A Node Architecture for 1000 Future Networks

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

***Abstract*—One possible key technology for the Future Internet is network virtualization. It allows to run numerous virtual networks in parallel, each of which can be adapted towards different requirements, intended use, or applications used. When consequently using network virtualization, it allows not only to have very specialized networks but also allows to run new protocols and services in different networks. This can give opportunities for rapid service deployment, especially for services based on new protocols.**

Currently a lot of research is concerned with network virtualization or related aspects like management or signaling of network virtualization. This paper however is different, since it looks on network virtualization from another angle. We describe our Node Architecture for the Future Internet, which uses network virtualization as a fundamental concept. It has the goal to give users access to a vast number of virtual networks and exploit the possibilities of network virtualization.

I. Introduction

One of the goals of the 4WARD Project[1] is to *let 1000 networks bloom*. With a vast number of networks one could supply networks for different applications, requirements, and even users. And all networks could use completely different protocols. This gives many opportunities to the users and providers, which today would be either not possible or too expensive. The development towards network virtualization can be seen as an enabler for such concurrent operations of multiple networks.

When having the vast amount of networks we envision, every node might be part of a multitude of virtual networks in parallel. As example, a user could use different virtual networks for different applications. He could have a high bandwidth network for high definition video streaming, a specially secured network for the transfer of sensible information, an information centric network for finding current news and reports, and a latency optimized network for voice communications. Each of which could use different *Network Architectures*, with, for example, different addressing schemes and different protocols.

To fully facilitate the possibilities of network virtualization, the architecture of the end nodes needs to be capable of supporting a multitude of networks. Within this paper details of such a Node Architecture (briefly introduced in [1]) are presented. The Node Architecture was designed with the following requirements in mind:

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

The main contributions of this paper include a Node Architecture using so-called Netlets for exploiting the possibilities of network virtualization, and enhanced application and network interfaces to communicate requirements and properties. A design process for future networks was sketched in [1], a selection process for automatic selection of virtual networks is presented in [3]. Together, this provides a tool-set for using a multitude of networks on an end node.

This paper is structured as follows: Section II presents the Node Architecture including the main concepts to achieve the aforementioned goals. In Section III an example of one of the core components of the Node Architecture – the Netlet – is presented. Related work is discussed within Section IV. Conclusions and outlook are provided in Section V.

II. The Node Architecture

The Node Architecture proposed in this paper was designed to support network virtualization and to run multiple networks side-by-side. It allows to add, modify, and remove highly specialized communication protocols for (virtual) networks with possibly different addressing schemes. 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 of possibly virtualized networks without requiring them to know about the underlying network architecture – a clear improvement for example over today's socket API, where the application is at least involved in name to address resolution and in the selection of the protocol it wants to use for communication.

Our proposal for a new Node Architecture is depicted in Figure 1 and described in the following subsections. This figure shows the architecture of an end system, but it can be easily transformed into an architecture for an intermediate system (e. g. a router) in removing application and user related functionality.

A. Overview

Netlets as used in Figure 1 are the fundamental concept of our Node Architecture to allow the Future Internet to smoothly evolve with multiple, parallel networks. A Netlet can simply be seen as a container to encapsulate today's and future network protocol stacks. With the principle of hiding protocol details but yet providing a number of properties via its interfaces, Netlets can be easily exchanged without the need to change application or network interfaces. They are described in more detail in Section II-B.

