IEEE Copyright Notice

Selecting Concurrent Network Architectures at Runtime

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

***Abstract*—The current Internet architecture nicely structures functionality into layers of protocols. While this reduces complexity, many tweaks have emerged because of the architecture's limited flexibility. Cross Layer Functionality corrodes the layer boundaries, intermediate layers had to be introduced for protocols like MPLS and IPsec, and middleboxes – like in case of NAT – further complicate the interaction of protocols. To overcome these problems, many publications have proposed modular solutions or protocol composition, allowing software engineering ideas to improve protocol design. Other publications state that instead of choosing a single common network architecture for the Future Internet, it might be advantageous to run multiple different architectures in parallel. We combine both approaches and make it possible to rapidly create and run different network architectures in parallel. While this allows for simplified Future Internet development, it requires the network architecture to be dynamically chosen. This paper not only presents a node architecture enabling the parallel operation of different network architectures but also introduces algorithms for their selection at runtime.**

I. Introduction

The Future Internet will most certainly not be based on the proven yet old TCP/IP layer architecture anymore but on a more versatile network architecture that could be built with flexible protocol composition as presented in [1]–[5], or even on concurrent architectures and network virtualization [6], [7].

A more flexible network architecture and network virtualization lead to greater flexibility and extensibility for future development, especially when used in combination. For one communication association, the optimal solution might be statically composed of communication protocols similar to TCP/IP, but for another association, a dynamically composed protocol stack without layers could be the much better solution. Such a flexible solution, however, is inherently coupled with a problem: the system must determine the optimal set of network protocols and their configuration at runtime. When additionally using different approaches and virtualization, this becomes choosing the optimal instance of a network architecture at runtime.

For the remainder of this paper we call an instance of a network architecture Netlet. A Netlet implements protocols for a network architecture using one or more approaches. For example, one Netlet could implement today's TCP/IP using the RNA approach, while another implements a SILO with a network architecture addressing data instead of nodes.

To achieve the goal of choosing the optimal instance of a network architecture at runtime, a flexible and generic meta architecture and selection process is needed. Consequently, this paper presents our node architecture and selection algorithm, which were designed to achieve this.

Our node architecture is a generic framework to instantiate and run different Netlets of arbitrary network architectures and approaches. The node architecture comes with some modifications to today's application interfaces because several assumptions have changed. For example, today's IPv4 and IPv6 name resolution is commonly done by the application using a resolver library. This clearly moves the decision of the address and the address family, and thus the network architecture, to the application. Some future network architectures could even choose to not have address but route directly on a name. Consequently, it is advantageous to hide the name resolution from the application and move it to the system, so that the name resolution can be done in respect to the available Netlets.

The node architecture also transparently supports the concept of network virtualization by introducing an abstraction for the network access interface. A necessity when a large number of network architectures is used.

Our selection approach can evaluate different properties of Netlets to choose the Netlet with the optimal communication characteristics for the applications current needs. It allows trade-offs between different aspects, e.g., allowing to optimize for QoS, energy consumption, and security at the same time. When using it with an composition based approach, like SILO, the properties of a Netlet (or in this cas a SILO) must be calculated. This can be achieved using our property model and property aggregation.

The paper is structured as follows: After this introduction, we start with related work in Section II. In Section III the node architecture is presented as the basis for running concurrent network architectures. Subsequently, we present the needed algorithms and approaches to choose the "best" network architecture at runtime. Section IV presents how the ratings of different Netlets are compared in comparison with user/application requirements, while the property aggregation is detailed in Section V. The paper closes with Conclusion and Future work in Section VI.

II. Related Work

The SILO architecture [1], [8], [9] consists of composable *services* that realize fine-grained protocol functions like "end-to-end flow control". For each service, multiple implementations (*methods*) may exist. The services needed by an application are layered vertically in a *silo* and their order is

determined by a global ontology that describes precedence relations between them. SILO's focus is more on the tuning of a given SILO at runtime than trying to compose the optimal SILO for a given application. We conclude that SILO can easily be extended by our algorithm since our approach can evaluate the properties of a SILO and therefore calculate the utility of it. This allows the automatic selection of the "best" SILO available.

A flexible architecture for non-layered protocols is proposed by [2]. Protocols are replaced with roles, which have neither strict order nor defined headers in the data units. While this approach is more powerful and flexible, it does not elaborate on how requirements and restrictions can be imposed on protocol heaps except *sequence rules*. The main contribution of [2] is a different approach on packet headers, which can be complemented by choosing the "best" alternative.

