Towards Dynamic Protocol Configuration and its Configuration and Control in Autonomous Communication Environments

David Wagner and Jens Mödeker

Fraunhofer FOKUS, Germany
{david.wagner,jens.moedeker}@fokus.fraunhofer.de

Abstract. The ability to dynamically adopt protocol functionality to the current situation will greatly improve performance, service availability and quality of service in the Future Internet. Nevertheless this provides a huge set of configuration and adaption options which have to be controlled intelligently and also in a timely manner. This creates need for a multi-level control and configuration framework. This paper presents a architecture for Dynamic Protocol Composition with some experimentation results and the design of a control and configuration framework that allows for efficient and reactive Protocol Composition and Execution while providing many options for configuration for a cognitive system and network management.

Keywords: Future Internet, dynamic protocol composition, cognitive network management, autonomic communication, communication protocols.

1 Motivation

The Internet as we know it is the result of thoughtful design that was supported by a repeated pattern of implementation and testing in the 1970's [1]. The fathers of the Internet decided to design the Internet Protocol as a narrow waist of oblivious datagram forwarding that avoids any connection state within the intermediate switching nodes. This decision paved the way to the Internet's success because it allowed the development of diverse novel applications and services on top of this very basic foundation. They also decided to protect state information from loss by gathering it at the endpoints of the net which are utilising the service of the network. Although this approach protects transport sessions against any number of intermediate failures, the limits and drawbacks of this design are apparent in today's networks: On one hand the simple mechanisms and assumptions of the Internet Protocol are being softened or even given up in many places, just think of middle boxes like firewalls, NAT-routers etc., VPNs, application proxies and so on. On the other hand devices, networks and service have become very heterogeneous, ranging from error-prone wireless sensor networks (WSNs) to multi-gigabit fibre-based storage area networks (SANs) but

P. Chatzimisios et al. (Eds.): MobiLight 2010, LNICST 45, pp. 334–345, 2010.

today's protocols don't consider the special features or liabilities of the actual link. It becomes more and more apparent that today's Internet protocols, namely Transmission Control Protocol (TCP) [2] and User Datagram Protocol (UDP) [3], fail to offer a service that is acceptably close to the best possible service based on the optimal configuration of protocol functionality on each link on the path.

The rise of Autonomous Communication (AC) opens new opportunities because it allows the network management to make complex and fine-grained decisions in a cognitive manner. We identified two major subjects that would allow overcoming current limitations in a transparent and evolutionary way: First, protocol functionality should be managed in a dynamic way that allows taking into account changing context. Second, we propose to give up the end-to-end principle and to establish protocol functionality in intermediate nodes in the network. The management of functionality and soft state in the network seems to be feasible in the Future Internet using the intelligent features of Cognitive Network Management. Nevertheless completely new control and configuration architectures are needed that are able to cope with the very different requirements and situations.

2 Research on Dynamic Composition of Communication Protocols

The idea to compose protocol functionality according to requirements attracted Internet researchers very early in the 1990's and since has been researched for different motivations, with varying focuses and assumptions. In the beginning frameworks like the early x-kernel [4] have been developed which allowed building static protocols according to the actual needs. These protocols had to be developed, compiled and distributed in a static manner and did not allow for runtime changes. With time, the proposed architectures got more dynamic, proposals like DaCaPo [5], Protocol Boosters [6] or DiPS/CUPS [7] allow dynamic runtime reconfiguration of the protocol stack, CUPS even allows hot swapping of components [8]. The focus of the DaCaPo and DiPS/CUPS architectures is still on the end systems whereas the authors of the Protocol Boosters proposal already described the deployment of additional protocol functionality within the network. Nevertheless there were no results published and this idea was not further pursued. Although the authors expected protocol boosters to evolve fast because their transparency with respect to the end nodes they still assumed homogeneous deployment within an administrative domain.

