ICT-317756

TRILOGY2

Trilogy2: Building the Liquid Net

Specific Targeted Research Project

FP7 ICT Objective 1.1 The Network of the Future

D2.1 Initial Liquid Net Architecture

Due date of deliverable: 31 December 2013

Actual submission date: January 30, 2014

Start date of project	1 January 2013
Duration	36 months
Lead contractor for this deliverable	Universidad Carlos III de Madrid
Version	v1.0 , 24-Jan-2014
Confidentiality status	Confidential to TRILOGY2 project and Commission Services

Abstract

Trilogy 1 successfully defined a set of mechanisms to provide optimal fault tolerant transport across an IP network by creating and exploiting multiple simultaneous path across the network. Trilogy 2 sets out to extend the results of Trilogy 1 to be cross-resource, cross-layer, and cross-provider. Cross-resource seeks a generalisation to all IT resources including processing and storage. Cross-layer seeks to generalise beyond the simple two layers of Trilogy 1 and include intermediate layers. Cross-provider seeks to provide solutions which can work across commercial boundaries. The architecture described in this document gives a framework which directly addresses the requirements of cross-resource and cross-layer and implicitly cross-provider. The architecture is based on an extension of functional block methodology to incorporate virtualisation and as a result includes features beyond current state of the art. It has been already been presented to and adopted by the ETSI NFV ISG for its work.

Target Audience

The ultimate target audience for this deliverable is the community of knowledge engineers who define the structure of ICT systems, and those who define the standards and frameworks that are necessary for these ICT systems to interwork across the industry. While this initial deliverable is confidential to the Trilogy 2 consortium, the audience is a) the project participants to ensure the whole is understood to be greater than the parts and b) the project's scientific advisory board and reviewers to articulate the approach being taken across the project in order to elicit useful feedback and criticism.

Impressum

Full project title	TRILOGY2: Building the Liquid Net
Title of the workpackage	WP2 Tussle over Liquidity
Editor	Andy Reid, BT
Project Co-ordinator	Marcelo Bagnulo Braun, UC3M
Copyright notice	© 2014 Participants in project TRILOGY2

Executive Summary

The Trilogy 2 project follows on from the Framework 7 Trilogy project (2008-2011). The the original Trilogy project (Trilogy 1) developed mechanisms to create and control 'liquidity' of bandwidth in the Internet. The aim of Trilogy 2 is extend the result of Trilogy 1 to create and control liquidity of other resources, in particular storage and processing, as well as to extend the applicable environments beyond the Internet to mobile devices and more general operator infrastructure.

This document describes an initial view on the architecture for the Trilogy 2 project. The requirements for the architecture start with a consideration of the various use cases which are software platforms being developed within the project which and include the following.

- *Bandwidth Liquidity*
  - *MultiPath TCP (MPTCP)* - enhancements to the protocol for controlling bandwidth liquidity developed in Trilogy 1
  - *PolyVersal TCP (PVTCP)* - extension of the concept of MPTCP to optimise liquidity across multiple protocols including protocols at different layers, for example HTTP
  - *ConEx* - a protocol designed to enable providers of communications infrastructure to give the correct economic incentives to data senders to manage their traffic load and to enforce limits if they do not.
- *Storage Liquidity*
  - *Irminsule* - a large-scale distributed database co-ordination scheme
  - *Trevi* - a transport protocol specialised for controlling data transfer to and from multiple storage replicas (i.e. bandwidth liquidity specific to storage I/O)
- *Processing Liquidity*
  - *Virtual Broadband Remote Access Servers (vBRAS)* - a high performance, distributed, load adaptive, fault tolerant implementation of a network operator's BRAS function
  - *Virtual CPE* - a flexible and virtualised implementation on customer premises equipment functionality which can be host in a datacentre and not at the customer site
  - *Virtual Machine Migration for Mobile Devices* - a mechanism which can migrate an application between mobile devices including the GUI
  - *Mobile Kibbutz* - a way for a localised group of mobile devices to take turns in using each other's cellular connections

These uses cases have then been distilled into three broad canonical problems which highlight particular common issues in the creation and control of liquidity across the broadened scope of Trilogy 2. These three are

- *Infrastructure as a Service (IaaS) Resource Control* - cloud technology have developed a number of 'as a service' concepts including Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). While each of these is a mechanism which creates liquidity, it is the last of these, IaaS that is most directly relevant to Trilogy 2 and on which many of the use cases are at least broadly based. IaaS allows physical servers and storage to create virtual servers and storage, that is 'virtual machines' and virtual machines are common to many use cases. Cloud technologies already have some mechanisms for controlling the liquidity of IaaS which the project seeks to both exploit and extend.

- *Consequences of Middleboxes* - middleboxes are in widespread use by service providers today in an attempt to control traffic at higher protocols layers than IP. These boxes directly contradict the end to end principle of the Internet and therefore create complexity as protocols above IP cannot be assumed to be transparent to higher layer protocols such as TCP and HTTP and so solutions based on adaptations to these high layer protocols may not work. Trilogy 2 has two reasons to focus on middleboxes

  - the existence of middleboxes hinders the operation of bandwidth liquidity mechanisms and therefore makes it worthwhile to extend liquidity mechanisms to even higher layer protocols (see PVTCP)
  - the middleboxes themselves are frequently implemented as software on generic servers and therefore are strong candidate for liquid implementation (see vBRAS)

- *Lightweight Virtualisation* - the standard virtual machines of cloud computing are generally designed to be of a size and performance needed to run applications designed to run on dedicated servers. As a result VMs tend to be large and a physical server will only support a small number of VMs. However, there are use cases where a much smaller and more flexible scale of VM would create more efficient liquidity of processing and storage resources.

When seeking to address these canonical problems, it is apparent that there is no general framework within which solutions can be developed. In particular, the general techniques of functional modelling and functional specification does not have the means of describing the essential properties of liquidity. This observation was also apparent in the work of the ETSI ISG on Network Functions Virtualisation (ETSI NFV ISG).
This deliverable develops and presents a basic functional architecture which can accurately describe and specify functionality, including processing and storage, in such a way that functional uses can be decoupled from the resources that implement these uses, thus allowing for liquidity in the implementation. This basic functional architecture can then give a technical framework and language for the rest of the architecture.

This basic functional architecture uses and extends the well know functional modelling method of systems engineering to include virtualisation. At this stage, virtualisation is treated in a very general way which can include virtual machines, virtual networks, and effectively any network service. Virtualisation is also shown to be conceptually identical to abstraction at this fundamental level, and more significantly, it is the basic mechanism which creates liquidity.

Having established this basic framework, this document then develops and presents an initial view of the abstract functional architecture for Trilogy 2. This is based on virtual machines (which present processing and storage resources) and virtual networks (which present bandwidth resources) and the basic mechanisms for creating and mapping virtual functions to these liquid VM and VN resources.

Finally, important aspect of security are developed into an initial security architecture. This is an area of particular importance as the creation of liquidity opens up a number of potential security threats.

Future work on the architecture will develop the Trilogy 2 abstract functional architecture as well as developing functional architecture for each of the use cases along with, respectively the associated Trilogy 2 abstract information model and the use case specific information models. These relationships are illustrated in the Figure below.

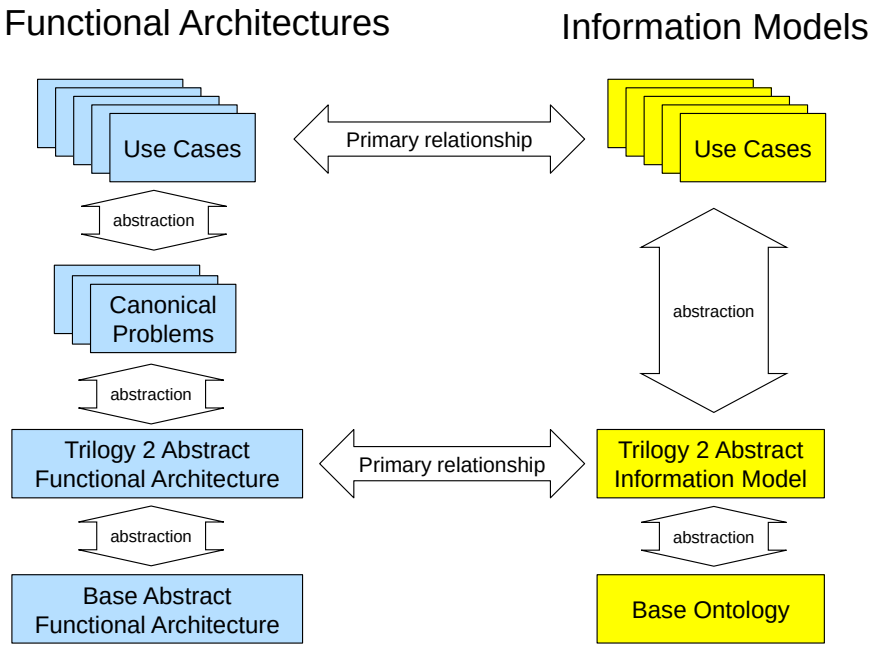


Figure: Levels of Functional Abstraction and Relationship with Information Models

List of Authors

Authors	Andy Reid, John Thomson, Bob Briscoe
Participants	BT, OnApp
Work-package	WP2 : Tussle over Liquidity
Security	CONFIDENTIAL (CO)
Nature	R
Version	v1.0
Total number of pages	72

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background to Trilogy 2

Trilogy 2 builds form the work of the first Trilogy project (Trilogy 1 [2]). A way of summarising the work of Trilogy 1 is that it devised a way of providing a set of parallel, fault tolerant paths through an IP network, and a way to couple their congestion controls that would balance congestion across all links even if up to about 2/3 of traffic was not using the approach. This technique has been adopted and published by the Internet Engineering Task Force (IETF) including the experimental standard called Multi-Path Transmission Control Protocol (MPTCP [15]), and multiple implementations have been deployed. Trilogy 1 also devised a way for network operators to encourage and ultimately enforce limits on the amount of congestion that network customers could cause, to drive the network towards economic optimality for all traffic, whether multipath or not, whether TCP-controlled or not, and whether in long or in short flows. The principals of this technique have been adopted by the IETF in an approach called Congestion Exposure (ConEx [11]), however the specific experimental standards are still emerging.

Trilogy 1 was firmly focused on the network problem and was seeking ways of improving on a) the limitations of the long established Transmission Control Protocol (TCP) that provides for control of resource utilisation and b) the routing protocols that separately provide for fault tolerance.

The concept of Trilogy 2 is to build on this successful work by seeking to extend its applicability such that it is

- cross-resource so that it also includes storage and processing;
- cross-layer so that it can extend beyond the two layer model of Trilogy 1; and
- cross-provider so a full and practical range of commercial interfaces can be accommodated.

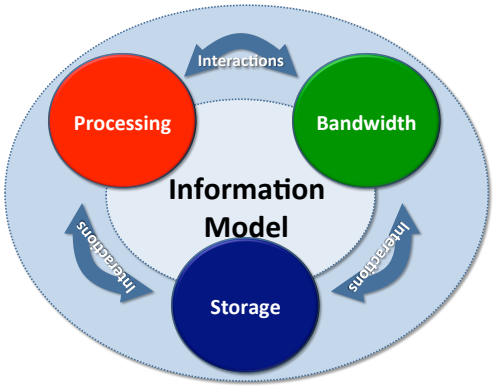


Figure 1.1: Conceptual Outline of Trilogy 2 Architecture

Description of functions and their resources is fundamental to this extension of scope. Therefore the present document motivates what needs to be described, e.g. the units to be used for describing resources, how to

describe the relationship between virtualisation and the underlying pool of hosted resources, and so forth. The centrality of description—the information model—is depicted in Fig 1.1. The companion deliverable, "D2.2 Basic tools for liquidity control" [4] catalogues the requirements for this information model in more detail.

1.2 Aims of the Architecture Task and This Deliverable Within Trilogy 2

The aim of the Architecture Task, Task 2.1, is stated in the project proposal as follows.

> The aim will be to produce an architecture for controlling pooled resources that crosses stakeholders, layers and different resource types. If this turns out to be too ambitious, the scope will be reduced, e.g. to solving each of these dimensions separately.
>
> The architecture glues together in one coherent whole the information sharing mechanisms (task 2.2), the incentives for behaving well (task 2.3) and the tools available for enforcement (task 2.4). Each of these also build on the techniques to create liquidity developed in WP1 and will aim to support the use cases described in WP3.
>
> The architecture must interface both applications and operating systems – it must be wider than just edge-to-edge scope. It must build on how applications use resources today, extending this to pools of resources. It must describe APIs into the information model that will extend and improve existing application practices.
>
> It will show how the incentives can be used in the system as a whole to encourage sharing, and how information can be used as a basis for effective enforcement.

The aim of this deliverable, deliverable 2.1, as stated in the project proposal is as follows.

> The deliverables described the initial version of the Liquid Net architecture, bringing together the tools for creating liquidity and the tools for controlling it. This initial version will be updated with the feedback of the work on the tools and the use cases.

1.3 Additional Relationship of Trilogy 2 with the ETSI NFV ISG

In late 2012, after Trilogy 2 was first proposed, agreed and established as a project, a number of network operators from across the world (led by BT and Telefónica the network operator partners in Trilogy 2) became the founding members of an ETSI Industry Specification Group (ISG) with the aim of defining architecture for Network Functions Virtualisation (NFV), identifying appropriate current standards where they exist, and promoting new standardisation work in appropriate expert standards bodies where no appropriate standard currently exists.[1] The architecture requirements and performance objectives of the ETSI NFV ISG have many parallels with the scope of Trilogy 2.

With the current implementation of network functions by network equipment vendors, many of the silicon

[1] Note that the ETSI NFV ISG does not set out to define standards of its own. The stated aim of the ISG is to produce documents of the style of 'White Papers'. In addition, the ISG aims to publish results from Proof of Concept (PoC) demonstrations relating to NFV.

chips are specific to the network function being implemented as are the plug-in units and normally the chassis as well. This means that the resource assigned to the network function is essentially preassigned by the equipment manufacturer. For example, when a manufacturer supplies a router, the capacity of the router is implicitly defined by the resources of the box that is sold. To the user of the router, this assignment of resources to the desired function, that is the router, is permanent and inflexible. It is not normally possible to add capacity to the router, nor is it possible to reassign the resources of the box to a different function, for example a Web server. These constraints disappear with the NFV approach.

One of the advantages of the NFV approach is that the assignment of resources to desired functionality can be flexible and dynamically optimised. The desired network function is supplied as a piece of software (normally expected to be in the form of a virtual machine image) and this is then activated on standard compute, storage and switching components using the same virtualisation techniques as cloud technology. The definition of the network function and the resources assigned to implement it are separated and a dynamic resource assignment and optimisation process is required to make this dynamic assignment.

This is an example of the architecture envisage by Trilogy 2 and therefore NFV represents a valuable use case for Trilogy 2. As a use case, it has a number of advantages:

- there is a practical overlap in the timing of the three-year Trilogy 2 project and the ETSI NFV ISG, which started in the same month and aims to complete its work within two years;
- the functionality of network functions is familiar to the majority of project partners;
- as the resources used for implementing NFV are the same generic resources used for more general functions, there is no loss of generality in the resource infrastructure architecture;
- there are a set of 'bootstrap' and security issues which arise directly and clearly with network functions and so an NFV compatible architecture must address these issues while other use cases might miss these important issues.

The project had identified IETF and IRTF as important standardisation bodies for the dissemination of the work of Trilogy 2. ETSI NFV ISG has now emerged as an important addition to these bodies.

2 High Level Architecture Overview

Trilogy 2 is about the creation and control of liquidity in the resources required to provide a wide range of ICT applications and services, not just telecoms services. By creating liquidity in the resources, it becomes possible to assign the resources to applications and services in a much more efficient way both in the short term and in the long term.

The example of Network Functions Virtualisation (NFV) illustrates this point. Currently, network functionality is supplied as equipment; boxes which have a predefined set of resources inside the box which implement the function that the box performs. There is no flexibility in the assignment of resources to the network function. From a resource assignment point of view, the box is 'solid'. With NFV, the resources are generic compute (eg server blades), storage (eg storage arrays), and infrastructure network (eg packet/optical carrier Ethernet network) resources and these can be deployed without reference to a precise network function, for example, BRAS, CDN, DPI, GGSN, Firewall, EPC, etc that the resources will be assigned to over the life of the resources. In this case, the resources are 'liquid'.

2.1 General Scope of Trilogy 2 Architecture

The general scope of Trilogy 2 is shown in Figure 2.1.

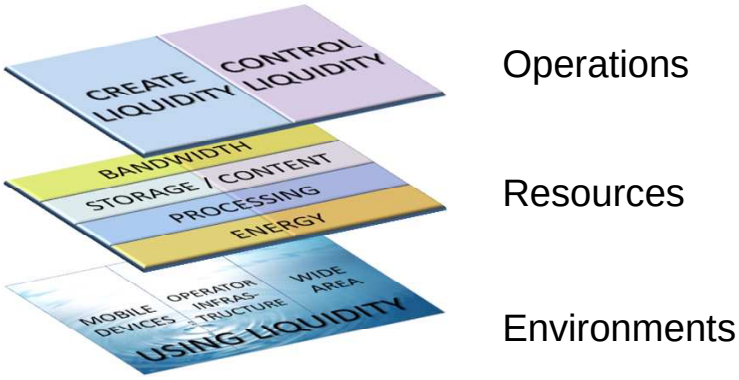


Figure 2.1: Scope and Conceptual Framework for Trilogy 2

From the top level perspective, Trilogy 2 is about the creation and control of liquidity. The creation of liquidity requires the creation of an infrastructural pool of generic resources which can be flexibly used by individual applications and services. Along with this creation of liquidity is the control of the liquidity which ensures that use of the pool of resources is economically efficient, which requires the optimisation to rapidly adapt and re-optimise over time.

There are three primary forms of resource which are relevant to Trilogy 2. These are bandwidth, storage, and processing. These are the basic raw materials of any information system. However, we also note that these resources themselves require powering and increasingly they are very hungry consumers of electricity and hence greenhouse gases. As a result we also track , the energy required to maintain the bandwidth, storage, and processing resources within the scope of Trilogy 2.

Finally, Trilogy 2 is applicable to a number of different environments ranging from the end user devices to major carrier networks and data centres and includes both fixed and mobile connectivity. Indeed, mobile devices and datacentres are two focal points of technological development in the wider industry. These environments provide the places where the liquidity of resources can be exploited.

2.2 Extension from Trilogy 1

Trilogy 1 addressed the creation and control of bandwidth liquidity and devised a number of mechanisms for creating and controlling it. These mechanisms were architected in three categories; first resource capacity measurement and exposure of the available capacity; second, calculation and control of multiple alternative routing options through the network; and third, the control of traffic load onto multiple alternative paths. Figure 2.2 is reproduced from the final communication report of Trilogy 1. Trilogy 1 resulted in a number of new mechanisms and standards including the MultiPath TCP (MPTCP) protocol and Congestion Exposure (ConEx). This of course was conceived within a networking context and therefore is naturally described within the general language of networking.

