Towards a Property and Requirement-based Application Interface for Future Networks

Helge Backhaus
Institut für Telematik, Karlsruhe Institute of Technology (KIT), Germany
backhaus@tm.uka.de

I. Introduction

In today's Internet many different services and applications like VoIP, instant messaging, or online video exist. We assume that in the future even more services, each with different requirements, will emerge. Sensor networks and social networking are just two possible scenarios, which make new features like energy awareness and data privacy management more important.

Our Node Architecture [1] (Figure 1) is a framework to run a multitude of current and future protocols and networks side-by-side. Contrary to today's architecture, it hides the actual protocol-stack from the application and provides a generic application interface. Thus, complete protocol-stacks, encapsulated in so-called Netlets, can be exchanged without the need to modify or adapt the application. Netlets are generic protocol containers, which contain network protocols or even complete protocol-stacks, like today's TCP/IP-stack. They are always associated to a certain network architecture, and thus share architecture specific properties like message formats or addressing schemes, and a multiplexer. A Management component gathers and exposes information about the underlying (virtual) network infrastructure. A Network Access abstracts from different (virtual) media and provides a generic and extensible interface, offering detailed information about the underlying network.

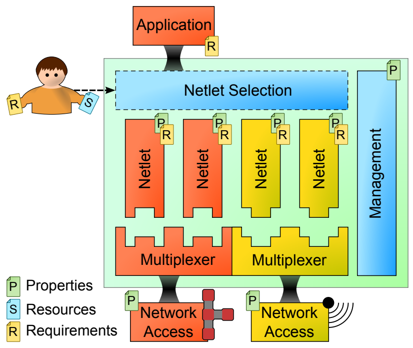

Figure 1. Simplified version of the Node Architecture

Today users expect communication to take place hassle-free and fast. Applications like Skype foster a *just works* mentality, hiding all the complex mechanisms from the user. This poses the challenge of providing an equal experience to the user in an environment, where several networks, each with its own application or service tailored protocols, exist side by side. Selecting the best available protocol for a certain task can become quite complex. Nevertheless protocol selection needs to be performed with a minimal user interaction. Therefore, (semi)automatic protocol selection is needed, which requires description languages for specifying protocol properties and application as well as user requirements.

In the following a concept of a flexible and extensible description language is presented, with a focus on the interfaces between applications, protocols, and users. The solution supports protocol designers, application developers, and users at specifying protocol properties and performance parameters. They are enabled to state their requirements on an underlying network infrastructure. While users gain full control over protocol selection, several automatisms and default settings ensure, that only minimal user interaction is needed in daily use.

II. Requirements and Properties

Figure 1 illustrates, where within the Node Architecture different requirements and properties are provided or needed for the selection process. Network Accesses expose technical properties, like maximum bandwidth or the medium used. A management component exposes further properties by monitoring links to other hosts or pulling information from an underlying virtual network. These properties are, e. g., the current packet loss rate, experienced minimum or maximum packet delay, or jitter on a specific link. Since Netlets replace today's protocols and protocol stack, they provide services like reliable transport, error detection and correction mechanisms, routing, or security. Netlets can also feature additional properties like data privacy, validity of delivered content, or access rights management for certain data.

Applications and users can state requirements which have to be met alongside a connection request. Netlets can also state requirements on certain system resources. A Netlet could require local disk space for storing data for instance. Ultimately, the user or system administrator has the control over system resources. He may decide which resources may be used by the Node Architecture or (groups of) Netlets. Netlets can request resources from the user on a per connection basis, but users are also able to specify default rules for different (groups of) Nelets or applications.

Application requirements have to be defined by the application's developer. They consist of a set of properties, which the application expects from the underlying Netlet and network to function properly.

User requirements are also a set of properties, which can be defined as a system wide policy for an application or per connection. A user can limit the bandwidth an application uses for instance. In the context of social networking, properties could include privacy preferences which define how long data may remain within a network, or who is allowed to access it. Further properties concerning content validity may be possible, e. g., networks that guarantee a trusted source for content instead of just guaranteeing non-manipulated data. If application and user requirements are conflicting with each other, user requirements override application requirements. This can however prevent the usage of certain applications.

For each connection request, a set of properties can be derived from application, user, and system wide defined requirements. To make the task of defining requirements easier, extensible profiles can be predefined for standard use cases. For application developers, profiles like WWW, P2P, multimedia data, or a generic reliable transport can be defined. Device manufacturers could set up predefined profiles for, e. g., mobile phones, which are tuned for low energy consumption.

III. Netlet Selection

The Netlet Selection process is shown in Figure 2. When an application wants to establish a connection, a set of properties alone is not sufficient. In most cases, there will be dependencies between at least some if not all properties. Therefore, constraints have to be defined against which properties can be checked. Constraints define functional dependencies between two or more properties. They can state simple dependencies, like two properties which can never be used together, but also more complex dependencies, like relations between two property values.

