Context-Adaptive Vehicular Network Optimization

Olivier Mehani, Roksana Boreli, Thierry Ernst

Context-Adaptive Vehicular Network Optimization

Olivier Mehani
Inria Rocquencourt, France, and
National ICT Australia, and
Mines ParisTech, France, and
University of New South
Wales, Australia
olivier.mehani@inria.fr

Roksana Boreli
National ICT Australia
Locked Bag 9013, Alexandria
NSW 1435, Australia, and
University of New South
Wales, Australia
roksana.boreli@nicta.com.au

Thierry Ernst
Inria Rocquencourt,
Imara Project-Team,
Domaine de Voluceau, BP 105,
78153, Le Chesnay Cedex, France
thierry.ernst@inria.fr

***Abstract*—We propose a framework to optimize the communication performance and mobility management in vehicular networks. By having a single unified decision algorithm taking into account both stack-related and external contextual information such as GPS localization or signaling from other nodes, advice can be provided to every layer in the network stack to allow for globally optimized, faster and more accurate adaptation to the current conditions. We present how key example scenarios would benefit from such a system. We describe an instance of this framework using a constraint satisfaction problem (CSP) approach. We also describe our prototype implementation of the network data collection system and give some timing evaluation for a given constraint solver.**

***Index Terms*—IPv6, mobility, multihoming, context-aware, performance optimization, constraint satisfaction**

I. Introduction

Vehicular networks present a heterogeneous environment based on a large variety of technologies. Several networking solutions can be used to establish communication channels with other vehicles (V2V), the infrastructure (V2I) or the internet. It is thus not uncommon to see vehicular communication solutions built on top of various physical technologies such as GPRS, UMTS, various flavors of the 802.11 standard or the more recent WiMAX.

However both IP- and non IP-based network protocols have been proposed, current standardization works advocate the use of IPv6 at the network layer [1]. Vehicles can then benefit from direct connectivity to the internet as well as native support for host and network mobility to maintain application-level communication. Having more than one network interface is additionally becoming common. Vehicles can then take part in several networks at once *e.g.*, a Vehicular Ad-hoc NETwork (VANET) for local communication and a WWAN for longer distance communication. Such multihoming, with peers potentially reachable via several disjoint networks, prompts the need for routes and interface selection support in order to achieve good performance *e.g.*, not using the WWAN when a direct VANET route exists [2].

Most currently available network stacks are designed following the layered design of the OSI model. As upper layers protocols aren't informed about the lower levels reconfigurations, it takes them time to detect and adapt to the new conditions on their own. Worse, when their parameters have been updated to reflect the new network conditions, it is the turn of the layers above to go through the same type of time-consuming stand-alone adaptation.

Due to the high dynamicity of vehicular environments, the completion of a full adaptation to new network conditions may be delayed beyond acceptable limits or even after said conditions have changed again. To overcome layer adaptability issues, a number of cross-layer designs have been proposed. Linking various protocols in order to pass relevant information directly, these approaches allow an improvement of the adaptation delay. It is however limited to the involved layers. Amongst others, the link and network/mobility management layers are usually coupled [3], [4], while other solutions target TCP's behavior over wireless networks [5], [6]. Most cross-layer solutions are however limited to a narrow set of use-cases and technologies and cannot be applied to more generic contexts. Additionally, as they do not take into account the rest of the system, two cross-layer mechanisms may interact with each other in unintended and destructive ways [7].

Some works have taken the cross-layer information passing out of the stack and delegated it to specific entities [8], [9], [10]. [11] proposed to use an external component in charge of collecting data from each layer and making it available to the entire stack in an abstracted form. The use of such abstraction allows to design adaptation mechanisms which are not tied to a specific layer implementation and protocol, hence more portable. Similarly, the Unified Link Layer API [12] can be used by applications desiring to get the necessary information to adapt their settings to the current conditions. In all these instances, though, optimization choices are still made locally by each application. This thus doesn't alleviate the risk of bad interactions.

To benefit from cross-layer information while ensuring globally valid optimization decisions, both the collection of stack information and the parameters adjustment decision have to be centralized. Protocol implementation are no longer responsible for interpreting the data collected from other layers. This loose coupling between information

source and parameters update has the advantage of limiting the modifications on standard implementations.

