Alexander Clemm
Ralf Wolter *Editors*

Advances in Network-Embedded Management and Applications

Proceedings of the First International Workshop on Network-Embedded Management and Applications
October 28, 2010, Niagara Falls, Canada

Advances in Network-Embedded Management and Applications

Alexander Clemm • Ralf Wolter
Editors

Advances in Network-Embedded Management and Applications

Proceedings of the First International Workshop on Network-Embedded Management and Applications
October 28, 2010, Niagara Falls, Canada

Editors
Alexander Clemm
Cisco Systems
Los Gatos
USA
alex@cisco.com

Ralf Wolter
Kalstert 1446
40724 Hilden
Germany
rwolter@cisco.com

ISBN 978-1-4419-7752-6 e-ISBN 978-1-4419-7753-3
DOI 10.1007/978-1-4419-7753-3
Springer New York Dordrecht Heidelberg London

© Springer Science+Business Media, LLC 2011
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

It is a great pleasure to present the proceedings of the 1st International Workshop on Network-Embedded Management and Applications, NEMA. NEMA was held on October 28, 2010, in Niagara Falls, Canada, in conjunction with the 6th International Conference on Network and Service Management (CNSM), the former Manweek. It was technically co-sponsored by the IEEE Communications Society and by IFIP. The goal of NEMA was to bring together researchers and scientists from industry and academia to share views and ideas and present their results regarding management (and other) applications that are embedded inside the network, as opposed to merely attached to a network. It is the first workshop dedicated to this particular topic. The workshop's Web site can be accessed at http://nema.networkembedded.org/, where also future editions will be announced.

The motivation behind NEMA is the general trend of modern network devices to become increasingly "intelligent" and programmable. Examples range from router scripting environments to fully programmable server blades. As a result, networked applications are no longer constrained just to servers that are interconnected via a network, but can migrate into and become embedded within the network itself. This promises to accelerate the current trend towards systems that are increasingly autonomous and to a certain degree self-managing. There are several drivers behind this trend: Equipment vendors continue to add value to the network to counter commoditization pressures. Network and service providers desire to adapt and optimize networks ever more closely to their specific environment. The emergence of cloud in the data center context has provided powerful evidence how programmable networking infrastructure which facilitates automation of management tasks can lead to entire new business models. In addition, there is growing recognition of the importance to make network operation and administration as easy as possible to contain operational expenses, pushing functions into the network that used to be performed outside, and to be able cope with control cycles that need to keep getting shorter from the time that observations are made to the time action occurs.

As network devices are being increasingly opened up to in a way that allows them to be programmed, the network itself is becoming a platform for a whole new ecosystem of network-embedded applications serving management and other purposes. The next frontier lies in applications that go beyond traditional management and control functions and that are becoming increasingly decentralized, not constrained in scope to individual systems. Examples include decentralized monitoring, gossip-based configuration, network event correlation inside the network across multiple systems, overlay control protocols, and network-aware multi-media applications. At the same time, another trend looks at leveraging increased programmability of networks, specifically programmability of data and control plane, to add more networking intelligence also outside, not inside the network. This is an exciting time for both researchers and practitioners, as these trends pave the way for another wave of exciting new opportunities for innovation in networking.

The six papers that were selected from the submissions to NEMA represent a wide cross section of varying interpretations of this theme and are divided into two parts. Part One covers enablers for network-embedded management applications – the platforms, frameworks, development environments which facilitate the development of network-embedded management and applications. Starting with the general topic of how to instrument systems for management purposes and transition from legacy command-driven to model-driven architectures, it proceeds with a set of papers that introduce specific examples of hardware- and software based programmable platforms, namely a programmable low-power hardware platforms, as well as an application framework for programmable network control that allows application developers to create complex and application-specific network services. Part Two covers network-embedded applications that might leverage and benefit from such enabling platforms, ranging from the determination of where to optimally place management control functions inside a network, then discussing how multi-core hardware processors can be leveraged for traffic filtering applications, finally concluding with an application of delay-tolerant networks in the context of the One Laptop Per Child Program.

We hope that you will enjoy these proceedings and find the presented ideas stimulating and thought-provoking. We would like to thank the authors of the papers without whom the program would not have been possible, the members of the NEMA Technical Program Committees who provided high-quality reviews that enabled us to make the final paper selection from the submissions that were received, and the organizers of CNSM who were hosting NEMA and allowed us to use their conference facilities. In particular, we would like to thank the team at Springer, first and foremost Brett Kurzman, without whom these proceedings would not have been possible and who in many ways got the ball rolling in the first place.

August 2010 Alexander Clemm and Ralf Wolter

Table of Contents

Preface ………………………………………………………………………………. v

Part One: Enablers

Chapter 1
Challenges and Experiences in Transitioning Management Instrumentation from Command-Oriented to Model-Driven ……………..………………………. 1
Sean McGuiness, Jung Tjong, Prakash Bettadapur

Chapter 2
A Low Power, Programmable Networking Platform and Development Environment …………………………………………………………………….. 19
Pál Varga, István Moldován, Dániel Horváth, Sándor Plósz

Chapter 3
Application Framework for Programmable Network Control ……………….…. 37
Rudolf Strijkers, Mihai Cristea, Cees de Laat, Robert Meijer

Part Two: Applications

Chapter 4
Facilitating Adaptive Placement of Management and Control Functions in Converged ICT Systems ……....…………………………………………………. 53
Dominique Dudkowski, Marcus Brunner

Chapter 5
Wire-Speed Hardware-Assisted Traffic Filtering with Mainstream Network Adapters ………………………………………………………………………………. 71
Luca Deri, Joseph Gasparakis, Peter Waskiewicz Jr, Francesco Fusco

Chapter 6
Embedded Rule-based Management for Content-based DTNs……….…….…… 87
Jorge Visca, Guillermo Apollonia, Matias Richart, Javier Baliosian, Eduardo Grampín

Contributors

Guillermo Apollonia, University of the Republic, Uruguay
Javier Baliosian, University of the Republic, Uruguay
Prakash Bettadapur, Cisco, USA
Marcus Brunner, NEC, Germany
Mihai Cristea, University of Amsterdam, The Netherlands
Luca Deri, ntop, Italy
Dominique Dudkowski, NEC, Germany
Francesco Fusco, IBM Research and ETH Zurich, Switzerland
Joseph Gasparakis, Intel, Ireland
Eduardo Grampin, University of the Republic, Uruguay
Dániel Horváth, Budapest Univ. of Technology and Economics, Hungary
Cees de Laat, University of Amsterdam, The Netherlands
Sean McGuiness, Cisco, San Jose/CA, USA
Robert Meijer, TNO and Univ of Amsterdam, The Netherlands
István Moldován , Budapest Univ. of Technology and Economics, Hungary
Sándor Plósz , Budapest Univ. of Technology and Economics, Hungary
Matias Richart, University of the Republic, Uruguay
Rudolf Strijkers, TNO and Univ of Amsterdam, The Netherlands
Jung Tjong, Cisco, San Jose/CA, USA
Pál Varga, Budapest Univ. of Technology and Economics, Hungary
Jorge Visca, University of the Republic, Uruguay
Peter Waskiewicz Jr, Intel, USA

Reviewers and NEMA Program Committee Members

Raouf Boutaba, University of Waterloo, Canada
Marcus Brunner, NEC Europe Ltd, Germany
Alexander Clemm, Cisco, USA
Waltenegus Dargie, Technical University of Dresden, Germany
Metin Feridun, IBM Research, Switzerland
Olivier Festor, INRIA Nancy, France
Silvia Figueira, Santa Clara University, USA
Luciano Paschoal Gaspary, UFRGS, Brazil
Lisandro Zambenedetti Granville, UFRGS, Brazil
Sven Graupner, HP Laboratories, USA
Masum Hasan, Cisco, USA
Bruno Klauser, Cisco, Germany
Jean-Philippe Martin-Flatin, Consultant, Switzerland
John McDowall, Cisco, USA
Aiko Pras, University of Twente, The Netherlands
Danny Raz, Technion, Israel
Jennifer Rexford, Princeton University, USA
Gabi Dreo Rodosek, Univ. of Federal Armed Forces, Munich, Germany
Volker Sander, FH Aachen University of Applied Sciences, Germany
Akhil Sahai, VMware Inc, USA
Joan Serrat, Universitat Politecnica de Catalunya, Spain
Rolf Stadler, KTH, Sweden
Radu State, Luxembourg University, Luxembourg
Burkhard Stiller, University of Zurich, Switzerland
Carl Sutton, F5 Networks, USA
Ralf Wolter, Cisco, USA
Xiaoyun Zhu, WMware Inc, USA

Chapter 1
Challenges and Experiences in Transitioning Management Instrumentation from Command-Oriented to Model-Driven

Sean McGuiness, Jung Tjong, Prakash Bettadapur

Cisco Systems Inc,
170 West Tasman Drive,
San Jose, CA 95134-1706, USA
{smcguine, jtjong, pbettada}@cisco.com

Abstract. The popularity of model-driven development has grown significantly in recent years pushing its rapid adoption in the management instrumentation space. While standards and tooling have been created for virgin management instrumentation applications, little has been done to address the challenges of transitioning existing applications into the model-driven arena. With management interfaces constructed with divergent stovepipe implementations to meet their differing requirements and data characteristics, moving the entire system to a model-driven environment is an expensive and impractical proposition. Discussed are the design challenges and implementation experiences encountered during the successful transition of a legacy management instrumentation system to a model-driven system, including major design choices and the rationale behind them.

Keywords: Management, instrumentation, modeling, command-oriented, design, development, CLI, SNMP, MIB, legacy, transition

1 Introduction

Historically, much of network management instrumentation has revolved around command-oriented interfaces. As Model-Driven Engineering (MDE) has gained popularity in the management instrumentation community, designers are looking to transition existing command-oriented management instrumentation applications to MDE-based designs. Transitioning systems where targeted commands have been constructed to directly manipulate or

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_1, © Springer Science+Business Media, LLC 2011

obtain hard-coded reports of configuration and operational data from specific instrumented features is a challenging proposition. While seemingly model-friendly data-oriented interfaces such as Simple Network Management Protocol (SNMP) exist, these interfaces tend to be confined to data monitoring rather than configuration, provide less functional coverage and are frequently developed as distinctly separate and parallel instrumentation paths from their command-oriented counterparts. Over time, this parallelism degrades the quality, reliability and consistency of the system and complicates transitions to a model-driven design.

Inconsistencies across interfaces are demonstrated when data and functions are exposed in one management interface path but not others, like Command Line Interfaces (CLI) providing instrumentation in one form while the same instrumentation is supplied in an SNMP management interface in a different form - or perhaps not at all.

Different and multiply redundant instrumentation results in different and redundant request processing, inconsistent request handling, and duplicate configuration synchronization and maintenance requirements. The handling of a CLI instrumentation request, for example, involves certain parameter and system state validation coupled with a specific response however; duplicate constraint validation and default processing can result in inconsistent handling and duplicate maintenance requirements. CLI has one instrumentation data access method while SNMP has another. With duplicated instrumentation access, in order to ensure consistency, quality and reliability, changes to instrumentation data and data constraints, must be applied and tested in all interfaces that access instrumentation. This burden is compounded as instrumentation and management interfaces grow.

The introduction of modeling concepts into a system constructed around a command-oriented paradigm will be met with difficulty, as there is a significant impedance mismatch between them. Command-oriented instrumentation uses specific management interface commands tailored for particular features that access data and services directly. Conversely model-driven instrumentation focuses on access to feature data and services through a common abstract interface shared by all management interfaces. Modeled instrumentation describes data and services for all management interfaces. The primary transition problem to overcome is determining the origin of the instrumentation model. One may utilize the characteristics of data and services embedded within the command-oriented implementations; one may create a model based on need and map implementation data and services to it. The choice is hindered by the inherent impedance mismatch introduced by multiple and inconsistent implementations of the command-oriented system

and differences between the implicit and imposed models.

A model-driven system requires an underlying implementation in order to access instrumentation data and services. This cannot be easily leveraged from the command-oriented implementation due to its parallel nature. Retrofitting model-driven instrumentation on a legacy application constructed with a hard-coded command-oriented instrumentation is difficult. Defining an accurate model based on the actual implementation is the most pressing problem. Designers are faced with the choice of whether to use existing implementations preserving their inconsistencies and redundancies or to address these problems by creating a new streamlined model-driven implementation from their domain knowledge and experience.

In this paper the understanding of how management instrumentation system designs are impacted by this transition is discussed. Challenges and experiences will be examined through the prism of a real-world development effort of transitioning a command-oriented management instrumentation system to a model-driven management instrumentation system.

The remainder of this paper is structured as follows: Section 2 provides some background on the existing command-oriented instrumentation methodology. Section 3 covers the design considerations for model-driven instrumentation derived from command-oriented instrumentation systems while Section 4 describes key transition implementation experiences. Section 5 covers work in related areas of interest and Section 6 concludes the paper.

2 Background

Prior to discussing the transition challenges and experiences, for a better understanding of the problem domain, a brief overview of command-oriented instrumentation methodology is provided.

Management Instrumentation interfaces constructed around command-orientation do not typically adhere to crisp layered interface and object-oriented data-hiding principles. Management instrumentation systems often begin with the simplest management method and grow as requirements grow, starting with support of feature-based commands targeted for particular command-line driven instrumentation needs. These CLI interfaces involve the parsing of a user-entered command that link to a monolithic action function. These action functions perform validation of system and feature state and either configure some data or display a hard-coded report describing

instrumented feature. These action functions are intimately linked with the parse result of their associated command line and the instrumentation they access.

The action function for all CLIs that manipulate data and services of a management instrumentation component provide the component's implicit management model. Transitioning to a model-driven system, designers define and impose a model outside of the command-oriented framework and based on instrumentation domain knowledge and management requirements. They ultimately face the issue of adapting their defined model to the implicit model of the CLI implementation. Difficulties arise when the actual instrumentation capabilities of the implicit model are not reflected in the defined model they are imposing and vice-versa.

A similar example of this impedance mismatch can be seen in today's management instrumentation environment in the area of SNMP MIB support. An SNMP MIB is a model specified outside of the domain of any specific instrumentation implementation. A MIB can specify access to management data and service capabilities that may or may not be reflected in the system instrumentation. Implementers of SNMP MIB interfaces must provide mapping between the imposed model of data and services requested by the MIB and the model implicit in the implementation.

Resolving this impedance mismatch – the differences between the implicit and imposed management instrumentation models is perhaps the largest and most complex challenge of the command-oriented to model-driven transition.

3 Design Considerations

Model-driven management instrumentation designs are approached from a practical inverse of command-oriented designs due to their focal differences. A command-oriented design focuses on particular management interface commands and their associated instrumentation with respect to a particular feature. Conversely, a model-driven design focuses on the feature instrumentation to be made available for all management interfaces. This section discusses key design considerations when transitioning from a command-oriented to a model-driven design.

Figures 1 and 2 below provide a comparison of command-oriented and model-driven architectures. The command-oriented architecture depends upon access to feature instrumentation data being tightly coupled with the management interface. In comparison, the management interfaces of the

model-driven architecture utilize a loosely coupled common abstract interface to access feature instrumentation.

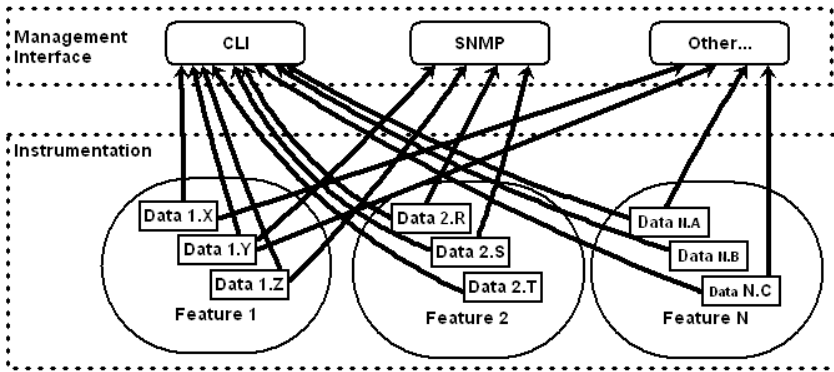


Fig. 1. Command-oriented architecture showing management interfaces directly manipulating feature data and services. Highlighted is the duplication and coverage between management interfaces. The CLI showing complete coverage, while others less so.

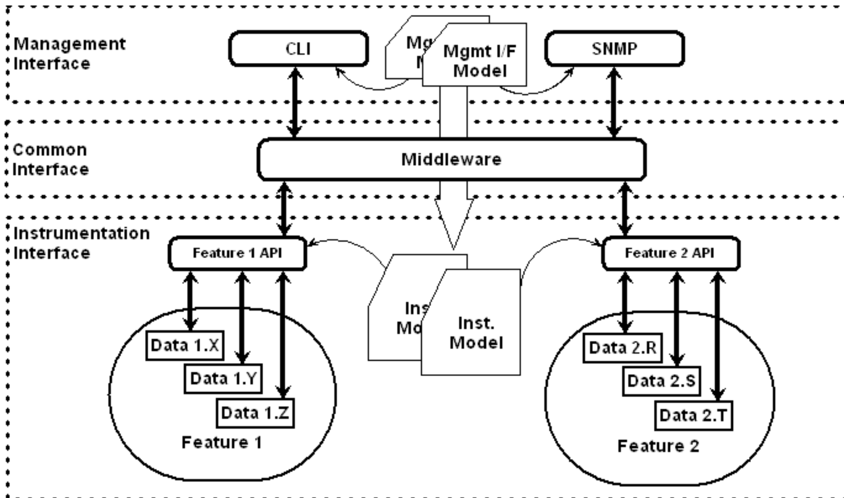


Fig. 2. Illustrates how the model-driven architecture abstracts access to instrumentation data and services through a common interface and normalizes data and services availability to all management interfaces. Models in the upper layer describe the management interfaces, while the models in the lower layer provide definitions of the instrumentation they manage. Together, these models describe the management instrumentation of the system in an end-to-end manner.

