# Lightweight Mechanisms for Self-configuring Protocols

Eleni Patouni and Nancy Alonistioti

Department of Informatics & Telecommunications, University of Athens
Athens, Greece
{elenip,nancy}@di.uoa.gr

**Abstract.** In next generation networks, the introduction of intelligence and flexibility in mobile devices and network nodes appears as a promising solution to the high degree of variability in the telecommunication environment. The materialization of these concepts under the autonomic networking notion paves the way for introduction of awareness and adaptation capabilities in various layers of mobile devices, also including the protocol stack. In this paper we investigate the problem of dynamic protocol stack adaptation and propose enabling mechanisms for the dynamic binding and replacement of their constituent components. Our work addresses the challenge of the application feasibility and performance evaluation of these concepts by quantifying the introduced delay. The obtained results show that the introduced protocol adaptation functionalities pose negligible performance impact in the system. Our work reveals that although flexibility and performance stand on the opposite sides of the system balance, the introduction of transparent mechanisms leads to great adaptability with minimum performance impact.

**Keywords:** self-configuring, protocol component, reconfiguration, feasibility, lightweight.

## 1 Introduction

In the heterogeneous environment of next generation networks, the coexistence of various radio access technologies (RATs) and the continuous launch of new and emerging paradigms increase the overall system complexity. At the same time various challenges should be addressed mainly related to the seamless and transparent integration of the new technologies. Such challenges are also tightly coupled with the vast proliferation of the user needs: user requirements impose the philosophy of seamless, anywhere, anytime connectivity and support of multiple types of mobile devices. In this context, one promising solution for addressing such requirements is the introduction of flexibility and intelligence in the system, including both the mobile devices and the network side.

One path in achieving this goal is realised through the emerging visions of reconfigurability and autonomic networking [1],[2] that bring forward new adaptation capabilities in the different layers of the protocol stack and system resources [3]. However, it is often argued that such features increase even more the system complexity. Autonomic networking is viewed as a solution that "fights complexity with complexity" [4]. Practically, this means that the complexity of managing a system is

P. Chatzimisios et al. (Eds.): MobiLight 2010, LNICST 45, pp. 112–123, 2010.

dealt by introducing complex mechanisms in the different system elements that enable them to be aware of the surrounding environment (self-awareness), manage themselves (self-management) and dynamically adapt to the environmental stimuli following high-level objectives and policy rules (self-configuration) [5].

In this direction, this paper addresses the aspect of dynamic adaptation and self-configuration in the protocol stack of mobile devices. In legacy systems, the protocol stack design is driven by the strict performance and resource requirements of mobile devices and network elements. Targeting the fulfilment of such requirements, protocol stack subsystems yield two main principles: a) monolithic, layered design and b) design tightly coupled and statically bound to the networking architecture. The abovementioned design principles lead to platform-dependent implementations that are highly optimized and often tied to the underlying operating system. In such systems, performance is a function of the allocation flexibility offered by the protocol stack's processing model and the minimization of context switches occurring in its operation. In this direction, evolutionary methods for flexible protocol layer design with seamless and transparent adaptation capabilities are necessary to boost the system performance.

Leveraging our previous work in [2],[6] for design of self-configuring protocols, this work focuses on the feasibility and performance evaluation of such concepts. This is realised by introducing the necessary support functionality and signalling between the mobile device and the network side for the realisation of protocol self-configuration. In addition, special focus is paid on the performance evaluation of these mechanisms in the mobile device. In order to test the feasibility of the proposed solution, there is a need to evaluate whether the proposed mechanisms are lightweight; this should be quantified in terms of the introduced delay in the system. The results prove that the presented mechanisms increase the system flexibility whereas at the same time impose minimum performance overhead.