Additionally, as a number of non network-stack-related conditions also impact vehicular communication performances, information about the environment (such as GPS and map-matched localization) could benefit to the optimization process. Indeed, the rather strict structure of the road system and driving rules make it a good field for condition prediction based on previous observations, as was proposed in [13] for personnal mobile devices.

This article presents such a framework. Section II gives further motivation by presenting example scenarios which would benefit from such a system. Section III presents the overall architecture as well as how contextual information is unified for use by a constraint programming solver running as the decision process. Section IV presents an early implementation of a prototype of the framework. Finally, conclusions and future work are laid out in Section V.

II. Motivating Example Scenarios

This section presents examples of vehicular communication (V2V and V2I) where adaptation of upper layers to new lower layers configuration or the ability to solve problems spanning several layers would be desirable.

A. Transport Adaptation to Network Changes

Performing mobility handovers, switching route or simply getting one hop closer to a peer usually changes the characteristics of end-to-end paths. While the network layer is quickly made aware of changes, upper layers tend to react more slowly to such changes.

Fig. 1 on the following page presents three typical use-cases where, due to the mobility of the vehicles, paths to communication peers change. Such changes are likely to have an important impact on higher layers, starting with transport protocols.

In multihop VANETs (Fig. 1(a)), getting closer to (resp. away from) another node reduces (resp. increases) the number of hops. The number of times a message has to be relayed in the air directly influences the round-trip time (RTT) observed along the path. Additionally, with a smaller number of hops, messages contend less for the wireless medium, thus reducing the probability of collisions. This can lead to an increase in the end-to-end throughput. Given the number of hops, it is then possible to estimate parameters that the transport should use to profit from this opportunity for better communication. The path-change scenario has an even more drastic impact in the case of a change from a NEMO to a MANET route (Fig. 1(b)), as several hops over networks with various characteristics and routing or peering policies as well as the triangular routing problem that happens between mobility Home Agents are replaced by a few direct hops. Conversely, it is desirable to inform the transport of degradations of the paths early to prevent creating congestion along the new path.

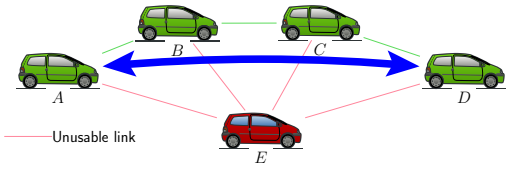


Fig. 2. Rogue node E advertises its presence but doesn't forward packets properly, thus hindering communication between A and D.

Correlating information from various layers can help make more educated handover decisions. When given the choice of several physical access points, previous observations can help choose the access network which will provide the best performance to current communications. In Fig. 1(c), a moving vehicle can decide to associate with a network it knows from previous observations has better connectivity to the one server it is currently communicating with.

B. Routing around a Rogue VANET Node

A very specific issue of unmanaged networks like VANETs is that of âĂIJrogueâĂİ nodes. Such a node participates in the route establishment algorithm but doesn't forward packets properly (either willingly or due to a misconfiguration), thus creating a "black hole". As shown in Fig. 2, rogue node E is on the shortest path from A to D. Their routing algorithm will then select that node to forward their packets to each other.