3 Our Concept of Dynamic Protocol Composition

We think that Dynamic Protocol Composition Frameworks (DPCFs) can be designed to allow for compatible interoperation with legacy protocols, namely the IP-based protocols, while keeping the ability to deploy more elaborated functionality with other Dynamic Protocol Composition (DPC)-enabled nodes. In order

to allow for highly efficient execution and minimal overhead the functional components, called Functional Protocol Elements (FPEs), handled by this framework shall be as small as reasonable: adding a sequence number, sending an acknowledgement or forwarding a packet are typical examples. This design goal leads to a fragmented world of micro-protocols which can be combined in a very efficient manner. Composed protocols can be tailored exactly to the needs of the service in a particular situation. To implement these composed protocols FPEs may express dependencies to other FPEs and even consist of a set of dependencies only, so called virtual FPEs (vFPEs). This allows single FPE instances and the respective header fields to be used to achieve several independent and more complex functionalities: A byte counter FPE can be used to achieve flow control at the same time as loss detection, maybe combined with automatic retransmissions.

It therefore is targeted on an efficient and highly dynamic packet handling which incorporates setting up of so called Functional Couplings like Automatic Repeat Request (ARQ) between nodes. To achieve this goal there is an architecture needed that on one hand allows fast and efficient packet processing but on the other hand allows for adaptation by the network management. These tasks can be supported by an efficient Self-Description mechanism that also supports abstract control interfaces by internally defining dependencies and conflicts and therefore allowing the DPC framework to solve many FPE related issues without external interaction.

The concept of minimal functional elements also facilitates the fine grained extension and adaptation of functionality within the network to the current situation in that local network: with such a framework it is possible e.g. to activate adding Forward Error Correction (FEC) to a (UDP-like) video streaming service for only the wireless last hop and at the same time activate local buffering and retransmissions for a reliable file transfer.

4 Applying DPC to Intermediate Nodes

The application of the principle of DPC to intermediate nodes i.e. routers creates new challenges in setting up and managing the DPC state in these nodes in an efficient and resilient manner. Current end-to-end protocols of the IP-based protocol family on purpose do not set-up any state in intermediate nodes which has several advantages, e.g. a clear separation of network management and end user equipment. Nevertheless a cognitive network management combined with soft state mechanisms (regularly refreshing signalling) will allow the use of DPC in the Future Internet. Applying communication functionalities, e.g. encryption or reliable transmission, dynamically between any nodes in the Future Internet allows to tailor the performance of each link to exactly the requirements of each flow and to the capabilities available in the current situation. We think that the potential benefit of more functionality within the network and the ability to adapt this functionality to the current context is promising a better network performance and higher efficiency. Therefore this topic is worth to be investigated more deeply.

5 Cognitive Control of Fine-Grained DPC Frameworks

Since DPC is only concept or tool to provide certain functionalities within a network, the management and the complexity of the management depend heavily on the goal that shall be achieved using this tool. Nevertheless some basic assumptions seem obvious and therefore lead our design of the management infrastructure. The first observation is that many decisions for (re-)configuration of DPC functionality require an understanding of the current network situation. To take the right decision, the decider has to know the exact location of a problem, e.g. on which link packets are lost, and, most important, has to determine the reason for the observed behaviour: packet loss may be induced by bad physical conditions but also by aggressive queuing strategies or overload of a link or a node. This also shows the need for continuous feedback in order to avoid amplification and oscillation in the network. Up to now there was no solution deployed fulfilling these requirements except the careful human administrator who intelligently gathers information, derives the current situation and the cause for the issue to be solved and then selects the action to be taken. Obviously this approach is not feasible for fine-grained protocol configurations and their frequent adaptations to changing network situations. The approach of cognitive network management or Autonomic Communication promises systems that are able to achieve a level of cognition that allows to assess the situation and to make intelligent decisions which take into account the expected impacts of the action to be taken. Therefore we assume situation awareness and cognitive decision making to be the key to successful broad and general deployment of DPC mechanisms.