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

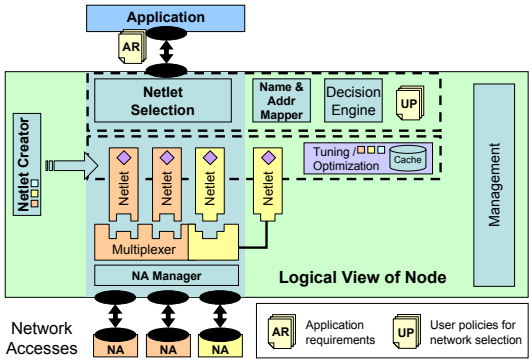

Figure 1. Node Architecture

The *Netlet Selection* component provides a node-local registry of all currently active Netlets. Based on the Netlet properties and the requirements the application has on the communication association, the Netlet Selection component chooses – with the aid of the Name & Address Mapper and the Decision Engine – the Netlet that fits the application's requirements best. If the underlying network's properties change during the life-time of a communication association, a Netlet may also be exchanged by another. If none of the active Netlets can be used, a new one might need to be instantiated (see Section II-F). While the Netlet Selection block contains the generic selection mechanism, the actual selection algorithm is executed within the *Decision Engine*. Besides the Netlet properties, the Decision Engine has to take several other parameters into account, e. g. user, network or administrator policies for the network selection. In addition, the Netlet Selection has to check, if the desired remote node or object is reachable via a specific Netlet. To obtain this information, the *Name and Address Mapper* has to be inquired. The Name and Address Mapper is responsible for any name resolution activities. More details about the Netlet Selection and Decision Process are presented in [1], [3].

Architecture-specific multiplexers allow to run multiple Netlets on a single Network Architecture. In order to have multiple different protocols running on parallel over the same network, streams produced by different Netlets need to be separated. An example in today's IP networks would be the use of Upper Layer Protocol IDs, e. g. to identify UDP and TCP. This example also shows that there might be a need for a common header field, and thus a common basic understanding of a PDU format is necessary. Since this would already be an invariant within a network architecture, we allow for multiplexers that differ from architecture to architecture – this avoids invariants across architectures.

A *Network Access* is an interface to the actual network. It may be compared to today's network interface, but it is intended as a more flexible abstraction for any type of networks, e. g. for physical networks as well as for virtual ones. The *Network Access Manager* can be seen as a registry where the different Network Accesses register and where the mapping to the network architecture running on the attached networks is done.

The *Tuning and Optimization Agent* constantly monitors the conditions of the Netlets, the network, and the applications. If changes in the conditions occur, it tunes configuration parameters to adapt the Netlet as good as possible to the new conditions. To support the tuning, Netlets may have configuration parameters that allow to modify their behavior accordingly. 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 hand-over from a wired to a wireless connection type). The Tuning and Optimization Agent has therefore to interoperate with the management component.

Finally, the *Management* component provides network and node management facilities. It may access any of the other components described here to obtain information or to trigger tuning and configuration of them via the Tuning and Optimization Agent.

B. Netlets: Container for Future Internet protocol stacks

Three different types of Netlets can be distinguished:

- *Regular Netlets* are used to implement communication protocols an application uses to communicate with another. One Netlet could hold, for example, a protocol stack comprising HTTP, TCP, and IP, so that it can be used to browse websites.
- *Control Netlets* commonly are used for control protocols. A routing protocol, for instance, would be implemented as a Control Netlet. Control Netlets allow us to structure protocol stacks using different Netlets: the regular Netlets for the application protocols and the Control Netlets for the control protocols.
- *Interop Netlets* are a combination of two Netlets and implement interoperability functions. They are used to connect different networks possibly containing different network architectures. Therefore, Interop Netlets may need to implement for example address mapping, protocol translation, or even content transcoding. Since interoperability between networks is a very complex, but interesting subject in itself, it is not covered further in this paper.

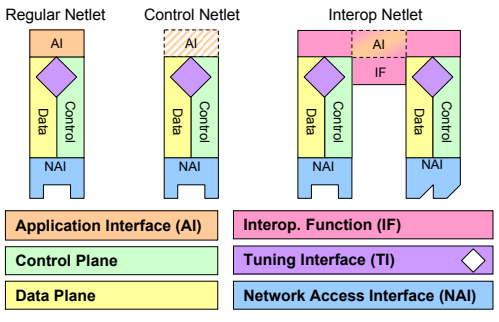

Figure 2. The types of Netlets

The interfaces of a Netlet need to be well defined in order to be able to easily exchange Netlets. A regular Netlet (left in Figure 2) has a set of important interfaces:

- *The Application Interface (AI)* is used to connect the Netlet to applications.
- *The Network Access Interface (NAI)* is used to connect Netlets to Network Accesses.
- *The Tuning Interface (TI)* is depicted as a diamond in Figure 2 and is used to adjust the Netlet and its behavior at runtime. This allows the Netlet to adapt to changed requirements of the application and changes in the underlying network. For instance, if the bit error rate of the network increases, the error control could be adjusted to better cope with these changed conditions.