The composition of fine-grained protocol functions or services to create application specific protocols is not a new approach. The Function-Based Communication Subsystem (F-CSS, [3]) allows the instantiation of *protocol machines* (PMs) for data streams with different requirements. Fine-grained protocol functions (PFs) may be implemented by multiple *protocol mechanisms* and are configured via protocol parameters. F-CSS defines an interface to the application, which can be used to specify service criteria. While quantitative service criteria (e. g. jitter, latency, rate) and qualitative criteria (e. g. stream synchronization, in-order delivery) influence the actual choice of the used protocol functions and mechanisms, possible side-effects on these criteria resulting from the composition of PFs are not fully considered. With the consideration of quantitative service criteria when selecting among different implementations, the composition algorithm of F-CSS went a bit further than SILO, but did not look into some properties, e. g. security. The F-CSS approach did look into composing a protocol stack and even used input from the application to influence that composition. Using our aggregation and weighting of properties, the changes of properties due to composition and the trade-offs can be reflected. Again, our approach could complement this solution.

RNA [4] attempts to avoid reuse of mechanisms on several layers by providing a single meta-protocol that is instantiated for every layer. The services these instances provide are configurable and thus can be tailored to the need of the upper and lower layer – this way, recapitulation of services is avoided and can be done at the most appropriate instance. The sequencing of fine-grained services follows best current practice and will be fixed within one layer instance. Our approach could also be used together with RNA, since RNA's current implementation is based on composition of modules, which will be turned on or off at instantiation.

The ANA System Architecture [10] focuses on an indirection interface called Information Dispatch Point (IDP). The smallest units of functionality (Functional Blocks, FBs) are interconnected via such an IDP. This allows for exchanging FBs very flexibly during runtime without disrupting the communication paths. The generic interface provided by the IDP could be easily extended to provide properties of the FBs they are currently connected to. Therefore, complementing the ANA Architecture with the concepts presented in this paper seems to be possible.

The presented approaches focus on the assembling of protocols. While some of them have foreseen the ability to assemble the protocols based on application or user requirements, most of them have no good solution for this yet. All of them, however, can generate a multitude of network stacks, which can be used for communication. In combination with our solution all these and possibly other approaches can run side-by-side and generate many possible network stacks from which our solution can choose. This way, even offline generation and online selection is possible allowing more thorough checking of the generated solutions.

III. The Node Architecture

The Node Architecture pictured in Figure 1 serves as the basis to support several network architectures concurrently. Together with the aggregation of properties and the selection algorithm presented in this paper, the dynamic selection of network architectures is achieved.

The Node Architecture focuses on the node-local modeling of a node's internals and how its functionalities relate to each other. Besides the interface definitions between applications, the communication components, and network interfaces, it provides the framework for the implementation of inter-working functionalities and protocols.

In this section, the general outline of the Node Architecture is described to set the context for this paper and the concepts detailed within. For an overview of the overall concept, refer to [11], for more detailed information about the Node Architecture refer to [12].

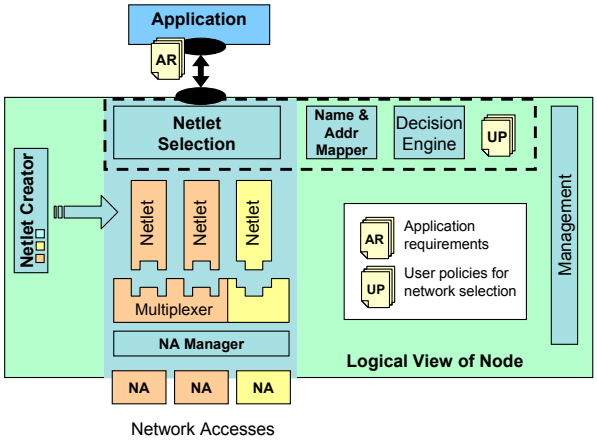

Figure 1. Node Architecture

The Node Architecture defines three fundamental abstractions, which differ from today's common architectures: The first one is the user application itself. In contrast to today's socket APIs, the application does not need to resolve names to obtain addresses, nor does it need to know which protocol it has to use – it just has to provide the desired properties of

the communication channel and, if applicable, the name of the remote object or node. The second one is the communication component called *Netlet*. Unlike today's protocol stacks, Netlets are flexible in their configurations and might even be exchanged during the life-time of a connection. In general, they bridge the gap between the application's requirements and the properties of the underlying networks. Access to the underlying networks is provided by the third fundamental abstraction, the *Network Access*. In contrast to today's network interfaces a Network Access allows any level of functionality already at the lowest level of abstraction: it might be a physical Network Access, a virtual one, or even one that already provides reliable and secure transport. These network properties may be communicated from the Network Access to the Node Architecture and influence the selection of Netlets.