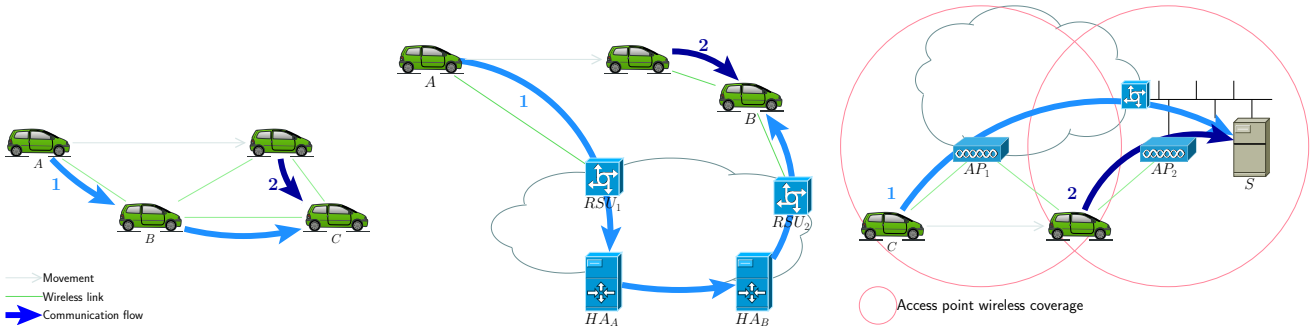
However, as E doesn't forward packets, communication establishment between A and D is impossible through it. IP routing being best-effort, the problem is not detected at the network layer. Without a human noticing that the application doesn't behave properly, the situation cannot be resolved (*e.g.* changing to a route not using E).

In that case a problem is observable at a given level, but solutions to the issue can only be undertaken at another layer, which isn't able to notice any malfunction. Using metrics from both the transport (*e.g.* unacknowledged packets) and network layer, A could correlate the problem with routing through node E, and adapt its routing table consequently to avoid it.

Extensions of this example can cover communication between the decision frameworks on A and B to synchronize their routes, as well as adaptation of the transport and application layers to parameters relevant to the new, longer, path, as previously emphasized.

III. Context-Based Optimization Framework

In this section, we present the architecture of a context-aware optimization framework. A general description of the components and their interaction is first given. The unification of synthetic context information and a proposal for a decision system based on a constraint satisfaction solver are then described in more details.



(a) Changes in the number of VANET hops impact the achievable end-to-end performance.

(b) MANEMO route switching can cause huge changes in the path conditions.

(c) Correlation of information can help choose the best combination of parameters.

Fig. 1. Several typical scenarios that could benefit from a better use of contextual information.

A. Overall Description

The proposed context-aware famework comprises several functionalities:

network state collection
: Information about the network stack (available networks and interfaces as well as network and transport metrics) need to be constantly updated to maintain a valid knowledge of the current conditions.

context collection
: Similarly, contextual data needs to be periodically updated from the available sources (such as battery monitor or localization) to keep an accurate description of external factors which may impact the performance.

context unification
: Once contextual information has been gathered from various sources, it is necessary to compose it into a synthetic representation to be used by the decision algorithm.

context history
: Optionally maintaining a history of the unified context (at a human scale *i.e.* days) can help the decision process predict future conditions. Such a history could be collectively built to allow provision of historical information even for new locations.

decision
: Main component of this framework, the decision algorithm takes synthetic contextual information (and possible future predictions) and derives the parameters the network stack should be configured with in order to use the available resources as efficiently as possible. One requirement for the decision algorithm is that it converges towards a stable solution and avoids oscillations.

parameters adjustment
: Once the parameters to use have been decided, they eventually need to be fed back to the network stack protocols to be taken into account. Maintain their genericity of the protocols is important. When adapted to take such intructions into account, they should not be modified so much that they can't function properly in a standalone fashion.

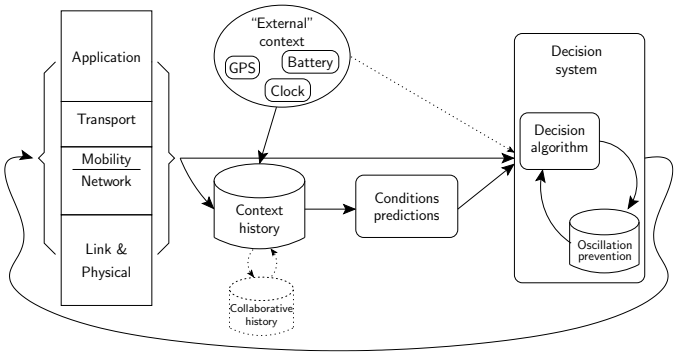


Fig. 3. The proposed architecture aggregates network as well as contextual and historical information. Based on this input, a decision algorithm then derives a good set of parameters for the various protocols and algorithms currently running within the network stack.

The functionnal blocks providing these functionnalities as well as their interactions are presented in Fig. 3.