This paper is structured as follows. Existing research activities on adaptable protocols are presented in section 2. The introduced mechanisms enabling self-configuring protocols are briefly analysed in section 3, focusing on the end-to-end support signalling for the realisation of dynamic component binding and configuration. Section 4 proves the arguments for applicability, feasibility and performance evaluation through simulations using the UDP-based Data Transport (UDT) protocol [7]. Finally conclusion remarks and directions for future research are drawn in section 5.

## 2 Related Work

The ongoing proliferation of networking standards and the proved inefficiency of traditional protocol stack design have motivated numerous research efforts on the adaptation and customization of protocol layers or entire protocol stacks. Specifically, adaptable protocol stacks are seen as a technological enabler of next-generation networks which leverage the introduced adaptation and customization capacities to achieve two main goals: the dynamic adjustment of protocols' operation mode and the performance optimization of the operating protocols/protocol stacks. Such targets have been the main research objective of various adaptable protocol stack approaches, which are classified into the following three main categories - a detailed survey analysis on dynamically adaptable protocol stack frameworks is available in our previous work in [8].

- Adaptable protocols: This design approach introduces an extension protocol layer besides the generic part, for the implementation of custom protocol functions. It should be noted that this category employs a coarse granularity design since the fundamental design unit is a protocol layer or a set of them. Adaptable protocol stack frameworks include Conduits, JChannels and POEM [8]. For example, Conduit+ [9] proposes the specification of explicitly specified connections between Conduit+ objects under a generic object-oriented framework. The primary design goal of this framework is the reusability among software protocols.

- Composable protocols: this concept employs flat, hierarchical and graph-based models for building a customizable protocol/protocol stack out of fundamental protocol functions. Composable protocol stack frameworks include DiPS/CuPS , x-kernel, Coyote/Cactus, Appia, Ensemble, Horus, RBA, Da CaPo, ADAPTIVE, DRoPS, DIPS+, ACCORD and DPS [8]. For example, the Distributed Protocol Stacks (DPS) framework [10] enables the distribution of protocol functionality in network elements/nodes, targeting the optimization of the operations through functions migration.

- Reconfigurable protocols: This design allows for extending the traditional protocol stacks' composition schemes to support the dynamic binding and replacement of protocol components or even entire protocol layers during runtime, enabling service continuity and no loss of protocol data. Reconfigurable protocol stack frameworks include THINK, FRACTAL, GRPSFMT, DRAPS and Alonistioti [8]. In DRAPS, the fundamental unit is the core component which implements the protocol layer functionality and also realizes the communication with the adjacent protocol layers [11]. In addition, DRAPS facilitates dynamic protocol reconfiguration during runtime operation, also ensuring transparency. This is achieved by specifying the necessary interfaces per protocol component for the control of the reconfiguration operations (e.g. start/stop functions) and the state management procedures. We should point out though that this framework does not provide any details as regards the internal blueprints of the protocol stack architecture are not detailed.

At this point it should be noted that our approach falls under the category of reconfigurable protocols. The main advantage of our work is the detailed specification of a framework enabling the dynamic, semantic-based binding and replacement of protocol components during runtime operation of the protocol stack. In addition, the necessary support management functionalities and state management mechanisms were defined, targeting transparency, robustness and seamless operation.

## 3 Self-configuration Mechanisms

### 3.1 Background on Protocol Reconfiguration

The key concepts of protocol reconfiguration have been thoroughly investigated in our previous work in [6]. This section presents shortly our work on the protocol

reconfiguration framework, which is based on the concept of protocol decomposition into components and their dynamic binding into specific protocol functionality. Specifically, this framework adopts a modular approach for both the protocol stack design and adaptation. The following design assumptions were made:

- A protocol stack is viewed as a composition of specific protocol layers,
- A protocol layer is viewed as a composition of protocol components.