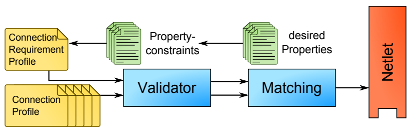

Figure 2. Netlet Selection via Connection Profiles matching

Property constraints and the set of desired properties are therefore combined into a Connection Requirement Profile. This is done with combined user and application requirements. Furthermore, a Connection Profile for each Netlet has to be created. Constraints have to be predefined for every property combination which has dependencies, and can later be automatically derived from the set of properties. Within the Validator, constraints are then applied to the properties. If constraints can not be fulfilled, a list of properties which do not work together is sent to the application. The application then has to modify its requirements where feasible. Alternatively, the decision which requirements have to be modified can be passed to the user or an error can be shown to inform the user that no connection could be established.

Once all constraints can be fulfilled, a Connection Profile is considered to be valid. Finding the best Netlet available for a Connection Requirement Profile is then done by matching the property-sets of all available Connection Profiles against it. A Netlet, whose Connection Profile overlaps completely with the Connection Requirement Profile is chosen. If several Netlets fulfill the requirements, additional properties can be specified or default behaviour, like choosing the first Netlet that fits can be set. If no overlapping Connection Profile can be found, the non-satisfiable properties are sent back to the application, and the same process is triggered as with non-satisfiable constraints.

Connection Profiles are split up into properties and property constraints, to allow for a flexible extension of the system. As constraints are not mandatory, new properties can be added without the need to create several constraints alongside of them. It is the responsibility of the application designer then, to define feasible requirements. This allows for properties that would otherwise be hard to verify. Also constraints can be added at a later time. Another advantage in relation to performace is, that a given set of properties and constraints can be pre-checked for validity and only a few checks or just a matching of properties is needed at run time, when a connection should be established. Finally new properties and constraints can be negotiated via the Management component and a signaling protocol to adjust application requirements between two or more hosts.

IV. Property and Property Constraints Format

Properties are currently represented in XML in an RDF-based [2] format, so that they are easy to extend and modify. In contrast to existing solutions like *WSDL* [3] or *OWL* [4] we propose a lightweight format, which is portable to different existing languages.

Properties are identified via a *PropertyName*, which is a unique resource identifier (URI) to avoid ambiguities. A *Description* provides a human readable explanation of the property, which can be displayed to the user, if requirements have to be altered. A *ValueComperator* defines how this property's value has to be compared during the matching phase. At the moment, we allow comparators like *greater*, *smaller*, *equal*, or *none*. The *none* comparator can be used for properties, which have no value and just need to be present. A *ValueFormat* parameter sets, whether the property *Value* is an integer, float, or per cent value for instance. Property constraints can be defined in a similar manner. Constraints identify two properties by their *PropertyName*. Dependencies between these two properties can be formulated with comparators as well. For example the *Value* of the *minimumBandwithProperty* always has to be smaller than or equal to the *Value* of the *maximumBandwithProperty*. Additionally dependencies between properties itself can be specified, with *and*, *or*, *xor* relationships.

V. Conclusion

We presented a concept for a flexible and extensible solution, to define requirements and properties for network protocols and applications in networks. The solution aims at recreating the straightforward and hassle-free experience users have come to expect from today's Internet, in an environment where several protocols and network architectures are running concurrently. We achieve flexibility by using an RDF-based format, which is portable and can be converted to other formats. Extensibility is supported by a design, which fosters the addition of new properties and property constraints step-by-step. In a next step we want to investigate how our approach can aid in finding the best protocol, in cases where several protocols satisfy requirements equally well. Therefore we want to extend a solution initially developed for selecting, e. g. security mechanisms [5] to incorporate our properties and requirements language.

References

[1] L. Völker, D. Martin, I. El 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.

[2] W3C Recommendation, “Resource Description Framework (RDF): Concepts and Abstract Syntax”, 2004, available at http://www.w3.org/TR/rdf-concepts/.

[3] W3C Recommendation, “Web Services Description Language (WSDL) 1.1”, 2001, available at http://www.w3.org/TR/wsdl.html.

[4] W3C Recommendation, “OWL Web Ontology Language Semantics and Abstract Syntax”, 2004, available at http://www.w3.org/TR/owl-semantics/.

[5] L. Völker, D. Martin, C. Werle, M. Zitterbart, and I. El Khayat, “Selecting Concurrent Network Architectures at Runtime”, in *Proceedings of the IEEE International Conference on Communications (ICC 2009)*. Dresden, Germany: IEEE Computer Society, Jun. 2009.