Ad Hoc Networks 13 (2014) 444–461

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Harnessing cross-layer-design

Ismet Aktas [*], Muhammad Hamad Alizai, Florian Schmidt, Hanno Wirtz, Klaus Wehrle

Communication and Distributed Systems (ComSys), RWTH Aachen University, Germany

ARTICLE INFO

Article history:
Received 11 April 2013
Received in revised form 7 August 2013
Accepted 20 September 2013
Available online 29 September 2013

Keywords:
Cross-layer design
Cross-layer conflict detection
Software architecture
Runtime adaptive systems
Cross-layer interactions

ABSTRACT

Applications and protocols for wireless and mobile systems have to deal with volatile environmental conditions such as interference, packet loss, and mobility. Utilizing cross-layer information from other protocols and system components such as sensors can improve their performance and responsiveness. However, application and protocol developers lack a convenient way of specifying, monitoring, and experimenting with optimizations to evaluate their cross-layer ideas.

We present CRAWLER, a novel experimentation architecture for system monitoring and cross-layer-coordination that facilitates evaluation of applications and wireless protocols. It alleviates the problem of complicated access to relevant system information by providing a unified interface to application, protocol and system information. The versatile design of this interface further enables a convenient and declarative way to specify and experiment with compositions of cross-layer optimizations and their adaptions at runtime. CRAWLER also provides the necessary support to detect cross-layer conflicts, and hence prevents performance degradation when multiple optimizations are enabled across the protocol stack. We demonstrate the usability of CRAWLER for system monitoring and cross-layer optimizations with three use cases from different areas of wireless networking.

1. Introduction

Developing real-world protocols and applications for wireless and mobile systems is difficult. The volatile nature of the wireless medium as well as network and channel dynamics induced by mobility complicate their comprehensive development. This is further aggravated by the isolated nature of today's applications, protocols, and the operating system. Although the isolation of applications from each other, protocols, and the operating system attains reasonable software engineering advantages, it disregards (i) access to relevant system information, such as protocol states, for monitoring and experimentation and (ii) coordination among different components to optimize the performance in the face of changing conditions or mobility effects.

In order to achieve in vivo access to such relevant system information, network analysis tools, such as wireshark [1], allow the inspection of traffic specific points in the protocol stack. However, such tools lack the ability to monitor protocol states, variables, and system components, e.g., battery, motion indicators, and CPU utilization. This is mainly because the protocol stack and system component drivers are deeply integrated into the operating system which strongly limits external access to their internal states. Therefore, application and system developers are unable to access vital system information for monitoring, experimentation and performance optimization.

However, breaking this isolated layer and application paradigm, recent research [2] has shown cross-layer information, i.e., information provided also over non-adjacent layers, to allow both diverse applications and protocols to be significantly more adaptive. For example, in mobile and wireless systems, even a single cross-layer optimization at the MAC layer can achieve TCP throughput

* Corresponding author. Tel.: +49 241 80 21418.
E-mail address: Aktas@comsys.rwth-aachen.de (I. Aktas).

1570-8705/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2013.09.003

speedups of up to 20 times and latency reduction of up to 10 times over unmodified systems [3]. However, despite its proven potential to enhance system performance and a fair share of research investment in recent years, the cross-layer paradigm has not been able to leverage its utility beyond few promising yet concentrated research efforts [4–8].

Although several static cross-layer architectures have been proposed, networking researchers and application developers lack a generic and flexible architecture that enables specification and experimentation with cross-layer optimizations. Specifically, existing *static* cross-layer architectures [5,9–11] facilitate manipulation of protocol-stack parameters and combine several dedicated cross-layer optimizations. However, in current architectures of this type, cross-layer optimizations are composed offline (i.e., at compile time) and are deeply embedded within the operating system (OS). This approach has three key limitations that motivate the ideas presented in this article.

First, the process of adding or removing an optimization is impractical: optimizations are hard-wired with the architecture, and because the architecture is deeply embedded into the OS, recompiling the kernel and rebooting the system are typical consequences when changing optimizations. Furthermore, the developer has to deal with too many system internals such as OS programming language, application programming interfaces (APIs) and primitives before actually experimenting with cross-layer optimizations.

Second, because of this static nature of the existing architectures, an optimization will change the system behavior even if it is not needed or intended to take effect. Precisely, an optimization that is specific to an application or environment is not required when that application is not running or the underlying conditions have changed. For example, energy saving optimizations may not be necessary if the device is plugged into a power supply. Therefore, this optimization and its interaction with the network stack is superfluous and may even adversely affect other active applications. We strongly believe that this is against the original spirits of the cross-layer paradigm [12] which emphasize the need for dynamic adaptation of the system behavior (i.e., protocols, system components, and applications) based on the current application requirements and the network conditions.

Third, compile-time installation of optimizations significantly complicates the detection of cross-layer conflicts, i.e., possible performance degradations [13] caused by multiple, contradicting optimizations. Detecting such conflicts thereby remains one of the major unresolved challenges in the cross-layer development domain [4,8,13].

In this article we present CRAWLER, a novel experimentation architecture for system monitoring and cross-layer-coordination that facilitates the evaluation of applications and wireless network protocols. CRAWLER thereby benefits developers of wireless and mobile applications, protocols, and systems and supports them in experimenting with and evaluating their cross-layering ideas. Specifically, CRAWLER provides the following key features that illustrate its departure from the existing work and mark the contributions of this article.

- CRAWLER simplifies the process of monitoring and experimentation by providing a unified interface for accessing application, protocol, and system information, independent from the OS internals.
- The generic, versatile design of this interface further facilitates specifying cross-layer optimizations by providing a declarative way of composing a set of optimizations and their adaption and adaptability at runtime.
- It offers (i) a very high degree of flexibility, to fluently experiment with changing compositions of cross-layer optimizations and (ii) extensibility, to include and remove heterogeneous protocol and system components in order to find the right set of optimizations for a certain use-case. Hence, CRAWLER is well suited as a rapid prototyping tool for application and system developers.
- It enables cross-layer conflict detection support to provide feedback to the developers regarding conflicting interdependencies when experimenting with multiple, concurrent cross-layer optimizations.

The remainder of this article is organized as follows. Section 2 presents a system overview, highlights our design goals, and comprehends the scope of our architecture. Based on our design goals, Section 3 describes our architecture from a conceptual point of view. The practical value of CRAWLER is demonstrated in Section 4 where three different use cases from divers networking fields are presented. In Section 5 we show how CRAWLER supports a developer to detect conflicting interdependencies between multiple cross-layer optimizations. The implementation details and the architectural overhead of CRAWLER are presented in Section 6. Finally, we discuss related work in Section 7 before concluding the article in Section 8.

2. Design overview

CRAWLER consists of two main components as shown in Fig. 1: the *logical component* (LC) allows cross-layer developers to express their monitoring and optimization needs in an abstract and declarative way. For this purpose, we have created a rule-based language customized to cross-layer design purposes. Using this language, developers can specify cross-layer signaling at a high level without needing to care about implementation details. Additionally, the LC offers a uniform interface that allows applications (i) to provide their own optimizations on demand and (ii) exchange information with the protocol stack, system components and other applications.

The cross-layer optimizations as specified in the LC are realized by the *cross-layer processing component* (CPC). Here, rules are mapped to compositions of self-written *functional units* (FUs). Finally, *stubs* provide read/write access to protocol information and sub-system states via a generic interface that abstracts from a specific implementation. Thus, additions and changes in optimization rules can be done at runtime using the LC. These changes are reported to the CPC, which adapts the FU compositions accordingly.