The updatable parameters, as determined by the decision algorithm, span over all the stack. Common examples can be cited such as

- next hop (or full path) determination in a VANET,
- decision to perform a handover to a wireless network with (or expected to soon have) a better signal-to-noise level,
- route selection and flow binding to specific interfaces or networks;
- update of the transport parameters *e.g.* for TCP: RTT, congestion window, slow-start threshold (*sshthr*),
- adaptation of the application to the available network capacity (*e.g.* change of codec, compression or data or reduction of the sending frequency).

Another desirable feature is the possibility to signal other instances of the framework over the network in order to synchronize common states. For example, in the case of

TABLE I
EXAMPLE INFORMATION AVAILABLE FROM VARIOUS SOURCES.

	Source s	Abstracted information I_s
Network stack layers	Application	Throughput and delay requirements, Data sending pattern (amount/frequency), Correctness of received data, Possible configurations (*e.g.* codecs)
	Transport	End-to-end metrics such as throughput, timings (RTT, jitter) or error rate
	Network	Next hops or full routes (source routing)
	(*Mobility*)	Change of network address (CoA)
	Link	Network identifier (*e.g.* VLANs, ESSIDs), Data rate, retransmissions and delays, Link-local neighbors
	Physical	Modulation, Signal strength
Context	Clock	Current time t
	System	Battery level, load average
	External network	Upcoming hand-offs at an uplink router, Overheard concurrent communications
	GPS	Map-matched localization, heading, speed
	...	

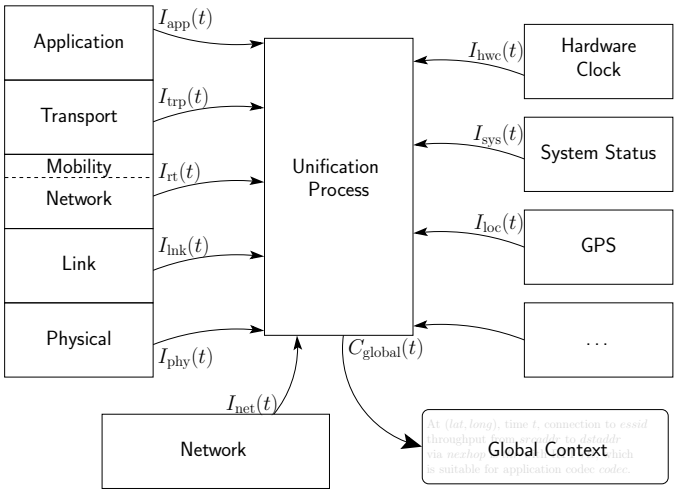


Fig. 4. Based on both in- and out-of-stack information $I_s(t)$, a unification process derives a description of the global context.

the binding of a flow or (resp.) change of next hop, one instance could signal its decision to its peer running on the mobility Home Agent or (resp.) the other involved vehicle for these changes to be mirrored. This is represented by $I_{\text{net}}(t)$ in Fig. 4.

B. Global Context from Separate Information

The global situation can be estimated from information both from the network stack and the external context. Table I lists typical examples of such information.

Each layer implementation, as well as external factors, have *states* varying with time. They represent the current dynamic conditions. States can be the mode a protocol is in, metrics currently observed (*e.g.* transport sending rate or physical signal strength from a neighbor) or current localization of the vehicle.

The global context information is scattered between the current observed states. It is necessary to aggregate all these data into a synthetic representation of the current situation. Fig. 4 shows a unification component which is in charge of collecting states from every source, and create such a representation. Achieved performances can thus be correlated with the associated settings and context.