Netlets are essentially containers for a collection of interworking protocols of a specific network architecture and for a specific task. This could include, e. g., a legacy TCP/IP stack or could be completely new architectures generated by approaches like SILO [1] or RNA [4]. A Netlet might be instantiated for one or more communication associations between a local application and a remote service. Some Netlets might also exist without an application association, which generally applies to network management or control functions, e. g., routing or distributed name resolution.

Unlike today's network interfaces, the Network Access (NA) is a clean interface to support attachment to any underlying network technology – this could include Ethernet, Bluetooth, or even virtual networks. In fact, this generic interface was designed to transparently allow for network virtualization. NAs may provide properties of the network they are connected to and will be attached to the Network Access Manager.

Netlets connected to the same network usually have some common understanding of the network. At a minimum, there needs to be a common understanding of the frame content and an identifier to allow multiplexing between the communication streams of different Netlets of the same architecture. To avoid choosing such invariants common to all architectures, we allow for architecture specific multiplexers, denoted in Figure 1 through different forms of Netlet bottom connectors.

The Node Architecture allows for running concurrent network architectures side-by-side – each architecture being implemented by a set of Netlets and a multiplexer. Functions needed across architecture borders like naming & addressing and management functionalities are supported as well, in an architecture independent way.

IV. The Selection Algorithm

The Selection Algorithm of the node architecture is based on the selection algorithm of Auto-Configuration for Communication Security (ACCS) [13], which we have designed to evaluate the effects and side-effects of adding security protocols to a TCP/IP stack. The selection algorithm compares different candidates and chooses the optimal one, based on user or application requirements. A candidate in the ACCS context is the combination of TCP/IP and one or more security ptotocols as wells as a configuration of that protocol. For the node architecture these candidates are more flexible, they are Netlets that are possibly composed of functional blocks.

When trying to find the optimum, it is important to find a trade-off between different properties of the candidates. Special care is needed for security, therefore, we have introduced a special security property in [13] called Effective Bit Strength (EBS), which will be also used for the new solution. The main idea behind EBS is that an encryption algorithm, for example, is only as strong as the easiest attack against it, even if that means attacking the key exchange.

In the first step of the evaluation of candidates, we filter all candidates based on system policy or user/application requirements. This means candidates that cannot fit the requirements will not be considered later on. The requirements are basically restrictions on the properties of the Netlet, e. g., the energy consumption must be below 100 J.

Based on these properties, we use a well-known decision-theoretic method, i. e., multi-attribute utility analysis (MAUT) [14], to aggregate the results of a candidate with respect to each single property into an overall ranking of each candidate. MAUT structures a complex decision process, which depends on multiple attributes, into a per-attribute utility evaluation. Therefore, for each attribute a utility function has to be defined which maps the value of an attribute to its corresponding utility. A simple exemplary utility function is shown in Figure 2. For valid attribute values, i. e., latency values smaller than the defined maximum, the utility increases with decreasing latency. The curve models the fact that smaller latency values are superlinearly prefered over bigger latency values. Subsequently, to get an overall ranking of the available candidates, the utility values of a candidate's attributes are aggregated using the weighted sum as aggregation function. By adjusting the weights, one determines the relative importance of attributes, i. e., the share they contribute to the final utility of a candidate. For n attributes a_i, n utility functions $u_i()$, and n weight parameters w_i where $1 \leq i \leq n$, the overall utility U of a candidate is then expressed by Equation 1.

$$U = \sum_{i=1}^{n} w_i \cdot u_i(a_i) \tag{1}$$

This overall rating is then calculated for each candidate and the highest-rated candidate is then chosen for the communication association.

This algorithm was not only designed to allow the node architecture's decision engine to choose between different Netlets at runtime, but it was also designed for performance. Our current implementation running on standard PC hardware achieves our goals with more than 1000 evaluations of connection request per second.

The determination of a Netlet's compound properties, which are used as the input into this process, will be presented next (in Section V).