The general architecture of model-driven instrumentation has five primary

characteristics that should be considered when designing a model-driven architecture based on a command-oriented system.

3.1 Defining the Instrumentation Model

The instrumentation model may be defined using either of two distinctly different methods. It may be derived from legacy source code or it may be explicitly constructed based on domain knowledge.

Derivation from Source Code

In an effort to minimize the impedance mismatch between the imposed model and the implicit model, deriving the instrumentation model from the management instrumentation implementation is often considered by designers. Since the CLI management interface often contains the most complete implementation it is frequently seen as the canonical source for model derivation; however, this task can be wrought with difficulty. Achieving this objective requires correct and complete interpretation of implementation source code sufficient to extract instrumentation configuration data elements, operational data elements, and services. Model element extraction alone is not enough to meet objectives, as the interpretation of the semantic relationship of features, data, and services is also required in the modeled system. Without a perfect interpretation, gaps and model generation errors will require exhaustive human interaction and domain experience to correct. Moreover, it should be carefully considered if the implicit instrumentation model in-fact meets the needs of the target model-driven system.

Designing from Domain Knowledge

Designing an instrumentation model relies heavily on the designer's knowledge of the instrumentation domain space. They must understand the configuration data, operational data and services offered by the instrumentation and the relationships between them. If modeling an existing system, it should be accepted that regardless of modeling choices, there is going to be some level of unavoidable impedance mismatch with legacy systems; however, when modeling new instrumentation in the model-driven system, this is not the case. Designers creating models for brand new instrumentation can ensure models have a 100% match to data and services. When constructing model architectures, designers should avoid sacrificing model extensibility by designs that are too rigidly tied to legacy structures.

3.2 Model Inheritance

When management instrumentation is considered as a collection of models describing the system's instrumentation, common elements emerge. These common elements may be collected into model libraries for leveraging across instrumentation models. This reduces duplication, streamlines maintenance and helps to promote consistency across instrumentation modeling. Constructed models that leverage a model library may implement or extend it. Derived child models may themselves be model libraries, further extending a reusable model hierarchy.

3.3 Dynamic versus Static Models

There are two types of methods for model use in the management instrumentation system – Static and Dynamic. Static models are used at build time to generate source code or other compile-time entities that are fixed in the run-time image. Dynamic models are interpreted at run-time at occasions determined by the management system. If a design is to employ both static and dynamic models, designers must consider what will occur if the model fails dynamic interpretation. How will the model's availability, as well as its possible dependent libraries be guaranteed? How will version control be enforced? Consider that dynamic models need not only be validated against their dependent libraries, but also against any static libraries used to build the runtime system in which they are being loaded. Designers must take into account the effects of the system's ability to successfully dynamically load a model in order to function and what affects this may have on system reliability and availability.

3.4 Model Versioning

Designs that include models that can implement or extend a model library must also consider management and enforcement of model versioning to ensure compatibility between parent and child models. Defined models must include a mechanism the model interpreter/compiler may use to determine if two models are compatible. In a system that utilizes dynamic models, pre-compiled models must also support version information for system validation during dynamic interpretation.

3.5 Model-Generated Instrumentation APIs

Modeled management interfaces obtain and manipulate instrumented features

through a common abstract interface design. This abstract middleware interface directs instrumentation access requests to appropriate instrumentation for manipulation of data and services through their instrumentation API.

The instrumentation API is collection of functions that independently access specific instrumentation data and services. The framework of functions may be generated from information in the instrumentation model, but not the particular code to access the actual data or service of the instrumentation – that must be supplied by the instrumentation developer.

When implementing APIs for modeled legacy instrumentation, there are two potential sources for instrumentation API implementation source code: 1) port it from the legacy implementation, 2) write it from scratch. It is important to consider which method is the most accurate, reliable, and most reusable. Frequently, only fragments of command-oriented instrumentation source code may be leveraged in a model-driven design. A careful evaluation of the effort required to port existing code or to write new and perhaps more efficient instrumentation code should be carefully considered.

Creating model-driven instrumentation can be more challenging than command-oriented instrumentation development if model-imposed restrictions are to be used. Command-oriented development allows direct and freeform access to feature instrumentation at anytime, from virtually anywhere, with no interface definition requirements. In contrast, because of crisply defined constructs, modeled instrumentation has the capability to ensure rigorous data validation and consistent instrumentation interfaces between clients and available instrumentation data and services. While this makes API definition somewhat more challenging than the freeform method, this is one of the premier benefits of model-driven systems and results in improved reliability and quality.

4 Implementation Experiences

This section will explore the challenges and experiences gained through the prism of a real-world development effort where an established command-oriented management instrumentation system was transitioned to a model-driven management instrumentation system. It will discuss design choices and the reasoning behind them.

The most design-influential aspect of the transition was resolving the

architectural differences between the existing command-oriented system and the target model-driven system. Viewed by many as an *inside out* to *outside in* transformation, there were two primary areas that stood out as the biggest hurdles to overcome – instrumentation API development and Instrumentation Modeling.

There was considerable effort devoted to developing automated tooling that could minimize the development effort by leveraging existing command-oriented source code to generate candidate instrumentation models and create a basic instrumentation API implementation. In order to perform such a task, the tooling was required to scan and interpret existing source code, derive APIs and candidate models from embedded domain knowledge. These efforts were unsuccessful. Resolution of run-time defined abstract function calls and model semantics proved impractical due to interface complexities and inconsistencies. The end solution was to not transition the command-oriented code and functionality but instead to build a framework that utilized the existing system for existing feature command requests while directing requests for newly implemented functionality to the model-driven framework. This allowed the management instrumentation system to maintain it's backwards compatibility while at the same time allowing its functionality to grow within the new model-driven paradigm. This coexistence allows existing feature and functionality to be transitioned from the legacy component to the model-driven framework on a piecemeal basis if desired. Maintaining legacy functionality as a coexisting component within the model-driven framework was found to be considerably more efficient, reliable and practical than attempting a manual transition of its entire functional feature set.

The second highest design hurdle was the construction and composition of the Model Framework's APIs. There are two APIs to consider:

1. Middleware Interface API – communicates with management interfaces such as the CLI, SNMP, Syslog and so forth.

2. Instrumentation API – Handles communication between the instrumentation data/services and the Middleware Interface.

These APIs must be well considered in order to ensure they satisfy the needs of their users. The Middleware Interface API communicates with management interfaces to supply access to managed instrumentation data and services without tight coupling to the particular kind of feature, instrumentation, data or service being manipulated. A Create Read Update Delete and eXecute (CRUDx) interface was selected to best satisfy the

abstract needs of the management interface clients. The CRUDx interface provides database-like functionality of management instrumentation resources to client management interfaces.

The Instrumentation API has a similar, but slightly more rigidly defined CRUDx interface. This interface definition choice satisfied the middleware layer's management instrumentation needs enabling it to operate on managed resource instances and on data and services directly.

Across both interfaces, request data is passed and response data is returned in a data-oriented fashion. Function oriented APIs were not used as they restrict input/output to the context of the specific function, making them less portable. Data orientation promotes efficient use of individual instrumentation APIs and allows a single request to more efficiently use the services of many different instrumentation components.

The third challenge was to determine how to design the system for optimal model management. Examination of the problem domain indicated that there was a large amount of overlap between the instrumentation models and potential for efficient model reuse streamlining both the design and development. Model libraries were defined to promote implementation and extension of common model components. This led to the imposition of model version constraints that impacted model content and how models, model libraries and model elements were referenced by the system.

Modeling decisions led to the proposition that the management interfaces could be models and such models could reference instrumentation data and services using abstract identifiers. This broached the subject of dynamic and static model management. Instrumentation models are static, compiled at build-time and are part of the application image. Dynamic models, such as those defining a CLI or an SNMP MIB, may be statically defined at build-time or loaded and compiled at runtime. Faced with the option of utilizing static or dynamic models, a half way solution was adopted. Management interface models were pre-compiled at build-time into quasi-object models moved the compilation and validation step to build time. This optimization enabled the run-time system to perform only the interpretation of validated and prepared pre-compiled models for operation.

An API Implementation Example

To illustrate the result of some of the API design choices made, a high-level implementation example is provided. For the sake of brevity, a simple Interface Flow Monitor was selected. This component monitors the flow of bytes or packets per second across a network interface and sends and event when a configured high threshold is exceeded. Additionally, it maintains the minimum and maximum observed flow rates and provides the capability to reset this operational data. The model for this component is shown in figure 3.

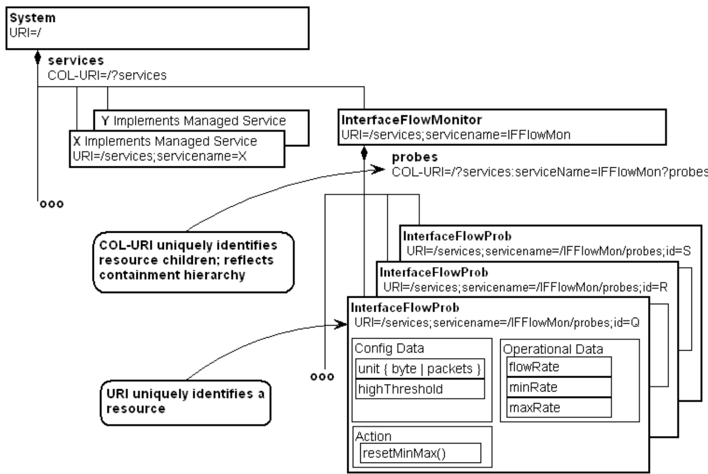


Fig. 3. Illustrates the model of the Interface Flow Monitor component.

The Instrumentation CRUDx API generated from the Interface Flow Monitor model is shown below in pseudo code.

```
errorcode createIntefaceFlowProb(Session);
value readIntefaceFlowProb_unit(ErrorOut);
value
readIntefaceFlowProb_highThreshold(ErrorOut);
value readIntefaceFlowProb_flowRate(ErrorOut);
value readIntefaceFlowProb_minRate(ErrorOut);
value readIntefaceFlowProb_maxRate(ErrorOut);
errorcode deleteIntefaceFlowProb();
errorcode InterfaceFlowProb_resetMinMax();
```

These functions have direct interaction with the instrumentation and are called by middleware in response to middleware API originated instrumentation access requests.

The construct of the middleware API does not depend upon the model, however, the value of arguments passed are model-dependent. For example, the `URI` arguments in the APIs identify the target object instance to create, read, update, delete or execute. It may also identify a collection of object instances as in the case of a Collection URI (COL-URI) for manipulation of a collection of objects and data. The `DataIn` and `DataOut` arguments signify the variant types of data that may be passed between the client and the identified target object(s). The middleware API is illustrated below in pseudo code.

```
errorcode create(URI);
errorcode readConfigData(DataOut, URI);
errorcode readOperationalData(DataOut, URI);
errorcode updateConfigData(DataIn, URI);
errorcode delete(URI);
errorcode execute(DataOut, DataIn, URI);
```

When the middleware receives a call across this interface, it resolves the URI to the object instance and performs the associated Instrumentation API call for the entity.

```
readOperationalData(long&
                  lRate,"/services;servicename=
                  /IFFlowMon/probes;id=/Q/flowRate"
                  );
```

The middleware resolves this information to a call to the instrumentation API shown here in trivial pseudo code:

```
Q=get("/services;servicename=/IFFlowMon/probes;id=
/Q");
lRate=Q->readIntefaceFlowProb_flowRate(ErrorOut);
```

While most of the fine details of the middleware and instrumentation API interactions have been excluded, the example illustrates the design choices and resulting concepts of middleware and instrumentation API construction from the instrumentation model.

5 Related Work

There is a great deal of work that has been done in the area of Model-Driven Engineering (MDE) over the past three decades and was crystallized with the formation of the Object Management Group (OMG) [10] in 2001. The OMG was formed to establish modeling and model-based standards. Since that

time, the promises of MDE have been elucidated for the development of new applications, providing modeling tools, development tools, domain specific languages, and the like. A virtual plethora of standards and applications have been created to support the development of new model-driven applications, however little has been done to address the cost effectiveness of leveraging existing systems in a model-driven environment. Douglas Schmidt [4] in his February 2006 Model-Driven Engineering overview states "When developers apply MDE tools to model large-scale systems containing thousands of elements, they must be able to examine various design alternatives quickly and evaluate the many diverse configuration possibilities available to them." He refers to the Integrated Modeling Approach of Lockheed Martin Advanced Technology Laboratories as an example of legacy integration with less than ideal results: "Reverse engineering is used to build models… from existing source code. Many previous attempts to reverse-engineer models from source code have failed due to a lack in constraining aspects of interest." A similar experience described in this paper.

In his article "The Pragmatics of Model-Driven Development", Bran Selic[6] discusses legacy integration mostly in terms of development tooling only tangentially touching upon the issue of leveraging application source code in the modeled environment. In this case, the recommendation was to take advantage of legacy code libraries and other legacy software where domain-specific knowledge often resides. While certainly true, this view overlooks the problems of impedance mismatch between the two designs and often-prohibitive implementation cost of custom glue code.

At the UML Conference of 2003, Jean Bezivin [2] presented "MDA: From Hype to Hope, to Reality" where it was stated: "The extraction of MDE-models from legacy systems is more difficult but looks as one of the great challenges of the future. There is much more to legacy extraction than just conversion from old programming languages". Indeed, it is the variability of legacy systems, platforms and ultimately the model impedance mismatch that is at the heart of these challenges.

Unfortunately, most work in the Model-Driven Engineering area focuses on the rapidly accelerating model-driven technologies and patently avoids dealing with the big white elephant in the middle of the room – how to leverage the existing application features and functionality in an efficient and cost-effective manner.

6 Conclusions

Management instrumentation designers are looking to shift their command-oriented management instrumentation to model-driven in order to utilize the benefits of these modern technologies but are daunted by the difficult challenges that complicate such a transition. Features supported through stovepipe CLI implementations and augmented with redundant and often only partial, alternate management interfaces complicate the problem. The practice of feature-specific/command-oriented implementations, while freeform in construct, culminates in multiple and redundant request handling, inconsistencies between management interfaces and differences across product versions. Perhaps most significantly, it geometrically increases maintenance requirements and costs due to duplicate and redundant code. Designers considering a transition to a model-driven system will find this impedance mismatch to be the most vexing problem.

In an ideal scenario, designers would like to leverage legacy code in the model-driven system by deriving models directly from the legacy source code, however this is seldom possible. The tight coupling of individual management interfaces with manipulated instrumentation data and services fundamentally blur the lines between the models they desire and the models implicit within their implemented instrumentation. This makes model derivation from legacy source an impractical proposition.

Experience has shown that neither reverse engineering nor model-derivation met expectations, but rather integrating a legacy system as a coexisting component was found to be the most desirable solution. Instead of attempting a re-design or fully modeling a seasoned management instrumentation system, the system itself was leveraged as an integrated partner of the model-driven framework. This technique allowed the supply and maintenance of existing features to the system while at the same time promoted the development of new features and functionality within in the model-driven framework.

The successful transition of a command-oriented system to a model-driven management instrumentation framework supporting both management interfaces and instrumentation involved the resolution of several key design considerations surrounding API development. Management interfaces reside above the middleware layer and exist in the management domain, requesting instrumentation data and services from the middleware interface as clients. Client services available from this interface are accessed in a manner decoupled from the instrumentation implementation using functions that provide well-known Create Read Update Delete and eXecute (CRUDx)

capabilities. Similarly, the middleware communicates with instrumentation implementations using a more rigidly defined model-generated CRUDx API that operates directly on an instance of the instrumentation object, invisible to management interfaces. Significant in the design of these APIs were their construction. Decisions that request data would be passed across each interface affected the kind of API generated. The API exposed to management interfaces is data oriented in order to facilitate optimal communication between management agents and middleware. The Instrumentation API required similar exposition of data and services for direct operational performance on an instrumentation instance by middleware.

Scrutiny of the instrumentation modeling problem domain revealed a large overlap of common elements among instrumentation models. Model leveraging was introduced using model libraries to share common components among models and model libraries allowing extension and implementation promoting model reuse. This concept further revealed the need for model and library versioning to ensure the integrity of referenced models during compilation and interpretation.

As the implementation of modeling paradigms took hold, the concept of utilizing modeling to describe management interfaces became clear. Modeling management interfaces utilizing instrumentation models to connect management elements to associated instrumentation data and services promoted an end-to-end management instrumentation development paradigm. This opened the door to dynamic model management – the idea that a management model did not have to be built within an application, but could be installed or removed in a running system dynamically. After considering dynamic build and interpretation options, designers chose to dynamically load pre-compiled/validated models. This design choice minimized the runtime compilation and validation burden on the running system and promoted better dynamism of the management interface models.

When faced with the daunting task of moving a command-oriented system to a model-driven paradigm, the impedance mismatch is at the heart of the matter. Finding a way to bridge the gulf the between the traditional “top-down” view and the modern “everything is an object” view is the crux of a successful transition.

Further Work

The transition factors highlighted herein focused on the primary design and implementation considerations. Additional work should be done to illustrate the details of integrating an existing management instrumentation system as a

coexistent component in a model-driven framework, covering details of the glue-logic and model-driven interactions. Moreover, the concepts developed for code generation from models should be provided to describe the techniques developed through the experience to provide end-to-end round trip model, code management and synchronization. Finally, and perhaps most importantly, the impedance mismatch between the command-oriented and model-driven paradigms are not restricted to design, but extend to development processes as well. The experience yielded significant changes to existing command-oriented development processes and involved much work with human factors engineering resulting in new, optimized processes requiring developer management and acceptance. All of these areas are in need of further research.