Such modular protocol stack design affects both the protocol stack bootstrap and the protocol reconfiguration procedures. During protocol stack bootstrap, the dynamic component binding takes place to form both the protocol layer and protocol stack functionality. This is realized evaluating a semantic layer of information that accompanies each protocol component: its metadata. Specifically, two types of defined parameters are distinguished: a) protocol ID parameters; such parameters are related to the identification of the protocol, i.e. name, modularity, vendor, etc. b) protocol component parameters which include ID and binding parameters per protocol component and performance parameters per protocol layer. The protocol component binding parameters include component input and output interfaces information.

During protocol reconfiguration, the new configuration of the protocol stack is specified which includes the protocol layers that are used as well as the components that form these layers. Depending on the specified reconfiguration actions, one or more protocol layers may need to be replaced or one/more protocol components within each layer.

*Semantic-Based Dynamic Binding of Protocol Components*

The semantic-based dynamic binding of protocol components is realised evaluating the dynamic characteristics of the component's metadata and identifying their communicating counterparts - components. After the validation and verification of the composition, the communication links for the protocol components are established. This is realised by creating a FIFO queue per communication link for each component. The composition verification and establishment procedures are repeated for all the protocol components that are bound to the specified component.

*Dynamic Replacement of Protocol Components*

After evaluating the new protocol configuration, the replacement of the old protocol component with the new one is realised by a coordinating management entity, the Component Binding Control Module (CBCM). At first, the CBCM module pauses the functionality of the component under replacement and retrieves its execution state. Next, it instantiates the new component and dispatches to it the retrieved state information. Based on the acquired state information and its metadata, the new component realizes its dynamic composition with existing software components (by accessing the FIFO queues corresponding to existing communication links). The above analyzed on-the-fly replacement process also allows the reliable operation of the protocol under configuration, since it applies state management models to ensure the transparent switching from the old to the new component [6].

### 3.2 End-to-End Signalling Supporting Self-configuring Protocols

This section focuses on the realization of protocol reconfiguration in a system analyzing the required end-to-end supporting signalling (Fig. 1). In this work we consider

the system model for dynamic adaptation of mobile devices in cognitive radio networks that is presented in our previous work [2]. Such system model encompasses the fundamental capabilities for the implementation of cognition in a system, namely monitoring the environmental stimuli, decision specification based on the contextual evaluation and realization of the adaptation through decision enforcement [12]. In such model two main physical entities are considered, namely the mobile devices and the network nodes. Based on the proposed capabilities, the following functionality is introduced in the mobile devices and network nodes:

- The Context Management (CTxM) module, which realises the administration of profile information (e.g. terminal and user profile, protocol stack stratifications).
- The Self-Configuration (Se-Co) module, which orchestrates the dynamic adaptation of the system protocol resources.
- The Autonomic Decision-Making module (ADM), which realizes the decision specification evaluating different options and alternatives. In this case study we consider that the decision is distributed between the network and the mobile device.

As regards the end-to-end supporting signalling, we assume that following a change in the environmental stimuli, a trigger for changing the device configuration takes place. Specifically, this could be the result of low QoS or low signal strength due to user mobility or could be caused by increased application requirements of a new service initiated by the user. In such cases, the ADM module located at the mobile device communicates with the CTxM module (also in the mobile device) to retrieve contextual information regarding the user and terminal profile and the protocol stack configurations (Fig. 1). After filtering this information and deciding the protocol stack requirements, ADM specifies a set of possible decision alternatives which are communicated to the network side ADM for evaluation. The latter should select the best one for the overall system optimality, taking into account various parameters including for example the network load, the a-posteriori operations that need to be implemented following the implementation of a specific alternative (e.g. handovers), load balancing schemes etc.

Upon receipt of the selected alternative, the ADM module at the mobile device communicates with the Se-Co module for the alternative implementation. In this scenario we consider that the selected option is the RAT upgrade which concerns the dynamic replacement of a protocol component in an operating protocol. Next, the Se-Co module implements all the necessary operations for this replacement. It communicates with the CBCM module which realizes the following operations: a) pauses the old protocol component, b) retrieves its state, c) instantiates and initializes the new component, and d) configures the state of the new component using state management schemes. The establishment and initialization of the communication links for the new component is also handled by the CBCM by evaluating semantic information for each component (section 3.1).

