An Architecture for Concurrent Future Networks

Lars Völker*, Denis Martin*, Ibtissam El Khayat†, Christoph Werle*, Martina Zitterbart*
*Institut für Telematik, Universität Karlsruhe (TH), Germany †Ericsson GmbH, Germany

I. Introduction

One of the key interests in Future Internet research is network virtualization [1]. It allows to operate multiple networks in parallel, each possibly having different applications, requirements, protocols, and even users. For the 4WARD Project[1] network virtualization is one of the core concepts to *let 1000 networks bloom*.

With such highly specialized virtual networks, users do not just use a single network but will use several virtual networks in parallel. A user could, for example, have one virtual network optimized for voice communication with family members, while another virtualized network is used for communicating sensitive information with business partners. A third general purpose network, like today's Internet, could be used for exchanging electronic mail. These networks could be even of different *Network Architectures*, e. g. using different protocols, and different addressing schemes.

To benefit from the possibilities of such specialized virtual networks, it is important to throughly consider the architecture of the end-node participating at a multitude of networks. We call this architecture a *Node Architecture*.

In addition, a good design process is needed to allow for rapid creation of protocols for new networks. It is very likely that different approaches, like composing or generating communications protocols, will accelerate the implementation and deployment in some cases significantly, while in other cases other approaches, like third party software, might give better results. Therefore, it is essential to support different approaches of constructing communication protocols, so that the network architect or user can choose the best approach for the virtual network and application at hand.

We define the following goals for our concepts:

- Support multiple virtual networks with possibly different architectures in parallel
- Support different approaches for creating protocols
- Support novel communication paradigms, like Network of Information [2]
- Allow new naming and addressing schemes
- Be efficient, flexible, and extensible
- Leave room for business cases and services

This paper is structured as follows: Section II will present our Node Architecture including the main concepts to achieve the aforementioned goals. After that, Section III will sketch the design process needed to accelerate the development and implementation of new networks and network protocols. We will close this paper with Related Work, Conclusion and Future Work.

[1]The FP7 4WARD Project – http://www.4ward-project.eu

II. The Node Architecture

The Node Architecture we propose does not require any specific protocols to operate. It can be seen as a framework or platform for a multitude of future protocols and applications that are able to run side-by-side. Changing such a Future Internet protocol *stack* will not require the application or the network interface to be adapted. Therefore, exchanging one set of protocols with another is possible without modifying any application software or hardware in the future. This way, applications may communicate via a multitude, possibly virtualized networks without requiring them to know about the underlying network architecture.

To achieve this goal, some changes have to be made to today's concepts. Our proposal for a new Node Architecture is depicted in Figure 1 and described in the following subsections.

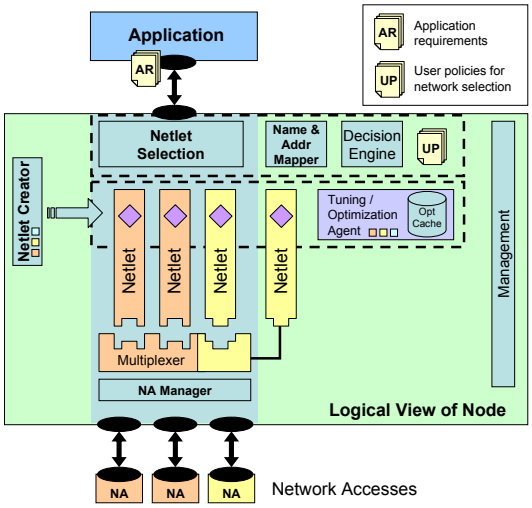

Figure 1. Node Architecture

A. The Netlet – a container for Future Internet protocols

Encapsulation by *Netlets* is the fundamental concept in our Node Architecture to allow the Future Internet to smoothly evolve with multiple, parallel networks. A Netlet can be compared with today's network protocol stacks and encapsulates the protocols of the Future Internet. Naively said, a Netlet

provides simple send/receive interfaces at its top and bottom. In practice, of course, they offer additional operations for configuration and attachment to the Netlet Selector at the top (see Section II-D) and the multiplexer at the bottom. Netlets will be dynamically instantiated by the Netlet Creator.