References

1. Poole, J. D.: Model-Driven Architecture: Vision, Standards and Emerging Technologies, ECOOP, Workshop on Metamodeling and Adaptive Object Models (2001)
2. Bezivin, J.: On the Unification Power of Models, MDA: From Hype to Hope, and Reality, UML Conference, San Francisco (2003)
3. Tolvanen, J.P.: Making model-based code generation work, Embedded Systems Europe, Aug 2004, http://i.cmpnet.com/embedded/europe/esesep04/esesep04p36.pdf (retrieved 2010)
4. Schmidt, D. C.: Model-Driven Engineering, IEEE Computer, Feb 2006, http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf (retrieved 2010)
5. Brown, A.: An Introduction to Model-Driven Architecture, IBM Technical Library, http://www.ibm.com/developerworks/rational/library/3100.html (retrieved 2010)
6. Selic, B.: The Pragmatics of Model-Driven Development, IEEE Software 2003, http://www.cs.helsinki.fi/u/przybils/courses/CBD06/papers/01231146.pdf (retrieved 2010)
7. Daniels, J.: Modeling with a Sense of Purpose, IEEE Software, Jan 2002, http://www.syntropy.co.uk/papers/modelingwithpurpose.pdf (retrieved 2010)
8. Bezivn, J., Gerard, S., Muller, P-A., Rioux, L.: MDA Components: Challenges and Opportunities, 2003, http://www.sciences.univ-nantes.fr/lina/atl/www/papers/MDAComponents-ChallengesOpportunities.V1.3.PDF (retrieved 2010)
9. Almeida, J.P.A: Model-Driven Design of Distributed Applications,

Telematica Instituut Fundamental Research Series, No. 018, 2006
10. Object Management Group, www.omg.org

Chapter 2
A Low Power, Programmable Networking Platform and Development Environment

Pál Varga, István Moldován, Dániel Horváth, Sándor Plósz

Budapest University of Technology and Economics
Inter-University Cooperative Research Centre for Telecommunications and Informatics,
Magyar Tudósok krt. 2, H-1117
Budapest, Hungary
{pvarga, moldovan, horvathd, plosz}@tmit.bme.hu

Abstract. Programmable networking platforms are getting widely used for customized traffic manipulation, analysis and network management. This propagates the need for exceptional development flexibility, for wide variety of high-speed interfaces and for the usage of high performance, yet low power technologies. This paper presents an FPGA-based programmable platform, capable of real-time processing, filtering and manipulating 10Gbps traffic. In order to expand its potential, besides the two 10GbE interfaces, the platform contains extension slots for COM express, mini PCI-e, and it has 16 onboard SFP connectors, towards which the fraction of the traffic, or even the full traffic can be forwarded to. The design is modular, programmable in both hardware (firmware) and software, aiming low power consumption. The full potential of the hardware can only be exploited with an easy-to-use development environment, with simple design customization and support for creating new applications. To fulfill this, a development environment is also presented, including a modeling framework that provides an easy way to create new networking applications on the platform. This framework allows modeling applications in SystemC, and eases the development of the hardware description code.

Keywords: 10GE, programmable platform, DPI, FPGA, low power

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_2, © Springer Science+Business Media, LLC 2011

1 Introduction

Networking at ever growingly high-rate connections constantly generates challenges for researchers and engineers developing algorithms and equipment to handle the demands of networking services. The increasing rate is not the only concern, but it brings some general, seemingly far-away problems into the limelight.

When data arrives to a system at 10 Gigabit per second rate, the time is very limited for analyzing or handling it. Moreover, there is no point for its single storage for further analysis (except for some targeted analysis). If the system cannot process continuously arriving data, it will not be able to process it later, when further data still arrives continuously. To optimize data processing in many levels, tasks should be distributed and made parallel. The processing level here does not only mean OSI levels, but levels of processing complexity determined by the given task. Examples for such tasks are flow assembly based on TCP- and IP-headers, routing and switching between interfaces, application classification by using DPI (Deep Packet Inspection), etc. Our system utilizes multiple processors with various capabilities for processing network traffic in various levels. The capabilities on the main board are distributed through FPGAs (used for time-stamping and initial packet header processing) and a general processor (used for management functions, and basic traffic analysis, statistics creation). Processors on the COM express PC and the PCIe-connected modules can be utilized for complex processing, including routing, switching and basic DPI. Furthermore, the system is prepared for very complex DPI (application analysis through fingerprints, deep flow analysis, etc.), by means of streaming digested packet and flow-data to external processors through its 1GbE interfaces.

Low power design is a current and very important requirement in all fields, including IT systems. The high demand for networking services is covered by ever increasing number of servers and networking equipment, which, if left uncontrolled, waste electrical power and simply turns it into heat. On the other hand, handling or analyzing high speed traffic requires high performance, which is by definition a term competing with low power consumption. The challenge of high-performance, yet low-power systems is to find out which costs less power: should we shut down resources that are not in use and urgently wake them up when required or should we leave the resource running. Measuring power consumption and optimization for low power with high performance was a very key requirement during the definition and design of our system.

We have also created a development environment together with the platform. The aim of this is to accelerate the development process of network applications on FPGAs in general. The environment provides a GUI and a set of hardware modules which builds up a variety of network devices. Key use-

cases are switching, routing devices, NAT devices, firewalls, deep packet inspectors, and traffic loopback devices.

The system has been designed and developed by applying a close hardware-software co-design methodology. This primarily defined the distribution of tasks among the various types of processors. Beside, this flexibility allowed the clarification of basic processing modules and algorithms, which enabled to create a programmable networking platform.

For design space exploration and to validate the design, a SystemC [1] based modeling environment is used. The results of the SystemC modeling can be used to construct the final hardware models and the corresponding software. The SystemC hardware components are also available in generic hardware description language (Verilog/VHDL) making the synthesis of the hardware possible. The developers will also be able to generate the top-level hardware model through the GUI. The modules required for the generic networking applications have been selected by identifying the most important use cases.

After the literature survey in the next chapter, we briefly describe the hardware, the firmware, and the development environment to be used for various networking applications hosted by the platform. Afterwards we highlight the usability of the environment through two use-cases: a network monitoring DPI scenario and a routing/switching scenario.

2 Related Work

In the literature, we have found similar work dealing with packet processing on FPGA-based systems.

Besides the industry leading Endace DAG packet capture products [3], the NetFPGA [4] platforms are largely in use in academic research to test for ideas and implement them on flexible hardware. The TenGig NetFPGA card is currently under development, and it will be capable of 10G traffic handling. It will provide 4 XFP ports RLDRAM II, QDRII, SRAM, PCIe 8x interface and extension connector, powered by a large Xilinx Virtex-5 FPGA, XC5VTX240T which is quite expensive. A similar platform is developed within the Cesnet Liberouter project [5], which already provides a 10G extension card to their extensible Combo system, making it capable of 10G packet processing. We would also mention an interesting application of FPGA-based design platforms for Gigabit Ethernet Applications. FPGA-based implementations offer the possibility of changing the functionality of the platform to perform different tasks and high packet-processing rate capabilities. In particular, the authors of [6] proposed a versatile FPGA-based hardware platform for Gigabit Ethernet applications. By introducing controlled degradation to the network traffic, the authors provided an in-depth

study on real-application behavior under a wide range of specific network conditions, such as file transfer, Internet telephony (VoIP) and video streaming. Other approaches include hardware accelerated routing, e.g. the work of D. Antos et al. [7] on the design of lookup machine of a hardware router for IPv6 and IPv4 packet routing with operations are performed by FPGA. In this framework, part of the packet switching functionality is moved into the hardware accelerator, step by step. This allows keeping the complete functionality all the time, only increasing the overall speed of the system during the whole development process. D. Teuchert et al. [8] also dealt with FPGA based IPv6 lookup using a pipelined, tree-bitmap algorithm based method.

The NetFPGA project also provides a development environment for the programmable hardware platform. Their approach [10] is to provide reference architectures (interface card, switch, router, etc.) as starting points for new development. To avoid the necessity of hardware level programming and provide a high level interface, a framework is presented in [11] to incorporate hardware G modules into NetFPGA based system designs.

Although there are several similar approaches [9], none of them fulfills the requirements of the C-Board. The existing hardware is not fast or not scalable enough, while also the development environment lacks the flexibility and the required simplicity.

3 The SCALOPES C-board

The ARTEMIS SCALOPES project aims at developing and utilizing novel methods in low power, high performance embedded platforms. Our SCALOPES C-board is the prototype platform for the communication infrastructure-related applications inside the project. The main purpose of the C-board is to provide a basis for high-speed data processing and manipulation. It could either host or serve monitoring, switching, routing, filtering and other applications that require real-time processing of 10 Gigabit Ethernet traffic. In the following sections the motivations, requirements and the state of the art is surveyed, followed by the brief description of the architecture.

3.1 Motivation and Requirements

Real time analysis and manipulation of 10 Gigabit Ethernet traffic requires scalable, high performance equipment. Clear and easy-to-use management and programming interfaces further ease the task of the user of such equipment. There are some programmable networking platforms already

available in the field, nevertheless, upto this date we have not found another platform that

- is both programmable in hardware and software,
- can manipulate the traffic by utilizing PCIe-connected controllers,
- has capabilities to directly forward 10Gbps Ethernet traffic to/from 1Gbps Ethernet or SONET,
- is designed for measurable low power consumption, and
- has lowered risks for extra developments since composability, predictability and dependability [2] issues are tackled.

The SCALOPES C-board was designed and developed with the ultimate intention of putting the above requirements into practice.

3.2 Internal Structure

The practical capabilities of any networking equipment are limited by its internal elements, their programmability, and its interfaces' types, modes, and speed. During the development of the SCALOPES C-board, the requirements were set high: it is a highly scalable device with a well-defined programming toolchain, capable of manipulating traffic arriving from SONET, ATM, Gigabit and 10 Gigabit Ethernet, routing/switching the traffic between these interfaces and further controlling or processing it through PCIe x4 extension modules. The simplified architecture of the board is depicted by **Fig. 1**.

The main components of the device are the following:

- XFPs (10 Gigabit Small Form Factor Pluggable Modules) connecting to XAUIs (10 Gigabit Attachment Unit Interface) for 10 Gbps traffic reception,
- SFPs (Small Form/factor Pluggables) to handle various Gigabit Ethernet ports, for output to devices,
- four FPGAs (Field-Programmable Gate Arrays) connected in a matrix, used for packet capture and manipulation, including interface handling, traffic flow handling firmware blocks, basic statistical modules,
- memory for packet buffering and flow tables,
- extra processor for on-board processing and management software,
- redundant power supply.

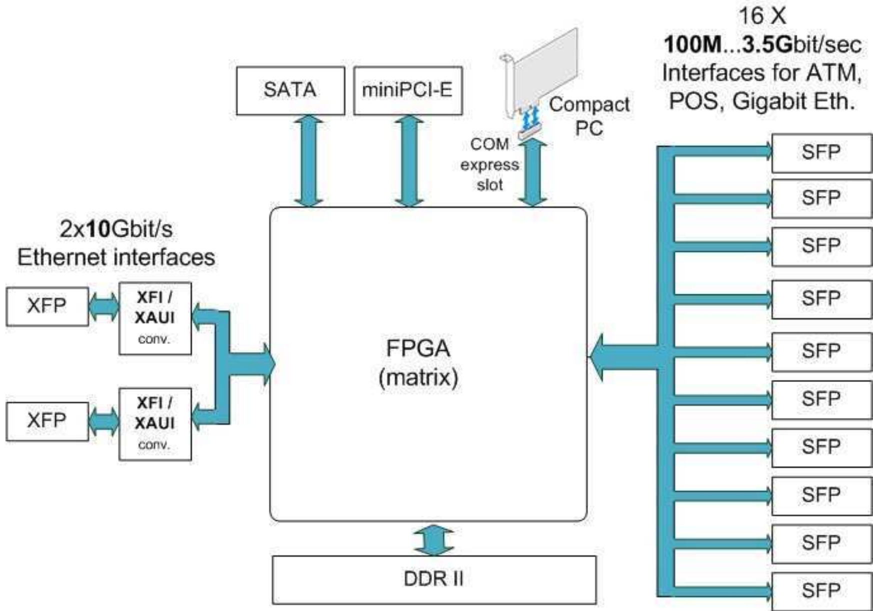


Fig. 1. The simplified structure of the SCALOPES C-board

Each element of this architecture meets the basic dependability requirements in order to support the overall dependability/survivability of the system that it is part of.

The main part of the device is the FPGA matrix (or ring), containing four FPGAs. The FPGA technology helps building a multi-purpose hardware. A great advantage of this technology is that a simple firmware switch enables us to switch between applications much faster than if we needed to switch the whole device.

The outside interfaces (SFP, XFP and COM express slots) connect to the FPGA's RocketIO ports, which allow high speed communication between interfaces. The optional variation of interfaces (Gigabit Ethernet optic, 10/100 Ethernet, STM-1 optic etc…) is possible by using SFP module receivers.

The physical and logical connection between the interfaces is defined by the FPGA firmware ensuring the hardware's flexibility. The current firmware is physically stored on flash memory connected to the chips: they load as soon as the hardware starts. I/O data is shared between FPGAs through a dedicated communication ring interface. There are two rings defined by the clockwise and counter-clockwise direction of communication, assigned to the two 10 Gigabit Ethernet interfaces. **Fig. 2** depicts the architecture of the board and the connection of the FPGAs.

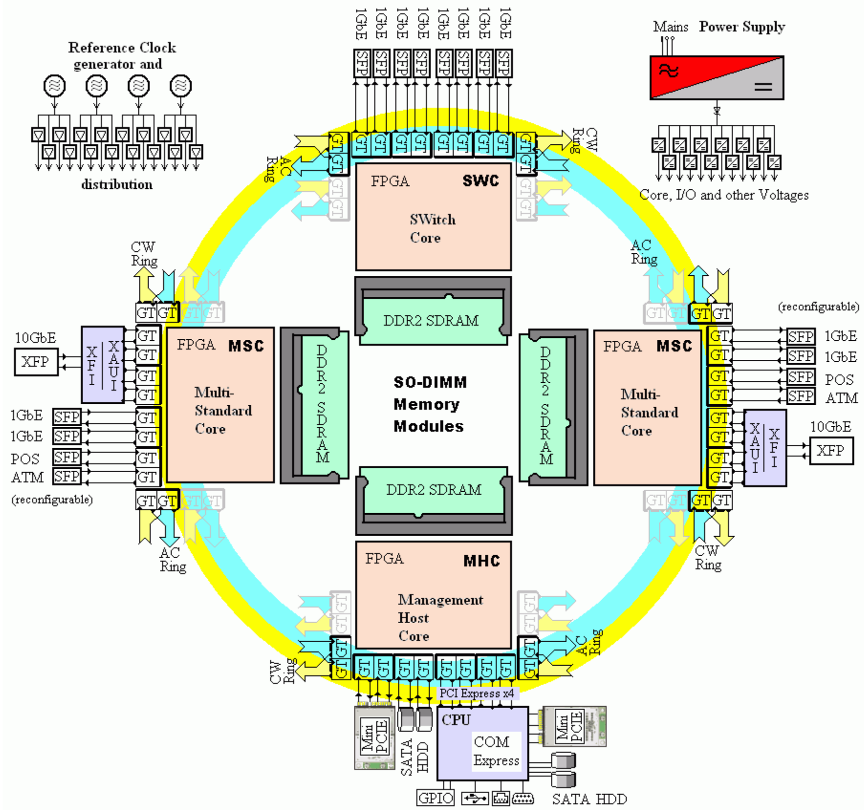


Fig. 2. Internal structure of the SCALOPES C-board and the two-way communication of the FPGAs

There is an additional interface designed for configuration and maintenance. The FPGAs can be reprogrammed during operation, and the running IP-core can be controlled/changed through a 10/100 Mbps Ethernet interface connected to the COM express PC.

3.3 Low Power Design

During the development of the C-board, one of the main, higher goals was to create a device of low power consumption. Depending on the network configuration, the used application and the traffic volume the power consumption of the C-board becomes significantly lower in comparison to systems that do not use sophisticated power control. As a static requirement, it can be reached by using low-power electronic elements. The operating power

consumption depends on the clock frequency as well. The C-board is configured to operate on the lowest frequency on which it is able to process all of the traversing packets on 10 Gbps interface.

Furthermore, lower power consumption can be reached by close power control of the programmable devices in a dynamic manner, while they operate. There are three areas in the SCALOPES C-board where such Dynamic Power Control (DPC) can be administered: the interfaces, the FPGA and the memory. DPC is managed by a central resource manager (governor) application, residing in the compact PC.

Naturally, if an Ethernet interface is not configured to be working in a given configuration (runtime), its controller is shut down, not consuming any power. Moreover, the Ethernet interfaces connect to the FPGA chips in a distributed manner, which means if the related interfaces are not needed, the corresponding FPGA chip can be assigned to stand-by mode, hence significantly decreasing the system's energy consumption. This power reduction scheme can be initiated runtime, in connection with the network configuration changes.

Internally to the FPGAs, power islands are defined for segregated functions: interface handling, packet filtering modules, management modules and low-speed/high-speed implementations of packet processing algorithms (depending on timing criteria, the low-speed implementation might be used for power considerations). Based on the running application and the traffic volumes the central power governor can decide to shutting down or waking up these islands.

DDR RAM memory is connected to the FPGAs, and its power-management can also be controlled from there, runtime: it can be set to stand-by or power down mode if a given application does not need to use the external memory.

In order to measure power consumption, sensors are placed at key areas of the hardware. These sensors send signals about the measured signals to the management interface, where the data can be collected, analyzed and used for system tuning.

4 The Development Environment

Developing or even modifying complex networking systems that consist of hardware and software processing modules require either incredible experience or extreme braveness – sometimes both. In order to develop applications to the SCALOPES C-board, efficient hardware and software co-design methodology must be administered.

Our approach follows a two-step process in the high-level, and based on elemental building blocks of a modular architecture. The first step in the

process is the development of simulation models, from the modeling blocks available or ready to be developed. The second step in the process is the actual hardware and software design, based on the experiences gathered from the modeling evaluation. This process requires both less experience and less braveness from the developer, since on one hand the building blocks already provide some design security, and on other hand he or she is going to be supported by the understanding the possible obstacles after learning the simulation models. Fig. 3 depicts the development environment from a high-level perspective.

