLZ<sup>+</sup>10] I. Houidi, W. Louati, D. Zeglache, P. Papadimitriou, and L. Mathy. Adaptive virtual network provisioning. In : *ACM SIGCOMM workshop on virtual infrastructure systems and architectures (VISA '10)*, 2010.
- [HLZB09] I. Houidi, Louati, D. Zeglache, and S. Baucke. Virtual resource description and clustering for virtual network discovery. In : *IEEE ICC09 workshops*, Dresden, Germany, 2009.
- [HMD13] Rasha Hasan, Odorico Machado Mendizabal, and Fernando Luís Dotti. Hybrid synchrony virtual networks: definition and embedding. In : *13th international conference on networks (ICN)*, pages 104–110, 2013.

- [HMOD14] Rasha Hasan, Odorico Machado Mendizabal, Rômulo Reis De Oliveira, and Fernando Luís Dotti. A study on substrate network synchrony demands to support hybrid synchrony virtual networks. In : *32nd simpósio brasileiro de redes de computadores e sistemas distribuídos (SBRC)*, pages 705–718, 2014.
- [HPN09] A. Haider, R. Potter, and A. Nakao. Challenges in resource allocation in network virtualization. In : *20th ITC specialist seminar*, October 2009.
- [HS91] Robert W. Haessler and Paul E. Sweeney. Cutting stock problems and solution procedures. *European journal of operational research*, 54:141–150, 1991.
- [HSWY12] W. H. Hsu, Y. P. Shieh, C. H. Wang, and S. C. Yeh. Virtual network mapping through path splitting and migration. In : *26th international conference on advanced information networking and applications workshops*, Fukuoka, Japan, 2012.
- [HW90] M. P. Herlihy and J. M. WING. Linearizability: a correctness condition for concurrent objects. *ACM transactions on programming languages and systems*, 12(3):463–492, 1990.
- [IR11] J. Infuhr and G. R. Raidl. Introducing the virtual network mapping problem with delay, routing and location constraints. In : *5th international networking optimization conference (INOC)*, Hamburg-Germany, 2011.
- [Jai91] Raj Jain. *Art of computer systems performance analysis techniques for experimental design measurements simulation and modeling*, chapter 16. Wiley computer, 1991.
- [Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, : *complexity of computer computations*, pages 85–103. Plenum Press, 1972.
- [Kle98] J. M. Kleinberg. Decision algorithms for unsplittable flow and the half-disjoint paths problem. In : *30th annual ACM symposium on theory of computing*, pages 530–539, New York, USA, 1998.
- [Koc04] T. Koch. *Rapid mathematical programming*. PhD thesis, Technische universität Berlin, 2004.
- [KR13] James F. Kurose and Keith W. Ross. *Computer networking: a top down approach*, chapter 7 (Multimedia networking), pages 648–655. Pearson, 2013.
- [Lam01] L. Lamport. Paxos made simple. *Journal of ACM SIGACT news*, 32(4):18–25, 2001.

- [LBB<sup>+</sup>13] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. Characterizing the impact of network substrate topologies on virtual network embedding. In : *9th international conference on network and service management*), pages 42–50, 2013.
- [LBB<sup>+</sup>16] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. How physical network topologies affect virtual network embedding quality: A characterization study based on isp and datacenter networks. *Journal of network and computer applications*, 70:1–16, 2016.
- [LCXX15] S. Liu, Z. Cai, H. Xu, and M. Xu. Towards security-aware virtual network embedding. *Journal of computer network*, 91(11):151–163, 2015.
- [Liu00] Jane W. S. W. Liu. *Real-time systems*. Prentice Hall PTR, 2000.
- [LK00] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*. McGraw-Hill, 2000.
- [LSS87] J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environment. In : *8th IEEE real-time systems symposium*, pages 110–123, San Jose, California, 1987. IEEE.
- [LT06] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. Technical Report 35, Washington university in St. Louis, 2006.
- [LZW15] H. Li, T. Zhou, and Q. Wang. The study of dynamic topology remapping in virtual network embedding. In : *international conference on information and communication technology convergence (ICTC)*. IEEE, 2015.
- [MF04] Zbigniew Michalewicz and David B. Fogel. *How to solve it: modern heuristics*. Springer, 2004.
- [MIMF00] Hiroyoshi Matsui, Michiko Inoue, Toshimitsu Masuzawa, and Hideo Fujiwara. Fault-tolerant and self-stabilizing protocols using an unreliable failure detector. *Journal of IEICE transactions on information and systems*, 83(10):1831–1840, 2000.
- [MLMJ14] A. Medina, A. Lakhina, I. Matta, and Byers J. Brite: Boston university representative internet topology generator [online]. available: <https://www.cs.bu.edu/brite/>, October 2014.
- [MMRT06] A. Mostefaoui, E. Mourgaya, M. Raynal, and C. Travers. A time-free assumption to implement eventual leadership. *Journal of parallel processing letters*, 16(2):189–207, 2006.