Following this description we identify that the concept of self-configuring protocols requires: a) message exchange between the mobile device and the network side for the evaluation and selection of the alternatives and decision-making functionality

at the network side and b) self-configuration functionality and mechanisms in the mobile device for the dynamic component replacement. The first one may impose additional overhead at the network side – this case is investigated in detail in [2]. The second case concerns the overhead that may be imposed in the mobile device and is the focus of this paper.

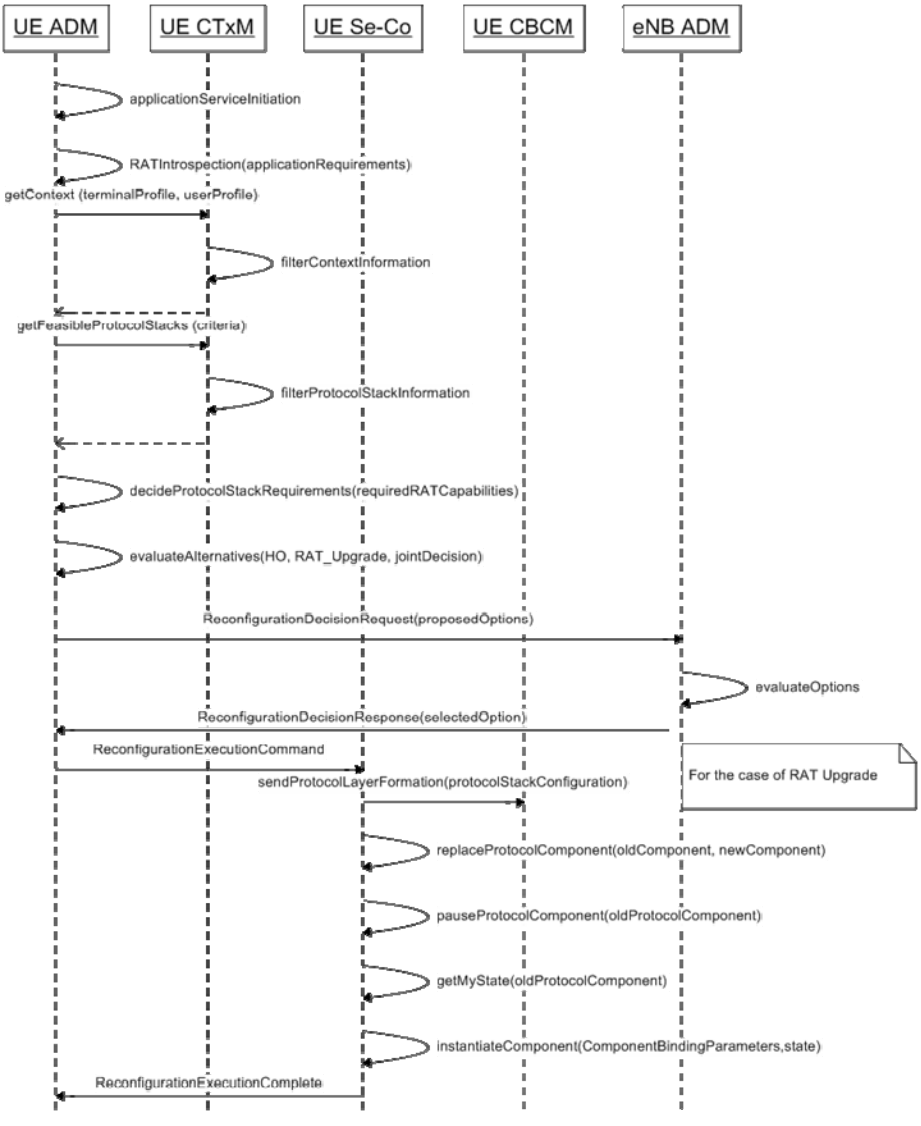


**Fig. 1.** End-to-end support signalling for protocol reconfiguration triggering and realization

# 4 Performance Evaluation

The introduction of dynamic reconfiguration capabilities in the protocol stack increases its flexibility but – inevitably – incurs a performance penalty [6]. Since one key issue in such a system is whether protocol reconfiguration forms an applicable and lightweight solution, the performance aspects should be investigated in the overall system operation. As regards the mobile device, the delay of the protocol reconfiguration procedure should be examined. Another objective concerns how the system operation is affected by the introduced delay and the protocol reconfiguration procedure, e.g. in terms of service continuity. Taking into account the component-based notion of the proposed solution, the component binding delay (the time required for the realisation of the binding operation) and component execution time (the time required for the execution of a specific algorithm/functionality that is incorporated in a component) are investigated using simulations.

## 4.1 Simulation Environment

The evaluation methodology used for the performance analysis of the self-configuring protocols solution is analysed herein. The performance analysis of the self-configuring protocols concept is evaluated through NS-2 simulations which target the component binding delay and the component execution time KPIs. In order to evaluate the key concepts of the self-configuring protocols solution, we selected to use the UDP-based Data Transport (UDT) protocol [7].