Regular Netlets are structured internally into a control and data plane. This is based on [4], which distinguishes between tightly bound mechanisms (Data) and loosely bound mechanisms (Control) of a protocol. Control mechanisms not at all related to data streams (e. g. routing) are commonly realized as separate Control Netlets and not within the control plane of a regular Netlet.

The implementation of a Netlet itself may be of 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. While we do not require a specific design for a Netlet, we are currently aiming to provide design methods for a structured, verifiable composition approach for protocols. Besides that, it is also possible to implement existing approaches, like e. g. RNA [5] and SILO [6] as Netlets.

C. A new Application Interface

Today's applications commonly use the socket API or a similar interface to communicate. To communicate with another application entity in the network, the application has to supply the address of the node and the address of the service it wants to reach, e. g. port 80 for HTTP. Furthermore, the application has to make a selection of the transport service to be used, i. e., UDP or TCP in order to demand either a best-effort or a reliable transport service. For example, if an application wants to access the web data of *http://www.internal.tld*, it has to contact e. g. the host 10.11.12.13 using the TCP protocol and port 80. In order to be able to provide this information, the application has to trigger a name resolution to obtain the address information related to the URL (commonly using DNS via a resolver library).

It becomes obvious that such a behavior is not suited for a node connected to 1000 networks with possibly different addressing schemes. The decision of which address a name is resolved to may have major consequences on the possible networks the system can select. Looking at IPv4/IPv6 stacks today, the common strategy is simple: if the name resolution returns an IPv6 and an IPv4 address, just try IPv6 first and IPv4 only if IPv6 fails. This strategy is an obvious choice for promoting usage of IPv6, but might not be actually what is best for the user. The path IPv6 takes through the network might be worse than the IPv4 path. For example, the IPv6 path could be much longer and introduces additional delay. So for latency critical applications, the IPv6 path might be unacceptable, while the IPv4 path would have been good enough.

With the vision of 1000 networks in the future, such a strategy should be replaced. A strict order of networks would not lead to efficient usage of networks based on the properties of them. Therefore, the decision which network should be used, has to be made by the system based on the requirements provided by the application and the properties of the underlying network.

This leads us to the following changes with respect to traditional application interfaces:

- Address by Name (and not only by address)
- Transport of application (and user) requirements

Address by Name will allow the application to just pass the name of the host, service, or information it wants to reach and to let the system take care of the next steps. Although this implies a global naming scheme across all network architectures, this also opens up opportunities for novel addressing schemes. One example of such an approach would be addressing information directly in a search engine like but distributed fashion. For such a global naming scheme, we envision a scheme similar to today's URI scheme. Any other hierarchical naming scheme would fit as well, of course. For name resolution, the Name & Address Mapper (compare to Figure 1) will query possible Netlets in order to obtain knowledge if a remote host or object given by a name can be resolved in a specific architecture – a vital information for the Netlet selection process. This ensures that novel approaches can be implemented inside Netlets and therefore can be easily used by any existing application without the need to modify these applications.

Application Requirements must be transported by the application interface, if the application wants to be able to affect the selection of the network. The application could, for example, require an encrypted and authenticated transmission. Thus, the application is relieved from protocol specific expertise.

D. A new Network Interface

When running 1000 networks in parallel, the common case will no longer be that a system is connected only to a single physical network. More likely, it will have connectivity to several virtual (and physical) networks. Today's abstraction of the network interface is based on physical network interfaces only and on systems with a small number of connected networks.

With the availability of 1000 virtual networks it is time to rethink the network interface. The redesigned interface should transparently support network virtualization and supply more information about the underlying network. For selecting the best Netlet for a communication association, we need information concerning the underlying network, e. g. with respect to latency, packet loss, and energy consumption. Therefore, the network interface needs to be able to supply such information. We call this new interface *Network Access* (NA).

The generic NA can be specialized by different types. For instance, a physical NA is used for physical networks, whereas

a virtual NA is used for virtual networks, respectively. While most of the interface functions and provided properties are identical for both (and therefore located in the generic NA), they slightly differ in their management and control functions, e. g. send power control for physical NAs and virtual network setup functions for virtual NAs.