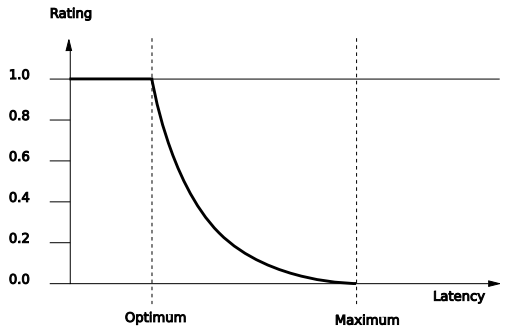

Figure 2. Example of a value function: Latency

V. Determining the Netlets' composite properties

As illustrated in Section III, a Netlet is composed of multiple functional blocks, each of which provides a certain service, e. g., realizing reliable data transfer or applying cryptographic functions. Thereby, each functional block influences the overall behavior of a Netlet. As it is possible that multiple Netlets are available, it is desirable to allow for comparison of Netlets with respect to the effects they cause, e. g., the increase in latency. Therefore, we need to derive a Netlet's overall behavior from the behavior of individual functional blocks, which we assume to be known. On the overall behavior of a Netlet, it is then possible to apply the decision mechanism presented in Section IV. Therefore, in this section we present a method to determine the overall behavior of a Netlet. For the remainder of this section, we will use the notation shown in Table I.

Table I
Notation

β_i	Functional Block i
Ψ_i	Effects of Functional Block i
Ξ	Global State
$\vec{\rho}$	Properties of the data stream
$\vec{\rho}_i$	Properties after Functional Block i
f	Transfer function

Referring to Figure 3, we will now explain our notation. Without loss of generality, assume that functional blocks β_i are ordered corresponding to their order of execution. The input to Netlet 1 is then processed by the functional blocks 1 to n, which affect the input while passing through with Ψ_1 to Ψ_n, respectively. We express the experienced effects of a functional block to the properties $\vec{\rho}$ using the transfer function shown as Equation 2.

$$\vec{\rho}_i = f(\vec{\rho}_{i-1}, \Psi_i, \Xi) \tag{2}$$

The transfer function is parameterized using Ψ_i which describes the effect of a functional block on the properties. The global state contains information about the node and its interfaces. The overall effects of a Netlet on the properties can then be described by Equation 3

$$\vec{\rho}_n = f(f(...(f(\vec{\rho}_0, \Psi_1, \Xi), ...), \Psi_{n-1}, \Xi), \Psi_n, \Xi) \tag{3}$$

where $\vec{\rho}_0$ corresponds to initial properties of the input. These initial properties also include a traffic profile of the application, i. e., a discrete packet size distribution.

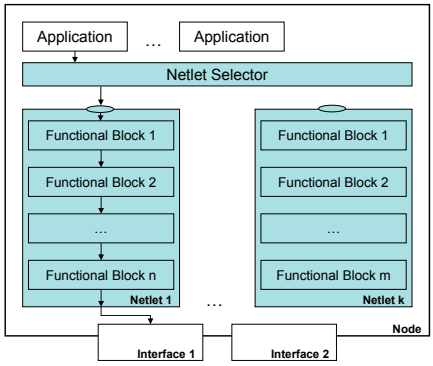

Figure 3. Concept Overview

After this short introduction into our notation, we will now focus on the modeling of sample properties and possible effects of functional blocks on these properties. Thus, showing how to determine the properties of a Netlet.

The first example will focus on latency of the data processing. Since *Latency* is cumulative, the latency of a Netlet can be estimated by summing up the Functional Blocks' individual latencies. So the effect on the property would be based on the add operation, as in Equation 4.

$$\rho_n^{la} = \rho_{n-1}^{la} + \Delta_n^{la} \tag{4}$$

Functional Block n will add Δ_n^{la} to the latency. This increment is commonly dependent on the hardware executing the Functional Block, which could be a CPU core or dedicated hardware. Furthermore, the latency added by a functional block might also depend on global state, like the current frequency of the execution unit, i. e. Δ_n^{la} depends on Ξ. The modeling of latency is very similar to that of energy consumption. The Functional Blocks also add Δ_n^{e} to the overall property and the energy consumption depends on the execution unit and, e. g., its current frequency.

Both energy consumption and latency also depend on the length of processed data. Considering the length of data units is very challenging because it often changes. We therefore choose length as the next example of a property.