The flexible programming capabilities provided by the SCALOPES C-board's processors and interfaces can only be exploited if a well-defined firmware development environment accompanies the device. Experience in HDL programming may be a serious limiting factor for customized system enhancements that require firmware development. Hardware, firmware and accompanying software programming knowledge is rarely present together in an organization that is not specialized exactly for these – and utilizing only one of these essential capabilities when experimenting with the networking platform leads to suboptimal results.

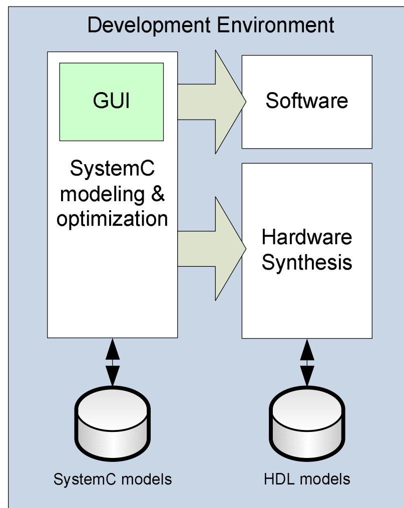


Fig. 3. The development environment

Therefore we provide a development environment that makes the development of generic applications possible, even without hardware knowledge. Beside this, custom hardware functions can be also developed easily, as new building blocks.

The development environment consists of a modeling environment, where the application is modeled in SystemC, and a hardware/software co-design methodology, utilizing commercially available programming tools as well.

The development environment gives access for different levels of complexity, from basic GUI-based modifications to full simulation/modeling and co-design.

4.1 GUI based development

Starting from a provided generic application architecture, custom firmware can be easily generated using available “filters”. There are several generic modules available for packet manipulations that can be inserted at the packet input/output processing stages of the architecture. These modules may also provide a software interface, for setting filter rules, reading statistics etc. For example, a simple firewall can be created by adding input and output filter blocks to the generic packet forwarding application. Adding NAT functionality means inserting a NAT module at the egress interface.

Fig. 3 shows the structure of the development environment. The whole development process can be done using the provided GUI. The GUI is written in Java, integrates the tools into a development environment and provides editing functions. The available modules are described in IP-XACT [12] XML format, and they can be selected from the toolbox and with simple drag-and-drop operations they can be inserted in the architecture. Composability is ensured by using the IP-XACT specification format, and is automatically checked while connecting the interfaces. Both the SystemC and VHDL sources for the different modules are also provided, along with their IP-XACT description. The new design can be tested by the SystemC simulation framework, which provides traffic generators for performance testing and predictability analysis. The top-level HDL file is generated automatically based on the connections made on the GUI, and the final firmware can be synthesized using the Xilinx ISE. As the new application is a modified version of the generic application with filter components added, its performance should not change, only an insignificant the latency is introduced by the filters.

Besides the composability checking, a dependability and predictability analysis is also possible. The framework will be extended to generate a Continuous Time Markov Chain (CMTC) based dependability model of the system stored in a view in the IP-XACT description, which can be used in open-source simulators like the PRISM model checker [13]. A queuing model of the hardware can also be created, which makes possible the analysis and queue dimensioning for different traffic mix scenarios.

4.2 Custom module development

The development of the new applications is also assisted by a number of available hardware component models that can be used for generic networking application development. The already available components define a generic packet processing/forwarding architecture with extensible filtering and processing properties, and a generic deep packet inspection architecture. All components come with SystemC and VHDL source code and IP-XACT description. The extensible, modular architectures are designed to allow easy integration of application-specific header operations at the ingress and egress. A method for buffering the packet payload is also provided. Most of the applications can be covered with the available modules. The architecture also features a hierarchical power management concept for low power operation [14].

However, if a custom module is needed for a hardware-level operation which is not yet available, new modules can be created and added to the model database by adding the source files to the SystemC and VHDL directories and creating the IP-XACT XML descriptor. The key issues are the interfaces of the new component, and the timing requirements. For the most common operations like input/output processing and flow handling well defined interfaces are provided with low timing requirements. However, for line-speed processing all modules must handle back-to-back packet arrivals at interface speeds. For even easier development generic filter prototypes are provided with full source code. First, the SystemC model should be created, for composability and predictability checking. Based on the SystemC model the HDL model can be created too, and added to the HDL model database.

4.3 Hardware accelerator development

The protocol processing task usually starts with basic input/output operations like packet reception and queuing. These operations require hardware processing, for fastest execution. After several stages of header processing, it may not be suitable for hardware processing because of increased complexity; software processing is more flexible and desirable. The conventional networking protocols had simple structure, binary headers and fixed fields.

As we climb the levels of the OSI model, the protocols become more and more complicated, with variable header lengths and fields. One of the most difficult protocols to decode are the textual protocols like HTTP and SIP, where very few things are fixed, fields can be mixed, string identifiers are used and all fields have variable length. Such protocols require software processing, involving parsing. There are several methods to enhance the processing in hardware with accelerator interfaces. Different protocols require

different interfaces, possibly tailored to the given application. Unlike for network processors, where hardware accelerators are given, we can design and implement custom hardware accelerators in order to achieve best performance. There are several hardware accelerators that are generic, like binary/ternary CAM tables, queue managers etc., but accelerators can be tailored by an optimization process. In our approach we consider that processors can reach the hardware accelerators through a simple and fast interface, like port I/O operation or memory operation to a very specific area. The optimal separation of software/accelerators/hardware however is an optimization "knob" that requires an iterative approach to select the optimal parameters.

A further gain for the hardware acceleration is the decreased power requirement, since hardware processing has lower cost form the point of view of energy consumption.

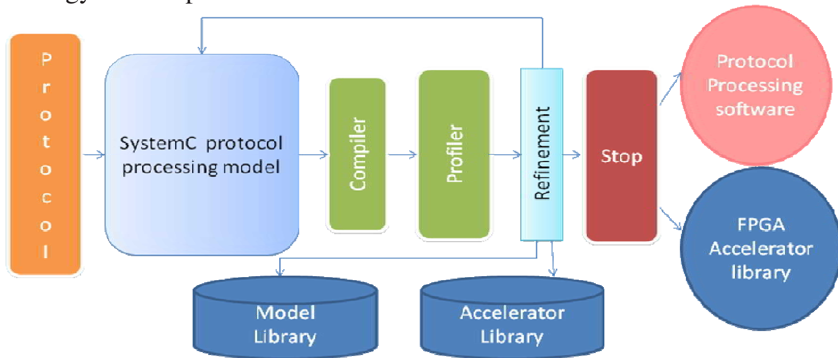


Fig. 4. Hardware/Software optimization process

The accelerator design tools are currently under development. Accelerator models are protocol-tailored hardware functions like table lookups, string operations or parallel lookup of several strings etc. The SystemC model of the protocol processing uses the library models of the hardware, accelerators and adds the software part too (see **Fig. 4**). The compiler expands the macros and includes the code from the model library, creating the simulation executable. The profiler tool collects data provided by the simulation and the results are timings, delays and also may estimate energy consumption for specific modules. Based on the profiler results new accelerators can be defined or existing ones can be refined and integrated into the SystemC model. The output of the simulation is the architecture, the hardware and accelerator modules on one hand, and the protocol processing software on the other.

5 Case Studies

The platform itself is capable of high-speed packet processing but can be extended with industrial PC boards for complex functions and increased processing power. Therefore it can perform all network related functions from the low level high-speed packet forwarding to the most complex protocol processing.

These packet processing architectures also utilize powerful energy management techniques. First, power islands can be formed, and unused islands may be powered down. Furthermore, local power management is used where possible by turning on the processing modules only if they are needed. Finally, a central manager provides a management interface for the control of the power islands.

In the following we present several specific use cases for the SCALOPES C-board.

5.1 A Network Monitoring System with DPI Capabilities

Traffic monitoring plays an important role in network management, network optimization and planning. Operators are usually aware of only the main characteristics of the traffic, which generally is limited to an average throughput with 1-minute granularity. On the other hand, information about fine granularity of the traffic allows for network tuning and more effective planning. As the operators switch to 10GbE connections, processing packets and flows real time at this speed is getting more and more important - no matter how complex this task really is. Deep packet inspection and flow analysis cannot be carried out with on this rate by using the currently available processors and memories, hence the traffic is going to be filtered and distributed over several processors outside the C-board for full analysis. In this use-case we present a monitoring system capable of DPI at line speed. Furthermore, we show that using our easy development environment the DPI can be easily tailored to the specific requirements. The high-level workflow of DPI and flow analysis in this system works the in the way depicted by **Fig. 5**.
The C-board receives the monitored traffic through the 10Gbps XFPs, and initially timestamp each packet. Depending on their configuration, the filter modules pass “interesting” packets to the forwarding buffer, and sends these to flow classification with the rest of the traffic. The classifier puts together flows based on the packet

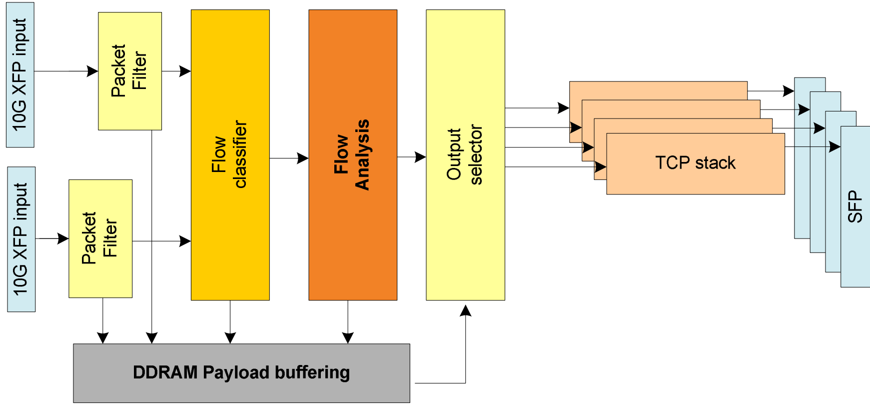


Fig. 5. DPI firmware architecture

headers. The flow analysis module provides statistics with fine granularity, and application identification information on each flow. The flow data and the chosen, "interesting" packets get forwarded from the buffer, through the output selectors to further processing entities over the 1Gbps Ethernet channels.

These processing entities are PCs called "Monitor Units" with high processing and storage capacity. In order to reduce loss of data between the outsider processing entities and the C-board, the packet information (headers and predefined parts of the body) get encapsulated in TCP flows and then forwarded. TCP is needed in order to assure lossless transfer of capture-data towards the remote units. The Monitor Units carry out complex traffic identification and traffic matrix analysis, as well as store bit-by-bit packet header information if configured so.

The basic DPI requires a specific firmware architecture, where the traffic is flowing from the XFPs towards the classifiers from where it will be de-multiplexed to 1Gbps speed. The complex DPI algorithms run on the Monitor Units.

The architecture shows the basic internal architecture for DPI application. The first and most important "knob" is the flow classifier. Several methods can be used, for example IP address range based, TCP/UDP port based, etc., selecting different portions of the traffic for deep inspection. It is even possible to configure the flow classifier from software, selecting the hardware filtering rules easily. Further user-selectable items are the statistic flow analysis modules. According to the required set of flow information, specific filters may be inserted in order to get the required information.

The C-Board with the DPI firmware architecture is capable of monitoring full speed the 10Gbps traffic in both directions, and it can forward it to the SFP interfaces without packet loss.

Finally, in a realistic use-case a part of the traffic can be streamed to the SFPs for packet level analysis. Up to 8 Gbps traffic can be streamed to the receiving monitor PCs dependably [13].

5.2 Generic Switch/Router architecture

The SCALOPES C-Board can also be used as a generic Switching/Routing architecture with hardware level packet forwarding. The architecture supports two 10 GbE ports, 16 GbE ports with an arbitrary combination of SFPs.

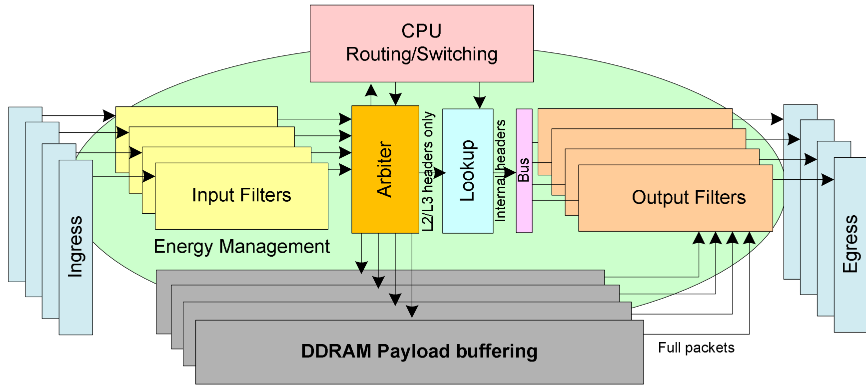


Fig. 6. Generic Packet forwarding architecture

The used packet processing pipeline is similar to the model recommended by Xilinx [15] and the model used by the Liberouter project [5]. The packets are filtered first, then all traffic is written in the DDRAM for buffering. Since the memory access can be the bottleneck in our packet processing pipeline, we tried to avoid copying the packet. We have decided to use the shared-memory packet forwarding model, this way avoiding unnecessary copying of data. The model has a further advantage: even multicasting/broadcasting can be done without actually copying the data.

The generic packet forwarding architecture is shown by **Fig. 6**. The arriving packets are buffered, while their header information is however forwarded to the lookup module. The lookup module is responsible to decide on which interface the packet should leave. For flooding, broadcast and multicast multiple interfaces may be selected. For different application types, different routing/switching modules can be designed, which may operate at different layers of the OSI model, like L2 switching, IPv4 or IPv6 routing, MPLS packet switching etc. Based on the decision at the lookup module, a short

internal header information is written in the egress interface queues (multiple queues in case of broadcast/multicast packets). The output filter block is responsible for scheduling and queuing, and retrieval of packet data from the DDRAM and transmitting on the egress interface.

The performance of this architecture is only limited by 2 main factors: lookup speed and DDRAM access. The lookup speed depends on the L2 or L3 forwarding table size and lookup algorithm, while DDRAM access may introduce delays in case of small packet sizes.

The generic packet forwarding architecture can be extended with a COM express based PC board, providing considerable processing power. This extension opens up further application possibilities for the board. Such a possible application is a Session Border Controller. Session Border Controllers (SBCs) have evolved to address the wide range of issues that arise when voice and multimedia services are overlaid on IP infrastructure. These include a wide range of operations from packet level monitoring tasks through flow level manipulation tasks to high level signaling processing tasks, all at high speeds. These put high demand on both hardware and software. With our C-board extended with a PC board a high performance SBC can be designed.

The key idea in this use case is the close interworking between the software processing on the PC board and hardware processing on the FPGA board. The low level processing handles the high-speed traffic and passes only the network signaling traffic to the processor. The routing protocols and forwarding control can be done just like in the previous case. An open protocol like OpenFlow [16] can be used to control the hardware flow processing.

6 Summary

The SCALOPES C-board is a versatile programmable platform capable of handling 10Gbps Ethernet traffic. It provides a base platform for various packet processing applications such as switching, routing filtering, monitoring, etc. Its modular structure allows its extension with processing cards to increase its applications with high-speed software processing as well.

The C-board is accompanied by a development environment to unlock its full potential. The environment supports the development process from design-space exploration and modeling in SystemC to modular design. It is based on predefined hardware modules, from which the basic applications (packet forwarding, DPI) can be constructed. The development environment will also feature a graphical user interface, providing easy development for customization. The scalability and low power requirements have been taken into consideration for both hardware and software design. We have demonstrated some major application fields for the hardware on use cases.

The DPI use case presents a configurable monitoring tool, with on-line processing capability and packet- as well as flow-level analysis. The Generic Switch/Router architecture demonstrates the use of the system as a high performance switch or router. We also highlighted a scenario for a session border controller device, capable processing of the traffic in both hardware and software.

The high-performance, low-power SCALOPES C-board can be widely used by operators, research engineers and application developers in order to tackle the challenges of effective traffic handling, network and service management tasks at 10Gigabit Ethernet connections.

Acknowledgments. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100029 and from the Hungarian National Office for Research and Technology (NKTH).

References

1. OSCI SystemC 2.2.0 Documentation: User's Guide, Functional Specifications, Language Reference Manual. Online: http://www.systemc.org/
2. M. Werner, J. Richling, N. Milanovic, V. Stantchev: Composability Concept for Dependable Embedded Systems, Proceedings of the International Workshop on Dependable Embedded Systems at the 22nd Symposium on Reliable Distributed Systems (SRDS 2003), Florence, Italy, 2003.
3. Endace packet capture hardware, Online: http://www.endace.com/high-speed-packet-capture-hardware.html
4. The NetFPGA project homepage, Online: http://www.netfpga.org/
5. Liberouter project homepage, Online: http://www.liberouter.org/
6. M. Ciobotaru, M. Ivanovici, R. Beuran, S. Stancu, Versatile FPGA-based Hardware Platform for Gigabit Ethernet Applications, 6th Annual Postgraduate Symposium, Liverpool, UK, June 27-28, 2005.
7. D. Antos, V. Rehak, J. Korenek: Hardware Router's Lookup Machine and its Formal Verification, ICN'2004 Conference Proceedings, 2004.
8. D. Teuchert, S. Hauger: A Pipelined IP Address Lookup Module for 100 Gbps Line Rates and beyond, The Internet of Future, pp. 148--157., ISBN 978-3-642-03699-6 (2009)
9. Martinek, T. and Kosek, M. NetCOPE: Platform for Rapid Development of Network Applications. DDECS 2008, April 16-18, 2008, Bratislava, Slovakia

10. M. Attig, G. Brebner,: High-level programming of the FPGA on NetFPGA, NetFPGA Developers Workshop August 12-14, Stanford, CA, 2009
11. J. Naous et. al., NetFPGA: Reusable Router Architecture for Experimental Research, *PRESTO'08,* August 22, 2008, Seattle, Washington, USA.
12. IEEE 1685-2009, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows, 2009
13. S. Plósz, I. Moldován, P. Varga, L. Kántor, Dependability of a network monitoring hardware, DEPEND 2010, July 18 - 25, 2010 - Venice/Mestre, Italy
14. D. Horváth, I. Bertalan, I. Moldován, T. A. Trinh, An Energy-Efficient FPGA-Based Packet Processing Framework, EUNICE 2010, LNCS 6164, pp. 31–40, 2010
15. N. Possley, Traffic Management in Xilinx FPGAs, White Paper, April 10, 2006. Online: http://www.xilinx.com/support/documentation/white_papers/wp244.pdf
16. N. McKeown et al.: OpenFlow: Enabling Innovation in Campus Networks, OpenFlow White Paper, Online: http://www.openflowswitch.org/

Chapter 3
Application Framework for Programmable Network Control

[1,2]Rudolf Strijkers, [2]Mihai Cristea, [2]Cees de Laat, [1,2]Robert Meijer

[1]TNO Information and Communication Technology, Groningen, The Netherlands,
[2]University of Amsterdam, Amsterdam, The Netherlands
{strijkers, m.l.cristea, delaat}@uva.nl,
robert.meijer@tno.nl

Abstract. We present a framework that enables application developers to create complex and application specific network services. The essence of our approach is to utilize programmable network elements to create a software representation of network elements in the application. We show that the typical pattern of an application specific network service is a control loop in which topology, paths, and services are continuously monitored and adjusted to match application specific qualities. We present a platform in which network control applications can be developed and illustrate possible use cases. Based on these use cases, new research questions are identified.

