Context-Adaptive Vehicular Network Optimization
Olivier Mehani, Roksana Boreli, Thierry Ernst

To cite this version:

HAL Id: inria-00426451
https://inria.hal.science/inria-00426451
Submitted on 26 Oct 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 personal 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 changes. 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.

B. Routing around a Rogue VANET Node

A very specific issue of unmanaged networks like VANETs is that of rogue 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. Overall Description

The proposed context-aware framework 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 instructions into account, they should not be modified so much that they can’t function properly in a standalone fashion.

The functional blocks providing these functionalities 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...
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 $I_{net}(t)$ 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 ($s = sock(A, B)$). At time $t$, the collected information from some of the relevant layers can be represented as

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

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 $nh_B$:

$C_{global}(t) = \{\{nh_B = nextHop(B, t), \} \}$.

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 intrinsically 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
\[
\min (\alpha \cdot \text{rtt} - \beta \cdot \text{thr} + \gamma \cdot C_{if})
\]
where \(C_{if}\) is the cost of using the chosen interface and \(\alpha, \beta, \gamma\) 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 on the main functions that are the network stack information collection and unification, and the decision process. The prototype is developed 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_INET_DIAG address family and binding to specific groups or querying information. More specifically, two channels are opened, 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_INTERFACE | 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 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.

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 benefiting 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
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. Additionally, the computation time of the chosen constraint solver has been evaluated with reasonable numbers of constraints to ensure the feasibility 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-naïve 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


### TABLE II
Examples of constraints expressed as tabulated relations for a simplified example.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Route</th>
<th>Interface</th>
<th>Throughput</th>
<th>Jitter</th>
<th>RTT</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr1</td>
<td>NH2</td>
<td>eth0</td>
<td>2 Mbps</td>
<td>$1 \times 10^{-4}$ s</td>
<td>$10 \times 10^{-3}$ s</td>
<td>0%</td>
</tr>
<tr>
<td>Addr1</td>
<td>NH1</td>
<td>wlan0</td>
<td>900 kbps</td>
<td>$1 \times 10^{-3}$ s</td>
<td>$100 \times 10^{-3}$ s</td>
<td>10%</td>
</tr>
<tr>
<td>Addr2</td>
<td>NH1</td>
<td>wlan0</td>
<td>450 kbps</td>
<td>$1 \times 10^{-3}$ s</td>
<td>$250 \times 10^{-3}$ s</td>
<td>30%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>eth0</td>
</tr>
<tr>
<td>wlan0</td>
</tr>
<tr>
<td>ppp0</td>
</tr>
</tbody>
</table>

### APPLICATION App1 parameters and requirements

<table>
<thead>
<tr>
<th>Quality</th>
<th>Throughput</th>
<th>Jitter</th>
<th>RTT</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>≥ 1.5 Mbps</td>
<td>≤ $10^{-3}$</td>
<td>≤ $10 \times 10^{-4}$ s</td>
<td>≤ $10 \times 10^{-3}$ s</td>
</tr>
<tr>
<td>2</td>
<td>≥ 1 Mbps</td>
<td>idem</td>
<td>idem</td>
<td>idem</td>
</tr>
<tr>
<td>3</td>
<td>≥ 500 kbps</td>
<td>≤ $10^{-2}$</td>
<td>≤ $10 \times 10^{-3}$ s</td>
<td>idem</td>
</tr>
</tbody>
</table>

### SOCKET BETWEEN APPLICATIONS AND DESTINATIONS

<table>
<thead>
<tr>
<th>Socket</th>
<th>Application</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>App1</td>
<td>Addr1</td>
</tr>
</tbody>
</table>