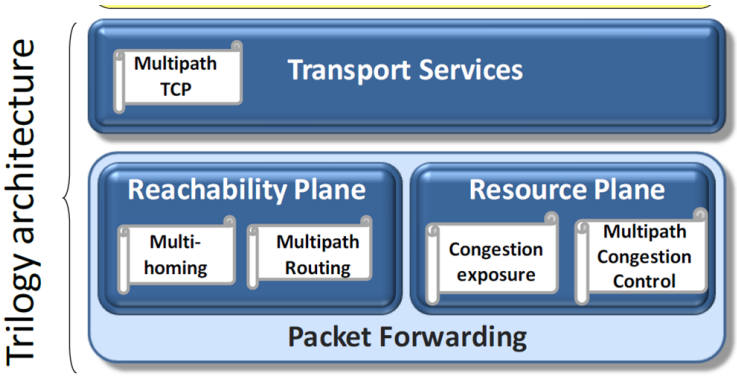


Figure 2.2: Architecture from Trilogy 1

This figure already shows some of the framework needed for generalisation. On the top are the services, in this case the Multipath TCP services while below are the resources needed to support the services. There are therefore a number of points from Trilogy 1 which were specific to network but the power of the liquidity concept can be greatly enhanced if these are generalised. These are:

- network bandwidth is the only resource which is considered;
- there is an implicit fixed two layer model of a lower network layer providing resource and an upper layer demanding this resource;
- coupled to the first point, any commercial boundaries within the architecture are limited to relationships between Internet Service Providers (ISPs).

The objective of Trilogy 2 is to extend beyond these specific restrictions to be *cross-resource, cross-layer, and cross- provider*.

Figure 2.3 is an alternative illustration of the Trilogy 1 mapping of the transport services to the resources and emphases that:

- the resources are fixed in relation to the services they support;
- the services are realised by a) a routing algorithm selecting equipment for the service to traverse and b) a resource assignment algorithm to assign the resource of each equipment to the services using the equipment, for example MPTCP.

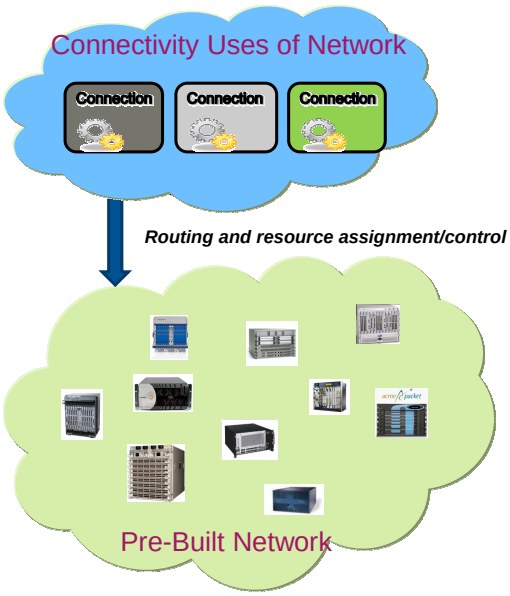


Figure 2.3: Representation of Trilogy 1 Architecture

In considering especially the *cross-resource* generalisation, it is possible to generalise to include both compute and storage resources with the pre-built network as part of a common infrastructure and to broaden the scope of the services to include general ICT applications and services. This generalisation is shown in Figure 2.4 below. This figure, at least in general form, has been used extensively for presenting NFV concepts. This illustrates how the Trilogy 2 architecture is both a simple extension and generalisation of the Trilogy 1 architecture and also illustrates how Trilogy 2 anticipated and is directly applicable to the needs of NFV.

2.3 Basic Features of the Trilogy 2 Architecture

There are a number of features of the generalisation which present considerable challenges compared to Trilogy 1.

- In Trilogy 1, the resource of the network is bandwidth which is essentially homogeneous across all forms of technology and is characterised by a minimal set of parameters (eg bit/s, latency, packet loss probability). The storage resource is potentially similarly homogeneous however, this is not the case with processing resource. With bandwidth and storage, the network and storage systems are transparent to what is being transported or stored and the user of bandwidth and storage is able to select their data

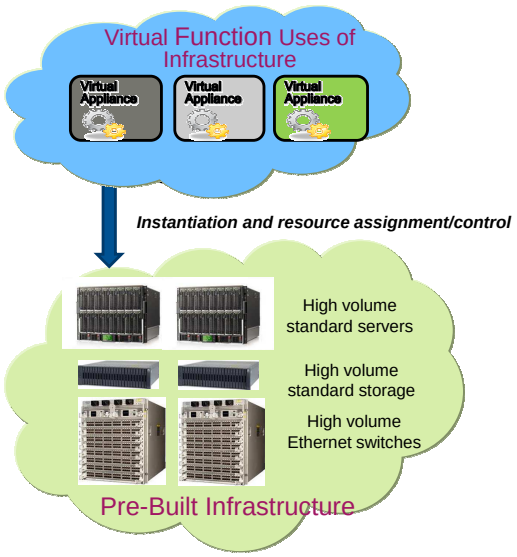


Figure 2.4: Representation of Cross-Resource Generalisation of the Trilogy 2 Architecture

completely independently of the network and storage systems. This is not the case with processing resources. Every processing resource has some form of atomic instruction set or logic elements. These atomic instructions and/or logic elements are normally specific to the design of the specific processing resource meaning that there is normally a strong dependence between service and the processing resource. A simple illustration in a CPU instruction set. The binary code which implements a required service will vary depending on the instruction set of a CPU.

- The lack of homogeneity of processing resource means that there is not a readily available measure of a processing resource's capacity in the same way that the 'bit' measures the capacity of a bandwidth and storage.

- Even in the real world of networks they are multi-layered, and this reality is more complex than "everything over IP and IP over everything" architecture. Flexible packet/optical transport networks now generally exist below the IP layer and "middleboxes" route on layers above the IP layer. When processing and storage is added, even with relatively simple examples such as content distribution networks (CDNs), the architecture is unavoidably multi-layered.

- The commercial models when processing and storage are added are also considerably more complex compared to the inter-ISP model. Both the range of functionality, and hence protocols, that commercial service providers need to interconnect necessarily increase with Trilogy 2, and the range of service providers extends to include cloud service providers and probably others as well.

Having said this, there is a strong correlation in the way in which resource assignment takes place in Trilogy 1. We should therefore expect that the algorithms and even the protocols of resource assignment should be adaptable to the generalisation.

The first task for the Trilogy 2 architecture is to identify the more general entities which are appropriate to Trilogy 2 and which, when specialised, correspond to the entities of Trilogy 1. This cannot be a 'mechanical' task as there are always many possible generalisations of any particular entity. We are seeking the generalisation which maintains the overall behaviour characteristics of liquidity.

As we describe in Chapter 5 below, a general and powerful architecture emerges from this generalisation which gives a framework for unifying a number of currently largely disparate disciplines. For now, we present a very simple summary of this architecture.

The first observation is that a general service in the context of Trilogy 2 will itself be a defined function. The service has input interfaces, output interfaces, and can carry out operations. This is the definition of a function. Second we observe that resources are always presented as belonging to host functions. We then note that functions have two basic recursive properties.

- A function can be recursively composed by assembling together consistent component functions with the composition being defined by a graph that defines how the interfaces of the component functions are joined to each other.
- Host functions can be recursively configured and once they are appropriately configured, they implement a virtual function. There is a functional equivalence between the suitably configured host function and the specification of the virtual function.

The basic entities of the generalisation are listed in Table 2.1.

Table 2.1: Generalisation of Concepts from Trilogy 1 to Trilogy 2

	TRILOGY 1	TRILOGY 2
SERVICE	transport connection	virtual function
RESOURCE	network capacity	host function
SERVICE COMPOSITION	*routing* through IP network	*graph* of component virtual functions *mapping* of virtual functions to host functions
CAPACITY ALLOCATION	MPTCP and TCP control and congestion policing	*allocation* of host function capacity to virtual functions according to mapping
CAPACITY PARAMETERS	*congestion-bit/s* latency	*congestion-bit/s* for bandwidth to be developed for storage to be developed for processing

Figure 2.5 below gives a diagrammatic summary of the generalisation of Trilogy 1 into Trilogy 2.

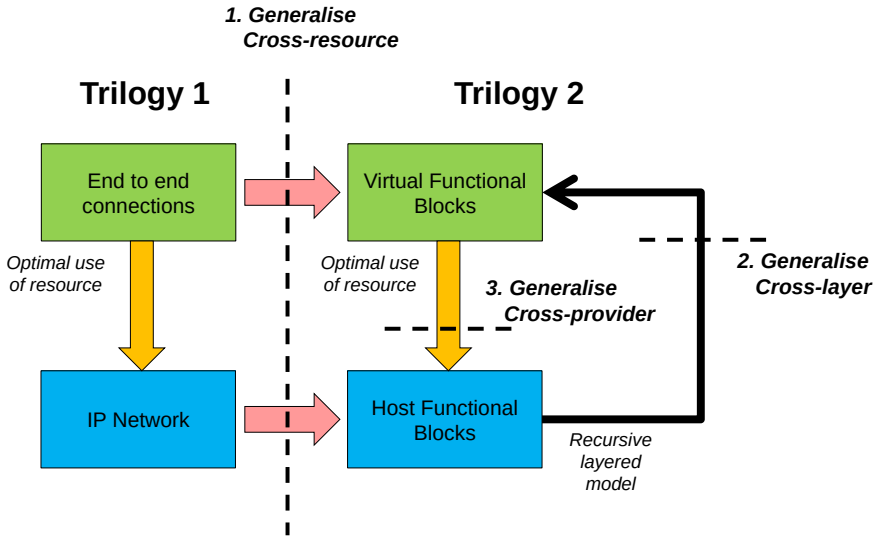


Figure 2.5: Generalisation to Virtual Functional Block and Host Functional Blocks

3 Use Cases

The project is considering a variety of use cases [7] covering and illustrating the scope and applicability of Trilogy 2. Some are immediately recognisable as application-level use-cases, while others develop lower layer infrastructure for highly generic use (e.g. MPTCP), but they can still be considered use-cases even though their domain of applicability is broad. To Control Liquidity (WP2), the primary focus of these use cases at this stage is within the Information Modelling Task (Task 2.2) and the requirements from each use case are set out in detail in Deliverable D2.2 [4].

From the perspective of the architecture task, the use cases are useful in setting the scope of the architecture and useful for validating the architecture. They also aid the interaction between the architecture task and the rest of the project by:

- providing a framework for capturing requirements, notably the information and behaviour requirements of the systems being developed within the project (WP1);
- providing a superset of use cases from which a smaller set of use cases can be selected for end to end development and demonstration of the Trilogy 2 concepts (WP3);
- providing a set of examples against which the overall results of Trilogy 2 may be evaluated, including examples beyond systems developed within the project;
- providing a set of examples some of which exemplify the architectural problem which the project is addressing.

This last purpose is taken up in the Problem Statement below (Chapter 4) where a selection of canonical architectural problems have been distilled from the use cases to illustrate the practical issues that the architecture needs to resolve. The use cases set out in D1.1 and D2.2 are reasonably detailed and oriented to the needs of specific developments within the project. This is their first and most important role. However, for the purposes of clearly setting out the scope and requirements of the architecture, it has been useful to distil all the use cases to inform and form the basis for three canonical problems for developing this stage of the architecture.

The initial canonical architecture problems are:

(i) IaaS (Cloud Infrastructure as a Service) and its tradable resource characteristics;

(ii) middleboxes and the way they interact with layering;

(iii) lightweight virtual machines.

Each of these canonical problems is discussed and analysed in Chapter 4 in order to assemble a set of observations which are then summarised as a set of requirements on the architecture. Table 3.1 shows that all use-cases exhibit at least two of these canonical problems.

Use-case	Canonical Problem		
	IaaS Resource Ctrl	Middleboxes	Lightweight Virtualisation
MPTCP		✓	
PVTCP	✓	✓	✓
Trevi	✓		✓
ConEx	✓	✓	
Irminsule	✓	✓	✓
vBRAS	✓	✓	✓
vCPE	✓	✓	✓
vM3	✓	✓	✓
Mobile Kibbutz	✓	✓	✓

Table 3.1: Mapping of Trilogy 2 Use-Cases to Canonical Architectural Problems

To keep the present document self-contained, a one-paragraph summary of each use-cases is also given below.

3.1 MPTCP Outline

Multipath TCP (MPTCP [15]) controls packet transport over multiple paths between two host processes. It improves resilience, performance and efficiency in the use of network capacity. It is applicable where at least one host has multiple network interfaces, and there is sufficient data to be transferred to warrant the initial connection setting up sub-flows. Typically the data to be transferred may be split over multiple paths and the congestion controls of all the paths are coupled in order to balance network load. However, but other path-scheduling functions can be implemented, such as duplication, hot-stand-by or lowest latency. The IETF has agreed experimental standards for MPTCP and multiple implementations now exist.

3.2 PVTCP Outline

Polyversal TCP (PVTCP [7, §2.2]) aims to control data transport for inter-process communication (IPC) by abstracting the underlying means of communication and the end-point addressing scheme from the application. The underlying transport may be sockets, shared memory channels, pipes, etc. and the end-point processes may reside within the same core, on separate cores, separate processors, separate machines or separate sites. It is being investigated whether PVTCP would benefit from using multiple paths in a similar way to MPTCP. The design of PVTCP is still under development.

3.3 Trevi Outline

Trevi [20] is a transport protocol specialised for controlling data transfer to and from multiple storage replicas. The read and write modes of Trevi use considerably different protocols, but both rely on fountain coding for reliable delivery. In write-mode, Trevi multicasts data blobs to the replicas using fountain codes, and each receiver uses a TCP channel to control the sending window for flow control and network congestion control. In read-mode, the transport is multi-sourced, using multiple unreliable channels to read randomised symbols

from each storage replica, each managed by a reliable control channel. The client continues to request parts of the blob (symbols) from each storage replica only until it has received enough data to reconstruct the whole blob. Thus the client enjoys the resilience of multiple replicas while only sending or receiving any one part of the data once, and always balancing demand in parallel across all the replicas while respecting the available rates of the different storage devices and the relevant parts of the I/O system and network. A detailed straw-man design of Trevi has been published and it is planned to be implemented.

3.4 ConEx Outline

Congestion Exposure (ConEx [11]) is a protocol designed to enable providers of communications infrastructure to give the correct economic incentives to data senders to manage their traffic load and to enforce limits if they do not. ConEx is an extension to IP that allows the sender to signal the amount of congestion it expects its traffic to experience in-band. These signals are then visible to traffic management nodes at the ingress to each network the traffic traverses. The integrity of the congestion information can be tested, and if the sender understates congestion signals relative to actual congestion experienced, an audit function can discard sufficient traffic to negate any benefit a sender would accrue from lying. ConEx is slowly being standardised experimentally by the IETF. The standards have not stabilised sufficiently for implementation, but prototype implementations exist of a similar protocol called re-ECN.

3.5 Irminsule Outline

Irminsule [7, §4.1] is a large-scale distributed database co-ordination scheme. Each distributed processor can use a part of a data structure by optimistically replicating it for local low latency access as a branch operation in the distributed version control system, git. The ACID (atomicity, consistency, isolation, durability) properties can hold within each branch, but there is loose consistency with the rest of the system. Conflicts when branches are merged have to be resolved by programmers exploiting conflict-free replicated data types, or data types with custom merge operators. Stored data is always immutable so there is never a need to lock data replicas while another replica is altered. This allows storage to be arbitrarily divided up between processes (e.g. by a load balancer), making it a liquid resource where it previously had to be treated as constrained and solid because of all the possibilities of different parts needing to be locked. Irminsule is in the process of implementation and refinement.

3.6 vM3 Outline

Virtual machine migration for mobile devices (vM3 [7, §3.3]) is the ability of move virtual machines between mobile devices or between a mobile device and a desktop. As virtual machine capability starts to emerge on mobile devices, the user interface is a large part of the capability of the mobile device and therefore a large part of any virtual machines on the mobile device. First this use case splits applications between front end (GUI) processes and back-end processes and so allow mobility for the resources used for the back end processes, thus creating a liquidity. However, this use case also addresses the challenge of moving the GUI and linkage

to virtualised devices (network, camera, phone, etc) between heterogenous hardware environments and well as achieve a minimal perceived delay between the VM being active on one device and then being active on the second device.

3.7 vBRAS Outline

The virtualised broadband remote access server (vBRAS [7, §3.2]) is developing a new generation of BRAS based on a full separation of the logic of the BRAS for the resources used to implement the BRAS thus creating a liquidity. As the BRAS is the primary security gateway to the network for all broadband services, content caches, for example storing video content, must architecturally sit on the network side of a BRAS and therefore the advantages of caches can only be fully achieved if BRAS can be highly distributed in the network. This use case is building highly efficient, small scale and highly parallelisable virtualised BRASs running in very low overhead virtual machines, thus creating liquidity not just for the BRAS but also for many other networked applications including content caching. A major part of this use case in the development of the lightweight virtual machines with a lightweight host operating system within which the vBRAS is hosted. The vBRAS is therefore itself a use case of these lightweight virtual machines and host operating system which in the project is called 'ClickOS'.

3.8 vCPE Outline

The virtualised customer premises equipment (vCPE [7, §3.3]) is creating liquidity in the way functionality is provided which currently requires specialised network equipment at customer premises. This specialised functionality is more efficiently provided in a datacentre where the allocation of resources to the implementation of these functions aggregated across many customers and therefore more efficiently optimised. This use case provides not just a solution to the virtualisation of these CPE functions but also a solution to the extension of the customers network for the customer's premises to the datacentre which does not change the logic of the customer's overall network.

3.9 Mobile Kibbutz Outline

The Mobile Kibbutz [7, §2.1.3.2] is a way for a localised group of mobile devices to take turns in using each other's cellular connections, because periodic use of a radio consumes disproportionate battery energy. The aim is to ensure fairness, accountability and privacy of participants. Participants use WiFi or BlueTooth for connectivity with each other, and they maintain a multipath TCP connection over all these links to exploit whichever one(s) open a path over the cellular medium that can reach the remote party. The scheme has been designed, implemented and evaluated.

4 Problem Statement

In order to develop the architecture, we focus on particular use cases which illustrate some of the major issues that the architecture needs to address. The purpose of this Chapter is to give a clear understanding of some of the more important issues from a practical point of view.

4.1 Canonical Problems

4.1.1 IaaS Resource Control

"as a Service" has rapidly become the 'buzz-phase' of public cloud computing. Within this, a number of things have been identified as being "as a Service":

- Infrastructure as a Service (IaaS) - essentially virtual machines within the cloud platform within which the user can load any operating system and applications the choose;
- Platform as a Service (PaaS) - a virtual machine with a specific operating system and possibly a specific platform application, for example a web server or database, which can be configured by the used for their own requirements;
- Software as a Service - end user software applications which run on cloud platform, normally accessed from a user's web browser, and which gives the appearance of being a local application.

