

# An Approach for an Evolvable Future Internet Architecture

Bernd Reuther and Joachim Götze

University of Kaiserslautern, Germany

**Abstract.** The development of the Internet during the last years has shown that it is difficult to integrate new functionality. Especially the core mechanisms (TCP/IP) are hard to change. We consider the tight coupling between several functionalities to be the main reason for this problem. In this work we present a clean-slate approach for a new loosely coupled network architecture. To achieve this, we adopt concepts of service-oriented architectures. We also briefly discuss methods for handling heterogeneity which is more likely in such a loosely coupled environment.

## 1 Motivation

While new transport technologies and new applications are constantly developed and introduced to the market, the core functionality of the current Internet is hard to change. There are alternatives to the protocols of the common TCP/IP suite, for example IPv6, DCCP or SCTP. Even though these protocols are offering several advantages compared to TCP/IP, the new protocols are used rarely. The inability to integrate new protocols is caused by the growing coupling of functionality. Among others this coupling is fostered by layer violations (e.g. cross-layer design) and erosion of the end-to-end model (middle-boxes, such as firewalls, NATs, proxies, caches, etc.). For example TCP and HTTP were designed as end-to-end protocols, but current practice is that port numbers of TCP and UDP are used by many network devices and HTTP is recognized or even interrupted by firewalls and proxies.

The problem of tightly coupled functionalities is not related to specific protocols or mechanisms of the current Internet but is mainly caused by the architecture of the Internet<sup>1</sup>. The main elements of the original Internet architecture are layers which cover a lot of functionality today. The Internet architecture does not specify internal structures of layers (i.e. layers are like black boxes) and the only evolutionary principle given is to exchange complete layers. Without a proper predefined method for introducing new functionality into the Internet manifold specific solutions have been used, which in turn increase complexity. As a consequence the architecture of a Future Internet should provide improved

---

<sup>1</sup> In the context of this paper we define an architecture according to [1] as the fundamental organization of a system, the relationship of components as well as the design and evolution principles (see [2] for further discussion).

evolutionary principles enabling to add, change and remove fine grained functionality to and from the network.

In our ongoing research we investigate concepts that should facilitate enhanced flexibility for a Future Internet. Our approach addresses two problems: 1) Within network nodes, it must be possible to change individual functionalities without changing (many) others. Therefore we adopt concepts from software engineering – especially service-oriented architectures (SOA) – to achieve loose coupling among functionalities. 2) Heterogeneity between network nodes must be handled. Being able to change functionality on individual nodes will inevitably lead to heterogeneity of functionality especially in large networks. This is a new problem, because currently all nodes in the Internet must support the protocols of the TCP/IP suite. We believe that solving both problems will be the key for a flexible network architecture that is able to evolve. The basics of our approach are described in section 2 and possible solutions to handle heterogeneity of network nodes are discussed in section 3. The paper concludes with an abstract view on the presented approach.

## 2 Basic Concepts

In order to be able to change functionality of a network node, the functionality within the node should be loosely coupled. Our approach is to apply concepts of service-oriented architectures on the network level to achieve this. In the following we provide a brief overview of such concepts. We have implemented these ideas in a first prototype and will use it for experimental implementations of network functionality soon.

*Services* Services are the essential building blocks of a SOA. A service provides self-contained functionality, has well-defined interfaces and must not make assumptions about internals of other services (loose coupling). In this context a service encapsulates some network functionality, for example a micro protocol or an agent providing name resolution functionality. Instead of focusing on specific algorithms and data structures, the interface of a service should reflect the benefit for the user and that is why it is called a service. For example, there may be a service for ensuring reliable transmission which can be implemented by different retransmission protocols or a routing loop avoidance service could be implemented by inserting TTL values in the data stream or might be ensured during connection setup.