As a simple example of the unification process, one can consider the situation in which node A communicates with B. The socket concept is used to represent and end-to-end connection ($s = \text{sock}(A, B)$). At time t, the collected information from some of the relevant layers can be represented as

$$I_{\text{app}}(s,t) = \{c = \text{codec}(t), \dots\},$$
$$I_{\text{trp}}(s,t) = \{thr = \text{throughput}(s,t), rtt = \text{rtt}(s,t), \dots\},$$
$$I_{\text{rt}}(t) = \{nh_B = \text{nextHop}(B,t), \dots\}.$$

By correlating data collected for socket s, the system could infer that a throughput of thr, as needed by codec c, is achievable towards node B along a path starting with nh_B:

$$C_{\text{global}}(t) = \{\{nh_B = \text{nextHop}(B,t), thr = \text{throughput}(s,t), c = \text{codec}(t), \dots\} \dots\}.$$

C. Constraint Satisfaction Problem

Both generic context and network stack layers have states. The latter also has *parameters*. They are sets of adjustable settings or possible features as well as their requirements on the other layers of the stack (*e.g.* possible video codecs and required bandwidth). The list of current parameter values and enabled features is however considered part of the states as it participates to the observed performance.

States lay out a set of constraints for the possible configurations of the stack. Not every combination of parameter values is intrinsicically valid but with the added constraints from the context, a number of others are temporarily ruled out (*e.g.* the use of a given codec when no underlying layer configuration can achieve at least the highest tolerated delay). The goal of the optimization process is to derive a valid set of parameters which would achieve good performance overall.

With an increasing number of interfaces, neighbors and routes, as well as several adjustable parameters for each layer protocol and a large number of concurrent connections, finding optimal solutions is computationally expensive. Indeed, it requires exploring an exponentially large number of configurations. For that reason, and given the formulation of the problem in terms of related constraints, it seems relevant to use a constraint solver as the decision engine.

Formally, a constraint satisfaction problem (CSP) is composed of a set of variables which can take values within specific ranges, and relations (constraints) between

these variables conditioning their simultaneous values [14]. Given this expression of the problem, a constraint solver is used to find valid solutions. A solution to a CSP is a binding of each variable with a value in its domain in such a way that all the constraints are satisfied. Table II on the next page gives an example of tabulated constraints derived from states and parameters of an application with three quality levels (codec) in a multihomed scenario with several routes to the destination.

To allow searching for a good (or optimal if time avails) solution, an objective function can be provided to be minimized by the solver. One such function to achieve a high throughput and low RTT while privileging the cheapest interfaces (*e.g.* in terms of money, user preferences or a combination of both) could be

$$\min\left(\alpha \cdot rtt - \beta \cdot thr + \gamma \cdot C_{if}\right)$$

where C_{if} is the cost of using the chosen interface and α, β, γ are weight parameters set according to the importance of each criterion (*e.g.* if the data flow is not large but has real-time requirements, $\alpha > \beta$).

Oscillation avoidance is not catered for directly by the constraint solver. This can however be modelled by adding additional constraints representing recent configurations and assigning high costs for returning into them.

IV. Prototype Implementation

A prototype implementation of the above framework is under development. Some non-crucial components have been omitted so far in order to focus one the main functions that are the network stack information collection and unification, and the decision process. The prototype is developped using the Python programming language for Linux 2.6(.30) systems.

A. Stack Information Collection

In order to collect stack information, the Netlink subsystem [15] has been used. This system provides an information bus between the kernel and userland applications accessible by opening a socket of the `AF_NETLINK` address family and binding to specific groups or querying information. More specifically, two channels are openned, using the route (`NETLINK_ROUTE`) and socket diagnostic (`NETLINK_INET_DIAG`) protocols.

The route socket is bound to the link, address, neighbor and route groups (`RTMGRP_LINK | RTMGRP_NEIGH | RTMGRP_IPV6_IFADDR | RTMGRP_IPV6_ROUTE`). Messages about these categories are sent to the relevant group whenever a parameter changes.

The diagnostic socket can be used to get in depth information about all sockets currently existing on the system. Unlike the route protocol, there is no group to bind to get information pushed by the kernel. It is necessary to to periodically send queries. TCP socket information obtained via the `TCPDIAG_GETSOCK` query details a number of metrics and parameters currently observed by the congestion control algorithm such as the congestion avoidance state, RTT or window sizes, amongst many others.