Although the impact on our architecture is limited, we assume the hierarchical model of Cognitive System and Network Management (CSNM) presented in [9]. It is based on a Network Element Cognitive Manager (NECM) running on each Future Internet Element (FIE) that controls the node with respect to the capabilities that nodes provides. NECM makes self-aware and situation-aware decisions based on the local monitoring information available and the (expert) knowledge it stores. If decisions need information about greater parts of the network, e.g. topology and routing information, NECM may request a higher level cognitive manager, the Network Domain Cognitive Manager (NDCM), to take a decision. NDCM gathers high level information about its network domain and proactively takes decisions if it sees need for based on his information or reactively takes decisions if triggered by a NECM of its domain.

Although this concept promises intelligent decisions the effort needed to take a decision is high: Gathering information from many sources, deriving a situation, maybe predicting the impact of several execution options and the final decision enforcement is expensive with respect to time and computing resources. With respect to control of fine-grained DPC frameworks this is not acceptable for many minor communication decisions e.g. deciding on a session initiation request or on activation of Cyclic Reduncy Check (CRC) checking for a certain flow. Therefore we propose a more elaborated set of control interfaces which is presented in section 8. First, we present the underlying architecture that is controlled by this interfaces and show shortly potential advantages of DPC application on intermediate nodes.

6 The DPC Architecture and Implementation

The DPC implementation developed in the Self-NET project [10] provides a unified framework that replaces the Linux IPv6 protocol stack but runs as a user space application. This implementation is based on the C++ open source Simple and Extensible Network Framework (SENF [11]) and uses packet sockets to read and write packets. For signalling IPv6 extension headers are used which allow the DPC signalling to be transparent for legacy IPv6 nodes: the highest-order two bits of the used option type are set to zero, by this advising nodes that don't recognise this option to skip over it and continue processing.

The main objects of this framework besides packets are:

Functional Protocol Elements (FPEs). A FPE implements network protocol functionality. Some FPEs require the cooperation with other FPEs so these define dependencies in their Self-Description, vFPEs even consist of other FPEs only and don't implement own functionality at all. FPEs that don't have any dependencies are called basic FPEs and only these FPEs can be instantiated in Composed Protocol Chains (CPCs).

Packet Filters (PFs). A PF is defined by a set of packet attributes, so called Packet Filter Element (PFE) which partitions all packets into matches and not-matched. A simple example is a destination IPv6 network as it is used in routing rules. More complex examples use more attributes like source address, traffic class, flow label, transport protocol, port address etc. Two different filters are either disjoint, have a partial overlap or one includes the other. If two filters match a packet, the finer one will precede.

Composed Protocol Chains (CPCs). A CPC consists of a PF and an ordered chain of FPEs that defines what this nodes does with a packet that matches the PF. So a CPC defines a unidirectional protocol. It should be noted that this inherent support for unidirectional functionality definition allows DPC functionality to be tailored to the actual needs that often are asymmetric. Therefore two nodes will usually have different CPCs active if they are configured to provide a certain functionality in a collaborative way, e.g. when providing ARQ for a streaming service (shown in figure 3).

6.1 Components and Operation

An overview on the modules of the implemented DPC architecture and their interoperation is given in figure 1. The main components and their roles are introduced in the following:

The Packet Inspector (PI). This module checks an incoming packet for so called Active Information Elementss (AIEs) which trigger the reconfiguration / activation / deactivation of a specific FPE, e.g. a packet could contain a activate CRC-check flag.