Key words: Distributed Computing, Network Management, Programmable Networks.

1 Introduction

Almost every type of network implements measures to guard against unexpected environmental changes, such as the effects of failing links, changing traffic patterns or the failure of network nodes themselves. Such measures can be considered as optimization of network resources with respect to network robustness. At the basis of the optimization of network resources are programs that control the response of the network to changes in and outside of the network. Moreover, actively controlling network resources is crucial to maintain the network service that is delivered to applications.

Optimizations have a certain penalty in realistic situations. For example, in sensor networks [1] minimizing the transmission power of sensor antennae optimizes battery lifetime, but impacts connectivity. Depending on the application and the actual situation, engineers will choose an optimum. Generally, the optimum network service is application-specific, yet in most

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_3, © Springer Science+Business Media, LLC 2011

networks, application programmers have no control over the network. One reason is that a general applicable, conceptual and technical framework to program the network is absent [2].

In the absence of any notion of specific application demands, as is usually the case, network providers offer typically a best or constant effort network service. Theoretically at least, computer programs can be so specific in their service requirement and optimal response to disturbances that network providers cannot configure and control the network for such applications anymore. If cloud infrastructures would only run on wind energy, for example, the amount and direction of wind will continuously change the energy available for computing and network resources. In such cases, (partial) control over the network must also be transferred to a computer program, i.e. the application domain, to automate continuous reconfiguration of the infrastructure.

Traditionally, networks have been designed according to well-defined requirements. One could say that at this point application domain knowledge enters the network domain. Conversely, application engineers may use the interface of a given network service, e.g. sockets in the Internet, to include the network in the application logic. Here, we extend the latter approach; any application-specific property of a network service becomes a network control issue programmed in the application domain, i.e. a dynamic user network interface. Moreover, we define the basic framework needed to design and build network control programs in the application domain.

In Section 2 we review state of the art of related areas in programmable networks, overlay network and sensor networks that allow network control from the application domain. Then, in Section 3, the application framework is presented and its functional components are described in Section 4. In Section 5, the implementation and test bed is introduced and Section 6 follows with examples of applications that control networks. The paper ends with conclusions and future work in Section 7.

2 Related Work

A basic approach to develop a programmable network is to use general-purpose computers as Network Elements (NE) and implement C programs that manipulate packet streams and network links [3-5]. The programmable and active network [6, 7] community developed the architectures for dynamic deployment and extensibility of functions in network elements. Other efforts provide programmability in the control plane of networks, while remaining backwards compatible with current Internet technologies [8-11]. These technologies enable network operators to offer better services to applications.

Basically, there are two types of limitations in networks that motivate application control: (1) limited network functions or (2) limited network resources. If the network does not offer enough functionality, a well-known

approach is to implement the network functions as part of the application, i.e. create and manipulate a virtualized network (overlay network). If the network has limited resources to accommodate application demands in a best-effort manner, frameworks exist to manage the quality of service on behalf of the application [12-14]. Next, we illustrate some approaches from related network research areas that deal with these limitations.

Overlay networks enable developers to redesign and implement, amongst others, addressing, routing and multicast services optimal to their application domain [15]. Overlay networks are widely used to support specific services, such as distributed hash tables [16], anonymity [17], and message passing [18]. Overlay networks might lead to sub-optimal utilization of network resources, because the mapping to the physical network resources is not open to the application developer. Moreover, overlay networks essentially duplicate functions offered by the physical network. Recently, some efforts [19] propose to expose physical network properties to applications to improve their mapping to the physical network. Assuming that networks are properly dimensioned, at least from the user's perspective, overlay networks are a straightforward solution to support their specific network service requirements.

Sensor networks illustrate best limitations in network resources. Sensor networks motivate tight integration of applications and network services [20]. Because of the resource constraints, sensor network designers attempt to use the scarce resources efficiently and various approaches to program sensor networks have been developed [21]. In *macroprogramming* [22], high-level programs use an intermediate language to abstract away concurrency and communication aspects in sensor application programming. A compiler translates the programs into basic instructions for individual nodes, and takes communication characteristics into account. In TinyDB [23], communication is integrated with a data query mechanism. *Macroprogramming* and TinyDB show that with a framework that structures the design space of network control applications, it becomes possible to design and implement reusable components for new applications.

Our research in advanced applications of networks [24-30] shows that applications have different optimal network services. Existing network management systems do offer APIs to configure network services [31]. Such APIs implement the network abstractions chosen by the network operator. We found that our use cases in hybrid networks and sensor networks require more flexible and specific network services than those designed and implemented by network operators. Because the application domain offers developers more flexibility, it might be more practical to implement network services as part of the application. Hence, we developed a model that enables developers to program networks as part of their application [32]. The resulting framework, User Programmable Virtualized Networks (UPVN), models the *interworking* between networks and applications and provides a conceptual framework to

investigate design patterns of application-specific network services. Here, we shortly introduce the model.

In UPVN (Figure 1), individual NEs are regarded as resources, which are used directly or through the Internet (open lines) as components in application programs. A NE component (NC) can be seen as a manifestation of the NE in the application, i.e. a virtualized NE. Consequently, all virtualized NEs together create a virtualized network, allowing interaction with user programs. To accommodate application specific packet processing, to set particular parameters of the NE, and to facilitate other functions NEs play in a UPVN, NEs have the ability to deploy Application Components (ACs).

UPVN's development is application driven; creating only those facilities that are crucial for applications while other operations remain automated. The NE uses technologies, such as Grid- and web services, to expose interfaces on the Internet. Through the interfaces a NE exposes, various applications interact simultaneously with the NE. As such, each application is capable to optimize the behavior of the NE accordingly. During application development, the NE appears as a software object, i.e. Network Component (NC), in the development environment. During run-time, state of the art technology allows dynamic extension of the set of NEs the applications interacts with.

The UPVN model leads to a practical framework in which network control is implemented as part of application domain programs and in which network services and optimizations are expressed in user-definable qualities. In the past, we developed a prototype UPVN that showed that the approach is feasible [33]. In Section 4 and 5, we present the design and implementation of a prototype that includes the control concepts we propose. In the following section, the application framework for programmable network control is introduced.

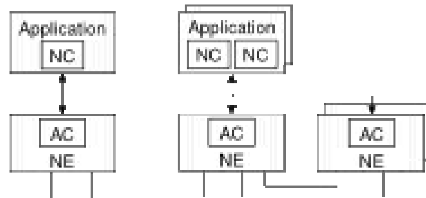


Fig. 1. Interworking model of applications and networks.

3 Application Framework for Network Control

Programmable network element technologies support dynamic network service composition for applications that need new network functions, such as network embedded trans coding of video streams. If changes occur in the network, however, applications must adapt to the new situation. The

adaptation process may be at the end-points, such as in TCP flow control process but may also be in the network, such as a process that changes the edge weights of a shortest path routing protocol [34]. The adaptation process typically consists of (1) inferring (possibly incomplete) network information, (2) calculating network state (3) and adjusting the network to a configuration that leads towards the desired optimum. A closed-loop control model, a well-known model in control theory to influence the behavior of a dynamic system [35], provides a minimal framework for network control (Figure 2).

In order to match the network to a state that is optimal to an application, the application has to collect (possibly incomplete) network information. The application developer chooses application specific abstractions (NC_x) to update a model the application uses internally. The application combines state information from all or a subset of NEs to update the internal model. In principle, the internal model can also include non-network related information, such as computing or hosting costs, sensor information and service level agreements.

The control application applies an algorithm to find the actions (NC_y) needed to adjust the network behavior in such a way that it matches the application needs (e.g. a stable, optimized state), which are described by the reference. To implement changes in the network, the control application translates decisions into instructions, such as create, forward or drop packets specific to each NE involved in the application. This means that the system needs to provide a distributed transaction monitor to keep network manipulations that involve multiple NE consistent.

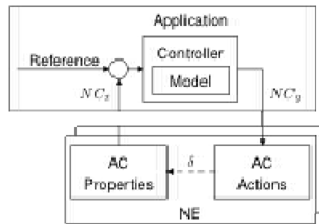


Fig. 2. The application framework to control networks contains a control loop.

In control theory, a measurement (AC Properties) from the system is subtracted from a reference value, which leads to an error value as input for the control application. In our framework, the measurements (AC Properties) that represent network state may use different metrics compared to the controlled state (AC Actions). For example, a controller may manipulate edge weights in shortest path routing based on throughput information. Such a scenario is meaningful if the relation between throughput and edge weights (δ) is known or can be learnt and would be useful to dynamically distribute traffic to avoid congestion, for example [34].

Applications exchange information ($NC_{x,y}$) with NEs over a communication network, possibly over the same network the application is controlling (in-band). Even though application developers may have access to a separate management network, the communication path between network and application complicates the design and validation of the controller. Network properties, such as latency and packet loss, limit the amount of information that can be exchanged or synchronized. So, NE state information can become incomplete, inaccurate or aged. The application developer has to understand the limits in information exchange of a given network, i.e. observability, when designing the control application.

This section introduced the abstractions needed to provide the basic framework for network control in the application domain. Next, the details related to interworking of applications and networks that lead to a functional model are described.

4 Functional Components

The OSI reference model organizes the interworking of applications and networks in seven layers [36]. The design principle of layering allows decomposition of a complex problem, but application specific details may be lost in the process. If network elements are virtualized in software, the application interface to the software (NCs) can be fine-tuned to the specific problem domain. However, the fine-tuning might lead to an application specific organization of network functions. Here, we define the organization of functional components to support fine-tuning of the application interface and organization of network functions. The functional organization preserves the context of the NEs by creating and managing the software representation of NEs in the application domain. For example, an application can use the software representation of NEs to manipulate traffic of a single strategic point in the network for filtering or anomaly detection purposes.

We identify three layers of abstraction in a distributed program: network

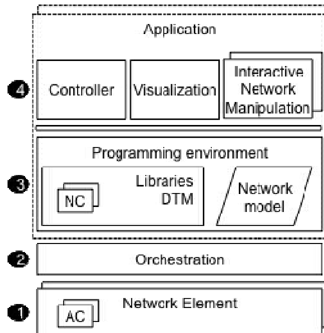


Fig. 3. Four functional layers characterize practical application domain network

element execution environment, middleware/orchestration, and application code. The latter can be subdivided in two sub layers, namely the programming environment providing reusable components such as programming libraries, and the application program. The result is a four-layer architecture (Figure 3). Clearly, the architecture resulting from the application point of view is similar to programmable network architectures [6]. However, the functional components between the application and programmable network need to be further defined to support network control from the application domain and is described next.

The orchestration layer (2) facilitates the interworking of software objects and ACs located on individual NEs (1). The orchestration layer may also supports basic mechanisms, such as discovery services, brokers, billing services, authorization, etc. The usefulness of these services depends on the network environment and application. In sensor networks, for example, there just may not be enough computational and storage resources to support an elaborate set of services.

The programming environment, layer (3), provides the NC implementation and reusable components, such as a Distributed Transaction Monitor (DTM) or breadth-first search algorithm, to support programming of a collection of NCs. Depending on the network environment, some abstractions can be implemented in the ACs, as a library in the programming environment or both. For example, the application developer might want to program network element interactions in a non-blocking manner. Hence, either the programming environment or the orchestration layer must facilitate non-blocking interaction mechanisms between ACs and NCs. In our implementation (Section 3) we use message passing in the orchestration layer and implemented (an easier to program) blocking interface to the application (Section 5).

Because network control is now part of the application domain (layer 4), developers can benefit from a large amount of existing software to implement network control programs. A characteristic of the control applications is that they operate on data structures that represent the network state. Therefore, the programming environment (3) explicitly contains a model of the network and the orchestration layer must supply the data with which the model can be updated. In Section 6, we discuss issues related to the accuracy of the network model.

Some applications support the construction of a network model that is close to mathematical concepts, such as graphs. The Mathematica [37] environment, for example, contains a graph data structure, which can be used as a basis for control applications that require graph algorithms. By enabling dynamic updates of network state into the Mathematica graph data structure,

domain experts can simply apply graph algorithms to find and remove (through network manipulation) articulation vertices; vertices that may disconnect a graph. Besides control, the application layer can also include visualization or other means of interaction with the network. The integration with toolboxes, such as those available in Mathematica, makes the application layer a powerful environment to develop network control applications.

5 Implementation and Test Bed

In the preceding sections, we introduced the framework for control applications as well as a four-layered functional model to implement such applications. We developed a test bed according to the presented functional model (Figure 3) to gain practical insight in the implementation of the application framework to support network control programs. The test bed implements the first three functional layers and enables further exploration of the network control applications that are part of the fourth layer.

5.1 Hardware

The test bed consists of eight machines (four dual processor AMD Opteron with 16GB RAM and dual port 10Gb NICs and four Sun Fire X4100 with 4GB RAM and 1Gb NICs) interconnected by two 1Gb switches and a Dell hybrid 1/10Gb switch. All machines run VMWare [38] ESXi hypervisor software and the virtual hardware is centrally managed and monitored with VMware vSphere management software. The test bed was bootstrapped with one Linux instance containing the software we developed, and iteratively grown to 20 instances to create a non-trivial configuration of networks and computers (Figure 4).

The setup involves two datacenter locations: a virtual infrastructure running in our datacenter in Groningen and an interactive programming environment including an interface to a multi-touch table running in our lab in Amsterdam. The multi-touch table enables users to interact with NCs (Section 6). The two locations are connected by two OSI-Layer 2 Virtual Private Networks (VPN) on basis of OpenVPN [39]: one for control traffic and one for data traffic. At the receiving host in Amsterdam, the control and data networks are separated by VLANs.

5.2 Software

The primary purpose of developing a prototype is to gain insight in the challenges and details to control a network from applications that require

dynamic traffic manipulation, and to enable experiments with various network control mechanisms. The implementation combines several open source

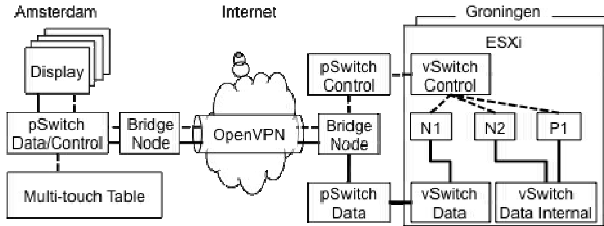


Fig. 4. Test bed and network connectivity.

software tools into one NE platform. We provide a global overview of the software that implements the functional layers.

Packet Processing and Token Networking. Fine-grained packet processing and manipulation facilities are implemented in Streamline [4], a tool originally developed for high-speed packet filtering and similar to other approaches presented in literature (Section 2). However, Streamline differs from other approaches by providing a simple and flexible query language to manipulate filter graphs on the fly (Figure 5) and a packet processing language FPL [40]. In addition, Streamline also allows dynamic loading of

```
(netfilter_fetch_in) >(fpl_tbs,expression="TOKEN")
\
  >(fpl_ipdest,expression="DST_IP")
>(skb_transmit)
```

kernel modules that provide specific packet manipulation functions.

We extended Streamline to support insertion, removal and filtering of tags in the IPv4 options field, which allows us to bind ACs to network traffic. A Streamline expression defines a chain of packet processing modules, which describes the network behavior for a particular application on a NE. Filters, such as *fpl_tbs* allow packets with specific tags to pass through a specific chain of packet processing modules. The expression is calculated for each NE separately by the control software and a distributed transaction monitor manages loading of each expression on the subsequent nodes to provision a path, for example.

Orchestration Software. The orchestration of ACs in the programmable network is implemented in Java. ACs available to applications, such as Streamline, are wrapped by Java objects. Network elements communicate using a peer-to-peer model. ACs register as a service on the network element. Each network element knows at least one peer to which it can connect (the controller). Currently, all peers connect to a single known controller, which provide basic message-passing functions over TCP sockets. The controller

also provides basic services that involve more than one NE, such as a distributed transaction monitor or topology discovery. The basic services are implemented as a set of ACs and can be used by network control applications. Currently only a single instance of the controller is used. Creating more controllers on-demand is a topic for future investigation. (Section 6).

Network Model. Our implementation provides various active and passive monitoring ACs that enables network control applications to create and maintain a network model:

Ping provides basic information about latency and jitter,

Network Mapper (NMap) [42] can detect nodes in the broadcast domain of an interface with ARP, and

/proc/dev/net is used to retrieve basic throughput information from the Linux kernel,

Uptime collects CPU load information.