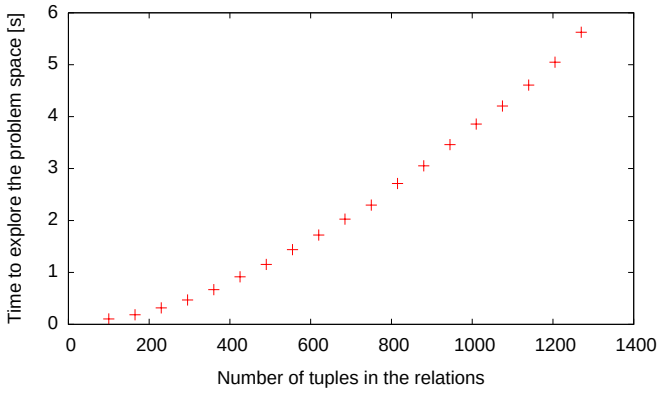


Fig. 5. Time taken for MiniZinc to explore the solution space for a varying numbers of constraints tuples (100 runs each).

B. Constraint Satisfaction Solver

The decision component of the prototype is based on the MiniZinc [16] constraint solver. The ruleset is similar to what is shown on Table II. It is however extended by a couple of relations representing the recent configuration for each socket, as well as a cost table for oscillations.

It is necessary that the solver is able to return solutions in timely enough a manner that the newfound configuration is still relevant. In order to check this property, timing tests have been run. For every run, coherent data files have been randomly generated to populate the model. To estimate a higher bound on the computation time, all solutions were requested and tree pruning was disabled. The computer used for these tests sports a 2 GHz Intel Core2 Duo and 1 GB of RAM under Linux (reported at 3991.24 "BogoMIPS"). The timing results are reported in Fig. 5. They present a reasonable computation time even for large problems (the last datapoint's 1270 constraints were created assuming 5 network interfaces, 190 known destinations and 95 sockets).

One problem of the current setup is that it currently can't function in an online loop. Support of dynamic constraints updates within MiniZinc is an ongoing research problem. Thus, for each iteration the solver has to be restarted with an updated set of data. This prevents benefitting from optimizations (for the solving algorithm) that could take advantage of the structure of the CSP.

V. Conclusion

We have presented a framework for network optimization taking into account parameters from within the networking stack as well as from the environment as reported by various subsystems. After unifying these data into a synthetic representation of the context, it is formulated as a constraint satisfaction problem for use by a solver along

TABLE II
EXAMPLES CONTRAINTS EXPRESSED AS TABULATED RELATIONS FOR A SIMPLIFIED EXAMPLE.

OBSERVED NETWORK PERFORMANCES

Destination	Route	Interface	Throughput	Jitter	RTT	PER
Addr1	NH2	eth0	2 Mbps	1×10^{-4} s	10×10^{-3} s	0%
Addr1	NH1	wlan0	900 kbps	1×10^{-3} s	100×10^{-3} s	10%
Addr2	NH1	wlan0	450 kbps	1×10^{-3} s	250×10^{-3} s	30%
			...			

INTERFACE COSTS

Interface	Cost
eth0	10
wlan0	100
ppp0	250
...	

APPLICATION APP1 PARAMETERS AND REQUIREMENTS

Quality	Throughput	Jitter	RTT	PER
1	$\geq$ 1.5 Mbps	$\leq 10^{-3}$	$\leq 10 \times 10^{-4}$ s	$\leq 10 \times 10^{-3}$
2	$\geq$ 1 Mbps	idem	idem	idem
3	$\geq$ 500 kbps	$\leq 10^{-2}$	$\leq 10 \times 10^{-3}$ s	idem

SOCKET BETWEEN APPLICATIONS AND DESTINATIONS

Socket	Application	Destination
1	App1	Addr1
	...	

with a cost function to identify good solutions. Based on the best found solution, parameters of the network stack are adjusted to achieve better performance.

An early prototype implementation under Linux has been presented. Additionaly, the computation time of the chosen constraint solver has been evaluated with resaonable numbers of constraints to ensure the feasability of this approach.

Future work will focus on refining the prototype into a complete implementation of our framework. This includes integrating more contextual information and a non-naive data unification process. We plan to evaluate the performance of our global optimization in real experiments and compare the results with similar control systems like the CALM Manager.

ACKNOWLEDGEMENT

The authors would like to thank Michael Maher for the insight he provided about constraint programming and the initial MiniZinc model of the problem.