In each case, the services includes the necessary connectivity for the end user to access the infrastructure, platform, or service. It is the first of these, IaaS, that is of particular interest. Here the liquid asset which is traded is the virtual machine.

We start by considering the basic model of liquidity 'trading' which is implicit in Trilogy 1 where the liquid asset is bandwidth. The network supplies bandwidth and connections demand bandwidth. The connections have an implicit price they are willing to pay for bandwidth while the different components of the network have different costs for their bandwidth (in fact, primarily based on congestion pricing rather than equipment cost). The liquidity pool matches the demand and supply and the MPTCP protocol is an efficient automated and distribute solution to matching this demand and supply. This is illustrated in Figure 4.1 below.

Critical to the operation of this bandwidth liquidity market is the uniformity of bandwidth: 1Mbit/s of bandwidth on one path is equivalent and tradable with 1Mbit/s of capacity on another path in the network. Bandwidth is homogeneous and essentially tradable as a commodity and all connections will treat all bandwidth equally. In economic terms, we can treat the liquid pool of bandwidth as a commodity market.

Even bandwidth is not quite as simple as this. Transmission errors and especially latency can differentiate the utility of bandwidth on different paths. However, these two extra parameters, errors and latency, do have some level of implicit trade off with bandwidth:

- errors can always be traded with bandwidth as error correction, for example as carried out by TCP, will trade errors directly into additional bandwidth and latency;

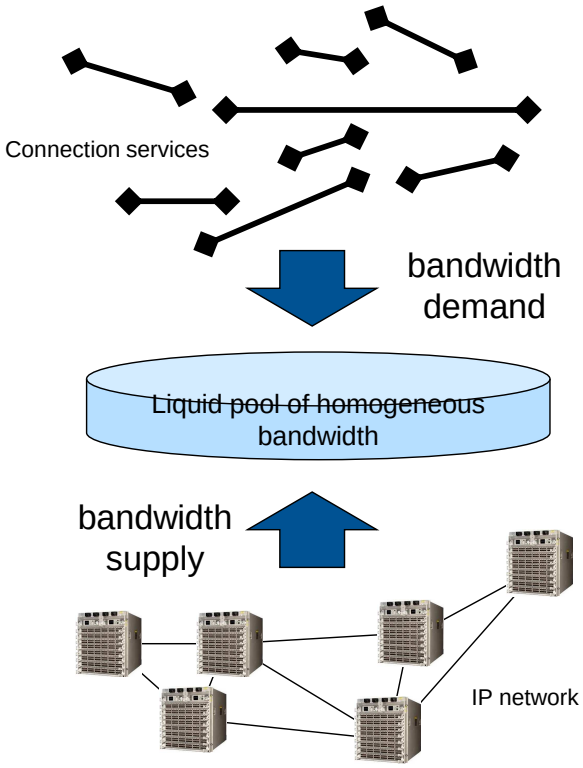


Figure 4.1: Liquidity with Homogeneous Bandwidth

- while latency cannot be fundamentally traded with bandwidth (although many buffer control algorithms do create a trade off), latency cannot be less than the separation distance divided by the speed of light which is often significant, and this can only be traded by the physical resiting of end points and, in addition, this finite speed of light latency can effectively dilute any buffer latency.

While noting these, it is also the case that in the great majority of cases, bandwidth is regarded by both connections and network as a homogeneous, uniform commodity.

This is not the case with virtual machines (VMs). A VM is a combination of several different resources which fundamentally do not trade with each other. These include a virtual CPU (vCPU), virtual RAM (vRAM), and one of more virtual disks (vDisk) [1]. A virtual machine is always a combination of these resources.

On the demand side, different applications will place very different emphasis on the different resources. Some applications may be CPU intensive, some may be RAM intensive, some may require very large disk storage but not overly concerned by the access time to the disk, while others may be especially concerned with the disk access time. And of course, there is a wide mixture between these. As a result there is no uniform VM profile. On the supply side, different suppliers of IaaS construct their own range of VM package options - a relatively small selection which cover broad classes of applications.

We see that the liquidity pool of VMs is not a homogeneous, uniform commodity. It is a set of differentiated packages. It is possible to trade off between packages, but there is discontinuity which means that some applications will have a certain stickiness to a certain package or small subset of packages. In economics

[1] There are other resources such as virtual NIC (vNIC), virtual DVD drive, virtual USB drive, and a virtual console but these are not so relevant to the current discussion, but do go to further enhance the need to treat the liquidity assets as heterogeneous

terms, this is not a commodity market but a differentiated market. This differentiated market for IaaS is illustrated in Figure 2.5 below.

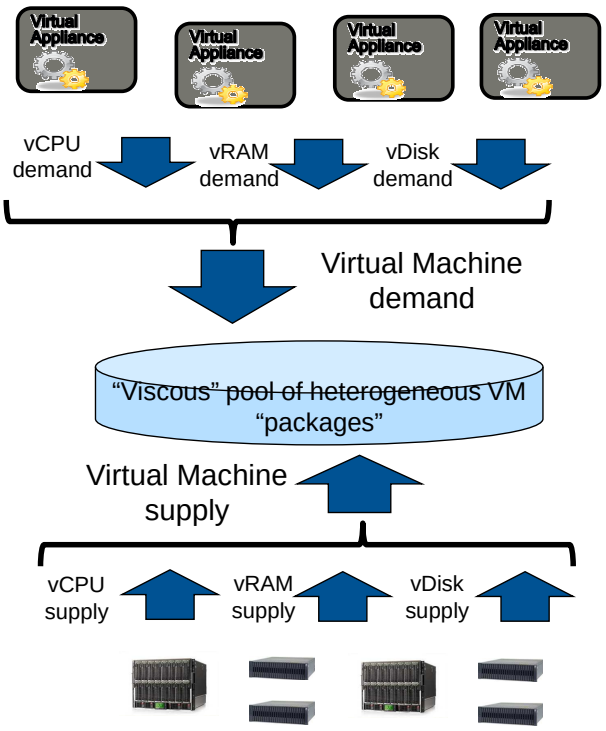


Figure 4.2: Liquidity with Heterogeneous Virtual Machines

Generally, assets in differentiated markets are not regarded as 'liquid' as those in a commodity market, indeed, we might describe them as 'viscous'.

Finally two further issues arise from this IaaS use case:

- why don't the IaaS suppliers make it possible be purchase increments of vCPU, vRAM, and vDisk separately and create individual commodity markets for each separately?
- vCPUs are only tradeable if they have the same instruction set [2], but this is not always the case.

The reason why IaaS suppliers create a limited number of packages is because its cheaper and more profitable for them. It is frequently the case that there are significant common costs which can be recovered against any or all of the components of the package, and there is normally no stable economic solution to determining how these should be split. In addition, there are often significant costs in managing complex variants so that one uniform package which roughly covers a range of applications is more profitable for the supplier than creating a bespoke package for each application. Moreover, the users do not necessarily want bespoke packages as these may increase the differentiation between their competitive alternative packages and increase their 'stickiness' to their bespoke package.

The fact that vCPUs are only tradeable if they have the same instruction set illustrates and introduces another fundamental difference between the general case and the specific case of bandwidth. Bandwidth and storage

[2] or there is some minimal cost means of translating (like JIT compilers within a Java runtime environment (JRE) if we take the JRE to be a form of vCPU)

are both transparent functions; transparent in that for the transport and storage functions themselves, there is no meaning (that is a behaviour dependency) to the values being transported or stored. This means that it is possible to measure bandwidth and storage in bits with complete generality. This is not the case for general functions as is evident from the vCPU example. In the general case of processing resources, or any more general resource where any level of behaviour processing is exposed outside the resource, then the way inputs affect the behaviour of the resource is fundamental to the tradability of the resource. Dealing with this new and profound complexity is a fundamental problem for this architecture.

4.1.2 Consequences of Middleboxes

Contrary to the original conception and architecture [3] of the Internet, 'middleboxes' are a widespread phenomenon throughout the networks of Internet service providers (ISPs). It is no longer the case that they can be dismissed as a "temporary aberration" or an "architectural violation to be opposed and resisted" and boxes performing functions such as NAT, Firewall, Proxy server, load balancers, and other deep packet inspection (DPI) are now effectively ubiquitous.

In general, these middleboxes are stateful and therefore quite different from IP routers and Ethernet switches which maintain no session/flow state. The need to maintain state means that efficient implementations increasingly rely on standard processing (such a x86 architecture) and now that virtualisation technology is maturing this means that full machine virtualisation need not create a packet forwarding bottleneck, middleboxes are emerging as virtual appliances.

In addition, there are a range of more formal gateway boxes which share many characteristics of middleboxes including BRAS in fixed networks and GGSN, SBC/G, etc in mobile networks. These also touch packets at a session level (even if the session may be an underlying session rather than an application session) and again these functions are starting to emerge as virtual appliances.

First, the very acknowledgement of middleboxes requires a multi-layer architecture. Second, if these middleboxes are virtual appliances, then there is considerably more flexibility in where they can be sited, and the siting of the these middlebox functions becomes a parameter for optimisation by the network. In addition, there is considerable flexibility in the choice of layers that the middlebox chooses to touch and this too can become a parameter for optimisation by the network.

However, in some cases middleboxes are a 'cat and mouse' game. The middlebox is trying to limit the behaviour of the end user and so the end user has an incentive to avoid the behaviour of the middlebox. For example a proxy server may wish to limit the range of applications that an end user can access, or at least limit the bandwidth given to those applications. This gives the end user an incentive to tunnel their application and masquerade their application as one that is allowed, for example tunneling the affected application through http. This tunneling may have a processing and performance cost, but this may still give the end user a better end result. A particular example protocol stack is shown in Figure 4.3 below.

[3] noting that the IETF has always preferred to work with a set of guiding principles rather than an explicit architecture, in this case the guiding principle being the end to end principle

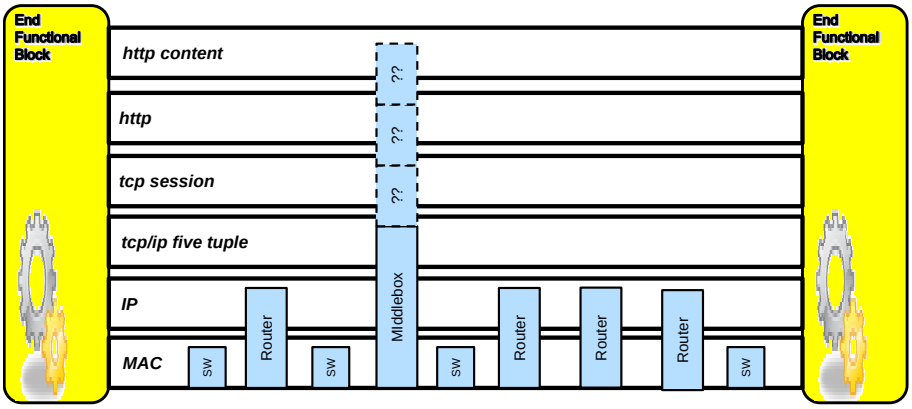


Figure 4.3: Liquidity with Virtual Middleboxes

This example use case illustrates three quite separate requirements:

- the need to include multiple layers and multiple siting options for functionality in the liquidity optimisation problem;
- multiple middleboxes frequently need to be deployed in parallel in order to cope with the load placed on them and therefore there must be some mechanism by which the load can be distributed amongst the instances, however, frequently the box deployed to do this, a load balancer, is itself a middlebox;
- end user can and do optimise their transport taking account of middleboxes.

The inclusion of multiple protocol/functional layers has the effect of adding a third dimension to the two dimensional optimisation on MPTCP which is itself an extension from the one dimensional optimisation of TCP. While MPTCP can spread traffic across the two dimensional plane of the IP network, the reality of middleboxes means that we cannot just consider the network to be a flat plane of IP.
The last requirement, to account for end user's multi-layer optimisation, directly gives rise to the polyversal TCP (PVTCP) development in the project which extend MPTCP to include tunneling at multiple different layers as additional transport paths.
In addition, in the case of MPTCP, the optimisation problem is quite clearly the optimal placement of traffic within the given and fixed topology of the IP network. MPTCP cannot change the topology of the network. However virtual middleboxes do two things to change this:

- the ability choose a protocol layer implicitly creates options to bypass nodes or, more likely, force routing through a node which substantially alters the way traffic flows on the network compared to a flat IP network
- much more significantly, virtualisation enables dynamic placement of middleboxes and a network operator can modify the effective topology at any one layer, for example, under light load, middleboxes can be centralised but under heavy load, more instances of middleboxes can be dynamically instantiated

near the end user in a much more distributed topology[4].

A further feature of multi-layer topology optimisation is the sensitivity and instability that can arise. Small changes in the 'price' of using a lower layer can sometimes cause dramatic changes in the optimal topology at a higher layer. Often, for a higher layer, there can be a tipping point of lower layer link costs which cause a radical change from a centralised network to a highly distributed network.

The need to distribute load across parallel instances of middlebox functions is really an extension of the dynamic topology requirement. However, historically it has not always been treated as such. As a result the fundamental and unavoidable tie between network addressing and network routing has frequently been missed and the so the routing needed to do the distribution of load has been forced up the protocol stack to a point where addressing can be sufficiently distinguished in order to identify the required end point for a particular packet, hence the need to introduce middleboxes.

This problem of load balancing making middleboxes become necessary is demonstrated by a simple example. Supposing a Web site for a domain name experiences load which requires more than one server, but the DNS entry for the domain name returns one IP address. This means that all the servers meeting the load must appear to be one IP address. But this violates the flat IP network as each server should have its own IP address. Therefore the load balancing cannot be carried out by a basic IP router. In this case, a load balancer can work using the five tuple of the destination IP address, the source IP address, the destination port number, the source port number, and the protocol. However, supposing the web site owner wishes to dedicate some server to delivery of large files which are all on a particular directory branch within the Web site URL, eg http://domain.name/largefiles, then the load-balancer must now parse into the http layer to find this directory branch in order to successfully route the packet. In this case, the packet forwarding is based on probing deep into the application layer.

4.1.3 Lightweight Virtualisation

When virtualisation was introduced into data centres through the introduction of hypervisors and virtual machines, the primary motivation was simply to reduce the number of servers required to handle a datacentre's workload. As the processing power of a basic server increased, it regularly became the case that some applications designed to run on a dedicated server were not consuming the full resources of the server. Virtualisation allows this spare server resource to be used by other applications and as a result many data centres saw a great improvement in their server resource utilisation efficiency.

With this objective, there was never any direct motivation to reconsider the scale and design of the applications. Indeed, the critical winning feature of virtual machines, is that there is no need to change the application software at all. To the software, the virtual machine behaves just like a physical server; it is prefect evolutionary innovation.

The motivation of the design of hypervisors was to create virtual machines that look and behave as close to

[4]While not generally considered a middlebox, CDN is another example of a dynamic topology under differing load.

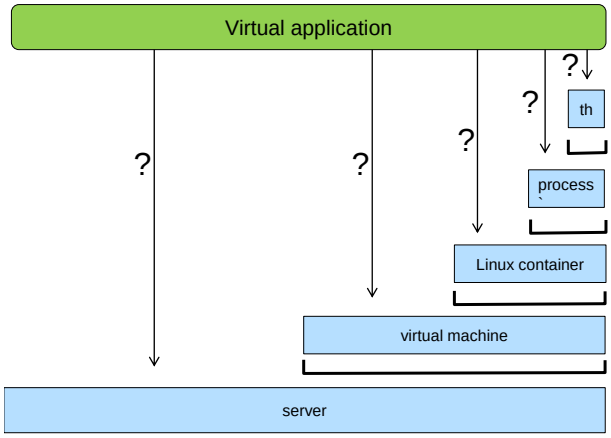


Figure 4.4: Options available to implementing a virtual application

physical servers as possible and this has led to hypervisors and virtual machines with the following characteristics:

- virtual machines are comparatively large and 'heavy weight' with a significant overhead needed to support each virtual machine;
- the number of virtual machines supported on any one server is comparatively small (units to tens);
- the boot time for virtual machines is similar to that of hardware servers.

However, virtual machines create a new dynamic flexibility, that is a liquidity, way beyond that of physical servers. It is now possible to reconsider the question of what the optimal size and number of virtual machines per server should be, and closely coupled to this, what scale of application should map to virtual machines. The current hierarchy of execution environments (that is server, virtual machine, possibly a Linux container, process, and thread) give different levels of performance and security isolation but also currently come with different levels of performance and resource overheads.

However, it is apparent that some of the performance and resource overheads associated with virtual machines are not fundamental and can be greatly reduced. The current level of overhead is as much a result of the history background to virtual machines as just described. Assuming that the performance and resource overhead of the virtual machine can be reduced, this opens an opportunity to find a different optimal scale of virtual machine and mapping of application to virtual machine. This is illustrated in Figure 4.4 below. Both the MirageOS and the ClickOS use cases with their associated software developments address this area within the project.

4.2 Requirement for a layers of Abstraction in the Functional Architecture

Based on the canonical problems, the following is a list of observations which emerge.

(i) The resources of cloud liquidity are packaged and differentiated and this creates a certain increase in trading 'viscosity' which increases the complexity in managing the liquidity when compared to bandwidth only liquidity.

(ii) General resources which include some level of processing are not transparent to their inputs in the way bandwidth and storage are transparent. For general resources there is a meaning and behaviour associated with each value of input and resources are only tradable if they have then same meaning and behaviour for the same input. For example the tradabiity of CPU processing resource depends on at least the different CPUs having the same instruction set or having some other common execution environment. By contrast, for bandwidth and storage, a bit is always a bit.

(iii) The siting of functionality at different layers is now open to optimisation in a third dimension of protocol/functional layers (compared to the two dimensional network of MPTCP, which itself added a second dimension to the one dimensional model of TCP).

(iv) Interactions between optimisation at different layers may lead to great sensitivity/instability to optimisation, especially at higher layers. Optimal results at higher layers may flip from highly centralised to highly distributed as a result of small changes at lower layers.

(v) Load balancing across multiple parallel instances of a function should not require arbitrarily deep packet inspection in order to identify which of the parallel instances to forward a packet.

(vi) If users move around such that the network characteristics change during the course of a session, or if middleboxes interfere with their applications, the user can choose to tunnel their application at different layer in order to optimise their transport.

(vii) Resource optimisation needs to find an optimal mapping of application to virtual machine and from virtual machine to server, and by implication find a new optimal size of virtual machine unrestricted by an assumption that an application will be unchanged from that running on dedicated servers.

This list is not exhaustive, nor may it be possible to fully address all the relevant issues which emerge from the canonical problems within the scope and timescale of the project. This list gives a broad target of requirements and details the more precise areas where the project can move beyond the current state of the art.