UDT is an application level data transport protocol for the emerging distributed data intensive applications over wide area high-speed networks (e.g., 1 Gb/s or above). UDT uses UDP to transfer bulk data and it has its own reliability control and congestion control mechanism. This new protocol is not only suitable for private or QoS-enabled links, but also for shared networks. UDT is also a composable framework that can accommodate various congestion control algorithms. Its reliability and congestion control mechanisms exist entirely in user space (not kernel space) and thus protocol variables are accessible at run time. However, UDT protocol configuration (congestion control algorithm selection) can only be done at compile time. In this layered architecture, the UDT layer is completely in user space above the network transport layer of UDP, whereas the UDT layer itself provides transport for applications. Meanwhile, applications provide optional control handlers to UDT as callbacks. In our work, the decomposition of the UDT protocol in 2 protocol components is modelled, considering a server node which implements UDT as a self-configuring protocol and a client node with a non-modular approach. A graphical representation of the component-based UDT protocol stack supporting protocol reconfiguration is provided in Fig. 2.

The simulations concern the evaluation of the communication control part of the two key protocol components, namely:

- the congestion control (CC) component which implements the congestion control functions,
- the UDT core component which implements the remaining functionality of the UDT protocol (i.e. connection establishment, memory and buffer management).

Both these components are implemented in two different agents and form the modular server node. The client node is implemented as a separate agent using a non-modular approach [13].