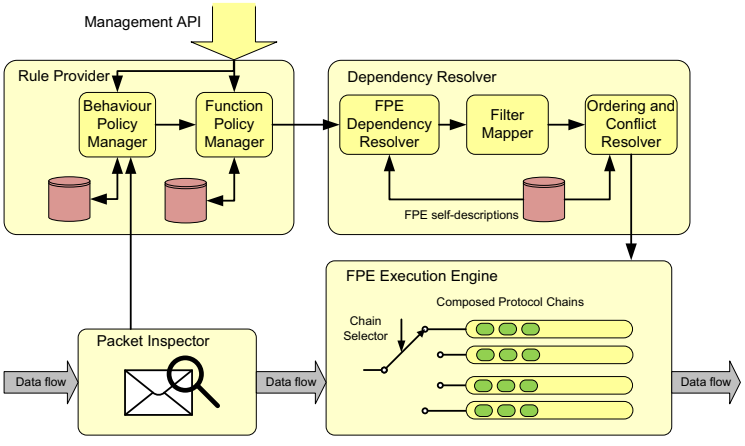

Fig. 1. Dynamic Protocol Composition Architecture

The Rule Provider (RP). This module manages two kinds of policies: **Function Policies** configure the application of a certain FPE for a certain PF and include routing rules, firewall rules, rules for adding or checking checksums, adding sequence numbers, triggering or sending acknowledgements etc. **Behaviour Policies** define how to decide on modification requests from other nodes received via AIEs (see section 8.2).

The Dependency Resolver (DR). This module composes the set of CPCs that is defined by the rule sets provided by RP. The process is detailed in section 9.

The FPE Execution Engine (FEE). This modules is responsible for the execution of the configured CPCs. For each incoming packet it selects the best matching filter, just like a router selects the longest matching prefix, and applies the FPEs in the order given by DR.

7 Initial Trials

In a first implementation step we implemented the basic components and six FPEs that allow two hosts to set up a ARQ functionality for the link or the path between them. The experimentation setup as depicted in figure 2 consists of seven Linux based PCs connected with Fast Ethernet. Two of them play the role of a streaming server and a streaming client and use a standard Linux IPv6 protocol stack. Four others, R1-R4, are equipped with the Fraunhofer DPC implementation and shall represent the routers of a wireless mesh network. The seventh computer V1 uses a standard Linux IPv6 protocol stack and allows to induce packet loss using the Linux kernel network emulation feature Netem [12].

In the experiment we start with a no loss and no DPC functionality except forwarding active on all routers. In a first step we induce 10% packet loss at

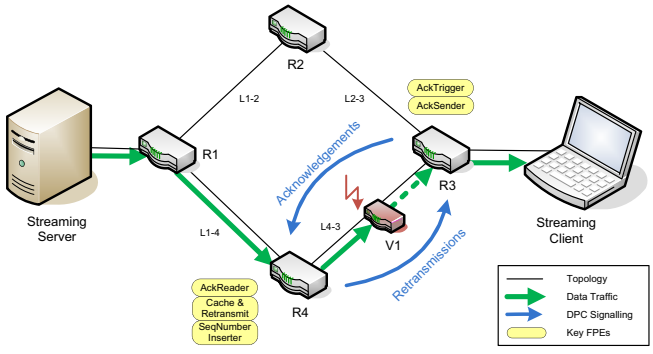

Fig. 2. Experimentation network setup

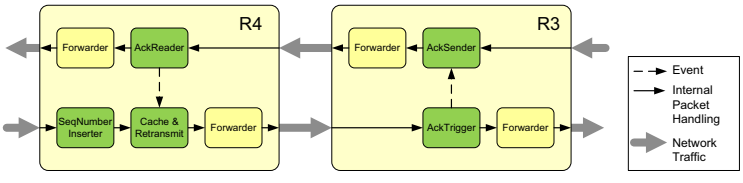

Fig. 3. CPC configuration when ARQ is enabled

V1. After some seconds we manually activate ARQ for transmissions from R4 to R3. This is implemented by two CPCs on R4 and two CPCs on R3, one sending and one receiving chain for this interface on each router. Please find the precise composition in figure 3.