The controller contains a Dispatcher AC that allows other ACs to subscribe to events, such as NEs registering to or detaching from the network and is the entry point for peers that connect to the control network. The Dispatcher AC subscribes to all known network elements and triggers network discovery requests when a new NE registers, consequently updating its network model to the new network state.

AC Management. Management functions, such as starting, stopping and manipulating AC of the programmable network implementation, are implemented in the Ruby [43] programming language. This allows new network behavior to be added at runtime, e.g. Java classes, kernel modules or installation of complete applications.

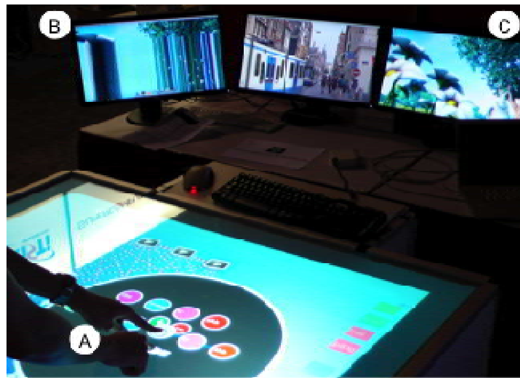


Fig. 6. A multi-touch table enables direct manipulation of programmable network components of 20 virtual machines. A user (a) modifies a sampler component of a streamline graph that multicasts a video to screen (b) and (c). As a result, the stream of (b) is distorted, while the other remains normal.

For example, a ruby script with instructions to compile new code for Streamline and insert it into the kernel can be remotely executed on NEs.

6 Network Control Programs

We showed a practical implementation of the model in Section 4, which enables a straightforward prototyping of network control programs. To test the setup an interface to view and modify the state of NCs was built. It allows manipulation of video streams produced by several nodes, which can be displayed on computers with a screen attached. By manipulating the NCs, a user can interact with the programmable network: create and modify paths and modify NE parameters, such as the packet processing chains of Streamline. We successfully demonstrated the setup at Super Computing 2008 in Austin, TX [44] (Figure 6).

We developed an interactive programming environment with Mathematica, which enabled automation of the possible user manipulations in the setup. Combining Mathematica with programmable networks allows advanced, yet straightforward implementation of network control applications. We implemented a Java adapter between Mathematica and the Management Agent (orchestration layer). The Java adapter deals with limitations of Mathematica's, such as real-time polling of the network, while being responsive to user input at the same time.

The Java adapter enables the Monitor AC to trigger the continuous updates of a number of data structures in the Mathematica kernel, such as *theNetwork*

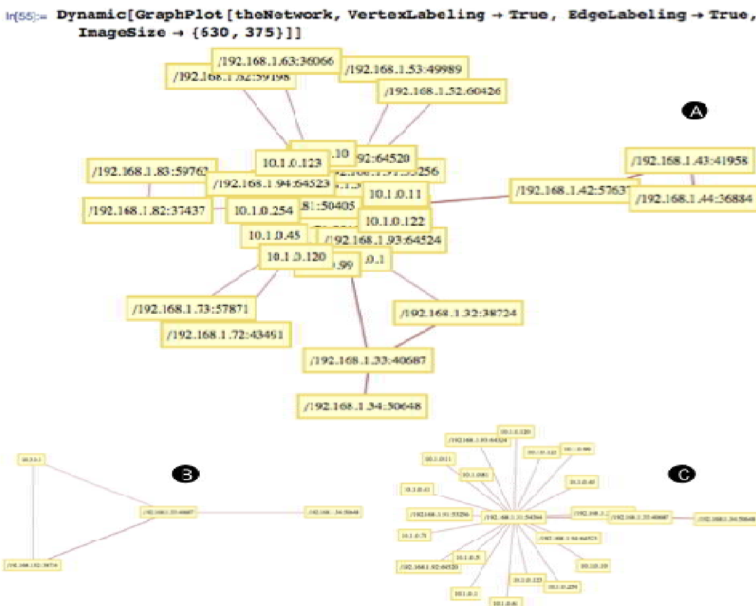


Fig. 7. Mathematica's function *Dynamic[]* facilitates continuous reevaluation of network state. The statement redraws the graph every time *theNetwork* data structure is updated with information of the network (a). Picture (b) and (c) show two stages of topology discovery.

or *thptNetwork,* and facilitates the development of control applications in Mathematica. An elementary control application is one that visualizes the network state while the data structures are updated (Figure 7). For example, the current IP network topology can be displayed while the discovery of the network is in progress; fully discovered in (a) and two intermediate steps (b) and (c). Another visualization example maps throughput measurements on a 3D contour plot (Figure 8). We also implemented various control applications using the test bed. For example, two control applications avoiding congestion were implemented by switching paths and by dropping packets on basis of throughput measurements. Another control application was to developed to continuously find and provision disjoint shortest paths [26]. Based on the experiments, we identify new research questions.

Application developers have to consider the accuracy of the network model. For network properties as throughput and delay some range of error can be tolerated. However, applications that require exact shortest paths require accurate topology information. The accuracy of the network model is influenced by the rate at which state information is (1) generated, (2) transported and (3) processed. At least (2) and (3) have architectural consequences for the control loop. One possible architectural consequence is to divide the network in multiple separately controlled domains, similar to areas in OSPF. In one extreme, dividing up the network into smaller individual control domains eventually leads to a fully decentralized architecture, i.e. peer to peer networks. In the other extreme, if network state can be generated, transported to and processed fast enough by one controller, then for practical purposes a centralized implementation might be preferred.

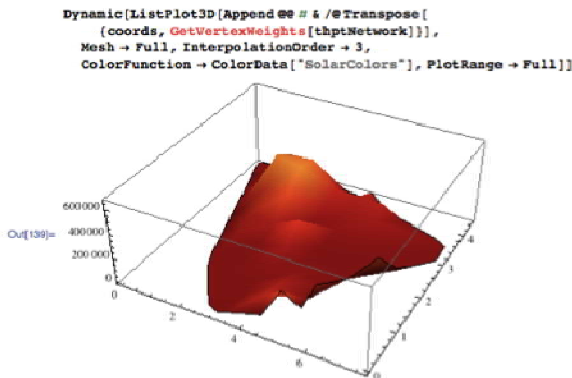


Fig. 8. Network throughput of the test bed visualized in Mathematica. The vertex weights in the *thptNetwork* data structure are updated with throughput values from the programmable network. The network topology is mapped to the x-y plane and throughput to the z-axis (in bytes per second). This way, a user can detect busy spots network and write algorithms to avoid such spots.

Application developers have to make a trade-off between state exchange and the processing capabilities of network elements. For example, an application that finds and removes articulation vertices can run as (1) a centralized component or, in the other extreme, (2) can run on each NE under its control. Because the computation of articulation vertices requires full topology knowledge, running the application on each NE (2) requires additional mechanisms to update and synchronizes changes in topology. Between centralized and decentralized implementations of control loops many architectural variants exist. Likewise, an enormous variety of control algorithms can be expected. On these points applications programmers would benefit from research [45] on design patterns of control loops.

7 Conclusion and Future Work

Until now, engineers optimize networks at design time and independent of application engineers. Examples from sensor networks, hybrid networks and overlay networks show a need to control networks at run-time. Past efforts created the programmable network element technologies to support dynamic network service composition. In this paper, we use these technologies in a framework for network service development in which each programmable network element has a software representation in a possibly distributed application. We presented an implementation of the framework and several network control applications.

Our implementations are limited to a single application that controls the network. In case many applications want control over the network, another control application is needed to manage (conflicting) resource demands, i.e. an operating system for networks. In the future, however, it can be expected that network management systems support mechanisms to host and run applications on the network. Recent research also continues in this direction (Section 2). More experience is needed to create reusable software components that enable and simplify control application development for large networks.

Control loops are a fundamental part of applications that optimize a specific network service as a response to changes in or outside the network. In subsequent research we shall determine the operational properties of a control application (e.g. how accurate is a given network state, what is the delay between network events and the application's ability to react, how fast can failures be detected). We have shown that architectural consequences can be expected when changes in the network occur faster than a single control loop can effectuate new adjustments, e.g. in large or unstable networks. In this case, the application framework needs to support decentralized network control. Hence, to extend the application framework to support multi-domain, multi-scale network control is a topic for further research.

Acknowledgments

We thank Wolfgang Mühlbauer, Burkhard Stiller and Bernhard Plattner for their comments and support.

References

1. Culler, D., Estrin, D., Srivastava, M.: Guest Editors' Introduction: Overview of Sensor Networks. IEEE Computer 37 (2004) 41-49
2. Ng, T.S.E., Yan, H.: Towards a framework for network control composition. Proceedings of the 2006 SIGCOMM workshop on Internet network management. ACM, Pisa, Italy (2006)
3. Elischer, J., Cobbs, A.: FreeBSD Netgraph pluggable network stack http://www.freebsd.org/, accessed at 10 August 2009
4. Bos, H., Bruijn, W.d., Cristea, M., Nguyen, T., Portokalidis, G.: FFPF: Fairly Fast Packet Filters. OSDI (2004)
5. Morris, R., Kohler, E., Jannotti, J., Kaashoek, M.F.: The Click modular router. SIGOPS Oper. Syst. Rev. 33 (1999) 217-231
6. Campbell, A.T., Meer, H.G.D., Kounavis, M.E., Miki, K., Vicente, J.B., Villela, D.: A survey of programmable networks. SIGCOMM Comput. Commun. Rev. 29 (1999) 7-23
7. Tennenhouse, D.L., Wetherall, D.J.: Towards an active network architecture. SIGCOMM Comput. Commun. Rev. 37 (2007) 81-94
8. Wang, W.M., Dong, L.G., Bin, Z.G.: Analysis and implementation of an open programmable router based on forwarding and control element separation. Journal of Computer Science and Technology 23 (2008) 769-779
9. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38 (2008) 69-74
10. Casado, M., Freedman, M.J., Pettit, J., Luo, J., Gude, N., McKeown, N., Shenker, S.: Rethinking Enterprise Network Control. Networking, IEEE/ACM Transactions on 17 (2009) 1270-1283
11. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: NOX: towards an operating system for networks. SIGCOMM Comput. Commun. Rev. 38 (2008)
12. Braden, R., Clark, D., Shenker, S.: Integrated Services in the Internet Architecture: an Overview. RFC1633 (1994)
13. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. RFC2475 (1998)
14. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. RFC3031 (2001)
15. Lua, K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. Communications Surveys & Tutorials, IEEE (2005) 72-93
16. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications.

Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, San Diego, California, United States (2001)
17. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation Onion Router. 13th USENIX Security Symposium (2004) 303-320
18. Open MPI: Open Source High Performance Computing http://www.open-mpi.org/, accessed at 11 August 2009
19. Xie, H., Yang, Y.R., Krishnamurthy, A., Liu, Y.G., Silberschatz, A.: P4p: provider portal for applications. SIGCOMM Comput. Commun. Rev. 38 (2008) ACM--362
20. Romer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wireless Communications 11 (2004) 54-61
21. Royer, E.M., Chai-Keong, T.: A review of current routing protocols for ad hoc mobile wireless networks. Personal Communications, IEEE 6 (1999) 46-55
22. Newton, R., Arvind, R., Welsh, M.: Building up to macroprogramming: an intermediate language for sensor networks. Proceedings of the 4th international symposium on Information processing in sensor networks. IEEE Press, Los Angeles, California (2005)
23. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30 (2005) 122-173
24. Meijer, R.J., Koelewijn, A.R.: The Development of an Early Warning System for Dike Failures. 1st International Conference and Exhibition on WATERSIDE SECURITY, Copenhagen, Denmark (2008)
25. Cristea, M., Strijkers, R.J., Marchal, D., Gommans, L., Laat, C.d., Meijer, R.J.: Supporting Communities in Programmable Networks: gTBN. IFIP Integrated Management 2009, New York (2009)
26. Strijkers, R.J., Meijer, R.J.: Integrating networks with Mathematica. 9th International Mathematica Symposium 2008, Maastricht (2008)
27. Cook, G.: ICT and E-Science as an Innovation Platform in The Netherlands. Cook Report on Internet Protocol. Cook Network Consultants (2009)
28. Portegies, Zwart, S., Ishiyama, T., Groen, D., Nitadori, K., Makino, J., Laat, C.d., McMillan, S., Hiraki, K., Harfst, S., Grosso, P.: Simulating the universe on an intercontinental grid of supercomputers. Submitted to IEEE Computer (2009)
29. Kruithof, N., Marchal, D.: Real-time Software Correlation. INGRID Workshop (2008)
30. Strijkers, R., Cristea, M., Khorkov, V., Marchal, D., Belloum, A., Laat, C.d., Meijer, R.: Network Resource Control for Grid Workflow Management Systems. SWF2010. IEEE, Miami, Florida (2010)
31. Haggerty, P., Seetharaman, K.: The benefits of CORBA-based network management. Commun. ACM 41 (1998) 73-79
32. Meijer, R.J., Strijkers, R.J., Gommans, L., de Laat, C.: User Programmable Virtualized Networks. Proceedings of IEEE International Conference on e-Science and Grid Computing. IEEE Computer Society (2006)

33. Strijkers, R.J.: The Network is in the Computer. Master Thesis. Informatics Institute, University of Amsterdam, Amsterdam (2009)
34. Fortz, B., Rexford, J., Thorup, M.: Traffic engineering with traditional IP routing protocols. Communications Magazine, IEEE 40 (2002) 118-124
35. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
36. Zimmermann, H.: OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection. Communications, IEEE Transactions on 28 (1980) 425-432
37. Wolfram Mathematica http://www.wolfram.com/mathematica/, accessed at 2 August 2007
38. VMWare http://www.vmware.com, accessed at 2 August 2007
39. OpenVPN http://www.openvpn.net/, accessed at 14 August 2009
40. Cristea, M., de Bruijn, W., Bos, H.: FPL-3: towards language support for distributed packet processing. Proceedings of IFIP Networking '05 (2005)
41. Linux Netfilter http://www.netfilter.org, accessed at 17 August 2009
42. Network Mapper http://nmap.org, accessed at 7 April 2008
43. Thomas, D., Fowler, C., Hunt, A.: Programming Ruby. Pragmatic Bookshelf (2004)
44. Strijkers, R., Muller, L., Cristea, M., Belleman, R., Laat, C.d., Sloot, P., Meijer, R.: Interactive Control over a Programmable Computer Network using a Multi-touch Surface. ICCS 2009. LNCS, Baton Rouge, Louisiana (2009)
45. Feitelson, D.G.: Distributed Hierarchical Control for Parallel Processing. Computer 23 (1990) 65-77

Chapter 4
Facilitating Adaptive Placement of Management and Control Functions in Converged ICT Systems

Dominique Dudkowski and Marcus Brunner

NEC Laboratories Europe, Network Research Division,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
{dudkowski|brunner}@neclab.eu

Abstract. Mechanisms for the management and control (M&C) of large-scale ICT systems, both established and innovative ones, generally follow a distinct approach on the dimensions from centralized to distributed and flat to hierarchical architectures. In this paper, we examine representative M&C frameworks and technologies and show that such a restrictive architectural choice is incompatible with system convergence, like computing/networking and fixed/mobile. To improve this situation, we propose a novel architectural approach that facilitates the adaptive placement of M&C functions by using different combinations of distribution and hierarchy patterns concurrently. Using a computing/networking systems scenario and a simulator prototype, we illustrate the potential of the proposed M&C approach in achieving more efficient overall M&C of converged ICT infrastructures.

Keywords: Management systems, control systems, centralization, distribution, hierarchy, convergence of ICT infrastructures, OpenFlow.

1 Introduction

Modern systems of information and communication technology (ICT) are in the process of converging on several dimensions, such as computing/networking and fixed/mobile. Powerful complementary paradigms, most notably virtualization, enable completely new and comprehensive system architectures, such as cloud computing, cloud networking, and amalgamations of both.

Managing such converged ICT systems becomes challenging because traditionally separated management realms now need to be considered within

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_4, © Springer Science+Business Media, LLC 2011

a single framework. For example, managing IT and network resources naturally differs significantly, and virtualization leads to new performance constraints between computing and networking resources that require the redesign of management functions.

We argue that management and control (M&C) systems must actively support both the static nature as well as the dynamic process of convergence in ICT systems in a way to achieve sustainable M&C for those ICT systems that are yet to be developed. Current M&C systems, however, are not flexible enough and competing, disparate views dominate state of the art management and control systems.

In the quite heterogeneous landscape of M&C approaches, distribution and hierarchy can be identified as two of the strongest architectural separators that hinder the management and control of converged systems' performance. Let us consider the dimension of distribution, delimited by centralized and distributed forms of M&C. Surveying the state of the art shows that established as well as innovative M&C control technologies prevail on both ends. In the centralized case, classical SNMP architectures, for example, have proven to be highly reliable, and modern control architectures, such as OpenFlow [1], [2], show that centralized M&C continues to make sense. On the other end, fully distributed architectures lead to M&C functions embedded in the network elements. Two examples are the established spanning tree protocol (STP) in Ethernet and novel in-network management (INM) approaches [3], [4], where distributed execution of M&C functions is strongly preferred and, optimally, does not require external intervention.

More than static choices, M&C is also characterized by transitional developments, where, for example, handover management in wireless telecommunication networks moves from centralized to more distributed solutions (e.g. [5], [6], [7]). It should further be left to the system administrator which M&C approaches he or she considers to be the most suitable ones, depending on any of a system's characteristics (e.g. size) that may require special consideration. Last but not least, system characteristics will likely change over time. For example, a system that is successful will grow in size and some of its M&C functions may have to be enhanced over time to maintain efficient system operation.