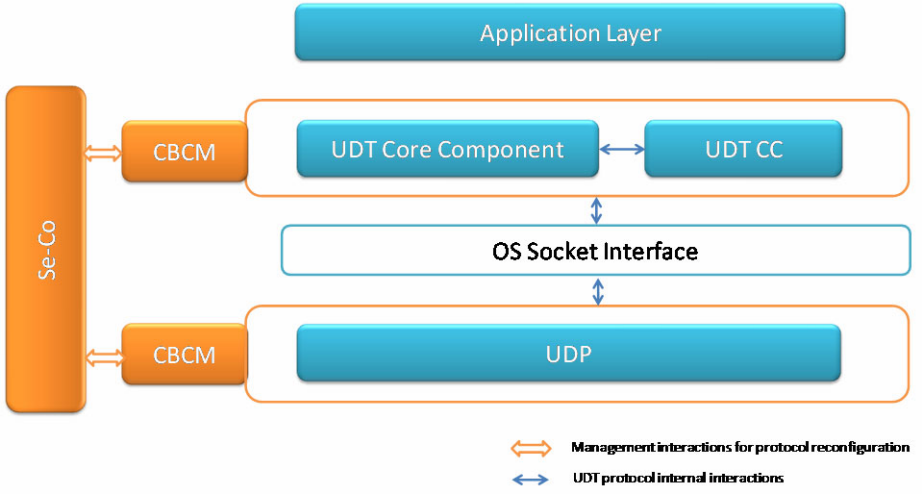


**Fig. 2.** UDT component-based protocol stack supporting protocol reconfiguration

The deployment scenario is based on the binding of the core UDT component (hereafter referred as *Core*) with a congestion control algorithm (hereafter referred as *CC*) according to the afore-described methodology. Initially, the Component Binding Control Module retrieves the components metadata. Then the composition of the two components is checked and the required structures are created. In this scenario, we also consider the dynamic replacement of the CC component implementing CVegas [14] with another CC component implementing CTCP [15]. CVegas is a delay-based algorithm that uses its own data structure to record packet departure timestamps and ACK arrival timestamps and then calculates accurate RTT values. CTCP is an application level implementation of the standard TCP algorithm using the UDT/CCC library [15]. In addition, it is assumed that the Core Component is bound to the same components as the CC component. At first, the Component Binding Control Module retrieves the metadata for the new component (CC with CTCP) and checks which components it is composed with. Then it pauses the function of the component to be replaced (CC with CVegas). Thereafter, the Component Binding Control Module retrieves the state of CC-CVegas component and sets the state of CC-CTCP to the same state. This is necessary since the replacement of a component, which is in a specific state, with a component that is in an idle or initial state, does not guarantee the reliable operation of the protocol [13]. It should be noted that besides CVegas, various algorithms for the congestion control component have been considered in the simulations.

### 4.2 Results

This section presents the results derived from the NS-2 simulations of the self-configuring protocols. In order to better compare the results, the minimum, maximum and mean value of the component binding delay and the component execution time are computed per algorithm implementing the congestion control component. The results are presented in Fig. 3 and Fig. 4. Such figures were obtained considering the following algorithms for the implementation of the CC component: TCP, Scalable TCP, High-Speed TCP, Binary Increase TCP, TCP Westwood, TCP Vegas and Fast TCP. Scalable TCP is a simple sender-side alteration to the TCP congestion window update algorithm which offers improved performance in high-speed wide area networks [16]. HighSpeed TCP is a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows [17]. Binary Increase TCP offers an optimized congestion control algorithm for high speed networks with high latency enabling the algorithm to aggressively increase its transmission speed toward the maximum allowed by the high-speed network [18]. TCP Westwood is a sender-side-only modification to TCP NewReno algorithm and it aims to better handle large bandwidth-delay product paths, with potential packet loss due to transmission or other errors, and with dynamic load [19]. Fast TCP is a TCP variation that measures a different factor (queuing delay instead of loss probability) to evaluate network congestion and can increases its congestion window more quickly than TCP [20].

As regards the component binding delay, the maximum value equals 337 μsec and it is obtained for the TCP Westwood algorithm, whereas the minimum value is obtained for the UDT algorithm and equals 72 μsec. In addition, as regards the component execution time, the maximum value equals 193 μsec and it is obtained for the Fast TCP algorithm, whereas the minimum value is obtained for the scalable TCP, Binary Increase TCP, TCP Westwood and UDT algorithms and equals 32 μsec.