The Netlet itself may be of an arbitrary nature – it could be compiled from lightweight, handwritten code or could be generated using a complex, formal definition taking best practices in Software Engineering into account. There are also countless other possible ways. While we do not require a specific design for a Netlet, we are currently aiming to provide design methods for a structured, verifiable protocol composition for various future network applications. Besides that, it is also possible to implement existing approaches, like e. g. RNA [3] and SILO [4] as Netlets.

Netlets of the same network architecture have a common basic understanding of the communication protocols, i. e. they understand the same basic PDU format. This is necessary, for instance, because there could be the need of a common identifier to allow multiplexing between the communication streams of different Netlets of the same architecture. To avoid such invariants common to all architectures, we allow for architecture specific multiplexers, denoted in Figure 1 through different shapes of Netlet bottom connectors. These multiplexers have to be installed on the system where a specific architecture should run.

Netlets may have configuration parameters that modify their behavior slightly. This could be, for instance, the maximum data unit size they are allowed to use or the encryption strength to be used by Netlets supporting encryption. These parameters may be needed to be adapted if, for instance, the properties of the underlying network change (e. g. because of a handover from a wired to a wireless connection type). For this case, a Tuning and Optimization Agent constantly monitors the conditions of the Netlets, the network, and the applications and tunes configuration parameters to adapt the Netlet as good as possible to the new conditions.

B. Application Interface – Moving forward from the socket API

Today's applications commonly use the socket API abstraction (or a similar interface) to communicate. This abstraction, however, has a major problem: the application needs to know the address of the peer in the address format used inside a given virtual network. Since applications often only know a (human readable) name of their communication peers, name resolution has to take place. This name resolution is commonly achieved by the application using a resolver library. Examining IPv4/IPv6 stacks these days, an obvious problem may be identified. The resolver might return two different addresses for one well-known name: one IPv4 and one IPv6 address. The application now uses a heuristic: try IPv6 first and fall back to IPv4 after a timeout – commonly a few seconds[2]. Such a simple heuristic however has its limitations. It means that the selection of two possible different paths through the Internet will be just based on "IPv6 first, IPv4 second". One could imagine the case that the path using IPv6 has a unacceptable high latency, while the IPv4 path would be still acceptable. In such a case a better approach for selection would be preferable.

For these and other reasons (e. g. support of novel naming schemes), our Node Architecture moves the name resolution to the system as Name & Address Mapper (compare to Figure 1) and lets the application work with names only.

[2]"IPv4 first, IPv6 second" would be also conform to [5], but this would not support a active transition towards IPv6.

C. Network Access Interface – Transparently supporting Network Virtualization

Our interface to access any underlying network infrastructure is called Network Access (NA). Unlike today's network interfaces, it was designed as a clean interface whose level of abstraction and functionality is not fixed but extensible to the needs of any future network architecture. NAs may provide properties of the network they are connected to, which are used for Netlet selection, tuning, and optimization.

Our goal is to allow running multiple different network architectures side-by-side. If network virtualization is used, new Virtual Network Accesses (VNA) are created to access the (virtual) infrastructures of the virtual networks. This way, the generic interface of the NA can hide the presence of network virtualization from the Netlets and from the Node Architecture in general. The VNA may be completely transparent to the Node Architecture.

Network Accesses are registered at a Network Access Manager. The Network Access Manager has the task to associate the Network Accesses to architecture specific multiplexers. This guarantees that Netlets accessing a network speak the same language as the network itself, e. g. in terms of PDU formats.

D. Automatic Selection – Choosing the best Netlet

A system running multiple Netlets side-by-side is not necessarily user friendly, since the user has to always choose a Netlet for a given communication association. Our Node Architecture therefore implements an automated selection approach inside the Decision Engine, based on [6]. This automatic selection takes many different properties into account. The properties (e.g. latency, throughput, effective bit strength) are grouped into different categories, e. g. Security, QoS, Energy Consumption, and Mobility.