- [Oli13] Rodrigo Ruas Oliveira. *Toward cost efficient, DoS-resilient virtual networks with ORE: opportunistic resilience embedding*. PhD thesis, Institute of informatics, Universidade federal do rio grande do sul, 2013.
- [OS03] Eric Osborne and Ajay Simha. *Traffic engineering with MPLS: design, configure and manage MPLSTE to optimize network performance*. Cisco Press, 2003.
- [Pad99] M. Padberg. *Linear optimization and extensions*. Springer-Verlag, 1999.
- [PG73] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. In : *4th ACM symposium on operating system principles*, pages 412–421, 1973.
- [Pli99] J.L. Pline. *Traffic engineering handbook*. Institute of transportation engineering, 1999.
- [RAB10] M. R. Rahman, I. Aib, and R. Boutaba. Survivable virtual network embedding. In : *9th IFIP TC 6 international conference on networking*. Springer, 2010.
- [RB13] M. R. Rahman and R. Boutaba. Svne: survivable virtual network embedding algorithms for network virtualization. volume 10, pages 105–118, 2013.
- [RSK<sup>+</sup>00] C. O. Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, and D. L. Levine. Evaluating policies and mechanisms for supporting embedded real-time applications with corba 3.0. In : *real-time technology and applications symposium*, pages 188–197. IEEE, 2000.
- [RSK<sup>+</sup>01] Carlos Ryan, Douglas C. Schmidt, Fred Kuhns, Marina Spivak, Jeff Parsons, Irfan Pyarali, and David Levine. Evaluating policies and mechanisms to support distributed real-time applications with corba. *Journal of practice and experience*, 13(2):1–21, 2001.
- [Sch93a] Fred B. Schneider. In Mullender, editor, *Distributed systems (2nd Ed.)*. ACM press/Addison-Wesley publishing co., 1993.
- [Sch93b] Fred B. Schneider. *Distributed systems (2nd Ed.)*, chapter 2 (What Good are Models and What Models are Good?), pages 17–26. Wesley publishing co., 1993.
- [Sie02] G. Sierksma. *Linear and Integer Programming: Theory and Practice, Second Edition*. Marcel Dekker, 2002.
- [SWP<sup>+</sup>09] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, and O. Maennel. Network virtualization architecture: proposal and initial prototype. In : *1st ACM workshop on virtualized*



- infrastructre systems and architectures, ser. VISA '09*, pages 63–72, New York, NY, USA, 2009.
- [Szt12] J. Sztrik. *Basic queueing theory*. University of Debrecen, faculty of informatics, 2012.
- [TEA11] T. Trinh, H. Esaki, and C. Aswakul. Quality of service using careful overbooking for optimal virtual network resource allocation. In : *8th Thailand conference on electrical engineering, electronics, computer, telecommunications and information technology (ECTI)*, 2011.
- [TT05] J. Turner and D. Taylor. Diversifying the internet. *IEEE global telecommunications conference (GLOBECOM05)*, 2, 2005.
- [VC02] Paulo Veríssimo and António Casimiro. The timely computing base model and architecture. *Journal of IEEE transactions on computers*, 51(8):916–930, 2002.
- [Ver06] Paulo E. Verissimo. Travelling through wormholes: a new look at distributed systems models. *Journal of ACM SIGACT News*, 37(1):66–81, 2006.
- [vsgNwp12] Network virtualization study group (NVSG) white paper. Advanced network virtualization: definition, benefits, applications, and technical challenges. Technical report, 2012.
- [Wax88] B. M. Waxman. Routing of multipoint connections. *Journal of selected areas in communications*, 6(9):1617–1622, 1988.
- [WWG<sup>+</sup>16] Z. Wang, J. Wu, Z. Guo, G. Cheng, and H. Hu. Secure virtual network embedding to mitigate the risk of covert channel attacks. In : *computer communications workshops (INFOCOM WKSHPs)*. IEEE, 2016.
- [XDH12] D. Xie, N. Ding, and R. Hu, Y.C. andKompella. The only constant is change: incorporating time-varying network reservations in data centers. In : *SIGCOMM conference*, 2012.
- [YAQS11] H. Yu, V. Anada, C. Qiao, and G. Sun. Cost efficient design of survivable virtual infrastructre to recover from facility node failures. In : *IEEE international conference on communications*, 2011.
- [YWK10] W. L. Yeow, C. Westphal, and U. C. Kozat. Designing and embedding reliable virtual infrastructures. In : *ACM SIGCOMM workshop on virtualized infrastructure systems and architectures (VISA '10)*, 2010.

- [YYRC08a] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *Journal of ACM-SIGCOMM*, 38(2):17–29, 2008.
- [YYRC08b] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *Journal of ACM-SIGCOMM*, 38(2):1011–1023, 2008.
- [ZA06] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In : *IEEE INFOCOM06*, 2006.
- [ZI07] D. Zhang and D. Ionescu. Qos performance analysis in deployment of diffserv-aware mpls traffic engineering. In : *8th ACIS international conference on software engineering, artificial intelligence, networking, and parallel/distributed computing*, pages 963–968. IEEE computer society, 2007.
- [ZQT<sup>+</sup>11] S. Zhang, Z. Qian, B. Tang, J. Wu, and S. Lu. Opportunistic bandwidth sharing for virtual network mapping. In : *global telecommunications conference*, pages 1–5, 2011.
- [ZQW<sup>+</sup>14] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein. Virtual network embedding with opportunistic resource sharing. *Journal of IEEE transactions on parallel and distributed systems*, 25(3):816–827, 2014.
- [ZQWL12] S. Zhang, Z. Qian, J. Wu, and S. Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In : *IEEE INFOCOM conference*, pages 2408–2416, 2012.
- [ZWJY10] M. Zhang, C. Wu, M. Jiang, and Q. Yang. Mapping multicast service-oriented virtual networks with delay and delay variation constraints. In : *IEEE global telecommunication conference (GLOBECOM)*, Miami, FL, 2010.