Whenever an application sends data, the Netlet, through which the application communicates, might need to add additional data, e. g., for headers or address fields. This increases the *length* of the data. The most simple model for *length* could therefore just add a Δ:

$$\rho_i^{l} = \rho_{i-1}^{l} + \Delta_i^{l} \tag{5}$$

The amount of additional data Δ_i^l may, however, depend on the original size of the data transmitted. Therefore, the application has to specify a predicted communication behavior. Since most applications do not only communicate with a single data sizes, we use a traffic profile. The traffic profile specifies the distribution of a limited number of data sizes, representing

the expected traffic. Bulk data transfers, for example, try to transport a large amount of data in one direction and only transfer a small amount of control data in the opposite direction. When looking at the data units, it is clear that for one direction the probability of large data units is high and for smaller ones low, while for the opposite direction the probabilities are the opposite: a high probability for small data units and a small probability for large data units.

Another functional block might not just increase the length by a constant Δ_i^l but could increase the data length based on the original data length. Block-based cryptography, for instance, might need to pad the data before processing; effectively increasing the data to the nearest block size b:

$$\rho_i^l = \lceil \frac{\rho_{i-1}^l}{b} \rceil \cdot b \tag{6}$$

Also a decrease in *length* is possible, e.g., by using data compression. However, one must differentiate between two cases: predictable compression and unpredictable compression. In the first case we can predict the changes to the length. This is usually the case with header compression algorithms. The unpredictable compression, however, is more difficult to handle. Some data compression algorithms could even increase the length by adding meta data, when the data is uncompressable. While this can be modeled with a stochastic models, the negative effects of the unpredictability of data compression may be reduced by allowing the data compression only to be used on data streams and not on data units. Negative runtime effects of the increased data unit size, like today's fragmentation, can be also avoided this way.

The previous examples showed how the functional building blocks influence transported data. Energy Consumption and Length are just examples for influenced properties to present our approach. We have also looked at other properties, e.g., latency jitter, bandwidth, packet loss rate, bit error rate, and effective bit strength.

VI. Conclusion and Future Work

In this paper, we presented our architecture and approach to support running different network architectures in parallel and selecting the "best" at runtime. The node architecture allows running concurrent network architectures side-by-side. This is done by seamlessly integrating virtualization using a Network Access abstraction with a modified application interface, which allows late-binding name resolution. By using our model of properties, interactions of functional blocks, and a modified selection approach, the goal of selecting the "best" network architecture at runtime is achieved.

In addition, our system allows to react to changing situations by re-evaluating the properties. For example, changing the network protocols because of security issues or the current state of underlaying (virtual) networks is possible. Also, different approaches for building protocol stacks can be used in parallel. Choosing the "best" one makes these approaches compete for every connection.

Future work includes adding a model for the (virtualized) network between the nodes. The major challenge will be to examine how properties of the protocol building blocks influence the path through the network and how that in turn influences other properties.

We are in touch with members of the RNA and SILO projects to discuss and evaluate our approach using different architectures. Future work, therefore, includes integrating RNA's and SILO's approaches with our implementation and evaluating this combination.

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 Commision. We would like to thank our colleagues in the project for valuable discussions. In addition, the authors would especially like to thank Christoph Sorge, Oliver Waldhorst, and Peter Baumung for feedback while preparing this paper.

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet Impasse through Virtualization", *COMPUTER*, pp. 34–41, 2005.

[8] I. Baldine, M. Vellala, A. Wang, G. Rouskas, R. Dutta, and D. Stevenson, "A Unified Software Architecture to Enable Cross-Layer Design in the Future Internet", in *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN 2007)*, August 2007, pp. 26–32.

[9] D. Stevenson, R. Dutta, G. Rouskas, D. Reeves, and I. Baldine, "On the suitability of composable services for the assurable future internet", Whitepaper, RTI and NCSU, Tech. Rep., 2007.

[10] 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.

[11] 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, Deutschland: GI/ITG Kommunikation und Verteilte Systeme, Nov. 2008.

[12] L. Völker, D. Martin, I. E. Khayat, C. Werle, and M. Zitterbart, "A Node Architecture for 1000 Future Networks", in *Proceedings of the International Workshop on the Network of the Future 2009*. Dresden, Germany: IEEE, Jun. 2009, (to appear).

[13] L. Völker, C. Werle, and M. Zitterbart, "Decision Process for Automated Selection of Security Protocols", in *Proceedings of the 33nd IEEE Conference on Local Computer Networks (LCN 2008)*. Montreal, QB, Kanada: IEEE, Oct. 2008, pp. 223–229.

[14] R. Keeney and H. Raiffa, *Decisions with multiple objectives: Preferences and value tradeoffs*. J. Wiley, New York, 1976.