When seeking to address these canonical problems, it is apparent that there is no general framework within which solutions can be developed. In particular, the generic techniques of functional modelling and functional specification does not have the means of describing the essential properties of liquidity. This means that there is no readily available appropriate technical 'language' to even describe these observations arising from the canonical problems. This observation was also apparent in the work of the ETSI ISG on Network Functions Virtualisation (ETSI NFV ISG).

The first requirement of the architecture is therefore to develop this basic functional architecture which can accurately describe and specify functionality, including processing and storage, in such a way that functional uses can be decoupled from the resources that implement these uses. This basic functional architecture can then give a technical framework and language within which the canonical problems can be addressed.
The overall Trilogy 2 architecture is therefore layered in different levels of functional abstraction. These level of functional abstraction also mirror abstraction in the Trilogy 2 information model.

- At the bottom is the basic functional architecture which is gives the technical framework within which the fundamental process of creating liquidity can be described.
- Above this and described in its terms, is the Trilogy 2 abstract functional architecture which is common to all the use cases and which can be used to develop solutions to the canonical problems. This functional architecture is also the basis of a Trilogy 2 abstract information model.
- Above and described in its terms, are the architectural solutions to the canonical problems and which also act a verification of the Trilogy 2 functional architecture.
- At the top are the functional architectures of each of the uses cases which are also each the basis of the specific information models for each use case.

These layers and their relationship to the information modelling is shown in Figure 4.5 below.

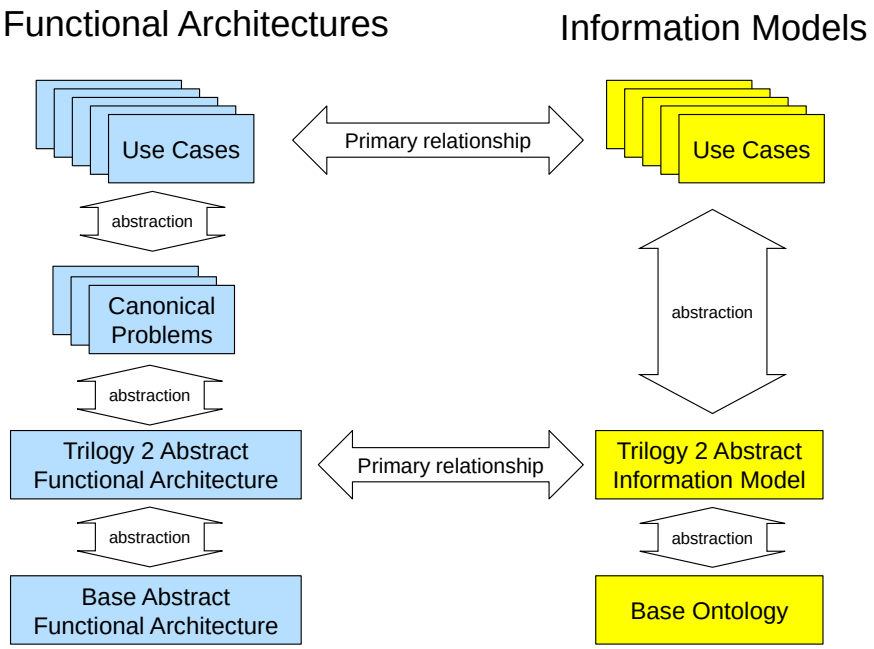


Figure 4.5: Levels of Functional Abstraction and Relationship with Information Models

5 Architecture

The Chapter defines the basic functional architecture and the Trilogy 2 abstract functional architecture and addresses, at least to some significant degree, each of the requirements set out in Chapter 4. This deliverable presents the current status of this work which is still on-going. Broadly, this current chapter presents the basic framework and the future work will be concerned with using this framework for the more specific aspects of the canonical problems and each of the use cases. It is therefore expected that the architecture presented here will not change greatly; future work will apply and add to it, rather than modify it.

The first section in this chapter describes the basic functional architecture and deals with the fundamentals which arise when uses are decoupled from the resources which implement the uses - the essence of creating liquidity. This first section can be regarded as more mature. The second section takes these fundamentals and applies them to the context of Trilogy 2 and develops a generic functional architecture for Trilogy 2. This second section is an initial view and will be reviewed and updated as required as work on the functional architecture of the individual use cases is developed in the next phase of the project.

5.1 Basic Functional Architecture

By way of introduction, we note that within most engineering disciplines, the language and general framework within which problems are expressed and documented together with their design solutions is normally taken for granted. However, Trilogy 2 is concerned with a strong convergence between the world of computing, notably cloud computing, and the world of networks. While 'convergence' has been on going for many years, there has been a continuing difficulty in this converging relationship in that the descriptive techniques used within networks and those use within at least some parts of computing do not align and people on one side can have difficulty in understanding the intention of people on the other side. For the purposes of Trilogy 2, there is not a readily available framework and all the standard toolkits do not have critical descriptive features. The basic aim of this architecture task does not include devising a new unified framework, however, the ambition of the Trilogy 2 objectives and the precision needed in specification does effectively require that one exists. The architecture in this chapter is to some extent such a unified architecture, however, it is developed as a 'means to an end' and it 'as not the end in itself'. Its purpose is to enable the successful development of solutions for creating and controlling cross-resource, cross-layer, and cross-provider liquidity.

The architecture is based on the well known technique of functional blocks which are common across many engineering disciplines. With a very large simplification, we can contrast some broad characteristics of a 'computation model' and a 'functional block model' as shown in Table 5.1. This comparison is not intended to imply that either model is right or wrong or used exclusively by one group of people[1]. The comparison seeks to illustrate why, for the purposes of Trilogy 2, it is better to start with the functional block model rather than a computation model, even though the objective is to include generic processing.

[1] for example, the architecture in this chapter draws strongly on automata theory which would normally be regarded as sitting on the computational model side.

Table 5.1: Comparison of Computation Model and Functional Block Model

COMPUTATION MODEL	FUNCTIONAL BLOCK MODEL
"does something in order to solve a maths problem"	"solves a maths problem in order to do something"
runs to completion	runs continously
sequential (time is a means of establishing sequence)	temporal (exists in real time)
measured by (information) *size*	measured by (information) *rate*
fundamentally sequential	fundamentally parallel
focussed on the *internal* specification of the 'maths problem'	focussed on the *external* behaviour at the interfaces
generally provides a universal acceptance language	generally has minimal acceptance language consistent with defined behaviour
is generally concerned with the ability to perform any task and to maintain maximum flexibility to options	is generally concerned with closing down options in order to keep interfaces as simple as possible consistent with high levels of reuse

The use cases of Trilogy 2 are interactive, real-world, real-time and so look very much like functional blocks. Moreover, they are temporal in that, in most cases, time rather than just sequence matters, and the natural units of capacity are rates not size (while it is part of the on-going work and not reported here, this is true even for storage where access time matters in many uses cases).

5.1.1 Virtual Functional Blocks as the Architectural Building Blocks

The representation of functional blocks is part of the working methods of many industries as well as different disciplines and perspectives within those industries. As a result, there is not a clear common representation of functional blocks which is unambiguous across different industries, disciplines, and perspectives.

As tools that describe functional blocks are most often used by engineers for the design, development and construction of functional blocks, quite naturally, many tools give considerable emphasis to these phases of the functional block life cycle. For example, in the construction phase, the reuse of common design features is especially important as reuse increases efficiency. Many tools therefore give considerable emphasis to the reuse of such features. In this case classification of functional blocks according to common design features is of considerable value and the natural starting point for describing functional blocks is the class. It is natural to start by representing a class of functional blocks which can be built using the same design. The class diagram can also contain hierarchy, for example an inheritance hierarchy, which can show increasing scope of design reuse at higher levels of the class hierarchy.

However, when describing the operation of functional blocks, the individual instances of functional blocks and the way individual functional blocks interact with each other are important. In this case the natural starting point is not the class but the individual instances. Classification and hierarchy of classification is much less relevant at the operations stage. More important is the way individual functional block instances

are interconnected and interact.

There are of course many other aspects to functional blocks which may be important to represent. For example the nature of what is passed between functional blocks may be important to differentiate and in the case of Trilogy 2, we restrict ourself to functional blocks which pass information and only information over interfaces. More general functional blocks may pass fluid (pressure and flow), electricity (voltage and current), rotation (revs and torque), money, etc.

Here we are concerned with the basic characteristics of information functional blocks. We can assume that all the parameters passed between functional blocks are information of one form or an other. This section considers some of the basic properties of information functional blocks. It focuses on the case where a functional block such as a server or a network acts as a host functional block, hosting virtual functional blocks such as virtual machines and virtual networks.

In this case, it is important to highlight the properties of functional blocks in operation and so all the discussion and diagrams in this document show functional block instances (and not classes of functional blocks) unless otherwise stated.

The section gives brief overview of the application of functional blocks to virtualisation. As we are aware, there are aspects of this architecture which break new ground and certainly some of the critical properties are not present in any standard modelling toolkit such as UML, SysML, or BPMN. Indeed, none of the currently available toolkits have any adequate means of describing virtualisation and in this they are importantly deficient. There are many consequences and conclusions which are not discussed in the deliverable, as they are beyond its scope and might otherwise distract from the objective of the architecture which is to set a framework for the rest of the project. The architecture has already been taken up the ETSI NFV ISG.

A functional block consists of:

- a set of input interfaces;
- state;
- a transfer function;
- a set of output interfaces.

When considering the functional block at its most fundamental level, the proper operation of a functional block is causal and the flow of causality from input to output is central to the methodology. In this case it is normally more convenient to consider all inputs as separate from all outputs. A basic view of a functional block illustrated in Figure 5.1 below.

There a number of fundamental properties of functional blocks.

- the transfer function is fixed and defining of the functional block;
- the set of all possible values of state is fixed and is defining of the functional block;

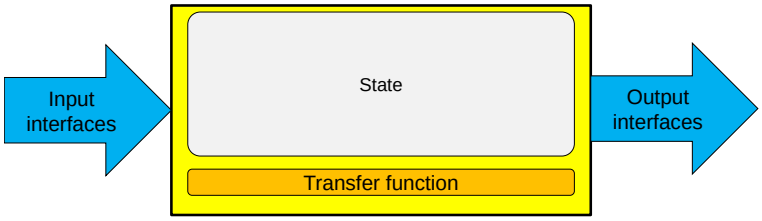


Figure 5.1: A Functional Block

- the set of all possible input values is fixed and is defining of the functional block;
- the set of all possible output values is fixed and is defining of the functional block;
- the transfer function is the set of mappings from a specific value of the tuple of input value and current state value to a specific value of the tuple of output value and updated state value;
- the state is the ability of the evolution of the functional block to be dependent on historic inputs and not just on the current input;
- the process of acquiring the current input value, the current state value, calculating the transfer function mapping, and setting the next state value and the next output value takes a finite amount of time;
- the exact amount of time may vary with each specific transfer function mapping.

A central property of functional blocks is the complete and formal separation of the static from the dynamic. Using a more IT oriented terminology, the input, output, and internal (ie state) data structures and all the methods (ie the transfer functions) are static. They must not change. Only the values of data within the data structures can change; these values are the only things which are dynamic.

For standardised functional blocks, in order to ensure proper interoperability, the goal would normally be to fully define all the static parameters of the functional block in the standard.

Having defined what a functional block is from the inside, the next fundamental property of a functional block is the ability to interconnect functional blocks. This is achieved by connecting an output interface of one functional block with the input interface of another functional block. For this to work the following must be true:

- the data structure of the output interface of the functional block on one side of the interconnection must be compatible with the data structure input interface of the functional block on the other;
- the output set of values must be a subset of the input set of values.

This interconnection of interfaces is called an interface binding. This arrangement is illustrated in Figure 5.2 below.

As illustrated in Figure 5.2, when a number of functional blocks are interconnected all together, the interface bindings form a topology graph between the functional blocks.

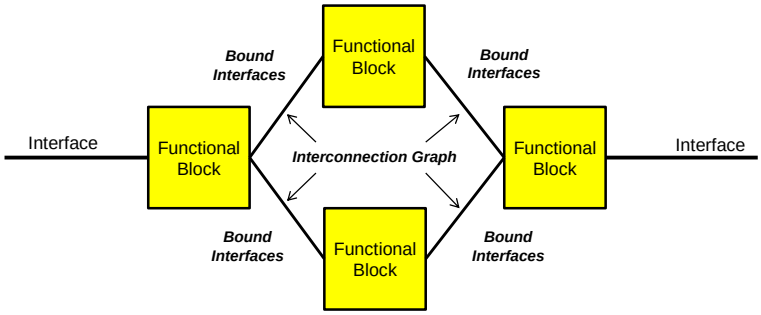


Figure 5.2: Binding of Functional Blocks and Interconnection Graph

When a number of functional blocks are interconnected in a topology graph, some input interfaces and some output interfaces remain. A property of the functional block methodology is that the entity as viewed through these remaining input and output interfaces is also a functional block and meets all the properties of a single functional block. This is illustrated in Figure 5.3.

This means that functional blocks have a fundamental recursive property. A larger functional block can be created by composing a number of a smaller functional blocks and interconnecting them with a specific topology.

Another fundamental property of functional blocks which is immediately apparent from this recursive property is that functional blocks are inherently parallel, concurrent, and asynchronous . Any sequential and synchronous properties will arise only as a special case, normally by imposing explicit design constraints on the static properties of all the constituent functional blocks.

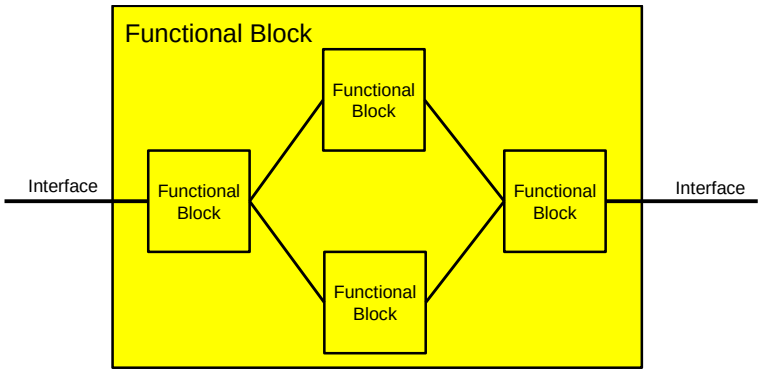


Figure 5.3: Recursive Composition of Functional Blocks

This section has restated well established properties of functional blocks. In the next section, we introduce an extension to this functional block methodology.

5.1.2 Virtualisation and Abstraction

Functional block methodology does not anticipate or directly support virtualisation. However, the foundations of the methodology are very general and mathematically robust and it is still possible therefore to understand virtualisation in terms of functional blocks. This section develops the extension of the methodology in terms which are still fully based on the same general, mathematical principles and so still retains the

formal robustness of the methodology.

The concept of virtualisation allows one functional block, a host functional block, to precisely emulate the interfaces and transfer function behaviour of a different functional block. For example:

- a hypervisor allows a server, the host functional block, to emulate a different server, a 'virtual machine' (VM) such that the virtual machine appears to have all the interfaces and behaviour properties of a server;
- a virtual machine running a specific operating system and application software can be made to emulate the interfaces and behaviour of network equipment such as a BRAS such that the virtual machine works indistinguishably from 'conventional' BRAS.

However, the conception of virtualisation can be more general that this. This new architecture addresses the concept of virtualisation by considering what happens when creating a virtual function in terms of functional blocks. We find that the essence of virtualisation is to revisit the boundary between the static and dynamic parts of the functional block specification and we see that virtualisation uses a process with the following steps:

- a host function has some dynamic state which can be set (configured) to a value and held constant for a prescribed period of time (which will be the lifetime of the virtualised function);
- this configuration allows the host to appear to operate according to the specification of the virtualised function – this configuration of the host implements the virtualised function.

If we consider a functional block which can host virtual functions, it is possible to regard a logical partitioning of it state into:

- state which is private to the host;
- state which can be held invariant for the duration of the virtual function;
- state which is allocated to be the state of the virtual functional block (VFB).

This host functional block (HFB) illustrated in Figure 5.4 below.

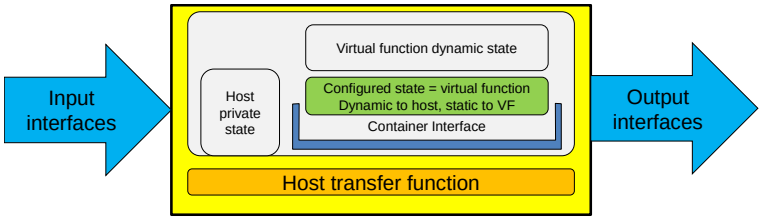


Figure 5.4: Partitioning of State to Create a Host Function

In addition to partitioning the state, we also partition the input interfaces according the state which is affected by the input.

- VFB operational interfaces which are the virtual interfaces to the VFBs;
- HFB configuration interfaces which allow the creation/deletion of VFBs;
- host private interfaces, which control aspects of the HFB which are neither part of the definition of the VFB nor of the operational interfaces of the VFB, control aspects private to the HFB.

This partitioning of the HFB interfaces is illustrated in Figure 5.5.

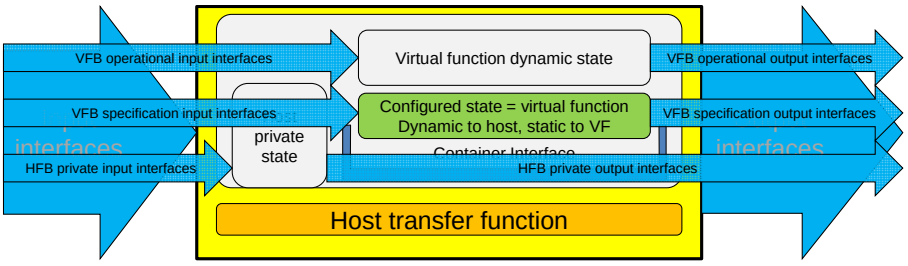


Figure 5.5: Partitioning of Interfaces

The state which can be held invariant is of particular interest for virtualisation. This state defines a set of possible configuration options for the HFB, which may be a small set or a very large set indeed.

Consider two examples, first consider a standard architecture computer being loaded with a programme, and second consider the forwarding engine of a packet switch being configured with a forwarding entry. The number of possible programmes that can be loaded into a standard architecture computer is extremely large, roughly the exponential of the programme memory size. Most of these are unlikely to be useful and most options for programmes will never be exercised, none the less there is still an enormous number of possible useful programmes that can be loaded. In the second case, the forwarding table of the forwarding engine can be configured with a limited number of entries and each forwarding entry is a configuration of the forwarding engine. Also note that in this second case, unlike the first case, changing the order of the entries will not create a different configuration. In this second case, the number of configuration options will be linear with the forwarding table size and not exponential.

In this context, we can also see that the data structure for the VFB specification interface is, by definition, a programming language.