7.1 Initial Results

Our first measurements are depicted in figure 4 and 5. In the monitored trial 10% packet loss (Gaussian distribution) have been induced at a time of 8.1 seconds on V1 resulting in a significantly increased packet loss as can be seen in figure 4. After 11.4 seconds the DPC framework activates a link ARQ mechanism which results in no packet loss but higher jitter. While before activation of DPC-based ARQ in figure 5 only one main cluster of delay can be recognised, after point 19.5 four clusters of delay can be recognised. These clusters represent packets that reach their destination after the initial transmission, the first automatic retransmission, the second or the third respectively.

The results show that certain properties of network performance can be significantly improved by the application of the DPC approach between routers. This solution provides several advantages compared to the usage of TCP since TCP causes acknowledgements and retransmissions to be forwarded along the complete end-to-end path.

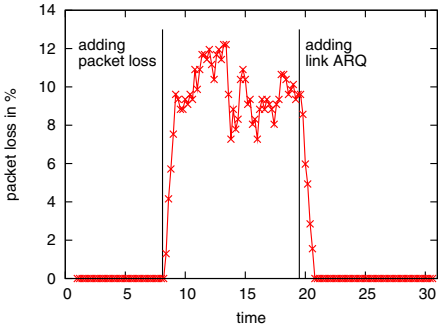

Fig. 4. Packet loss at receiver

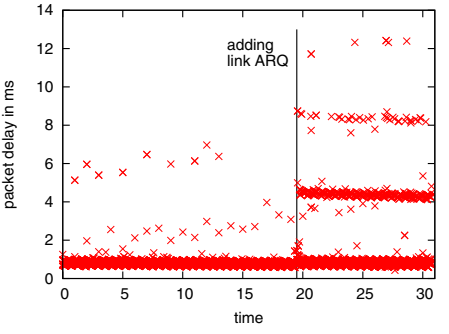

Fig. 5. Packet delay at receiver

8 Management of DPC Functionalities

As explained in section 5 it doesn't seem feasible to put all DPC-related decisions under direct control of the CSNM. The many different decisions required to control a dynamic communication protocol suite differ heavily in frequency, impact and immediacy. Our DPC architecture design exploits this fact and provides several options to take decisions that differ in the level of response time and intelligence with respect to situation awareness and cognition. All of decision making options base on the NECM who influences decisions taken within the DPC framework by providing preconfigurations that guide the decision. The four different ways of decision making considered in our DPC framework are presented in the following.

8.1 Reflex-Action Decisions

First, our architecture supports reflex-action decisions that are realised as preconfigured CPCs within the FEE component, e.g. to establish a connection when a request for a certain local service access point from a specified host or network is received. These CPCs are configured by the NECM inserting special Function Policies in the RP. The decisions on this level are prepared in advance and only wait for a trigger in form of a packet matching the PF of the CPC. Because of this lean structure they don't require significant computation time but the decision can only take into account information that can expressed in PFs, that means it is restricted to packet related information.

For security and resilience reasons we assume that a packet may only trigger configuration changes that apply for exactly the filter matching this packet: The configuration change will add or alter the CPC for exactly the PF of the triggering packet, i.e. the change will only affect packets coming from the same source address, same flow label etc. This postulation allows for flow-specific reconfigurations, in particular management of sessions while minimizing the risk of denial of service attacks. This new established state has to be refreshed periodically and is not part of the permanent ("hard") state which is stored in the RP.

Although these CPC-based decisions are very simple these rules are defined and can be changed at any time by the NECM who may use a very sophisticated situation aware cognitive and knowledge-based decision making framework which may even evaluate historical data and predictions.

8.2 Automated Rule-Based Decisions

The second option supported in our architecture is based on Behaviour Policies stored in the RP. These policies define rules for automated reconfigurations requested from other DPC-enabled nodes by a AIE and allow for a broader set of conditions as well as actions compared to the CPC-based rules described above. The Behaviour Policies may base on external configuration information the DPC framework can access fast, e.g. the current link speed of a WiFi interface. The result of a decision based on Behaviour Policies are Function Policies that define hard state of the DPC framework and are put into action as described in section 9.