REFERENCES

[1] ISO/TC204 WG16, "ISO/DIS 21210:2009: Intelligent transport systems — Communications Access for Land Mobiles — CALM IPv6 Networking," 2009. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=46549

[2] M. Tsukada, O. Mehani, and T. Ernst, "Simultaneous usage of NEMO and MANET for vehicular communication," in *TridentCom 2008, 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, March 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1390576.1390592

[3] L. Qin and T. Kunz, "Survey on mobile ad hoc network routing protocols and cross-layer design," Carleton University, Tech. Rep. SCE-04-14, August 2004. [Online]. Available: http://kunz-pc.sce.carleton.ca/Thesis/RoutingSurvey.pdf

[4] N. Montavont and T. Noël, "Stronger interaction between link layer and network layer for an optimized mobility management in heterogeneous IPv6 networks," *Pervasive and Mobile Computing*, vol. 2, no. 3, pp. 233–261, September 2006. [Online]. Available: http://dx.doi.org/10.1016/j.pmcj.2006.02.001

[5] M. Chiang, "To layer or not to layer: balancing transport and physical layers in wireless multihop networks," in *INFOCOM 2004, 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4, March 2004, pp. 2525–2536. [Online]. Available: http://dx.doi.org/10.1109/INFCOM.2004.1354673

[6] A. Baig, L. Libman, and M. Hassan, "Performance enhancement of on-board communication networks using outage prediction," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 9, pp. 1692–1701, September 2006. [Online]. Available: http://dx.doi.org/10.1109/JSAC.2006.875108

[7] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross-layer design," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 3–11, February 2005. [Online]. Available: http://dx.doi.org/10.1109/MWC.2005.1404568

[8] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for Internet hosts," *SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 175–187, October 1999. [Online]. Available: http://dx.doi.org/10.1145/316194.316220

[9] B. L. Tierney, D. Gunter, J. Lee, M. Stoufer, and J. B. Evans, "Enabling network-aware applications," in *HPDC-10 (2001), 10th IEEE International Symposium on High Performance Distributed Computing*, April 2001, pp. 281–288. [Online]. Available: http://dx.doi.org/10.1109/HPDC.2001.945196

[10] T. Dunigan, M. Mathis, and B. Tierney, "A TCP tuning daemon," in *SC 2002, ACM/IEEE conference on Supercomputing*, November 2002, pp. 9–25. [Online]. Available: http://dx.doi.org/10.1109/SC.2002.10023

[11] E. Borgia, R. Bruno, M. Conti, F. Delmastro, E. Gregori *et al.*, "MobileMAN architecture, protocols and services," EC âĂİ Information Society TechnologiesâĂİ Programme, Tech. Rep., October 2004. [Online]. Available: http://cnd.iit.cnr.it/mobileMAN/deliverables/MobileMAN_Deliverable_D5.pdf

[12] M. Sooriyabandara, T. Farnham, M. Wellens, J. Riihijärvi, P. Mähönen *et al.*, "Unified link layer API: A generic and open API to manage wireless media access," *Computer Communications*, vol. 31, no. 5, pp. 962–979, March 2008. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2007.12.025

[13] S. Herborn, H. Petander, and M. Ott, "Predictive context aware mobility handling," in *ICT 2008, 15th IEEE International Conference on Telecommunications*, 2008, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ICTEL.2008.4652647

[14] B. M. Smith, "A tutorial on constraint programming," University of Leeds, Tech. Rep. 95.14, April 1995. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.9886

[15] J. H. Salim, H. M. Khosravi, A. Kleen, and A. Kuznetsov, "Linux Netlink as an IP services protocol," RFC 3549 (Informational), July 2003. [Online]. Available: http://tools.ietf.org/html/rfc3549

[16] K. Marriott, N. Nethercote, R. Rafeh, P. J. Stuckey, M. García de la Banda *et al.*, "The design of the Zinc modelling language," *Constraints*, vol. 13, no. 3, pp. 229–267, September 2008. [Online]. Available: http://dx.doi.org/10.1007/s10601-008-9041-4