Using these properties, applications can state their requirements for the network (application requirements in Figure 1). Also user or administrator policies can be defined this way. All of them can define which properties are important, so that the selection approach can take this into account.

Based on these policies the selection approach will now calculate the utility of the available Netlets using Multi Attribute Utility Theory (MAUT). It therefore converts every property (e. g. latency) to a property independent utility by using a utility function. The utility function defines how much utility a certain value of the property has, e. g. low latency might be

valuable for a VoIP call; thus having high utility. It should be clear that every property needs a specific value function to convert its values to the utility. In the next step the combined utility of a Netlet is calculated from the utilities of the different properties and the policies.

The selection algorithm itself, however, does not support composed Netlets directly. To support Netlets, which are composed of different functional blocks, we use an aggregation algorithm, which calculates the overall properties of a composed Netlet by the properties of the functional blocks it is composed of.

The current implementation of the selection algorithm is fast enough to make decisions for more than 1000 connection attempts per second, using a moderate number of alternatives. It is however subject to further investigation, whether the aggregation algorithm can work for different composition approaches.

III. Design Process

Providing a platform for running multiple network architectures in parallel does not automatically guarantee success for running a multitude of virtual networks. It is also necessary to ease the development of network architectures and protocols. The complexity of protocol design may lead to common errors that are repeated over and over again. To avoid these errors and to allow for rapid architecture prototyping, we are developing a structured design process. This process lends from Software Engineering methodologies (MDA [7]) and provides general guidelines for network architectures. Such structured methodologies have been proven indispensable in software development, and we now aim at developing similar techniques for network development. Additionally, a catalog of standard functionalities (e. g. for routing, naming and addressing, transport, mobility, QoS, and security) is provided. Together with the design guidelines, a network architect may choose among alternatives and opt for the ones that fit his requirements.

The design process is currently split into three phases: (1) the identification of requirements, (2) the definition of the Network Architecture Model (NAM), and (3) a deduced Software Architecture Model (SAM). From the Software Architecture Model, an implementation of the network architecture is finally created. The phases are not self-contained but will provide feedback to the other phases as depicted in Figure 2. Therefore, as in Software Engineering, the whole design process is an iterative process that will refine all intermediate models until the end-state is satisfactory.

During the first phase, the requirement specification for a new network architecture is developed. In the common case, there is a business idea or service request at the beginning that is highly incomplete and most likely from a non-expert. A domain expert, i. e. a network architect, identifies in this phase the detailed technical requirements imposed on the network architecture that are needed to realize the service or idea. This phase is very similar to the requirement specifications that are best practice in Software Engineering.

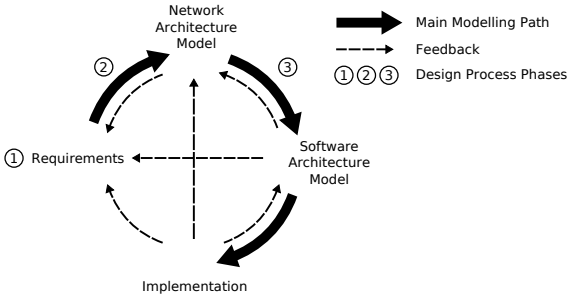

Figure 2. Design Process

In the second phase, the network architect walks through different abstraction levels or views. First, he defines the basic network elements (e. g. node types, link types, etc.), what basic functionalities they have, and how they are interconnected. From this fairly abstract view, he looks closer at the nodes and identifies which parts of the aforementioned functionalities are run on the respective nodes as protocols. Groupings of these protocol functionalities build *Netlets* that are specific to the new network architecture. Getting even more into detail, the construction of the Netlets themselves is defined: here, an off-line composition approach may be considered, or standard Netlets supplied by third parties may be used as *black boxes*.