It is now possible to treat this configuration, for the duration for which the configuration option is held constant, as being coupled with the host transfer function. This combination of a specific configuration with the underlying host's transfer function defines a new transfer function. In the first example of the standard architecture computer, to the outside, the computer is behaving according to the execution of the programme; the observed transfer function is that of the executing programme. In the second case, to the outside, the forwarding engine behaviour now is to forward packets within the forwarding table according to the forwarding table entry. This creation of a virtual transfer function is illustrated in Figure 5.6 below.

Finally, as shown in Figure 5.7, when we take together the new virtual transfer function with the state allocated to the VFB and the partition of the interfaces for the VFB, we have a complete virtual functional block. To the

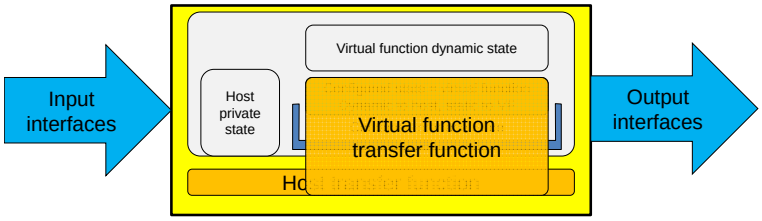


Figure 5.6: Creating a Virtual Transfer Function

outside, when accessed using the VFB partition of the interfaces, the HFB *is* the VFB. The configured HFB is, by definition, an *implementation* of the VFB. This capability of an HFB to be configured/programmed to be a VFB is also equivalent to a "container interface" and an "execution environment".

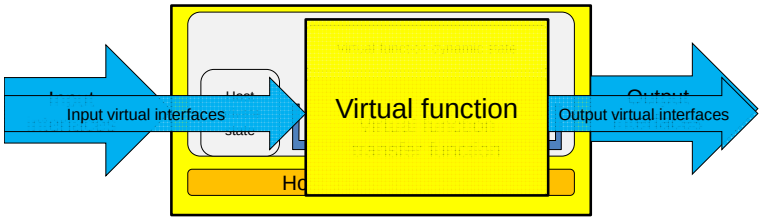


Figure 5.7: Creating a Virtual Function

In fact, it is the case that *all* implementation is exactly this process[2]. It is indeed, this mechanism of virtualisation that allows any implementor freedom to choose and optimise their own implementation of the virtual function. Moreover, all implementation independent specification is a specification of a virtual function. This means that the requirement to extend functional block methodology to virtualisation has also provided a complete and robust solution to implementation independent specification. Or expressed the other way around, all functional specification is an abstract specification given in a specification language and this specification language must be translated into the specific implementation language for any given HFB (assuming the HFB is capable of implementing the required function).

Having precisely defined the process of virtualisation, it can be seen that the existence of a VFB depends precisely on the specific configuration of the HFB which is hosting the VFB. Given that the definition of a functional block requires that the transfer function is static and is defining of the functional block, this means that the configuration which creates the VFB must not change during the lifespan of the VFB. Indeed, the lifespan of the VFB is defined by the period during which the configuration which defines it remains unchanged.

This means that there is a profound and fundamental difference between input interfaces of the HFB that determine the configuration state and the input interfaces that determine the dynamic state of the VFB.

Having set out this framework of virtualisation and taking a hypothesis that it is universal from a functional point of view[3], we can deduce a number of important conclusions from this hypothesis.

[2]While beyond the scope of the project, the concept of configuration of HFBs can equally apply to physical components. For example, a circuit board is a spatial configuration of chips and copper tracks, and a network is spatial configuration of equipments, fibre cables, and other interconnecting cables.

[3]We noted above that the purpose of the (informational) functional block is "to solve a maths problem in order to do something",

Compared to 'hardware' implementation, virtualisation of functional blocks results in:

- division of a functional block between a HFB and a VFB;
- creation of a new container interface between a HFB and a VFB;
- division of the interface between an infrastructure interface to the HFB and a virtual interface to the VFB;
- the VFB is *not* a functional block independent of it host function;
- the container interface is *not* an interface between functional blocks equivalent to other interfaces.

When considering the definition of "abstraction", we note that

- The VFB is, by nature, abstract;
- every viable[4] abstract function is a virtual function;
- Every (abstract) VFB can be implemented on a wide variety of hosts functions.

This is virtualisation shown in Figure 5.8 below.

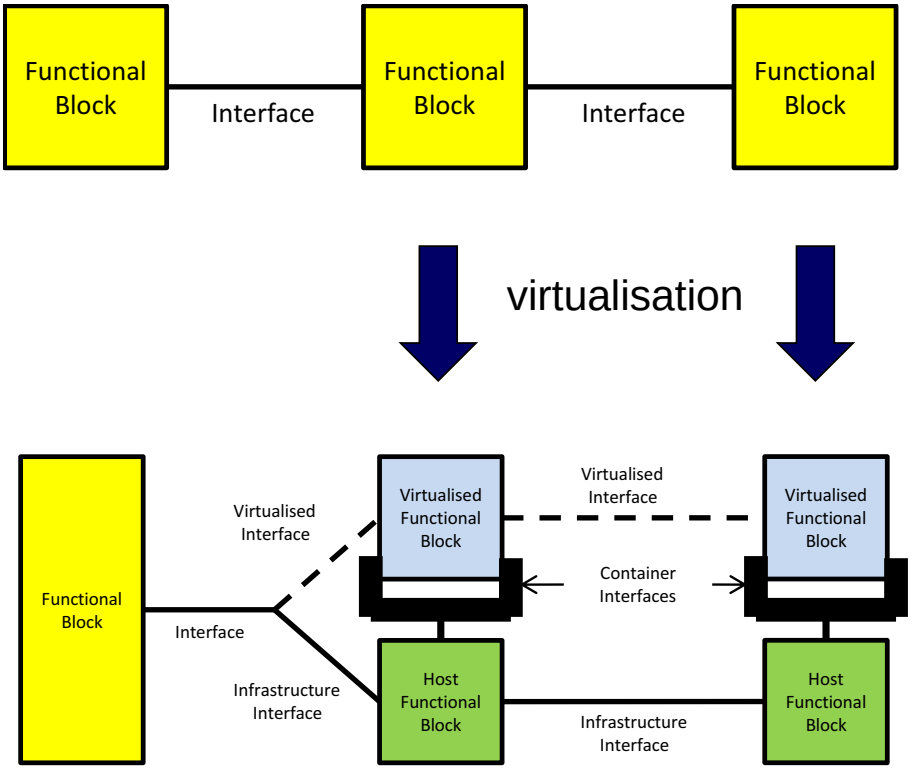


Figure 5.8: Virtualisation of Existing Functional Blocks

There is therefore a fundamental relationship between the abstract view of a functional block - viewed purely from the point of view of what it does and not how it does it - and the implementation of a functional block

and the universality is in the context of the things to be done, not in the universality of maths problems.

[4] as opposed to "abstract" in the context of an incomplete specification of a function

which is concerned with how the functional block does its job. Importantly, there may be very considerably simplification in the apparent behaviour. The behaviour specification of the abstract VFB may be very simple, however, the behaviour of the configured HFB may be highly complex. Consider the example of a connection across a network. The behaviour seen at the interface to the connection (ie its port) is extremely simple - pass packet to far end port. The implementation, on the other hand, has a great complexity of functions including flow control, loss detection and retransmission control, attachment of network address(es), address lookup, forwarding according to address lookup entry, etc. However, these two are exactly equivalent.
Expressing this equivalence is an area where current modelling toolkits appear to be deficient. For example,

- it is not generally possible to show that a UML/SysML activity diagram of an HFB, when certain state is configured[5], is equivalent to a separately defined UML/SysML activity diagram of a VFB;
- the above is the case whether the equivalence is simply asserted or whether it is logically deduced/confirmed by the toolkit;
- abstraction is really only considered from an interface point of view and not a behavioural point of view.

Figure 5.9 below illustrates the equivalence of an “abstract functional block”, a VFB, and a suitably configured HFB.

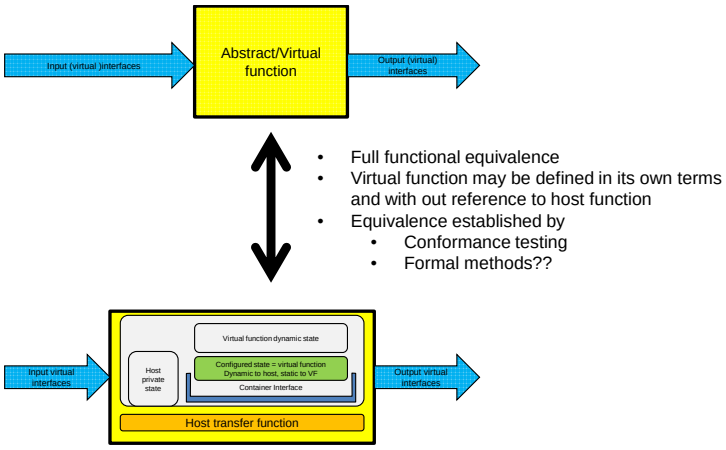


Figure 5.9: Equivalence of Abstract Function, Virtual Function, and Suitably Configured Host Function

We can return to the primary goal of the project, that is the creation and control of liquidity. We can see that the virtualisation framework provides for the creation of a liquid infrastructure, that is a large heterogeneous HFB which is an interconnected set of processing, storage, and network resources. It also provides for the creation of useful services which can use the infrastructure resources, that is VFBs, together with the way in which these are hosted on the infrastructure HFB.
In addition, we can see that the framework also provides the way for describing how the resources of an HFB are assigned to a hosted VFB. Each VFB ’consumes’ resources in so far as a HFB is using its capacity

[5]Indeed, it is generally very hard to link state into an activity diagram

to implement that VFB. This framework is a generalisation of the standard traffic capacity model used for measuring the usage of bandwidth. This is illustrated in Figure 5.10 below and it is this application of the architecture framework which is taken up in detail in the project.

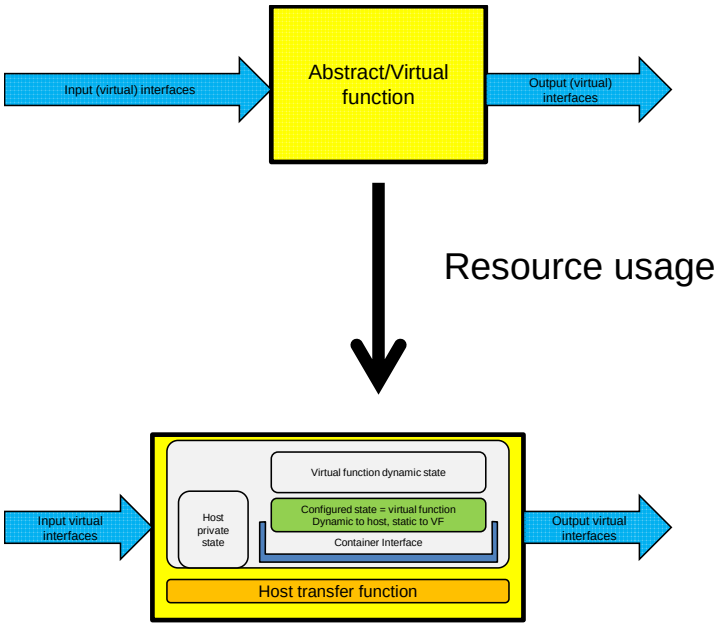


Figure 5.10: Definition of Resource Usage

Finally, when considering this resource usage in more detail, we note that both HFBs and VFBs are recursively composable and decomposable and that the functional block methodology is fundamentally parallel and concurrent. This means that VFBs and HFBs may be distributed with high degrees of concurrency. and when decomposed, the mapping of VFBs to HFBs may be a complex many to many relationship. In particular:

- one HFB may host more than one VFB;
- a single VFB may be hosted across many HFBs.

This complex mapping may be viewed in two ways:

- the mapping arises from the configuration of HFBs which created the VFBs and therefore the mapping is simply an observation of what is in actual operation;
- HFBs will have certain properties based on their level of concurrency which will place constraints on the time required for some specific states to evolve and this will place significant constraints on the mapping if certain performance for the evolution of state is required.

The second of these is especially important where state must be common across geographically distributed HFBs. Managing this constraint on the mapping between VFBs and geographically distributed HFBs is very often a primary design consideration for the performance and/or stability of an end to end VFB.
This mapping is illustrated in Figure 5.11 below.

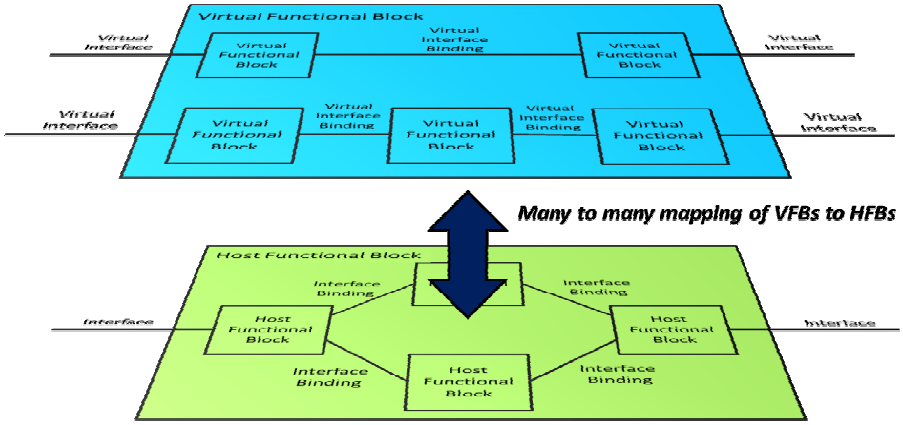


Figure 5.11: General Mapping of Virtual Functional Blocks to Host Functional Blocks

5.1.3 Recursive Virtualisation and A Middle Out Perspective

In section 5.1.1 we derived the well established recursive property of functional blocks, composition/decomposition. We can now develop a second new recursive property for functional blocks, virtualisation.

The recursive property of virtualisation is immediately apparent form the observation that a virtual functional block (VFB) is a functional block and therefore is itself capable of being a host functional block (HFB), as illustrated in Figure 5.12 below. This enables a layering of configuration of an underlying HFB. As an example, consider an IP network.

- There is a physical configuration which creates a network of interconnected routers which is the underlying physical network HFB and is a large scale, distributed, parallel functional block. However, at this low level of virtualisation, there is no specification of routing protocols, forwarding, or end to end sessions. In principle it could be configured to a variety of different protocols and host a variety of different logical networks.

- At the next level of virtualisation, each router is configured with IP addresses, to run an IP routing protocol on each interface, and to have an IP forwarding function. Each router is now an executing router functional block in the network. At this level of virtualisation, an IP network functional block has been created but there is no specification of forwarding or end to end sessions.

- At this next level of virtualisation, the routing protocol configures forwarding tables in each route which establish one or more flow trees to each destination address and the network actually becomes a large number of flow tree functional blocks. The routers disappear and the network inputs simple see access to destinations selected by destination address. However, there is no specification of end to end sessions.

- TCP creates session functional blocks and at this higher level of virtualisation, the flow trees have also disappeared and all this is visible is a large number of point to point session functional blocks. Each

session connection is a complete and properly specified functional block and has abstracted the network on which it is hosted. In fact, for most purposes, only a few point-to-point sessions are normally of any relevance at any one time, and so there is never normally any need to consider the totality of sessions on a network at any one time. In this way, the layered virtualisation automatically create context appropriate abstraction and automatically handles scalability and 'scopability' [6].

In essence, the HFB/VFB framework gives the language to describe the mechanisms that are actually used in reality to handle scale and scope, including the ability to reuse resources in many different ways at many different layers.

A particularly significant consequence of this recursive virtualisation architecture is that it gives a new perspective on "top-down" and "bottom-up" approaches to specification, design, and development.

The merits of 'top-down' and 'bottom-up' are much debated and in most cases, a practical conclusion is that both are needed. A 'top-down' perspective is effective at identifying overall requirements and context as well as the overall architecture while a 'bottom-up' perspective is effective at identifying reuseability of component functions as well as scalability of the final solution. the 'top-down' approach gives the end users perspective while the 'bottom-up' gives the atomic building block perspective.

However, two basic questions inevitably arise:

- who is the ultimate end user for the 'top-down' perspective?
- what degree of smallness constitutes the atoms at 'the bottom'?

In the above example, the user of the TCP session was the ultimate end user. However, as we extend to include processing and storage, many 'end users' of TCP sessions are actually now firmly within scope of the project as the TCP sessions interconnect processing and storage functions within the architecture.

Similarly, the router equipments were regarded as atomic. While this may seem appropriate for the network operator, from the perspective of the equipment vendor, the router equipment is the end product and not the least atomic. From the equipment vendor's perspective, component chips might be atomic as they set about specifying a configuration of chips on plug-in boards and a configuration of plug-in boards in a chassis. And again, as we consider the extension of Trilogy to include NFV, any hard boundary between 'equipment design' and 'network design' is not as clear and robust as it was. Again, we see that the idea that atomicity is absolute is simply not the case.

However, recursive virtualisation does not assert or require an absolute top layer or an absolute bottom layer; indeed, the opposite is the case. Recursive virtualisation is always relative as one layer is defined relative to other layers and any notion of 'top' or 'bottom' only defined as a practical reference which can always be re-defined if necessary.

[6] that is the ability to handle a large and/or increasing scope

- There is always some predefined host functional block and so any identification of a 'bottom' HFB is always only a practical convenience and never fundamental. Someone or something built (ie configured) that HFB out of something more elementary.

- Any VFB block that is capable of receiving inputs is, by definition, capable of further configuration and hence further virtualisation and so any identification of a 'top' is again only of practical convenience and not fundamental. As long as the function can receive an input, it is not ultimately configured.

This framework therefore directly addresses the 'cross-layer' requirement of Trilogy 2. We might call this recursive virtualisation a 'middle out' perspective, where the middle is simply defined as the layer of design and operation of interest. The perspective can then extend up and down as far as in useful and need not assert any absolute top or bottom. This is illustrated in Figure 5.12 below.

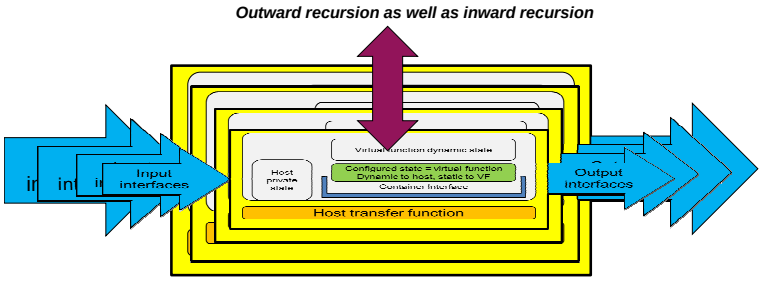


Figure 5.12: Recursive Abstraction Leading to "Middle Out" Architecture

An important final observation on recursive virtualisation is the link with what is static and what is dynamic and the link with layering.

- A fundamental part of the definition of a functional block is that the transfer function is static and does not change.