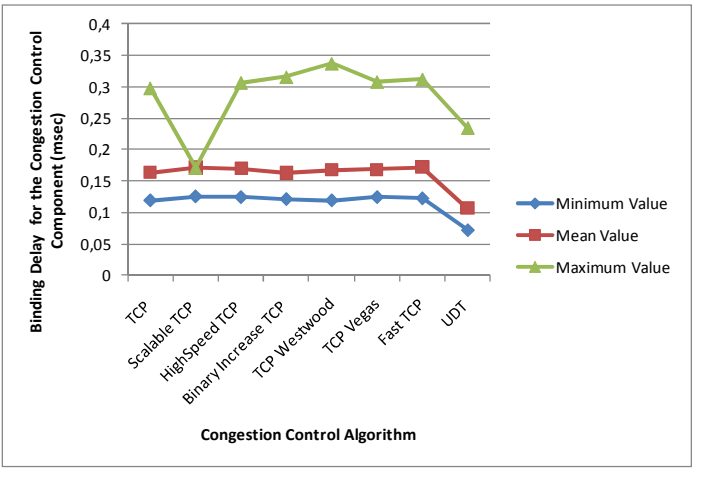


**Fig. 3.** Minimum, maximum and mean values of the binding delay for the congestion control component based on simulation results for different algorithms

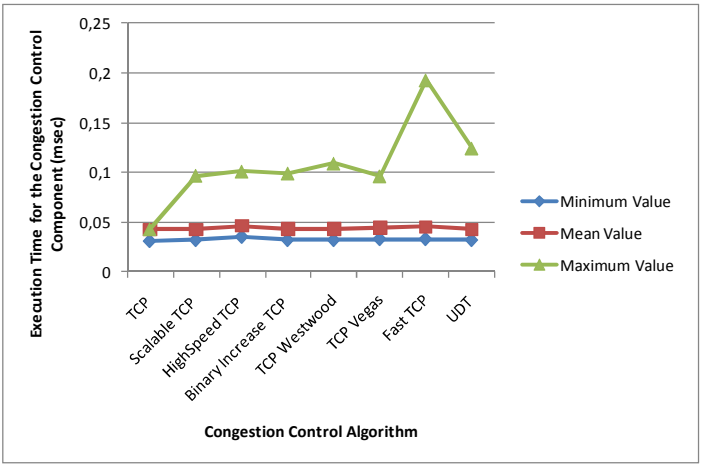


**Fig. 4.** Minimum, Maximum and Mean values of the Execution Time for the Congestion Control Component based on Simulation Results for Different Algorithms

The simulation results have proven the feasibility of the self-configuring protocols solutions. More specifically, the simulation results proved that the UDT protocol components are successfully bound together forming the full functionality of the UDT protocol. The correctness of the presented framework was also confirmed by the successful execution of the congestion control operations. It should be noted that the induced component binding and component execution delays were very low for all the tested congestion control algorithms and did not affect the overall system operation [13]. In addition, the simulation results have proven that the dynamic binding and execution of different types of functionalities/algorithms does not affect the performance of the reconfiguration procedures. This fact validates the feasibility and applicability of employed framework and mechanisms in a component-independent manner.

It should be noted that the evaluation of the concept through experiments in proof-of-concept prototype platforms has been also realised [13]. Such results complement the simulation analysis and have also proven the applicability of the protocol modularisation and self-configuration operations during real system operation. In addition, it should be noted that the presented framework does not affect the system functions (the applications runs smoothly despite the changes) - however the results in the RTT delay are significant. Based on real time measurements acquired during demonstrations, the RTT delay with CTCP is reduced to 1/3 of the RTT delay with CVegas.

## 5 Conclusions

This paper discusses the introduction of enhanced autonomous functionalities in mobile terminals to enable their intelligent protocol auto-configuration. Specifically, the basic system model for the support of such functionality and respective end-to-end signalling has been proposed. The operational stages required for protocol reconfiguration have been analyzed, discussed, and assessed through simulation results.

The results proved that the mechanisms for component-based protocol are light-weight, yielding a feasible approach. Such approach introduces negligible delay to the system, but it does not affect the viability of the system's operation.

## Acknowledgements

This work was funded by the European Commission Seventh Framework Programme ICT-2008-224344 through the Self-NET Project (https://www.ict-selfnet.eu). The authors would like to thank their partners in the IST SELF-NET project for insightful technical discussions that triggered the development for the material presented herein.