The third phase of the design process focuses on the software architectural aspect of the Node Architecture. With the specification taken from phase two, actual interface definitions are created for the elements within the Node Architecture, the Netlets, and the functional blocks within the Netlets. For this, the Network Architecture Model can be transformed into a Software Architecture Model using model-to-model transformation. During this transformation, missing details are added as required for the software platform.

As a possible final step, code or stub code may be generated with the help of the Software Architecture Model via model-to-code transformation. Since this is mainly Software Engineering research, we do not focus on this part, but we pave the way for any solutions in this domain.

IV. Related Work

While there is a lot of research regarding network virtualization and new communication protocols, most of them only focus on small problem spaces and look there into the details – we instead are looking for a framework around all those approaches.

The driving forces behind composable protocol stacks are mostly described as follows: (1) overcome strict layering and allow for clean cross-layer interaction, (2) avoid recurring functionality at several layers, (3) allow reuse of smaller functional units in several protocols, and (4) simplify the design of protocols.

While composition approaches in the past were mostly looking at dynamic composition at run-time with all its complexity (e. g. F-CSS [8]), recent research shifted towards more structured approaches where the solution space is restrained and part of the decision process is done off-line, either at

design-time or at network configuration time (e. g. RNA [3], SILO [4]). Another approach (RBA [9]) proposes a unified protocol header allowing that stored information can be used by several units of functionality, regardless on their order of operation or the layer they appear in.

Virtualization techniques allow to run several isolated networks in parallel over a single physical substrate [1], [10], [11]. The architectures of these networks do not matter as long as they are able to interface with the virtualization infrastructure. The focus of the different virtualization techniques lies more on the signaling for reservation of virtual resources.

Although, virtualization offers great opportunities to run novel architectures in parallel without big investment in infrastructure, it may not be needed in all cases. For instance, if there is no real need for strict isolation of network architectures, the aforementioned composition approaches could share the same (physical or virtual) channel – provided that there is some generic Node Architecture around them as presented in this paper.

V. Conclusion and Future Work

In this paper we have presented our Node Architecture for the Future Internet. It allows for arbitrary, parallel network architectures on a single node and the smooth transition towards the Future Internet as well as the evolution beyond. For the evolution beyond, we are working on a design process that allows for fast development of a multitude of future network architectures. Here, we are investigating possible tool support for the concepts and patterns we develop. Future work also includes the further refinement and evaluation of our concepts.

Acknowledgment

Parts of this research were carried out within the 4WARD project of the 7th framework programme (FP7) and are partially funded by the European Commission. We would like to thank all the partners involved in valuable discussions and contributions about the concepts presented here. In addition, the authors would especially like to thank Sören Finster for valuable input during writing this paper.

References

[1] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time", *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, 2007.

[2] V. Jacobson, M. Mosko, D. Smetters, and J. J. Garcia-Luna-Aceves, "Content-centric networking", Whitepaper, 2007.

[3] J. D. Touch, Y.-S. Wang, and V. Pingali, "A Recursive Network Architecture", ISI, Tech. Rep., Oct 2006, iSI-TR-2006-626.

[4] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet", in *Proc. IEEE International Conference on Communications ICC '07*, G. N. Rouskas, Ed., Glasgow, Scotland, 2007, pp. 1899–1904.

[5] E. Nordmark and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213 (Proposed Standard), Oct. 2005.

[6] L. Völker, C. Werle, and M. Zitterbart, "Decision Process for Automated Selection of Security Protocols", in *33nd IEEE Conference on Local Computer Networks (LCN 2008)*. Montreal, Canada: IEEE, Oct. 2008.

[7] T. Stahl and M. Völter, *Model-Driven Software Development*. John Wiley & Sons, 2006.

[8] M. Zitterbart, B. Stiller, and A. Tantawy, "A model for flexible high-performance communication subsystems", *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 507–518, May 1993.

[9] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture", *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 17–22, 2003.

[10] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation", in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. Pisa, Italy: ACM, 2006, pp. 3–14.

[11] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization", *Computer*, vol. 38, no. 4, pp. 34–41, 2005.