E. Connecting Netlets and Network Accesses

When a Netlet is instantiated, it connects to the Netlet multiplexer for the network architecture it belongs to. The Netlet can then communicate via all Network Accesses that are connected to the multiplexer. NAs can only connect to a multiplexer that at least has a basic understanding of the network architecture running in the network to which the NA provides access (see Section II-A). Commonly, it only makes sense to use protocols in one network that are at least at a basic level compatible to each other, so that they can run side-by-side and that they can be identified and separated from one another. Since protocols are part of the network architecture, one can even say that Netlets have to fit to the network architecture of the network provided by the NA.

To determine the network architecture of a given NA, different methods can be used. These range from simple static solutions to auto-discovery mechanisms. Simple solutions include manual and application-based associations of NAs to a multiplexer. The manual setup is commonly done by a user or administrator and is useful for small networks with a limited number of nodes that only connect to a limited number of networks. However, the high amount of user control of this method yields to very low user friendliness. Alternatively, the application can specify via the requirements it has to provide to the Netlet Selector that a certain network and network architecture should be used. This can be very useful if applications come with their own specialized networks, e. g. for television streaming or video telephony.

Also auto-discovery of network architectures may be possible. Automatically determining the network architecture of a NA's network can be achieved by different approaches. One solution would be that the network signals which architecture it is using. For virtual networks, this signaling could be done inband or outband: outband signaling can be done within the management plane of the virtualization infrastructure, while inband signaling would run inside the network architecture. The latter would have the advantage that it can also be used by networks connecting directly via a physical NA.

Examples for inband signaling of the network architecture are the use of magic packets or frames, or the use of shim headers providing the required information about the architecture. Magic packets/frames are announcements PDUs, which can be listened for. These special PDUs have to be defined for (possibly) all network architectures and would be therefore an invariant for all network architectures. Hence, inband signaling of network architectures are not a viable overall solution, but could be advantageous for a family of network architectures designed to work side-by-side this way.

F. Netlet Creator and Netlet Repository

If none of the existing Netlets fulfills the requirements a application has for a communication association, new ones need to be instantiated. Also, if a node is connected to a new network running a different architecture, Control Netlets for e. g. routing information need to be instantiated. Therefore, the Netlet Creator shown in Figure 1 is an important part of our Node Architecture. The Netlet Creator instantiates the Netlets using a Netlet repository. This repository holds all available Netlets and can be filled with additional ones if the node must be able to connect to new networks running a yet unknown network architecture. In order to fill this repository with Netlets for instantiation, we are also developing a design process, which allows to develop Netlets and virtual networks in a very short time [1].

III. A Netlet Example

In order to concretize the interior of Netlets, we sketch a brief example for a Netlet that provides reliable transport functionality. The Netlet is depicted in Figure 3.

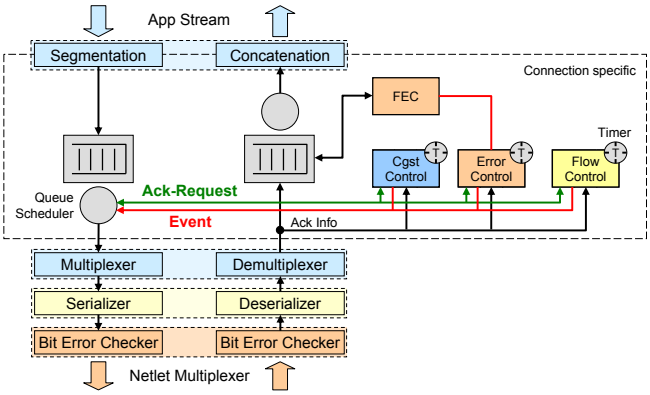

Figure 3. Example Netlet for Reliable Transport

There are basically two main data paths, one for outgoing data on the left and one for incoming data on the right. Most operations done for outgoing data have its counter parts at the remote node for incoming data. Therefore, building blocks realizing such operations are grouped in the figure.

At the top, the application data is passed from and to the Netlet via the Application Interface. Since this example is inspired by TCP, we also provide a stream-oriented interface for this Netlet. Therefore, as a first step, we need to segment the application stream that is to be sent. Likewise, received data needs to be concatenated again in order to deliver a stream to the application. At the bottom, incoming data as received for instance directly from the Netlet multiplexer is first checked for bit errors. The serializer (resp. deserializer) block is responsible for marshaling (resp. unmarshaling) the data and header information to be sent (resp. received). The multiplexer above the serializer does connection multiplexing, e. g. with port numbers as used in TCP.