- Therefore all the configuration of the state which determined the transfer function of a VFB must not change during the life of the VFB.

- At a technical level, therefore, the layer boundaries are directly tied to which state is static and which state is dynamic.

A corollary is that if a process does change some state, it is, by definition, working within the layer which sets that state. This becomes especially important when considering optimisation for two reasons:

- any optimisation, which is always a selection of a member solution from a set of possible solutions, must assume some underlying infrastructure which is fixed to define the set of possible solutions from which the optimal may be picked, and so it is important be clear what this fixed infrastructure is;

- with multi-layer optimisation, layers are formed as a necessary solution to scaling and scoping and therefore any temptation to dynamically and simultaneously optimise across layers is unlikely to scale.

In summary, in a scalable solution, upper layers will treat lower layers as fixed and optimise within their own upper layer on the assumption that the lower operates within it fixed and declared parameters. If the lower layer decides to make changes within its layer, then the upper layer must reoptimisation based on the new parameters from the lower layer. A simple example is a routing protocol. A routing protocol optimises the placement of traffic on a network topology, but it implicitly assumes that the topology is fixed and invariant. Should a lower process building the topology change the topology, then the routing protocol needs to reconverge based on the new topology.

5.2 Initial Trilogy 2 Abstract Functional Architecture

In this section we now use the framework of the basic functional architecture to develop an initial view of the Trilogy 2 Abstract functional architecture. At this level of abstraction, the functional architecture needs to be:

- general enough to be a single functional architecture to be common across all the Trilogy 2 use cases;
- give reasonable expression to the canonical problems and to their solutions;
- develop a generic framework for creating, measuring, monitoring, and controlling liquidity across the scope of Trilogy 2, that is across the resources of bandwidth, storage, and processing and across the environments of mobile devices, operator infrastructure, and the wide area.

5.2.1 Create Liquidity (1) - Establish an Infrastructure of Resources

As was set out in section 5.1.3 above, there is always an underlying host functional block (HFB) which is the underlying infrastructure. Importantly, the point at which this is determined to be fixed, invariant, and atomic is defined pragmatically. The architecture fully understands that when this was defined, there was a lower level process that designed and built this infrastructure. For the purposes of NFV, it is practical to choose the fixed infrastructure to be a set of servers and end user devices (providing processing resources), storage arrays (providing storage resources), interconnected with a carrier Ethernet infrastructure network (providing bandwidth resources). This physical infrastructure is shown at the bottom of Figure 5.13 below.
At the next layer up, the servers and end user devices of this base infrastructure provide a processing and storage container interface (equivalently, an execution environment) which can be configured and its resourcs assigned by a hypervisor. The virtual Ethernet switch (vSwitch) of the hypervisor can be bound to the infrastructure carrier Ethernet network to form and end to end carrier Ethernet network. The hypervisors on the servers and the carrier Ethernet network can now be configured to produce:

- virtual machines (VMs) which embody the processing ans storage resources and are VFBs of the hypervisor (and server below that) and are also HFBs for whatever function might be implemented by the code running on the virtual machines.
- virtual networks (VNs), which embody the bandwidth resources and are VFBs of the carrier Ethernet network and are also HFBs to specific end to end communications (in practice these are likely to be

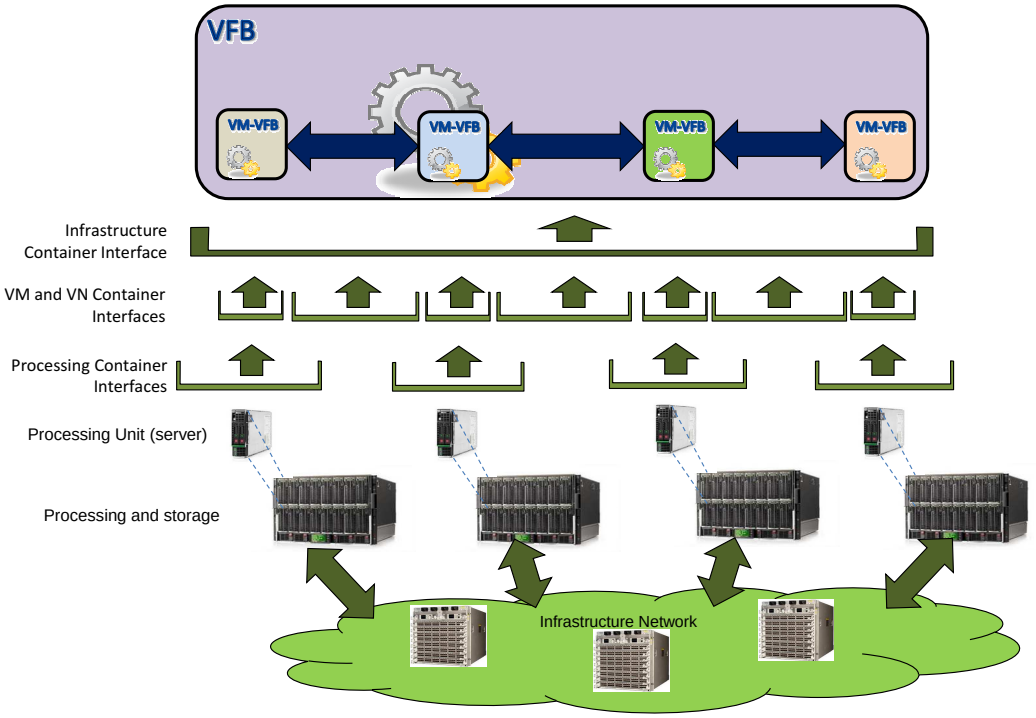


Figure 5.13: Liquidity Infrastructure

E-LANs, that is multipoint to multipoint virtual private LANs which interconnect vNICs of appropriate VMs).

Finally, within this infrastructure, we compose an appropriate set of VMs and VNs as a single HFB for a full scale, distributed, cross-resource application.

The layers above the physical infrastructure are shown as layers of container interfaces in Figure 5.13 and the highest infrastructure container interface is shown implementing a distributed application which itself is decomposed to match the VMs and VNs of the infrastructure.

One particular point worthy of note it that IP routing can be implemented within virtual machines, and hence the Internet is included as a particular usage example of the infrastructure. It would be fully possible to include IP routing with the infrastructure, however, this creates security complications as discussed in section 5.3. The decision to base the infrastructure on Ethernet rather than IP is a practical one and can be readily revisited and does not alter the fundamental framework of virtual functional blocks.

This infrastructure is therefore the heterogeneous resource pool required for the extension of resource control and optimisation to processing and storage as well as to bandwidth and can underpin development of all the use cases.

- *MPTCP* - as the infrastructure can readily host Internet routing and conceptually include the Internet, it supports the necessary resource and functional descriptions required for the development of extensions to MPTCP.

- *PVTCP* - the infrastructure supports the layered fucntionality and description of resources needed for

PVTCP. It can directly support everything from MAC layer inter-VM commnuication within a hypervisor to higher layer protocols. Moreover, as the framework is middle out, even though the VM is taken as the base resource of processing and storage, this can be fully decomposed if need be to reveal inter-process communication even inter-thread communication if so desired.

- *Trevi* - again Trevi is a network protocol and so is accommodated by the layered network framework.
- *ConEx* - the infrastructure framework not only provides a for ConEx in its current form, it also gives insight as to how the principles of ConEx can be applied cross-layer within the network, and possibly cross-resource as well.
- *Irminsule* - provides a framework for distributed storage and therefore allows for the optimisation of storage resources. This is fully hosted by the VM/VN framework.
- *vM3* - this is directly built using a VM/VN framework and allows for end user device resources to be optimally exploited.
- *vBRAS* - this is directly based on a the VM/VN framework and allows processing resources to be optimally exploited
- *vCPE* - this is directly based on a the VM/VN framework and allows processing resources to be optimally exploited as well as defining the necessary network protocols needed to maintain security when the vCPE are moved between hosting sites.
- *Mobile Kibbutz* - the VM/VN infrastructure includes wireless networking including the layering of protocols and so the resource optimisation on this use case is also accommodated by the VM/VN infrastructure framework.

More generally, this VM/VN infrastructure is a direct extension of the canonical problems. It directly includes the basic building blocks of IaaS, that is hypervisors and virtual machines. It also addresses the canonical problem of middleboxes as the framework is properly layered and so protocol layers are accurately described both technically and their mapping to resources. Finally, as the treatment of virtualisation is very general, the canonical problem of lightweight virtual machines is also addressed directly.

The aim of Trilogy 2 is to extend the techniques of Trilogy 1 so as *not* to have to carve off a fixed (solid) partition of the host infrastructure for each VM and VN. Instead the resources of the host infrastructure remain as a liquid pool available to all VMs an VNs, but separate control mechanisms are configured to limit excessive usage and incentivise optimal balancing of load.

This infrastructure framework allows for the complete and accurate description of all the resources. The next phase in the development of the Trilogy 2 architecture is to appropriately parameterise these resources so that the dynamic control mechanisms and algorithms can be applied to their allocation.

5.2.2 Create Liquidity (2) - Definition of Abstract Specification of Desired Functions

With the liquidity infrastructure in place, the next step is to create configuration files for the infrastructure which will implement a required application. This process starts with the abstract specification of the desired functional block. This desired functional block is specified in abstract, implementation independent terms, and in order to implement it, a configuration specification is needed for the intended HFB which will host the implementation. This configuration specification is in the form of a file, that is, it is a piece of information, which when loaded into the HFB causes it to behave according to the specification of the abstract functional block.

Given the host infrastructure described in section 5.2.1 above, the implementation file will consist of three basic information elements:

- a set of virtual machine specification files each of which will describe the necessary configuration of the hypervisor to create an appropriate virtual machine together with a virtual machine disk image which is the binary code of the complete running virtual machine for a VM-VFB component of the required overall VFB;
- a set of VN specification files which specify the E-LAN services needed to interconnect the VMs (these services are effectively a configuration of the infrastructure carrier Ethernet network);
- a set of additional constraints which restrict the possible mapping of VMs and or VNs to specific hosting servers or networks.

This process is illustrated in Figure 5.14 below.

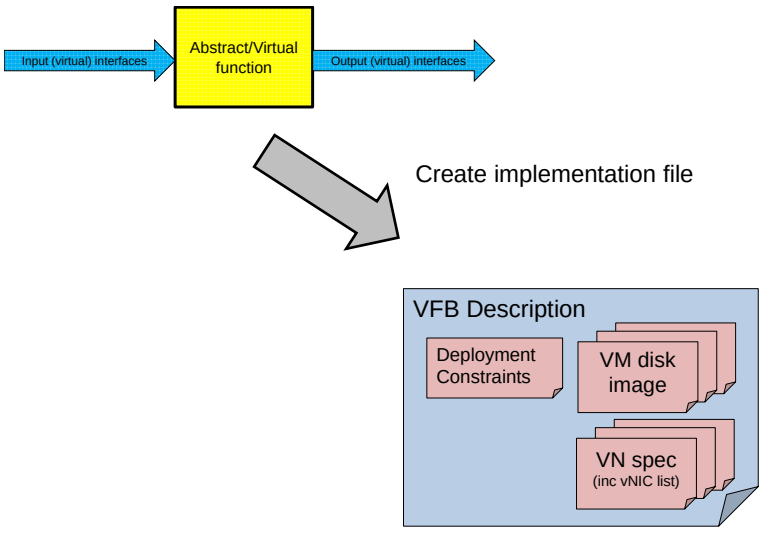


Figure 5.14: Creation of an Equivalent Implementation File from an Abstract Specification

For Trilogy 2, the aim is for the configuration file not to fix (solidify) the resources assigned to each VM and VN, but instead to define the parameters of the control mechanisms that will enforce limits on the use of the resource pool and that will provide the economic incentives for guest functions to balance load across the

pool. Thus, the configuration of the *behaviour* of virtualised services is solidified at creation time, but the configuration of the resources needed to serve that behaviour is left liquid. Then the function's behaviour itself fixes (solidifies) the amount of resources it needs at run-time, on much shorter, more dynamic, timescales.
It is possible and indeed likely that many such implementation files will be created for different abstract functional block/HFB pairings ahead of time. This means that when a particular application functional block is required, it can be created by selecting the appropriate implementation file from a library of implementation files. This motivates the microkernel approach in which small functional blocks decomposed from larger ones each carry minimal virtual machine overhead for rapid execution.
In summary, the creation of a host infrastructure together with the creation of a library of implementation files enables the creation of liquidity.
However, we can also note that the creation of the VFB descriptions is itself a process with a strong analogy to compiling of source code into binary code. With compiling there are several variants that compile at different timescales, ranging from standard compilers, through just in time compilers to interpreters. There are a similar range of options for the creation of VFB descriptions. This form of optimisation of VFB description creation was not directly anticipated in the project objectives and so is only noted at this stage. If time, resources, and opportunity allow, it may be possible to develop such optimisation within the project, but, more likely, this will be work beyond the scope of Trilogy 2.

5.2.3 Control Liquidity (1) - Creation and Activation of Applications and Services

The creation of a host infrastructure together with a library of implementation files allows for a set of applications to be instantiated on the infrastructure. We can trace the basic process which results in the instantiation of an application instance.

(i) An application VFB description (created as described in section 5.2.2 above) is registered and lodged in a VFB description repository which is accessible by an application orchestration system.

(ii) A user request for an application instance is received.

(iii) An orchestration functional block identifies the application and finds and collects the correct VFB description from the VFB description repository.

(iv) Using the VFB description for the application, the orchestration functional block works out the required VM instances and VN instances and issues requests for their instantiation.

(v) An infrastructure mapping and resource optimisation functional block receives the requests and decides to which hypervisors and which infrastructure network controllers to send the VM and VN instantiation requests. This system also tracks the mapping of VFBs to HFBs. Note that Trilogy 2 aims for the infrastructure mapping and resource optimisation functional block to be logically divided into host and guest parts, so that the pool of resources can remain largely unassigned at activation time, and only fixed (solidified) later at run-time.

(vi) The hypervisors and infrastructure network controllers create the required VM and VN instances.

(vii) A VFB may be able to request new VMs and/or VNs for itself within previously defined limits.

This process is shown in Figure 5.15 below.

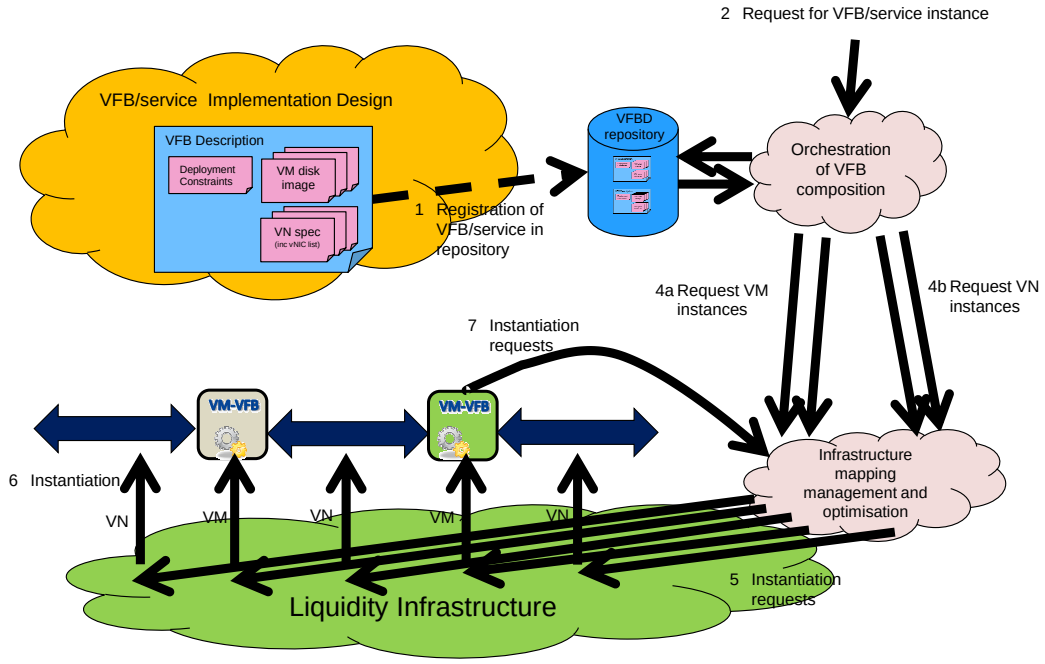


Figure 5.15: Creation and Activation of a VFB Instance

At this point, we can also define the relationship between the concepts of 'application' and 'service'[7]

- An application is a VFB and emphasises the static characteristics of a functional block that is created.
- A service is normally defined as a usage of a functional block and therefore emphasises the dynamic characteristics of a functional block.

However, we can see that using the layered virtualisation architecture these are in fact different viewpoints of the same thing. When the time scale is altered appropriately, we can see that a service is a VFB with a shorter lifespan. This is readily apparent when we consider a session. A session is normally viewed as a service instance and is a dynamic usage of the network supporting the session. However, from the point of view of the session user, the session is a static functional block which provides connectivity (or whatever functionality is associated with the particular type of session) for the duration of the session.

5.2.4 Control Liquidity (2) - Resource Optimisation

Finally, there is the process which is at the heart of the Trilogy 2 objectives; the optimisation of the resource usage. There are three stages to this process:

[7] By service, we primarily mean service in a technical sense rather than a commercial sense. The full development of the spectrum of service from technical to commercial is the subject of future work.

- characterising pooled HFB resource capacity together with notifying current cost of usage of a particular HFB's resource (which is directly analogous to and a generalisation of the resource plane in Trilogy 1);
- an optimal mapping of VFBs to HFBs (which is directly analogous to and a generalisation of the reachability plane in Trilogy 1);
- a dynamic assignment of HFB resource (such as CPU clock cycles and different forms of storage such as cache, RAM, and disk to different storage instances) to VFBs (which is directly analogous and a generalisation of the use of the resource plane by transport services in Trilogy 1);
- enforcement of contracted limits on usage and settlement of costs incurred to incentivise the optimal mapping and assignment of VFBs to HFBs.

Having now established the architectural framework as set out in this deliverable which can accommodate the cross-resource, cross-layer, cross-provider context of Trilogy 2, detailed work on these three is the next phase of work.
It is useful to note that at this stage, ETSI NFV ISG is generally assuming a centralised control model for resource mapping and assignment which will inevitably have both scaling limits and have difficulties with cross-provider interfaces. Trilogy 2 is seeking solutions without these limitations.

5.3 Trilogy 2 Security Architecture

This section organises discussion of the security of resource pooling into cross-resource (§5.3.1), cross-layer (§5.3.2) and cross-provider (§5.3.3). §5.3.4 then brings together these three viewpoints by briefly summarising the security disciplines that all three draw on.
Trilogy 2 aims to build systems for pooling resources built on security that is intrinsic to the structure of the architecture. Intrinsic or structural security techniques are often not even recognised as security techniques, but they need to be the focus of security at the architectural stage. Examples of structural security are:

- Ensuring that resources are described using units that relate to the underlying economics, which gives the system intrinsic security against gaming;
- Structuring the system to naturally exploit various forms of isolation;
- Designing protocols to exploit in-band signalling, which inextricably binds signals to the data they relate to, thus avoiding the need to cryptographically bind control messages to the information objects that they control.