*Service Interaction* To provide complex functionalities different services within a network node need to interact, but services should not call or exchange data directly with each other. Services should create notifications instead of calling another service. A notification broker will delegate notifications to all services that have registered for a notification or notification class. For example a queuing service that detects the lack of memory resources may then publish a notification. The notification broker can forward this notification to a flow control service and

to a resource monitoring service. This way the queuing service needs not to be aware that there is also a resource monitor in addition to a flow control service. Services exchange data indirectly using a tuple space (a set of <name,value>-pairs) enabling temporal and referential decoupling of services [3]. For example, a routing service may provide information about an outgoing port without having knowledge when or which service may read that information from the tuple space. All status information of a service is kept in the tuple space so that each service is stateless. This fosters the autonomy of services and improves their stability.

*Messages* Basically all communication between network nodes is implemented by message exchange between instances of the same service on different nodes. To achieve a high degree of flexibility it should be possible to reorder messages<sup>2</sup> and it should be possible that the length of messages varies. On that condition it seems naturally to use a TLV (Type, Length, Value) like format for messages.

*Flows* The term “flow” usually denotes a sequence of packets which are semantically related to each other. Within our approach, a flow denotes a sequence of semantically related messages. Flows are separated by specific flow-ident messages. It is assumed that all messages from the same source belong to the same flow until a new flow-ident message is received. Thus on unreliable links, each data unit (e.g. packet or frame) must begin with a flow-ident message to ensure that messages of different flows are not interleaved. The value of a flow-ident message can contain arbitrary data types, typically some kind of address or label. Such a flow identifier is valid only on one link. Nodes sharing the same link are free to change flow identifiers as needed.

If two services process messages belonging to the same flow, then these services share the same tuple space, i.e. these services can exchange data. Further notifications can be related to a flow. Services receiving such a notification also gain access to the tuple space of that flow. Thus a flow defines a context for services.

Note that the concept of flows does not imply a connection-oriented behavior. Connection-less behavior (i.e. using stateless protocols) is implemented by deleting a tuple space (i.e. the flows context) whenever a new flow-ident message is received, meaning the context is transient. Connection-oriented behavior (i.e. using stateful protocols) is implemented by storing the tuple space for later reuse whenever a new flow-ident message is received, meaning the context is persistent.

*Channels* Channels represent links between two or more nodes. A channel can be provided physically by a local accessible transport technology or a channel can be a virtual link provided by an existing flow. Embedding a flow into another flow enables reuse of functionality similar to layering in the OSI model. But in contrast to OSI layers, the functionality of each flow can be defined as needed.

---

<sup>2</sup> Reordering may be limited by dependencies between messages, e.g. it is preposterous to send payload messages before sending a destination address.

A channel should be able to transport messages of different flows. Therefore (de-)multiplexers are required. A multiplexer should aggregate messages of the same flow<sup>3</sup> and send them to the channel. Messages of different flows are separated by a flow-ident message as described above. A demultiplexer forwards messages to an appropriate dispatcher for processing.

*Dispatchers* The message and event processing for each flow is controlled by dispatchers. Dispatchers receive messages from demultiplexers and notifications from the notification-broker and forwards them to appropriate services. The task of a dispatcher is to control the sequence of message and event processing. Thus a dispatcher corresponds to a workflow manager in a SOA. A trivial dispatcher can process messages sequentially. But independent services might run in parallel. In case of known dependencies, a dispatcher might even reorder messages. Such dependencies might be specified explicitly by an administrator or may be derived from read and write accesses to the tuple space<sup>4</sup>. Note that even the dispatcher does not need to interpret messages and does not have knowledge about the semantic of services. Messages and notifications carry identifiers which are used by a dispatcher to assign them to registered services.

*Nodes* In our approach a network consists of linked nodes. From the network's point of view applications<sup>5</sup> are also considered to be nodes. Thus the interface to an application is also a channel. Of course there should be tools to hide the complexity of interacting services for application developers, for example by offering abstract data transport services to applications [4]. The purpose of a node is to provide a framework for services on a network level, i.e. the basic concepts described here. Within a physical device multiple nodes can be used to separate classes of services from each other, e.g. one node may provide basic communication functionality and another node may handle specific or even experimental services used in overlay networks.