Because fixed architectural choices put tight constraints on the distributed and hierarchical placement of M&C functions, they provide a limit for adaptability of converged system M&C. In contrast, there is a need for freedom of functional allocation. Central control and management have benefits, such as network element simplification (off-box), in terms of information to be used for decision-making, and it is a convenient single point for attaching policies, also high-level, such as business objectives. Strong distribution may lead to significant gains in performance, reliability, scalability, flexibility, and robustness [8], e.g. when functions are only locally

relevant they may significantly reduce the risk of producing computing and communication bottlenecks. Hierarchical structures, in addition, have the benefit of improving scalability with little impact on algorithmic logic.

It seems inadvisable to attempt to reconcile M&C approaches by simply bundling them within a single M&C system, because this would lead to tremendous complexity and redundancy of functions. It is more appropriate to conceive a homogeneous approach that allows for different mechanisms to coexist.

In this paper, we propose a novel architecture for the management and control of converged ICT systems, which targets at the two dimensions of distribution and hierarchy. In Sec. 2, we first show how state-of-the-art M&C frameworks can be characterized along these dimensions in order to illustrate the lack and need for a homogeneous architectural approach that is able to unify individual frameworks. In Sec. 3, we introduce a compact architectural framework that facilitates the coexistence of different M&C paradigms and the adaptation of M&C function placement over time to keep up with the dynamics in system convergence. In order to show that the chosen M&C framework is adequate for practical deployment, we apply it in Sec. 4 to the scenario of an OpenFlow-enabled data center, in which computing, networking and virtualization must be considered by management and control. In the same section, we discuss briefly our simulator prototype and relevant implementation details and sketch a number of qualitative results in order to assess the proposed M&C framework. We conclude in Sec. 5 with a summary and a brief outlook on future work.

2 Related Work

Management and control frameworks in both the literature and deployed systems are abundant. We therefore focus here on a number of representative approaches that we present in Fig. 1 in the design space that is formed by the two dimensions of *distribution* and *hierarchy*, based on an extension and combination of previous classifications in [8], [9], [10], [11], [12]. Each M&C approach is represented by a shaded rectangle that indicates its approximate location in the design space and relative to other approaches. The dashed line indicates a hybridization of approaches, and arrows indicate selected trends in one or more dimensions.

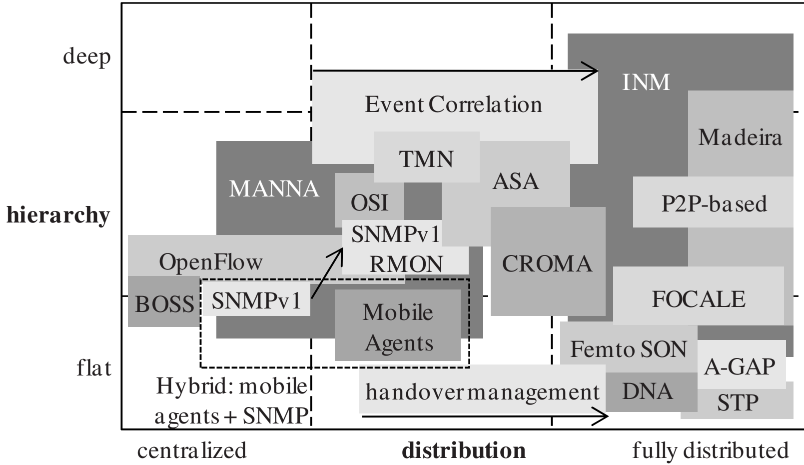


Fig. 1. ICT management and control systems on the distribution and hierarchy design space.

Several distributed management frameworks make use of highly distributed function placement, for example, In-Network Management (INM) [3], [4], the Autonomic Service Architecture (ASA) [13], the CROMA architecture [14], DNA [15], Madeira [16], Focale [17], and the framework proposed in [18]. These frameworks also vary in the depth of applied hierarchies, for example, INM and CROMA allow for a flexible stacking of hierarchical levels for objective management and policy management, respectively, while the framework in [18] is restricted to three layers and Focale's autonomic management elements suggest cooperation on a relatively flat hierarchy. These frameworks follow distribution and hierarchy patterns that are in line with innovative management and control algorithms, such as handover management [5], [6] and femtocell self-organizing interference management in [7] in the context of fixed/mobile convergence, flat management and control protocols such as A-GAP monitoring [19] and STP, and distributed event correlation frameworks with hierarchical patterns such as [20].

In wireless sensor networks, although highly distributed in nature, management frameworks capture virtually all coordinates in the design space [21]. For instance, the centralized system in [22] performs management operations purely external to the network, while in [23] management tasks are performed by cooperating network nodes. The motivation for choosing any of such approaches is to achieve the best compromise between computation and communication overheads that are dictated by specific management functions. Hence, WSNs substantiate the claim that coexistence of different architectural patterns is essential for flexible function placement.

The authors of [24] also motivate the coexistence of centralized and distributed management approaches, and concrete hybrid management architectures are proposed, for instance, in [25], [26]. In these architectures, more traditional centralized and weakly distributed management frameworks, e.g. SNMPv1, SNMPv1 + RMON, and TMN [27] (indicated in Fig. 1 according to [8], [9], [10]), are combined with distributed management approaches that make use of mobile agent technology (e.g. [28], [29]). It is clear from these hybrid approaches that coexistence is vital for achieving optimal performance when different management functions are to be combined, such as high-level versus localized management.

Of specific interest to network virtualization in particular is the recently proposed OpenFlow network switching technology [1], [2], which is characterized by a centralized controller that manipulates traffic flows by parameterizing the flow tables of individual network elements, which hence is in direct opposition to distributed management frameworks. Complementary, virtual switching in virtual machine monitors (hypervisors), such as [30], provide new abstractions to manipulate multiple virtual switches (Open vSwitches in [30]) to be controlled via a single logical image from a central point. At the same time, trends towards deeper embedding of control functions from host to programmable network interface card (NIC) space are pushed by new performance constraints that are dictated by virtual switching in the host. Such transitions are facilitated by e.g. [31] and lead to a shift of general networking functions [32] and in particular OpenFlow-related functions [33], [34] to the NIC.

3 Adaptive Placement of Management and Control Functions

In this section we introduce the principles and the framework for implementing a management and control system that is suitable for converging ICT systems. In particular, this system is focusing on the facilitation of different degrees of distribution and hierarchies as we will explain.

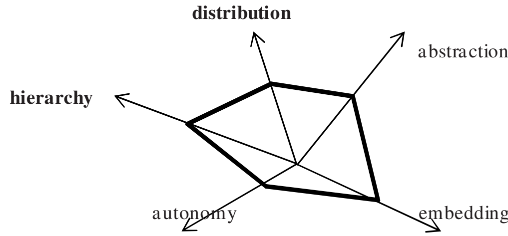


Fig. 2. Design space of management and control on 5 dimensions.

Figure 2 is an extension to the design space that we introduced previously in [3], where we have identified the degree of autonomy, abstraction, and embedding as three fundamental dimensions along which various network management functions can be designed. In this paper, we focus on distribution and hierarchy; the concepts we present in the following are orthogonal to the work presented in [3], [4], [35].

3.1 Management/Control Capabilities

Let us start by examining the distribution and hierarchy scale in more detail, where we can identify an intersection between both as a suitable starting point of our design. In that intersection, a transition from vertical to horizontal processing of management tasks (flows) occurs. We explicitly distinguish the semantics of these flows into organization (vertical) and collaborative (horizontal) to indicate the different nature of these flows. For instance, organization tasks may be used to inject high-level objectives or report key performance indicators, while collaboration implements the algorithms that obtain the values for these.

Based on this view, we define the basic constituting component of *management capability* (MC) which we have introduced on a high level in [3]. In order to define its role in acting as a transitional element (mediator) between organization and collaboration flows of management, we also define its internal structure in such a way that it can flexibly support that transition without defining the specific management task. Fig. 3 depicts the structure of a management capability.

External Interfaces of Management Capabilities

Let us consider the management capability's external interfaces first that integrate the capability into more complex management and control structures (Sec. 3.2). Two distinct interfaces, termed organization and collaboration

interface, define how the management capability interacts with other management capabilities on the level of organization (vertical) and collaboration (horizontal).

More specifically, the collaboration interface is related to algorithm logic and deals with exchanging information or commands such that a certain management algorithm can be performed collaboratively across several instances of the same type of management capability, or across different functions collaboratively in order to achieve a certain management task. The particular implementation of this interface may follow different communication and programming styles, for instance, an RPC or REST style, discussed in more detail in Sec. 3.4. A typical interaction of an aggregation algorithm, such as A-GAP [19], for instance, may be the exchanging of partly aggregated values, and the management algorithm component would make them a full aggregate eventually that is finally sent in upward direction.

The organization interface has two major tasks. Firstly, it exposes information to higher levels of abstraction along the organization hierarchy, the information typically being aggregated or filtered towards a high level of presentation (e.g. at the management console). Second, the interface receives objectives that are expressed in a higher level of abstraction and transforms them into lower level ones. This interface may also follow different programming styles (Sec. 3.4).

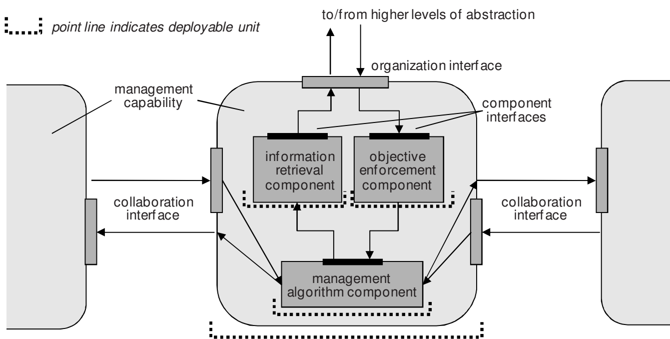


Fig. 3. Structure of management/control capabilities.

Both organization and collaboration interfaces exist virtually independent of the location in which a management capability is executed, which is the key enabling concept for placing a management capability in any location of the system that is to be managed (see Sec. 4).

Internal Structure of Management Capabilities

While a management capability's interfaces allow for the necessary flexibility in placing the capability into the managed system's runtime environment as a whole, a more detailed decomposition of MCs is used to make placement also flexible with respect to organization and collaboration tasks independently.

In order to allow this, without loss of generality, three basic components make up a management capability's interior. Within the management algorithm component, the actual management algorithm, such as A-GAP, is located. This component is associated with collaboration and only exposes a lean interface via which information that is relevant for the organization perspective is transferred. This interface is usually well understood by the algorithm designer and can be defined appropriately, but also be extended easily. For instance, in the case of A-GAP, the top-level aggregation result may be accessed via this interface in upstream direction, and basic threshold parameters of the aggregation algorithm may be set in downstream direction. The management algorithm component is further mapped to the management capabilities external collaboration interface to allow for communication with other MCs.

The information retrieval component and objective enforcement component are responsible for handling information retrieval and composition in upstream and downstream direction along the organization hierarchy. Note that objectives might also be called policies. In this paper, we strive to be independent of the particular mechanisms used to specify objectives, so policies but also simple configuration of thresholds may all be executed via the organization path. Both components perform tasks that are related to organization only. For instance, the information retrieval component may simply receive the top-level aggregate from one management capability and hand it over to any other management capability via the organization interface. The objective enforcement component receives objectives (policies) from upper levels in the organization hierarchy and may parameterize both the internal policy and the management algorithm component. An example for the latter case is to adapt the performance of a management algorithm, e.g. increase the aggregation latency for a tradeoff of aggregation accuracy.

3.2 Management and Control Structures

From individual management capabilities, arbitrarily complex management and control structures can be created. Such structures extend vertically from one or several global management points that terminate the information chain at the most abstract level (e.g. management consoles) down to MCs that terminate the management/control hierarchy at the lower end, e.g. at

individual network elements or at one of the lower layers of a network protocol stack, down to the hardware of individual network elements (e.g. network switches).

In a typical management task, for instance, a low-level objective is violated in a management capability and resolved by the next upstream capability using a specific self-adaptation mechanism without the need to report to a general management point. This example illustrates the ability of the proposed framework to adopt self-management principles, which are considered as an important means for local self-management operations in order to improve the management/control systems overall performance and to avoid explicit interactions with human beings.

3.4 Realization Options

Implementing MC-based network management and control works by defining the logic of management processes first, and then deploying management and control functions onto the system in question. The definition of management processes can be done by a language that is runtime-independent, which allows specifying functional elements (e.g. network elements), the functions these elements are to carry out, and how communication between elements is done.

In particular, the organization and collaboration functions may be implemented using different kinds of programming/communication models. Based on our assessment, REST (representational state transfer) and RPC (remote procedure call) are suitable for organization and collaboration, respectively. The reason is that organization tasks are likely to be more abstract in that an objective is modeled as a resource and manipulated via one of the few generic methods according to REST. For collaboration, it is naturally the case that algorithms (e.g. for monitoring, fault detection, anomaly detection) are rather diverse and require specific interfaces to communicate with each other, specific to the algorithm. These may be more suitable designed via RPC or a new protocol suit is defined for that.

Once a management process is defined, it is to be deployed. This phase requires specification of how to deploy a management function, and available support, e.g. in terms of compilers, will determine possible target runtime environments (also hardware is possible). For instance, OSGi is considered a suitable candidate to implement individual management functions, and brings with it already great flexibility and system support (such as discovery of services, which would be mapped to the management capabilities, etc.) that allows to easily relocate functions. Moreover, it is possible that existing hypervisor environments can be used as runtime environments in virtual environments. From these, in a transitional step, similar capabilities can then

be also recompiled and redeployed to programmable network interfaces, which is beginning to be applied in virtualized systems (e.g. [32]).

A major feature of the M&C framework is the support for deployment on different levels. While the management capability usually encapsulates functions that are specific to a management task, such as the aggregation and upward propagation of a key performance indicator, deploying the components of a single management capability is key for allowing various kinds of tradeoffs, such as performance versus memory space. An example is that of a virtualized environment, in which a management capability's algorithm component is located in the runtime environment of a programmable NIC due to performance reasons, while the information retrieval and objective enforcement components are located in the virtual hypervisor that is located "closer" to the virtualization system's policy engines.

4 Scenario: OpenFlow Control and In-Network Management

We now describe an example scenario that combines some of the features of centralized, hierarchical, distributed, and flat technology and show how management capabilities facilitate the transition in parts of that scenario.

4.1 Application to Converged Systems

We consider how to apply management capabilities to a typical converged system, in this case, IT and network convergence occurring for example in large data centers. Fig. 4 illustrates how management capabilities can be applied to an IT/network converged management and control system in general. In the figure, both hierarchical and distribution concepts are indicated and some of the management capabilities translate between these two directions.

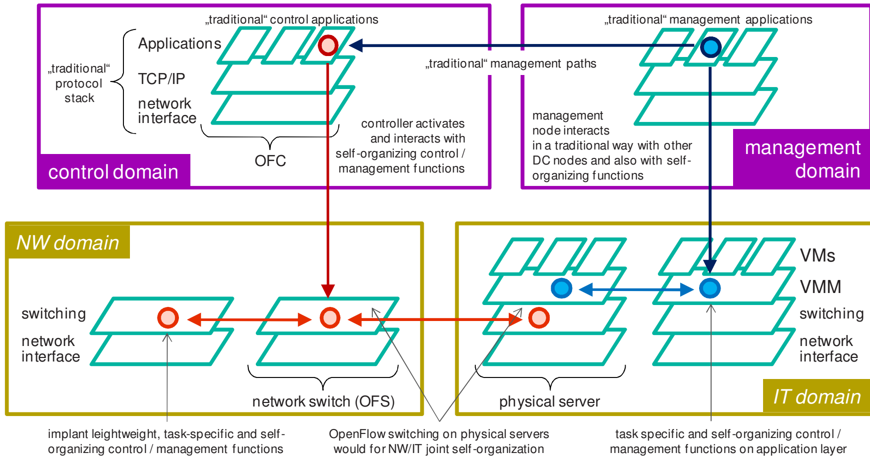


Fig. 4. Application of management capabilities to a converged IT/network system.

In this figure, SNMP in a weakly distributed form, OpenFlow centralized control, and INM distributed network management with objectives on the IT side are all combined in a single architecture. The figure shows in particular how centralized approaches merge with decentralized management functions, that is, INM. To provide uniform handling of the technologies, adapters are used that encapsulate e.g. SNMP agents to allow standardized communication between management capabilities of other types of management and control protocols. Note in particular the resulting homogeneous switching layer, where each network element (including servers) supports OpenFlow switching functionality homogeneously.

4.2 Scenario Description

The scenario we consider is a data center with IT and network resources, and virtualization. We consider two use cases that are closely linked: 1) anomaly (congestion) detection, and 2) virtual machine migration with flow rerouting that follows after an anomaly has been detected (also see Fig. 5).

Assumptions

1) An anomaly occurs locally, e.g. due to the exhaustion of CPU capacity at a host because of high load that is incurred by virtual machines.
2) Local performance checks find out about other physical hosts to which some virtual machines may be migrated.

3) One or more virtual machines are migrated, and flows are automatically adapted without contacting the central OpenFlow controller. This requires that a subset of controller functions must be implemented in the host initially, that is, the hypervisor that is executed on the physical machine.
4) In a transitional step, the required OpenFlow controller functions are moved from hypervisor space to the programmable network interface for improved efficiency. This step is the offloading of network functions that follows the idea of [32].

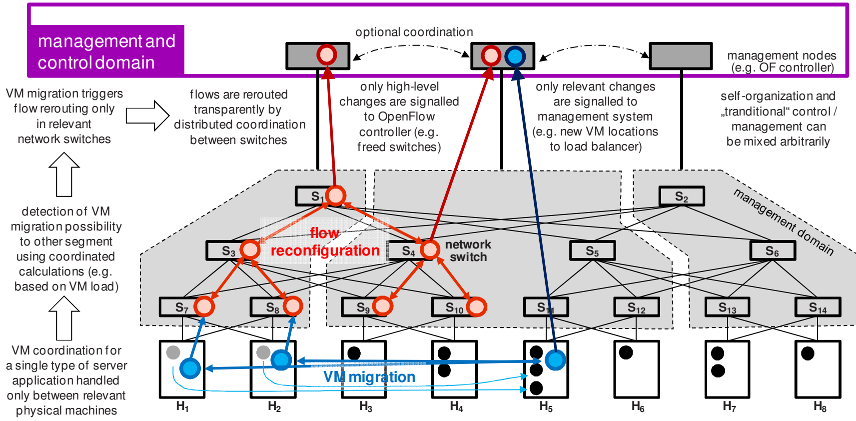


Fig. 5. Converged IT/network scenario in a typical data center.