Structural security techniques, including these examples, appear throughout the following three sections. Then they are revisited in §5.3.4, which provides a taxonomy of the structural techniques employed. For

instance, isolation techniques are further categorised into information isolation, performance isolation and failure isolation.

This survey of relevant security disciplines also highlights that, although sound foundations using intrinsic, structural techniques are good, it is still necessary to explicitly engineer (non-structural) security techniques. For instance, information security techniques such as authentication, encryption, integrity verification, etc. still play an important role and the control protocols still need to be carefully designed to prevent abuse.

Finally, §5.3.5 offers general security guidance that is important but too general to fit elsewhere.

5.3.1 Security for Cross-Resource Liquidity

5.3.1.1 Narrowing the Security Problem Space

If processing, storage and bandwidth resources are each considered separately, there is already work in progress on securing each of these areas separately. Although we have to be aware of these separate security areas, this is not the focus of Trilogy 2.

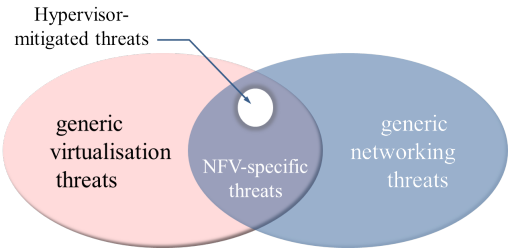


Figure 5.16: Change in the Threat Surface by Combining Networking with Compute Virtualisation

Combined virtualisation of computation, storage and network infrastructure requires defence against the known threats in each area separately, as well as the new threats from combining them. Cloud technology already virtualises applications that combine processing and storage, while NFV adds network functions to the virtualisation mix. So the focus of our security attention should be the new vulnerabilities due to this new combination—the intersection in Figure 5.16.

This intersection is the focus of the ETSI NFV industry specification group Security Problem Statement [6], which is an output of Trilogy 2 (providing editorship and much of the technical material). In particular, this problem statement identifies performance isolation as an area where solutions are lacking and where research is required, particularly for the I/O (bandwidth) resource.

5.3.1.2 Economic Metrics for Performance Isolation

The main architectural focus of Trilogy 2 is to identify the metrics that capture the essence of the economics of using each resource. Then, even if this metric is not directly used in commercial contracts, technical mechanisms can be designed to control this metric, and the configuration of those mechanisms can be derived from the applicable commercial contract governing its usage.

For bandwidth, the metric that expresses the economics is the congestion-bit-rate [b/s], which is the product of bit-rate [b/s] and the probability of congestion [%]. The congestion-rate is measurable as the rate that

bits are dropped due to packet discard or the rate that packets are marked by explicit congestion notification (ECN [21]), both measured in b/s.

In the original Trilogy project, identification of this metric led to the development of congestion policing mechanisms [10] so that a network operator could limit the level to which users/customers could impinge on the service being provided to others. It also led to the realisation that the congestion-rate metric was not visible to network operators in the Internet architecture, which in turn led to the development of the congestion exposure protocol (ConEx), and the creation of an IETF working group to standardise ConEx experimentally as an architectural change to IP and TCP, which is still in progress [11, 18].

This lesson from Trilogy 1 illustrates that it is important not only to identify the right economic metric, but also to ensure that the system architecture makes it practical to use the metric to control resource usage. In the case of packet traffic, the sender ultimately governs consumption of the bandwidth resource, so the metric has to be usable where the sender's traffic enters a network operator's domain, not just at the receiver.

Applying this lesson to Trilogy 2, this means we still have work to do on controlling the bandwidth resource, as well as extending the lessons to other resources. For instance, even in a case where the receiver nominally controls the sending rate, as in the Trevi transport protocol [20] that uses multicast to write to storage, the receiver is merely asking the sender to control its sending rate, and the sender is still ultimately in control.

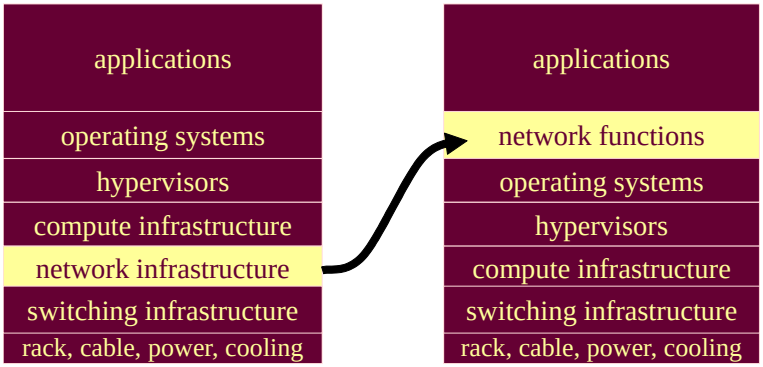
The Trilogy 2 project has therefore been developing alternative ways to make the congestion-rate of traffic streams accessible to network operators, without having to wait for adoption of an architectural protocol change. The approaches fall into two categories:

- The network operator simply uses local congestion at a predominant bottleneck as an approximation of the congestion along the paths through the network [9];
- the network operator creates tunnels across its network and generate its own congestion feedback from each tunnel egress for a congestion policer to use at the ingress [13].

A specific aim of this architectural approach is to make it unnecessary to hard-partition I/O resources between storage and data networking, whether within internal server I/O lanes, internal vSwitches, top-of-rack switches or switches and routers on the way to more remote storage. It will then be impossible for congestion within the I/O system to exceed the sum of everyone's congestion allowances, because whenever anyone exceeds their own allowance, their congestion policer will focus congestion at their entry into the I/O system, which will prevent further load entering the system by focusing discard solely on those exceeding their allowance.

A congestion policer can be distributed so that it enforces a single logical allowance, even though it is distributed physically over a customer's multiple entry-points into the I/O system. This approach is designed to be agnostic to the transport being used, whether unipath like TCP & UDP, or multipath like MPTCP [14] or Trevi [20].

It is more challenging to make the congestion metric in multicast reflect the underlying economics, because multicast duplicates the economic metric that represents congestion (a loss is duplicated to every downstream leg of the tree). This multicast metric problem is solvable in theory, at least for explicit congestion notification, by randomly picking only one copy of the packet on which to forward the ECN marking at each multicast branch point. This preserves the economic meaning of a congestion signal and it can still signal congestion to all the other multicast receivers using a different codepoint [12, §5], which allows a multicast session to use either sender-driven or receiver-driven congestion control (eg. Trevi [20] writes to storage using multicast with receiver-driven congestion control). However, this would require a change to existing multicast forwarding hardware, and it's not immediately apparent that even an SDN-controlled switch would be able to do this.



(Note: 'compute infrastructure' is taken to mean both processing and storage.)

Figure 5.17: Rearrangement of the Layers of Dependency in a System when Virtualising Network Functions

5.3.2 Security of Cross-Layer Liquidity

The layering of an architecture that combines compute and network infrastructure will include the OSI reference model used in networking, but go beyond it. Figure 5.17 shows a stack of systems that an application relies on, ordered by dependency. Those nearer the top can treat the resources below them as more liquid, and conversely solidity increases towards the bottom. The shift from the left stack to the right shows how network functions sit higher in the dependency stack when they are virtualised (NFV), so they can exploit the greater resource liquidity below them, much as virtualised applications can today.

However, this re-arrangement of the layers also inverts the security dependencies. Compute infrastructure still depends on rudimentary network infrastructure (generic hardware switching, etc.). However, any network functions that are virtualised become more dependent on the security of the compute infrastructure that they sit on. This could open up new vulnerabilities, but it also offers better opportunities to secure the infrastructure, e.g. the infrastructure operator can:

- use the introspection capabilities of hypervisors;
- spin-up virtualised ad hoc security monitoring processes;
- rapidly patch security bugs in network code once no longer implemented in hardware.

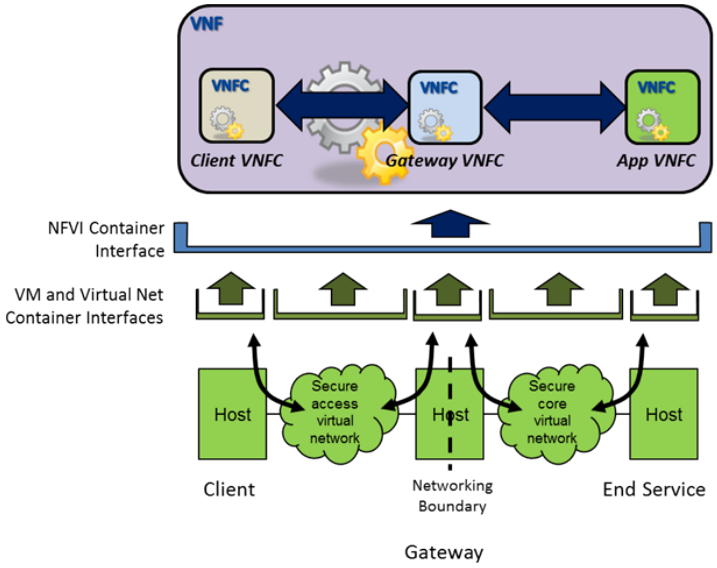
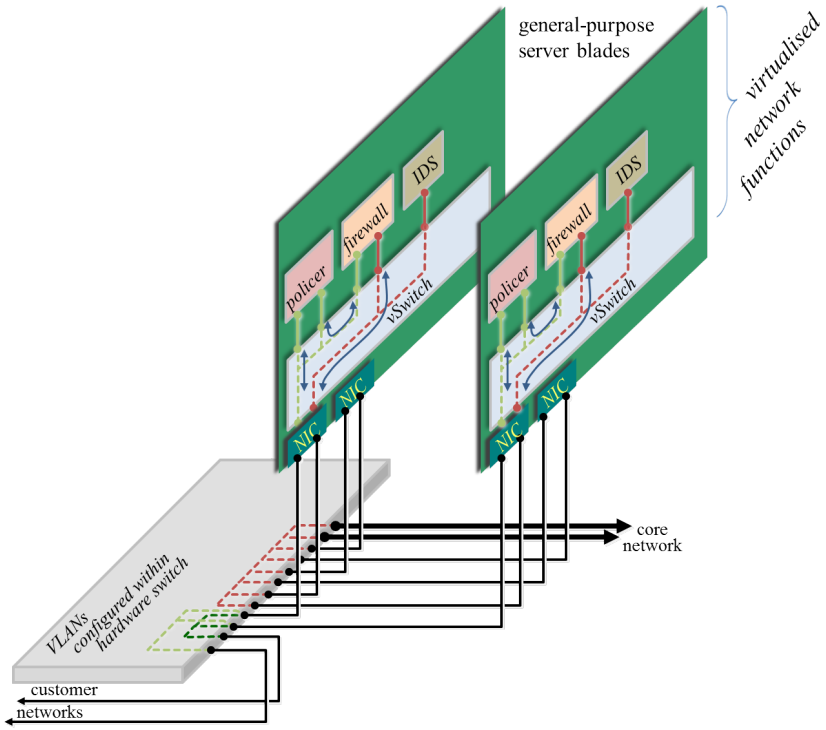


Figure 5.18: Partitioning Boundaries Between Infrastructure Networks

Likewise, compute infrastructure now solely sits on top of an infrastructure network that is simpler and more stable, therefore potentially more secure. But on the other hand compute can no longer depend on connectivity through virtualised network functions (e.g. during boot).

The main security consequence of implementing part of a network's connectivity in a higher layer is that the lower layer has to consist of many disconnected network partitions that will only be connected together when the higher layer software is executed. For instance, in Figure 5.18 the virtualised network function component



(The hypervisor that oversees the life-cycle of the virtualised network functions is not shown.)

Figure 5.19: Example Cross-Layer Network

at upper centre offers a gateway function that forwards certain traffic between the physical access and core networks shown at the lower layer. The essence of a gateway is to provide conditional connectivity, e.g. it might pass all routing adverts from core to access, but only one-hop routes from access to core. It would defeat this conditional function of the gateway if the core and access were also connected at the hardware layer. The dashed line labelled 'networking boundary' represents the complete disconnectedness between the two networks at the infrastructure layer.

Figure 5.19 illustrates this point with a security-related example. It shows a couple of server blades each running a firewall, intrusion detection system and rate shaper as virtualised software, all connected via a vSwitch to two of the network interfaces of each blade. The network interfaces are connected to red-side and green-side networks configured as VLANs within the infrastructure layer (the hardware switch), shown as different coloured connectivity within the switch.

Clearly, the operator intends that there should not be direct physical connectivity between the red-side and green-side networks, without going through a firewall and a shaper, and without the oversight of an IDS. Figure 5.20 illustrates this by showing the position before the firewall boots, or after it shuts down or crashes. The no-entry signs emphasise the lack of connectivity between each little disconnected partition of the infrastructure network.

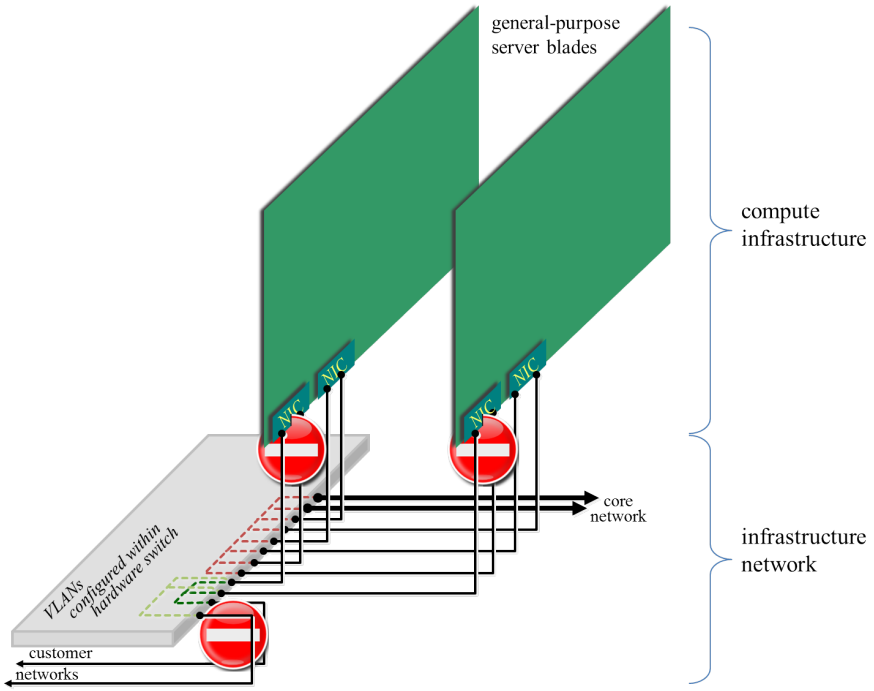


Figure 5.20: Example Partitioned Infrastructure Network

This has profound implications for how routing is controlled. All the partitioned infrastructure networks have to be able to complete their connectivity while disconnected from each other. So, if they are centrally configured on the SDN model by an open-flow controller, the controller has to be able to connect to each partition (e.g. over a separate management network). Then the idea that an SDN controller understands the

connectivity of its network can only be realised for each disconnected partition. It is not feasible for an open-flow controller to describe, let alone understand, the conditional connectivity through a complex application layer stateful firewall moderated by an IDS, and potentially being shaped to various rates conditional on the firewall's rules. Instead, one can envisage that the virtualised network functions would be instantiated by an orchestration function that only understands each network function as an opaque blob of software. All orchestration will know is that each blob has to be connected in a certain way to the interfaces of the infrastructure network. Unlike an SDN controller it will not know the internal behaviour of the blob, which would require machine understanding of arbitrary code.

The NFV Security Problem Statement [6, S6.1] highlights these issues, and further discusses the problem of validating whether a topology satisfies an operator's security policy.

5.3.3 Security of Cross-Provider Liquidity

There are two forms of cross-provider system, each covered in the following sub-sections:

Vertical: Different providers operate different layers of a stack of dependent services;

Horizontal: Different providers operate different domains of the service in one layer of the stack.

5.3.3.1 Vertical Cross-Provider

Figure 5.21 (which is a simplified version of Figure 5.17) illustrates the elements with potentially separate security responsibility in different deployment scenarios: the building, the host compute hardware, the hypervisors and the guest virtual network functions within their virtual machines.

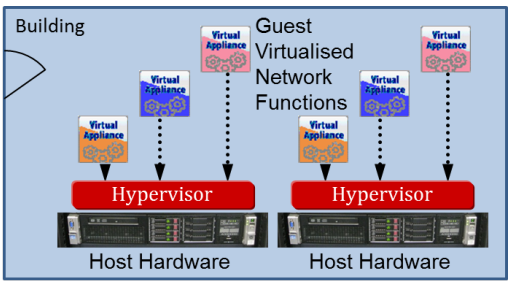


Figure 5.21: Security Dependencies between Providers in Various Deployment Scenarios

Below some deployment scenarios are described that are likely in realistic contractual arrangements. These include:

- Scenarios that have become common in cloud computing, as recorded in the NIST Definition of Cloud Computing [19];
- Scenarios considered likely to be realistic for NFV; a Trilogy 2 output into the NFV security reference framework provided in [6, S5];
- Scenarios used in Trilogy 2 use-cases but not already included in the above.

For convenience these scenarios are also summarised in Table 5.2. The right-most column also identifies the NFV deployment scenarios that are similar to the common cloud deployment models identified by NIST. All but the last row (currently purely a Trilogy 2 scenario) have been defined in the NFV Security Problem Statement [6, S5] for the ETSI industry specification group (ISG) and wider industry to use as a reference framework, so that everyone can be clear which security scenario they are discussing.

Deployment Scenario	Building	Host Hardware	Hypervisor	Guest VNF	cf. NIST Cloud Model
Monolithic Operator	N	N	N	N	Private Cloud
Network Operator Hosting Virtual Network Operators	N	N	N	N, N1, N2	Hybrid Cloud
Hosted Network Operator	H	H	H	N	
Hosted Communications Providers	H	H	H	N1, N2, N3	Community Cloud
Hosted Communications and Application Providers	H	H	H	N1, N2, N3, P	Public Cloud
Managed Network Service on Customer Premises	C	N	N	N	
Managed Network Service on Customer Equipment	C	C	N	N	
Network Service on Peer Customer Equipment	C1	C1	C1	C2	

The different letters are not meant to be overly significant. They are chosen to represent different traditional industry players, e.g. H = hosting provider, N = network operator, P = public, C = customer, although it can be seen that these 'player' names do not always reflect the bundles of roles adopted by the player.

Table 5.2: Some realistic deployment scenarios.

In general, in order to determine the security implications of a deployment scenario, the two main factors are:

(i) The different parties that operate each of the levels (building, host hardware, hypervisor, guest virtualised network function).