## References

1. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. 1(2), 223–259 (2006)
2. Patouni, E., Alonistioti, N., Merakos, L.: Cognitive Decision Making for Reconfiguration in Heterogeneous Radio Network Environments. IEEE Transactions on Vehicular Technology (TVT), special issue on Achievements and the Road Ahead: The First Decade of Cognitive Radio, PP(99) (2010)
3. Mahmoud, Q.: Cognitive Networks: Towards Self-Aware Networks. Wiley, Chichester (2007) ISBN: 978-0-470-06196-1
4. Miller, J., Thomson, P.: Beyond the complexity ceiling: evolution, emergence and regeneration. In: Proc. GECCO 2004 Workshop on Regeneration and Learning in Developmental Systems (2004)
5. Bouabene, G., Jelger, C., Tschudin, C., Schmid, S., Keller, A., May, M.: The autonomic network architecture (ANA). IEEE Journal on Selected Areas in Communications 28(1), 4–14 (2010)
6. Alonistioti, N., Patouni, E., Gazis, V.: Generic architecture and mechanisms for protocol reconfiguration. Special Issue on Reconfigurable Radio Technologies in Support of Ubiquitous Seamless Computing, Mob. Netw. Appl Journal 11(6), 917–934 (2006)
7. Gu, Y., Grossman, R.L.: UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks. The International Journal of Computer and Telecommunications Networking 51(7), 1777–1799 (2007)
8. Gazis, V., Patouni, E., Alonistioti, N., Merakos, L.: A Survey of Dynamically Adaptable Protocol Stacks. IEEE Communications Surveys and Tutorials 10(1), 3–23 (2010)
9. Sahlin, B.: A conduits+ and java implementation of internet protocol security and internet protocol, version 6, Master's thesis (1997), `http://citeseer.ist.psu.edu/286786.html`
10. Kliazovich, D., Granelli, F.: Distributed protocol stacks: A framework for balancing interoperability and optimization. In: IEEE International Conference on Communications Workshops 2008 (ICC Workshops 2008), pp. 241–245 (2008)
11. Niamanesh, M., Jalili, R.: A dynamic-reconfigurable architecture for protocol stacks of networked systems. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), Washington, DC, USA, vol. 1, pp. 609–612 (2007)

12. Mitola III, J.: Cognitive Radio for Flexible Mobile Multimedia Communications. Mob. Netw. Appl Journal 6(5), 435–441 (2001)
13. Patouni, E., Holland, O., Alonistioti, N.: Cognitive Functionalities for Mobile Terminal Self-Recovery and Protocol Auto-Configuration. In: The Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2008 (PIMRC 2008), Cannes, France (2008)
14. Brakmo, L., Malley, S.O., Peterson, L.: TCP Vegas: New techniques for congestion detection and avoidance. In: Proceedings of the SIGCOMM 1994 Symposium, pp. 24–35 (1994)
15. Tan, K., Song, J., Zhang, Q., Sridharan, M.: A Compound TCP Approach for High-speed and Long Distance Networks, Technical Report MSR-TR-2005-86, Microsoft Research (2005)
16. Kelly, T.: Scalable TCP: improving performance in highspeed wide area networks. ACM SIGCOMM Computer Communication Review 33(2) (2003)
17. Floyd, S.: HighSpeed TCP for Large Congestion Windows, RFC 3649, Experimental Standard (2003)
18. Xu, L., Harfoush, K., Rhee, I.: Binary Increase Congestion Control for Fast Long-Distance Networks. In: Proceedings of IEEE Infocom 2004, Hongkong, China (2004)
19. `http://www.cs.ucla.edu/NRL/hpi/tcpw/`
20. Jin, C., Wei, D.X., Low, S.H.: FAST TCP: motivation, architecture, algorithms, performance. In: Proceedings of IEEE Infocom 2004, Hongkong, China (2004)