Everything above this within the dashed box in Figure 3 has connection specific state. The central blocks here are the ordered queues for incoming and outgoing data with their respective queue schedulers. While the queue scheduler of the

incoming data queue is rather simple (i. e. pass all complete and correct data to the application if it is ready to receive it), the outgoing queue scheduler is a bit more complex because of congestion, flow, and error control. On one hand, the scheduler has to request for permission to send data (congestion and flow control), and on the other hand, the scheduler has to ask for which data unit is next to be sent (retransmission control). To avoid polling of these information, we are using an event mechanism, where the congestion, flow, and error control blocks trigger events upon incoming ACK information or timer events. The send queue scheduler can request ACK information from these control blocks to piggy-back it with the data to be sent.

Eventually, we added a new building block that is not found in regular TCP: a forward error correction (FEC) block. This block tries to repair lost information in the incoming packet queue with redundancy information sent with other packets.

This example here is missing a lot of functions found in TCP, like connection establishment and tear down, and detailed specifications of the building blocks and PDU formats. Further description is beyond the scope of this paper.

IV. Related Work

Most of the related work that is relevant in this context focusses on protocol composition or on network virtualization. On contrast to this, the presented Node Architecture is a general architecture for network nodes that enables the usage of a vast number of different network architectures on top of physical or virtual networks.

Composition approaches for communication protocols try to: (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, 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. Examples for such approaches are F-CSS [7], RNA [5], SILO [6], and x-Kernel [8]. 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.

Network virtualization allows to run several (isolated) networks in parallel over a single physical substrate [10]–[12]. A survey of network virtualization can be found in [13].

In parallel to our work, related ideas have been proposed in the context of Future Internet research [14], [15].

V. Conclusion and Future Work

In this paper we have presented a Node Architecture for the Future Internet, which allows for arbitrary, parallel network architectures on a single node and the smooth transition towards the Future Internet as well as the evolution beyond. We identified important properties of interfaces needed for such a Node Architecture and introduced the concept of Netlets as containers for current and future protocol stacks. Together with the generic Network Access presented here, we are ready to support virtualization which we consider will be the general case in a Future Internet. Current and future work includes the further refinement and evaluation of our concepts, as well as the refinement of a design process that eases the development of a multitude of future networks.

Acknowledgment

Parts of this research were carried out within the 4WARD project of the 7th EU 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 like to thank Peter Baumung and Helge Backhaus for valuable contributions to this paper.

References

[1] L. Völker, D. Martin, I. El Khayat, C. Werle, and M. Zitterbart, "An Architecture for Concurrent Future Networks", in *2nd GI/ITG KuVS Workshop on The Future Internet*. Karlsruhe, Germany: GI/ITG Kommunikation und Verteilte Systeme, Nov. 2008.

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

[3] L. Völker, D. Martin, C. Werle, M. Zitterbart, and I. E. Khayat, "Selecting Concurrent Network Architectures at Runtime", in *Proceedings of the IEEE International Conference on Communications (ICC 2009)*. Dresden, Deutschland: IEEE Computer Society, Jun. 2009, (to appear).

[4] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall International, Jan 2008.

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

[6] 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., 2007, pp. 1899–1904.

[7] 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.

[8] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols", *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, 1991.

[9] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture", *SIGCOMM Computer Communication Review*, 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] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time", *SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 61–64, 2007.

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

[13] N. M. K. Chowdhury and R. Boutaba, "A Survey of Network Virtualization", Technical Report, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Tech. Rep. CS-2008-25, Oct 2008.

[14] A. Keller, T. Hossmann, M. May, G. Bouabene, C. Jelger, and C. Tschudin, "A System Architecture for Evolving Protocol Stacks", in *17th International Conference on Computer Communications and Networks (ICCCN)*, Aug 2008.

[15] C. Vogt, "Towards a hostname-oriented network protocol stack for flexible addressing in a dynamic internet", Whitepaper, Nov 2008.