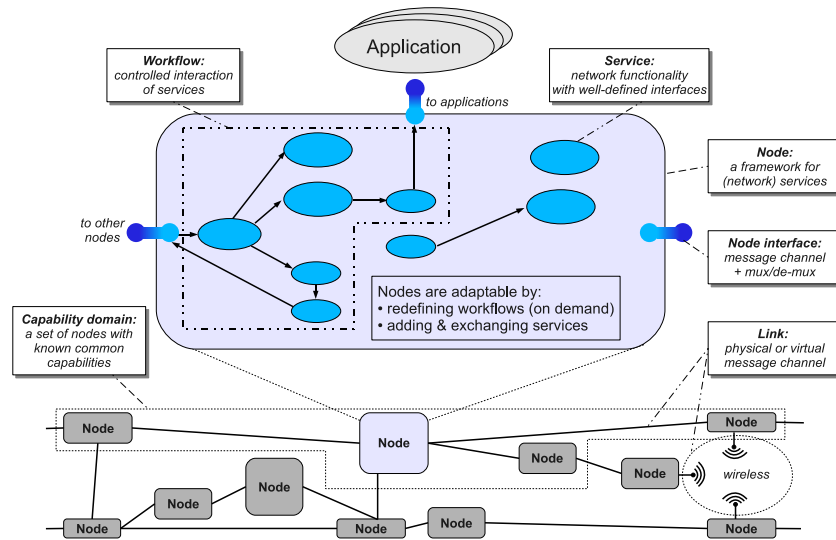
### 3 Discussion of Handling Heterogeneity

Communication in general requires a common basis or language; in case of computer networks common protocols are required. Thus different instances of the same service on different nodes have to use the same protocol. But changing and introducing services in a large scale network will inevitably lead to a heterogeneous network according to the available protocols. From the point of view of a single node, heterogeneity causes an uncertainty about the protocols that are supported by other nodes. A node can basically handle this in two ways: handling uncertainty dynamically at runtime or removing uncertainty in advance of communication:

<sup>3</sup> In our prototype the aggregation of messages is performed more efficiently by the dispatcher.

<sup>4</sup> Two services writing and reading the same element in the tuple space have a producer/consumer relationship.

<sup>5</sup> More precisely any kind of process on endsystem



**Fig. 1.** Basic concept of a service-oriented approach for a Future Internet architecture

(A) Handling uncertainty at runtime can be done at least by three different methods. (1) For some protocols it is appropriate if these are just ignored by nodes which do not support these protocols. For example, having flow-control support within the network fosters efficiency, but it is not necessary that all nodes support flow-control mechanisms. (2) If there is uncertainty only about few alternatives then it might be possible to offer all alternatives. An example is to use two address types for the same destination. (3) Finally, it is possible to delegate the processing to another node which is able to handle the protocol or to inform the originating host to change its behavior.

(B) The second way is to remove uncertainty in advance of communication, which in turn requires communication between two or more nodes. Nodes may negotiate their capabilities with each other or may consult a registry immediately before user data will be exchanged. This causes a delay before communication can start, even if caching techniques are used. Thus these methods should be used rarely, e.g. when using highly specific protocols. Nodes that interact often could decide to negotiate their capabilities in advance. Such nodes can build capability domains to share information about common capabilities (see figure 1). A node may be a member of several capability domains covering different types of capabilities. For example a set of nodes may share the capability of supporting specific address types. Capability domains could be defined statically or dynamically. The latter requires distribution of information. This might be implemented similarly to routing protocols and will also produce overhead by additional communication.