Although the automated decision making mechanisms of this stage do not create situation awareness themselves, again it is the NECM who provides the Behaviour Policies to the Behaviour Policy Manager. The ability to define Behaviour Policies for the DPC framework allows NECM to define reactions for several expected situations that can be executed very fast based on a triggering message from neighbouring nodes and a certain set of context information. This system is expected to be used by the NECM to prepare for Functional Couplings like ARQ as described in section 7 with other DPC-enabled nodes.

Since the NECMs on R3 and R4 know that L4-3 is a wireless link which might be improved by Local ARQ, they prepare to serve as an ARQ-Acknowledger if requested. To take this decision NECM needs to know that the resources are sufficient to provide the additional service, e.g. bandwidth for the acknowledgements, computing power to check CRC and create acknowledgements etc.

8.3 Configuration by NECM

Third, our architecture provides a local management interface that allows configuration of the DPC framework by the local NECM. This interface to the RP is designed to store and retrieve the Function Policies that define the CPCs configuration. These policies define the "hard-state" of the DPC framework that e.g. would have to be restored in case of a reboot.

Since NECM provides the full set of cognitive features the policies configured using this interface are expected to represent high quality decisions without having strict immediacy. The interface was designed to allow for simple policies in contrast to the complex CPCs that are instantiated by the FEE. One FPE-policy consists of one PF and just one FPE which may be a virtual one. This keeps the configuration interface lean and is in line with the limited abilities of automated rule-based reconfigurations that are also stored in the Function Policy Manager.

8.4 Configuration by NDCM

The fourth control option is indirect management by the NDCM. The NDCM as a higher level cognitive manager has access to more information like topology, routing configuration, load and actual capabilities of the nodes in the network. This guarantees highest quality decisions but also leads to high delays and comparatively high costs for decision making and decision enforcement. Since NECM and NDCM communicate frequently we assume NECM to forward decisions taken by NDCM to the DPC framework using the interface described in section 8.3.

8.5 Comparison of Decision Making Levels

The proposed four levels of decisions vary in many properties. Table 1 gives a short overview on the most important properties of the four options.

Table 1. Properties of the different decision making levels

	CPC-based	**Rule-based**	**NECM-based**	**NDCM-based**
speed	++	+	–	– –
considered context	packet and CPC	simple local	local	compartment
level of cognition	– –	–	+	++
location of decision making	FEE	RP	NECM	NDCM
state affected	soft only	soft and hard	soft and hard	soft and hard
external signalling	inband	inband	outband	outband

The given rating of the speed of decision making on the different levels will be measured once the experimentation phase is concluded. When estimating the times between the trigger and the decision enforcement, the expected times lay between few microseconds for the CPC-based configuration changes up to many seconds or even few minutes for the decision to be taken by the NDCM. The given rating of level of cognition shall abstractly show the impact that the limited access to context and knowledge will have for the preconfigured automated decision processes in FEE and RP. This may be neglected except for abrupt changes in the network because the underlying rules, be it a policy in RP or a CPC in FEE, are defined with maximal knowledge and cognition.

Altogether the first two levels of decision making could not provide a sufficiently intelligent protocol configuration mechanism because the lack of knowledge and context information while NECM and NDCM alone could not achieve that goal because the required resources, in particular time. The combination of all four levels of decision making promises being able to deliver fast and intelligent decisions at the same time.

9 Handling of Policy Changes in the RP

Policy changes in RP may have different sources therefore measures have to be taken to guarantee the consistency of the protocol configuration in FEE and its consistency with the policies stored in the RP. Once changes are completed by a commit command, the resulting ruleset has to be checked for conflicts due to explicit incompatibility, ordering conflicts or contradicting parameters. The proposed architecture therefore realises the ACID properties [13] known from database transaction theory. The detailed process consists of three steps that are performed by subcomponents of the DR as depicted in figure 1.