(ii) Whether the party at any one layer has exclusive or non-exclusive use of the resources of the lower layers and, if non-exclusive, are the resources available to all other parties or only to a restricted set (e.g. only allied operators? competing operators? the general public?)

Monolithic Operator: The same organisation that operates the virtualised network functions deploys and controls the hardware and hypervisors they run on and physically secures the premises in which they are located.

Network Operator Hosting Virtual Network Operators: Based on the 'Monolithic Operator' model, except as well as hosting itself, the network operator (e.g. BT, Verizon) hosts other virtual network operators (e.g. Virgin Mobile, Tescos, Walmart) within the same facility. It would probably isolate each virtual operator on separate hardware. However, in theory, the virtual machines of different virtual network operators could run alongside each other over the same hypervisor.

Hosted Network Operator: An IT services organisation (e.g. HP, Fujitsu) operates the compute hardware, infrastructure network and hypervisors on which a separate network operator (e.g. BT, Verizon) runs virtualised network functions. The premises including cable chambers, patch panels etc. are physically secured by the IT services organisation.

Hosted Communications Providers: Similar to 'Hosted Network Operator', except the IT services organisation hosts multiple communications providers. Alternatively, the IT services organisation may host one (or many) wholesale network operators, which in turn host multiple virtual retail communications providers. In this latter case the IT services organisation would give controlled rights to run virtualised network functions to the wholesaler, which could in turn delegate rights to the virtualised retailers.

Hosted Communications and Application Providers: Similar to 'Hosted Communications Providers' except servers in a data centre facility are offered to the public for deploying virtualised applications (Cloud) while in the same physical facility network operators and communications providers deploy virtualised network functions on the same type of generic hardware platforms (probably not sharing the same server hardware, but in blades and racks alongside and sharing the same data centre network).

Managed Network Service on Customer Premises: A network operator runs virtualised network functions on its own generic server hardware located on a customer's premises and physically secured by the customer. This model could be in a residential or enterprise scenario, for example, respectively, a remotely managed home gateway or remotely managed VPN gateways, firewalls, etc.

Managed Network Service on Customer Premises Equipment: Similar to 'Managed Network Service on Customer Premises', except the compute hardware is supplied and operated by the customer not the network operator. The customer allocates a blade to the network operator, which runs a hypervisor on this blade, and in turn runs all the necessary network functions within virtual machines over this hypervisor. This model does not involve the customer running virtual machines on the same hypervisor and hardware as the network operator, although this would be another valid scenario (similar to 'Hosted Network Operator' except the customer both hosts the virtual network functions of the network operator and runs guest applications in virtual machines alongside them).

Network Service on Peer Customer Equipment: One customer operates a service on their compute hardware (e.g. their mobile handset) on behalf of another peer customer, as in the Trilogy 2 'Mobile Kibbutz' [5].

5.3.3.2 Horizontal Cross-Provider

Horizontal federation applies particularly to networking, which is supplied as paths that can be comprised of segments from different providers in order to achieve global coverage, as well as coverage between end-users across a competitive local market.

The information exposure needed to control usage of the bandwidth resource across horizontal federations was one of the outputs of the Trilogy 1 project, by exposing downstream congestion information at inter-domain borders [8]. This then allowed translation between the metric used in:

- horizontal inter-provider and provider-customer contracts
- the congestion control algorithms used in transport protocols like TCP, whether unipath, multipath or polyversal.

When expanding from bandwidth to also include compute and storage resources, the Trilogy 2 project has so far focused solely on the vertical cross-provider model. Horizontal chains of provision of compute resources (i.e. within a layer) are common where the result of one process depends on another. For instance cloud Web services (application layer use of compute) or the working group in formation at the IETF on service function chaining (SFC) (network layer use of compute).
For application layer use of compute resources, the mature web-services market already has well-established interfaces that incorporate usage charging models that reflect local compute, storage and bandwidth usage. However, these interfaces are not translatable between different commercial models and therefore not amenable to standardisation between cloud hosting providers. Therefore there is scope to introduce compute and storage metrics based on the underlying resource *economics* that can be standardised as a lingua franca for interworking between the different *commercial* models.
For networking layer use of compute resources (NFV), it seems unlikely that chains of network functions will interact between operators, and more likely that each operator will introduce and control its own independent service function chains.

5.3.4 Security Disciplines

Trilogy 2 draws on two main security disciplines: information security and system isolation boundaries, mainly focusing on the latter for information isolation and performance isolation when using pooled resources. A brief taxonomy of these disciplines is given below to help explain which areas of security are applicable to the different parts of the architecture. Then further details are given on the categories central to Trilogy 2:

(i) ***Isolation***

Information isolation: Used for instance:

- During Forwarding, e.g. using virtual networks (see §5.3.4.1)
- In memory and registers (see §5.3.4.2 on virtualisation)
- To protect identity material (physical isolation in tamper-proof storage & processing)

Performance isolation: The main focus of Trilogy 2 for isolating performance between users of pooled resources (see §5.3.4.3)

Failure isolation: e.g. clean-up after crash as part of virtualisation technology

(ii) ***Information Security (Cryptography)***

Identity verification: for access control, authorisation and message authentication

Configuration Integrity checking: for checking code integrity on execution

(iii) ***Secure Time:*** This embodies both isolation and infosec inseparably, so requires a category of its own. Secure time will be addressed in future work.

(iv) ***Protocol Security:*** Hardening protocols for controlling and trading resources against abuse. See, for example:

- the security analysis of the MPTCP protocol [7, §2.1.2]
- the support for information Integrity and Audit in the ConEx protocol [18, §5.5]

Similar security review will be required for Trevi [20], as a transport protocol for storage I/O.

Isolation is distinct from information security in that it uses physical or logical separation to divide up access to information or to authorise access to a system, rather than relying on knowledge of cryptographic secrets. Isolation may be used as a defence in depth to complement cryptographic protection. For instance a VPN may prevent data being forwarded to sites that are not members of the VPN, but some or all of the data forwarded over the VPN may also be encrypted, which prevents active eavesdropping and protects against accidental mis-routing.

5.3.4.1 Information Isolation using Virtual Networking

Figure 5.22 uses the example scenario already introduced in §5.3.2 to illustrate how virtual networking can be used to isolated a network for management and control from the networks for customer data. The management interfaces of machines are generally protected by cryptographic security (keys or passwords), but isolating their connectivity onto a separate virtual network provides defence-in-depth, given support engineers regularly move to new posts and access control procedures may not always have been followed correctly.

Isolation of a management network is useful for resilience as well as security. The management network is the penultimate resort for recovering a system from a hard crash or from the operator accidentally locking themselves out (the last resort being to physically visit the machine). Given that NFV makes networking dependent on compute rather than the other way round (as shown in Figure 5.17 earlier), a useful principle is to ensure that the management network is not dependent on compute virtualisation, at least only for stub connectivity, not for interior connectivity within the management network. This principle has been articulated in the ETSI NFV Security Problem Statement [6, S6.2].

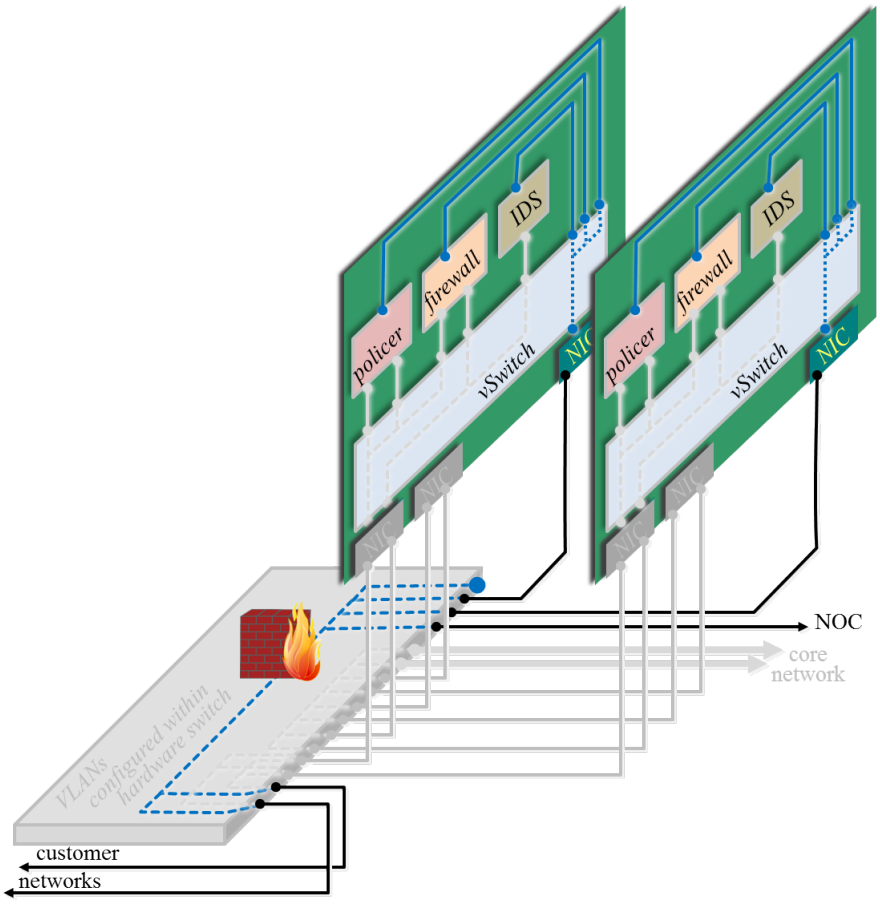


Figure 5.22: Example Partitioned Management Network

5.3.4.2 Information Isolation using Virtualisation

Virtual networking only prevents data from leaking out of a network at unintended output interfaces. On its own virtual networking only provides isolation between the different virtual networks when viewed at the access interfaces; it doesn't prevent snooping the information during transit and it doesn't provide isolation within the nodes of the network. For the former, encryption is required. For the latter, virtualisation is the technology for isolating information so that the operator of one virtual machine (VM) cannot snoop on information being processed by another VM.

In Trilogy 2, the microkernel approach being used for MirageOS [16] and for ClickOS [17] provides fundamentally the same separation as traditional virtualisation technology. However, by using a functional language, the resulting image is inherently typesafe and also compiles down into solely those parts of the VM needed for the virtualised application. This aggressive removal of unnecessary complexity is more important than most other security advances.

In the extreme, it may be required to isolate the capabilities of one set of administrators from those of another, e.g. to minimise the risk of internal fraud within a network operator or a hosting operator. This would require trusted platform module hardware, which can ensure that keying information is isolated within a tamper-resistant module. This will be an area of further work.

5.3.4.3 Performance Isolation

Early virtualisation technology made performance suffer, but more recent advances have streamlined the architecture, for instance by adding support for I/O virtualisation into network interface hardware (e.g. single root I/O virtualisation defined by the PCI SIG [3]). Although the intention was to preserve the isolation boundaries of virtualisation, this has required compromises and the additional complexity has led to more bugs and security flaws.

The NFV security problem statement describes the state of the art in this space [6, S6.5] for sharing all the following categories of infrastructure:

- cores
- memory
- network & I/O generally
- acceleration hardware
- virtualisation infrastructure (the hypervisor and its supporting software)

Network workload is particularly hard to isolate from that of other guests. The NFV Security Problem Statement [6, §6.5] explains that this is because network load can range widely over different distributed network resources and it can be highly variable at any point, making any static partitioning very inefficient, and in data centres tenant churn even puts dynamic partitioning of each switch beyond feasibility.

In order to at least localise the problem, a network's topology is often arranged so that the capacity at a server's interface will tend to be the bottleneck. The interior of a network can be made strictly non-blocking using sufficient capacity and multipath routing. However, often competitive pressure drives providers to save on core capacity and rely on the low probability of traffic all focusing on one core link. Any partitioning between customers on high capacity core links would require too much per-packet processing and/or too much per-guest state and churn, therefore they still rely solely on edge techniques.

Algorithms like weighted round-robin are sometimes used to divide the edge bandwidth resource only between active workloads. Participants in the Trilogy 2 project have been instrumental in implementing hierarchical QoS including WRR in the open-source data plane development kit (DPDK[1]). However, algorithms such as WRR hold no memory of how often each guest has been active over time, so intermittent workloads lose out considerably to persistent ones. In other words, by design, these algorithms favour flooding attacks, even in the outgoing direction.

Isolating usage of local network resources is hard for another reason. Large numbers of remote users can send incoming traffic that makes demands on the local resources, both bandwidth and interrupt processing. This problem has as much to do with preventing identifier duplication/spoofing as with scheduling technology, but even without any malicious intentions, isolation is hard for incoming workloads.

Network QoS (quality of service) differentiation can be used to ensure more critical tasks are less likely to contend for network resources. However, QoS adds extra dimensions of complexity for network function developers and network operators and it deliberately does not address the performance isolation problem for the bulk of regular traffic.

Therefore, the Trilogy 2 project has been developing congestion policing algorithms. These use a congestion-rate metric to reflect the underlying economics and they correctly take account of usage over time. §5.3.1 should be referred to for more details and references to IETF and other documentation.

5.3.5 General Good Security Practice

The weakest aspect of security is nearly always the procedures involving humans. Although it is down to the ultimate operators of technology to get this right, the technology itself can help by automating as much as possible, and minimising the complexity of the human parts of the processes.

Microkernel technology has already been discussed as a simplifying principle, and the Trilogy 2 focus on using the right metrics within the mechanisms avoids additional system complexity having to be introduced to correct for commercial use of metrics that do not reflect the economics of the underlying isolation problem.

The Trilogy 2 project has also contributed general warnings on safety vs. complexity; and altered norms and procedures into the NFV Problem Statement [6, S7]. We also espouse the principle of using open-source code and open security processes to ensure the widest possible validation.

6 Evaluation and Future Work

6.1 Dissemination and Standards Already Achieved

The architecture described in this deliverable has already been presented and adopted by the ETSI NFV ISG. This architecture has provided a number of critical concepts for the work of the ISG.

- The nature of hosting, that is the relationship between a VFB and an HFB had been a source of considerable confusion and many early diagrams showed them as distinct functional blocks with a conventional interface between them. This architecture firmly established this is not the case and that the VFB is the HFB and that there is an execution environment (or equivalently a container interface) between them. This is now clearly described in the NFV Framework Architecture which is already published.

- The more detailed architecture of the liquidity infrastructure has been adopted by the Infrastructure (INF) Working Group and a summary is described in the NFV Infrastructure Overview document. A more complete and detailed description is given in the Interfaces and Abstraction document. Both of these documents have been agreed by the INF working group and are at the final draft stage.

- The architecture has resulted in constructive dialogue on the relationship between functional block architecture and information models and the nature of functional construction versus functional operation in preparation on developing the necessary information models of the northbound management interfaces for NFV.

6.2 Future Architecture Work

The further work on Trilogy 2 architecture includes the following items.

- Building on mechanisms being developed WP1, development of capacity measures for storage and processing resources.

- Building on mechanisms being developed WP1, development of resource mapping algorithms which can support distributed, automated mapping of VFBs to HFB in a cross-resource, cross-layer, cross provider environment.

- Building on mechanisms being developed WP1, development of resource assignment algorithms which can support distributed, automated assignment of VFBs to HFB in a cross-resource, cross-layer, cross provider environment.

- Further elaboration of the architecture, in particular as more specific detailed levels and worked examples, as required by WP3.

- A further iteration of the architecture together a linked iteration of the information models of task 2.1.

- Further dissemination of the architecture working through with WP4.

In addition there are some items which merit further which were not anticipated in the original project proposal. While it may be possible to develop some work in these areas, it would anticipates that these are areas for work beyond the scope of Trilogy 2.

- Development of toolkit methods for representing equivalence of a VFB to a suitably configured HFB and dissemination to OMG and any other relevant bodies.
- Development of optimisation tools for the optimal creation of VFBDs.

Bibliography

[1] Intel(R) DPDK: Data Plane Development Kit. Online: http://dpdk.org/.

[2] Trilogy Project. http://trilogy-project.org/.

[3] Single Root I/O Virtualization. Specification v1.1, PCI-SIG, 2009. (member-only access).

[4] Basic tools for liquidity control. Deliverable D2.2, Trilogy 2 EU 7th Framework Project ICT-317756, December 2013.

[5] Initial Cross-Liquidity tools. Deliverable D1.2, Trilogy 2 EU 7th Framework Project ICT-317756, December 2013.

[6] Network Functions Virtualisation; Security; Problem Statement. Group Specification ETSI GS NFV SEC001 v0.0.8, ETSI NFV ISG, December 2013. (work in progress).

[7] Software Platforms. Deliverable D1.1, Trilogy 2 EU 7th Framework Project ICT-317756, December 2013.

[8] Bob Briscoe. Using Self-Interest to Prevent Malice; Fixing the Denial of Service Flaw of the Internet. In *Proc Workshop on the Economics of Securing the Information Infrastructure*, October 2006.

[9] Bob Briscoe. Nice traffic management without new protocols. Presentation to ISOC bandwidth management roundtable, Online: http://www.bobbriscoe.net/present.html#1210isoc, October 2012.

[10] Bob Briscoe. Network Performance Isolation using Congestion Policing. Internet Draft draft-briscoe-conex-policing-00, Internet Engineering Task Force, February 2013. Work in progress.

[11] Bob Briscoe, Alissa Cooper, and Richard Woundy. ConEx Concepts and Use Cases. Request for Comments 6789, Internet Engineering Task Force, July 2012. (Status: Informational).

[12] Bob Briscoe and Jon Crowcroft. An Open ECN service in the IP layer. Technical Report TR-DVA9-2001-001, BT, February 2001.

[13] Bob Briscoe and Murari Sridharan. Network Performance Isolation in Data Centres using Congestion Policing. Internet Draft draft-briscoe-conex-data-centre-01, Internet Engineering Task Force, February 2013. Work in progress.

[14] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, and Janardhan Iyengar. Architectural Guidelines for Multipath TCP Development. Request for Comments 6182, Internet Engineering Task Force, March 2011.

[15] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. Request for Comments 6824, Internet Engineering Task Force, January 2013.

[16] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS'13)*, 2013.

[17] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling Fast, Dynamic Network Processing with ClickOS. In *Proc ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[18] Matt Mathis and Bob Briscoe. Congestion Exposure (ConEx) Concepts and Abstract Mechanism. Internet Draft draft-ietf-conex-abstract-mech-08, Internet Engineering Task Force, October 2013. Work in progress.

[19] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Special Publication 800-145, NIST Computer Security Division, September 2011.

[20] George Parisis, Toby Moncaster, Anil Madhavapeddy, and Jon Crowcroft. Trevi: Watering Down Storage Hotspots with Cool Fountain Codes. In *Proc ACM Hot Topics in Networking (HOTNETS-XII)*, 2013.

[21] K. K. Ramakrishnan, Sally Floyd, and David Black. The Addition of Explicit Congestion Notification (ECN) to IP. Request for Comments 3168, Internet Engineering Task Force, September 2001.