Several recent research projects propose the definition of overlays networks, which are tailored for specific tasks or application classes (for example ANA [5] and SpovNet [6]). In these approaches a node can participate in an overlay network only if it offers all functionality required for that overlay. This way uncertainty is also removed in advanced by managing who can participate in an overlay network. The differences between defining specific overlays (a) and defining capability domains (b) are: there will be many different networks with more or less fixed functionalities (a) or there will be one (or few) networks with many different functionalities (b). The advantages and disadvantages of both approaches are considered an open issue.

There are other reasons for heterogeneity than varying sets of functionalities, which might require different solutions. For example nodes may be different according to the available resources. This could be handled by QoS mechanism if required. Further there may be network users with different and contradictory intentions (see [7]). This might be solved by offering disjoint (virtual) networks.

## 4 Conclusion

The goal of our approach is to define a network architecture that is able to evolve. We favor a clean-slate approach therefore we do not have to consider limitations of the current Internet. We expect that the key for such an architecture is the ability (1) to change functionality locally within a node and (2) to handle the inevitably arising heterogeneity of network nodes. We have identified basic concepts to solve the first problem. The applicability of these concepts will be validated soon. Methods for handling heterogeneous network nodes are considered to be an open issue. From our point of view mechanisms for maintaining capability domains are especially required.

Parts of the concepts presented here are not new and can be found in recent and older research work. For example in the context of configurable protocols (e.g. [8] or [9]), active networks [10], service-oriented applications [12], [11] and some recent clean-slate research work (e.g. [13] [14]). What is new is the combination of concepts and the mechanisms used to implement them.

Central to our approach is the concept of self-contained services on a network level which contain micro-protocols or agents. From an architectural point of view such services replace the layers of the OSI model: an n-layer interacts with the layer above and below, whereas a service may interact with any other service. This interaction is supported by tuple spaces and a notification-broker to ensure loose coupling and is controlled by a dispatcher which acts as a workflow manager. Services logically communicate with instances of the same service on another node, which is comparable to layers which also logically communicate with the same layer on other nodes.

The presented approach is the first step toward a future Internet architecture and will be the basis for further investigations.

## References

1. IEEE Std. 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems -Description.
2. How Do You Define Software Architecture? *Software Engineering Institute, Carnegie Mellon* <http://www.sei.cmu.edu/architecture/definitions.html>
3. D. Gelernter: Generative communication in Linda. In: *ACM Transactions on Programming Languages and Systems*, volume 7, number 1, January 1985
4. B. Reuther, D. Henrici: A Model for Service-Oriented Communication Systems *Journal of Systems Architecture*, Vol 54/6 pp 594-606, June 2008
5. ANA: Autonomic Network Architecture <http://www.ana-project.org/>
6. SpovNet: Spontaneous Virtual Networks <http://www.spovnet.de/>
7. D. Clark, J. Wroslawski, K. Sollins, R. Braden: Tussle in Cyberspace: Defining Tomorrow's Internet. *ACM SIGCOMM*, August 2002
8. S. W. OMalley, L. L. Peterson: A Dynamic Network Architecture. In *ACM Transactions on Computer Systems*, Vol 10, No 2, May 1992, Pages 110-143.
9. M. Zitterbart, B. Stiller, A. N. Tantawy: A Model for Flexible High-Performance Communication Subsystems. In: *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 4, May 1993
10. D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden: A survey of active network research. In: *IEEE Communications Magazine*, Jan 1997, Volume: 35, Issue: 1
11. T. Erl: Service-Oriented Architecture Concepts, Technology, and Design. *Prentice Hall, 2005*
12. OASIS Reference Model for Service Oriented Architecture 1.0, Official OASIS Standard, Oct. 12, 2006
13. R. Braden, T. Faber, M. Handley: From Protocol Stack to Protocol Heap - Role-Based Architecture. In: *Proceedings of the First Workshop on Hot Topics in Networking (Hotnets-I)*, *ACM SIGCOMM*, Princeton, NJ., October 2002.
14. R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, D. Stevenson: The SILO Architecture for Services Integration, control, and Optimization for the Future Internet. In: *IEEE International Conference on Communications, ICC* 2007.