FPE Dependency Resolution. In the first step for each policy the dependencies of the included FPE are resolved so that each rule is translated to a so called CPC description consisting of one PF, a set of FPEs and their parameters. The PFs of this set of CPC descriptions in general contains overlaps, inclusions and identity.

Filter Mapping. In the second step the Filter Mapper function resolves this diverse set of PFs to a tree consisting of internal nodes containing one PFE each and leafs containing a set of FPEs and their parameters. If two sets of FPEs have to be joined into one leaf because of overlapping PFs in the policies their parameters are checked for consistency. If they carry contradicting parameters, all changes are discarded and the process is rolled back. If the policy change request has been requested by the NECM, it is informed about the failure.

Ordering and Conflict Resolution. The third function is called when the tree is complete and works on the leafs only: The FPEs of each leaf are checked for conflicts and ordered according to their requirements expressed in their Self Description. In case of conflict or unresolvable position requirements, the processing of the set of policy changes is stopped and rolled back as described above.

If the policies could be merged, they are activated in FEE, stored in the RP and NECM is notified about the change.

10 Conclusion

The concept of applying DPC principles to intermediate nodes proves to be advantageous and the first experiments show that DPC allows to compensate packet loss and to greatly improve quality of service. The approach to focus on selected use cases and to research their aspects in detail has been productive and since the set of potential use cases and parameters is overwhelming, this approach should be continued for further research. In general the combination of DPC mechanisms with its many degrees of freedom with cognitive network management that allows to autonomously make situation-aware decisions based on knowledge is very promising and opens up many new opportunities to increase the efficiency of operating communication networks and by this save scarce resources like spectrum and energy.

Acknowledgement

The presented work is supported by the European Commission Seventh Framework Programme ICT-2008-224344 through the Self-NET Project [10].

References

1. Clark, D.: The design philosophy of the darpa internet protocols. In: SIGCOMM 1988: Symposium proceedings on Communications architectures and Protocols, pp. 106–114. ACM, New York (1988)
2. Postel, J.: Transmission Control Protocol. RFC 793 (Standard), Updated by RFC 3168 (September 1981)
3. Postel, J.: User Datagram Protocol. RFC 768 (Standard) (August 1980)
4. Hutchinson, N.C., Peterson, L.L.: The x-kernel: An architecture for implementing network protocols. IEEE Trans. Softw. Eng. 17(1), 64–76 (1991)
5. Vogt, M., Plattner, B., Plagemann, T., Walter, T.: A run-time environment for da capo. In: Leiner, B. (ed.) Proceedings of International Networking Conference, INET 1993, San Francisco, California, pp. BFC-1–BFC-9 (August 1993)
6. Feldmeier, D., McAuley, A., Smith, J., Bakin, D., Marcus, W., Raleigh, T.: Protocol boosters 16, 437–444 (April 1998)
7. Vcrbaeten, P., Janssens, N., Michiels, S.: Dips/cups: a framework for runtime customizable protocol stacks. Tech. rep. (2001)
8. Towards Hot-Swappable System Software: The DiPS/CuPS Component Framework (2002)
9. Mihailovic, A., Chochliouros, I.P., Kousaridas, A., Nguengang, G., Polychronopoulos, C., Borgel, J., Isral, M., Conan, V., Belesioti, M., Sfakianakis, E., Agapiou, G., Aghvami, H., Alonistioti, N.: Architectural principles for synergy of self-management and future internet evolution. In: Cunningham, P.C.M. (ed.) Proceedings of the ICT Mobile Summit 2009 (June 2009)
10. Self-NET (Self-Management of Cognitive Future InterNET Elements). EU FP7 project INFSO-ICT-224344, `https://www.ict-selfnet.eu`
11. The simple and extensible network framework, `http://senf.berlios.de`
12. Netem homepage, `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`
13. Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. 15(4), 287–317 (1983)