In the described scenario, it is essential that some modifications in the deployment, e.g. from host (hypervisor) space to NIC space, are likely not to be available immediately but require upgrading on the hardware side, for example. A typical occurrence of events is that during the detection of anomalies at a physical host, the management system that is currently deployed in the data center sets a limit on the maximum number of virtual machines that may be deployed on a host. With the availability of more advanced NICs supporting programming, the transitional step in 4) can be applied after the redeployment of the corresponding management capability to the programmable NIC execution environment.

4.3 Discussion of Scenario

In order to evaluate the feasibility and benefits of proposed framework for adapting the placement of management and control functions, we have implemented a simulation prototype for simulating large-scale data centers,

and specifically, data center networks. The prototype is able to perform migration of virtual machines and flows by cooperation between network and virtual machines.

In previous work, the advantage of different architectural choices for network management and control in terms of hierarchies and distribution has been shown to a large extent in quantitative terms. For example, Du et al. [28] evaluate the benefits of a distributed agent-based management approach versus a centralized approach, where the distributed architecture incurs significantly smaller communication cost in terms of number and size of data packets compared to a centralized solution. Furthermore, most of the architectural choices that the discussed frameworks and algorithms in Sec. 2 follow are motivated by certain desired performance characteristics.

Rather than providing additional quantitative results, we focus here on evaluating the central claim of our framework, which is the *adaptive support* for functionality placement along the hierarchy and distribution dimensions. From the scenario that we have implemented in the simulator prototype, we can summarize the following points where dynamic placement is highly beneficial. We focus on a number of concrete practical situations of the described scenario:

- **Pushing OpenFlow control functionality down the hierarchy:** in the first OpenFlow implementations, network control remains within the OpenFlow controller. The scenario has shown that dynamic placement is highly desirable to push OpenFlow controller functions down to network elements. Moreover, it is beneficial to do this in at least two steps. In the first step, control functions to manipulate network flows can be pushed to a host (e.g. to hypervisor space). In a second step, control functions can be pushed further to the programmable NICs. Our framework supports this function pushdown on the level of individual management capabilities.
- **Increasing scalability with scenario size:** When the size of the scenario changes, e.g. when additional network elements and physical servers are added, it might become necessary to change function placement in order to maintain scalability and to avoid bottlenecks incurred by management and control functions. It is easy to expand the management and control structures via our framework in this case by extending the structures on the collaboration and/or organization direction by introducing additional management capabilities. Assuming deployment frameworks are used that support discovery and other basic runtime functions, which are provided e.g. by OSGi, it is straightforward to adapt structures dynamically using the mechanisms provided by the proposed management capabilities.

- **Function replacement during runtime:** While hierarchical and distribution structures of the management and control system do not always change, it sometimes is necessary to replace certain components. In a concrete case, an aggregation algorithm that aggregates data related to the anomaly of the described scenario may change. In this case, replacing just the management algorithm component of a management capability (see Fig. 3) is necessary. This is also supported by the proposed framework and can be readily implemented by frameworks such as OSGi.
- **Performance optimization of a management and control system:** Dynamic placement allows to "experiment" with different locations of a function in the system and to choose the best placement after having assessed different trials. For example, it might not be immediately clear what is the best function placement due to system complexity. This would correspond to a more short-term dynamic placement, rather than a long-term dynamics for evolution in the managed/controlled system, which is both supported by our framework.

5 Conclusion

In this paper, we proposed a framework for adaptive placement of management and control functions in converged ICT systems. The framework introduces management capabilities and their internal composition as the basic component to compose complex management and control structures that are flexible with respect to the placement and wiring of individual functional parts.

The framework particularly considers the dimensions of hierarchical and distributed composition of management functions, and provides the means to implement more flexibly combinations that are compatible and do not have to coexist without intersection. In other words, the proposed concepts do not force the adoption of specific technologies that work in a specific distribution scheme, but these can be combined. Fundamentally, that is possible if management capabilities are described and the interfaces are understood between different components, which might require a certain degree of standardization. In this paper we also studied a scenario integrating management and control (M&C) of different technology domains, namely, IT and network, and different M&C technologies, OpenFlow and SNMP.

While our simulation prototype gives valuable insights into the adaptive placement of management functions and shows that it is beneficial to have support for flexible function placement, the framework needs yet to be

specified in more detail. Additionally, more detailed analysis of how the deployment of functions and adaptive placement can be accomplished in technical terms via suitable execution environments (e.g. OSGi) needs to be carried out.

References

1. Appenzeller, G.; Balland, P.; Casado, M.; Erickson, D.; Gibb, G.; Heller, B.; Kobayashi, M.; McKeown, N.; Pettit, J.; Pfaff, B.; Price, R.; Sherwood, R.; Talayco, D.; Tourrilhes, J.; Yap, KK; Yiakoumis, Y.: OpenFlow Switch Specification. Version 1.0.0 (Wire Protocol 0x01), OpenFlow Consortium, December 31, 2009.
2. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J.: OpenFlow: Enabling Innovation in Campus Networks. White Paper, OpenFlow Consortium, March 14, 2008.
3. Dudkowski, D.; Brunner, M.; Nunzi, G.; Mingardi, C.; Foley, C.; Ponce de Leon, M.; Meirosu, C.; Engberg, S.: Architectural Principles and Elements of In-Network Management. In: Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM'09), pp. 529-636, Hempstead, NY, USA. IEEE Press (2009)
4. Prieto, A. G.; Dudkowski, D.; Meirosu, C.; Mingardi, C.; Nunzi, G.; Brunner, M.; Stadler, R.: Decentralized In-Network Management for the Future Internet. In: Proceedings of the IEEE International Conference on Communications Workshops (FutureNet'09), pp. 1-5, Dresden, Germany, 2009. IEEE Press (2009)
5. Schröder, A.; Lundqvist, H.; Nunzi, G.: Distributed Self-Optimization of Handover for the Long Term Evolution. In: Proceedings of the Third International Workshop on Self-Organizing Systems (IWSOS'08), pp. 281-286, Vienna, Austria, 2008. LNCS 5343, Springer Berlin / Heidelberg, Germany (2008)
6. Suciu, L.; Guillouard, K.: A Hierarchical and Distributed Handover Management Approach for Heterogeneous Networking Environments. In: Proceedings of the International Conference on Networking and Services (ICNS'07), Athens, Greece, 2007. IEEE Computer Society (2007)
7. Haßlinger, G.; Andersen, F.-U.: Business Value Scenarios of Inherent Network Management Approaches for the Future Internet. In: Proceedings of EUROMEDIA 2009, pp. 91-95, Bruges, Belgium (2009)
8. Martin-Flatin, J. P.; Znaty, S.; Hubaux, J.-P.: A Survey of Distributed Enterprise Network and Systems Management Paradigms. In: Journal of Network and Systems Management, pp. 9-26, vol. 7, no. 1 (1999).

9. Martin-Flatin, J.-P.; Znaty, S.: A Simple Typology of Distributed Network Management Paradigms. In: Proceedings of the 8th IFIP/IEEE International Symposium on Distributed Systems: Operations and Management (DSOM'97), pp. 13-24, Sydney, Australia (1997)
10. Martin-Flatin, J. P.; Znaty, S.: Annotated Typology of Distributed Network Management Paradigms. Technical Report SSC/1997/008, SSC, EPFL, Lausanne, Switzerland (1997)
11. Kakadia, D.; Thomas, T. G.; Vembu, S.; Ramasamy, J.: Enterprise Management Systems Part I: Architectures and Standards. Sun BluePrints™ OnLine, Sun Microsystems, Santa Clara, CA, USA (2002)
12. Pras, A.; Schönwälder, J.; Burgess, M.; Festor, O.; Pérez, G. M.; Stadler, R.; Stiller, B.: Key Research Challenges in Network Management. In: IEEE Communications Magazine, vol. 42, no. 10, pp. 104-110 (2007).
13. Cheng, Y.; Farha, R.; Kom, M. S.; Leon-Garcia, A.; Hong, J. W.-K.: A Generic Architecture for Autonomic Service and Network Management," In: Computer Communications, vol. 29, no. 18, pp. 3691-3709 (2006)
14. Davison, R.; Hardwicke, J.: A New Architecture for Open and Distributed Network Management. In: Proceedings of the 6th International Conference on Intelligence and Services in Network (IS&N'99), pp. 25-38, Barcelona, Spain, 1999. LNCS 1597, Springer-Verlag, London, UK (1999).
15. Binzenhöfer, A.; Tutschku, K.; auf dem Graben, B.; Fiedler, M.; Arlos, P.: A P2P-based Framework for Distributed Network Management. In: Wireless Systems and Network Architectures in Next Generation Internet. LNCS, vol. 3883, pp. 198-210, Springer, Heidelberg (2006)
16. Zach, M.; Parker, D.; Fahy, C.; Carroll, R.; Lehtihet, E.; Georgalas, N.; Marin, R.; Serrat, J.; Nielsen, J.: Towards a Framework for Network Management Applications Based on Peer-to-Peer Paradigms. In: Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS'06), Vancouver, Canada (2006)
17. Jennings, B.; van der Meer, S.; Balasubramaniam, S.; Botvich, D.; Foghlú, M. Ó.; Donnelly, W.: Towards Autonomic Management of Communications Networks. In: IEEE Communications Magazine, vol. 45, no. 10, pp. 112-121 (2007)
18. Panisson, A.; Rosa, D. M.; Melchiors, C.; Granville, L. Z.; Almeida, M. J. B.; Tarouco, L. M. R.: Designing the Architecture of P2P-Based Network Management Systems. In: Proceedings of the IEEE Symposium on Computers and Communications (ISCC'06), pp. 69-75, Pula-Cagliara, Sardinia, Italy (2006)
19. Prieto, A. G.; Stadler, R.: A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives. In: IEEE Transactions on Network and Service Management, vol. 4, no. 1 (2007)
20. Martin-Flatin, J. P.: Distributed Event Correlation and Self-Managed Systems. In: Proceedings of the 1st International Workshop on Self-*

Properties in Complex Information Systems (Self-Star'04), pp. 61-64, Bertinoro, Italy (2004)

21. Lee, W. L.; Datta, A.; Cardell-Oliver, R.: Network Management in Wireless Sensor Networks. In: Denko, M. K.; Yang, L. T. (eds.) Handbook of Mobile Ad Hoc and Pervasive Communications. American Scientific Publishers (2006)
22. Song, H.; Kim, D.; Lee, K.; Sung, J.: Upnp-based Sensor Network Management Architecture. In: Proceedings of the Second International Conference on Mobile Computing and Ubiquitous Networking (ICMU'05), Osaka, Japan (2005)
23. Ruiz, L. B.; Nogueira, J. M.; Loureiro, A. A. F.: MANNA: A Management Architecture for Wireless Sensor Networks. In: IEEE Communications Magazine, vol. 41, no. 2, pp. 116-125 (2003)
24. Meyer, K.; Erlinger, M.; Betser, J.; Sunshine, C.: Decentralizing Control and Intelligence in Network Management. In: Proceedings of the 4th International Symposium on Integrated Network Management (IM'95), pp. 4-16, Santa Barbara, CA (1995).
25. Greenwood, D.; Ghanbari, M.; O'Mahony, M.: A Hybrid Centralised – Distributed Network Management Architecture. In: Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99), pp. 434-441, Sharm El Sheikh, Egypt (1999).
26. Pagurek, B.; Wang, Y.; White, T.: Integration of Mobile Agents with SNMP: Why and how. In. Proceedings of the Network Operations and Management Symposium (NOMS'00), pp. 609-622, Honolulu, Hawaii, USA (2000)
27. Pras, A.; van Beijnum, B.-J.; Sprenkels, R.: Introduction to TMN. CTIT Technical Report 99-09, University of Twente, Enschede, The Netherlands (1999)
28. Du, T. C.; Li, B. Y.; Chang, A.-P.: Mobile Agents in Distributed Network Management. In: Communications of the ACM, vol. 46, no. 7, pp. 127-132 (2003)
29. Tsekouras, G. E.; Anagnostopoulos, C.: A Mobile Agent Platform for Distributed Network and Systems Management. In: Journal of Systems and Software, vol. 82, no. 2, pp. 355-371. Elsevier (2009)
30. Pfaff, B.; Pettit, J.; Koponen, T.; Amidon, K.; Casado, M.; Shenker, S.: Extending Networking into the Virtualization Layer. 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York City, NY (2009)
31. Lockwood, J. W.; McKeown, N.; Watson, G.; Gibb, G.; Hartke, P.; Naous, J.; Raghuraman, R.; Luo, J.: NetFPGA – An Open Platform for Gigabit-rate Network Switching and Routing. In: Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE'07), San Diego, California, USA (2007)

32. Maccabe, A. B.; Zhu, W.; Otto, J.; Riesen, R.: Experience in Offloading Protocol Processing to a Programmable NIC. In: Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'02)
33. Luo, Y.; Cascon, P.; Murray, E.; Ortega, J.: Accelerating OpenFlow Switching with Network Processors. In: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'09), Princeton, NJ, USA (2009)
34. Naous, J.; Erickson, D.; Covington, A.; Appenzeller, G.: Implementing an OpenFlow Switch on the NetFPGA platform. In: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'08), San Jose, CA, USA (2008)
35. Dudkowski, D.: Co-Design Patterns for Embedded Network Management. In: Proceedings of the 2009 Workshop on Re-Architecting the Internet (ReArch'09), pp. 61-66, Rome, Italy, December 2009. ACM, New York, NY, USA (2009)

Chapter 5
Wire-Speed Hardware-Assisted Traffic Filtering with Mainstream Network Adapters

Luca Deri [1], Joseph Gasparakis [2], Peter Waskiewicz Jr [3], Francesco Fusco [4] [5]

[1] ntop, Pisa, Italy
[2] Intel Corporation, Embedded and Communications Group, Shannon, Ireland
[3] Intel Corporation, LAN Access Division, Hillsboro, OR, USA
[4] IBM Research - Zurich, Rüschlikon, Switzerland
[5] ETH Zurich, Switzerland

deri@ntop.org, joseph.gasparakis@intel.com,
peter.p.waskiewicz.jr@intel.com, ffu@zurich.ibm.com

Abstract. Modern computer architectures are founded on multi-core processors. In order to efficiently process network traffic, it is necessary to dynamically split high-speed packet streams across cores based on the monitoring goal. Most network adapters are multi-core aware but offer limited facilities for assigning packets to processor cores.
In this paper we introduce a hybrid traffic analysis framework that leverages flexible packet balancing mechanisms available on recent 10 Gbit commodity network adapters not yet exploited by operating systems. The main contribution of this paper is an open source hardware-assisted software layer for dynamically configuring packet balancing policies in order to fully exploit multi-core systems and enable 10 Gbit wire-speed network traffic analysis.

Keywords: High-speed network traffic monitoring, hardware-assisted dynamic packet filtering, commodity hardware, operating system design.

1 Introduction

The complexity and heterogeneity of monitoring tasks, such as anomaly and intrusion detection, traffic classification and application level analysis [1], gradually caused a shift from dedicated network devices toward hybrid software and hardware architectures which are more flexible and easier to maintain than dedicated monitoring devices [2]. Along with hardware-based

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_5, © Springer Science+Business Media, LLC 2011

solutions [20], researchers have demonstrated that the performance of traffic analysis applications running on commodity hardware can be substantially improved by properly accelerating selected operating system tasks [19, 21, 22]. However, the performance gap between pure software solutions and hardware assisted ones has been significant. Recent advances in off-the-shelf server technologies suggest that the gap can be substantially reduced. In fact, modern servers are based on advanced multi-core processors featuring integrated memory controllers and high-speed and low latency interconnections. In addition, off-the-shelf network interface cards (NICs) are supporting new advanced features such as message signaled interrupts (MSI-X), multi-queue capabilities and virtualization support, which have been designed to boost the network performance in specific scenarios. The trend is to introduce into NICs the logic for offloading workstations from computationally intensive network operations. With the advent of multi-core processors, balancing the networking workload among cores is necessary in order to increase the networking performance of network services. Therefore, modern interface cards provide multiple independent reception (RX) and transmission (TX) queues and hardware traffic splitting techniques to distribute the traffic among cores.

Unfortunately, traffic monitoring software did not fully benefit from these new breakthrough technologies. The reason is that software layers on top of which network monitoring applications are implemented, such as network device drivers and operating systems, are not designed for exploiting these features for network monitoring purposes.

In this work we present a flexible and extensible framework that simplifies the development of complex and yet efficient traffic analysis applications running on commodity hardware. The main contribution of this work is a novel traffic balancing and filtering networking layer optimized for traffic analysis purposes that fully exploit advanced features implemented by modern off-the-shelf NICs. The framework is characterized by the following properties:

- It provides an API for hardware-assisted traffic filtering and balancing across cores.
- It can be deployed on sub-1000$/port commodity network adapters which are more than an order of magnitude cheaper than dedicated traffic monitoring devices.
- The filtering mechanisms are flexible and able to address common problems monitoring scenarios such as adaptively balancing the incoming traffic among cores or dynamically filtering incoming traffic.
- It can be used as a building block for designing complex yet efficient monitoring applications.
- It is publicly available at no cost under the GNU GPL license.

The rest of the paper is structured as follows. In section 2 we describe how the software framework we designed few years ago could benefit from modern NICs in particular for supporting in hardware those features we previously implemented in software. In section 3 we position the work described in this paper against similar efforts. Section 4 describes the design and implementation of a new software layer that allowed us to offload traffic filtering to modern NICs. Finally section 5 describes some common use cases we used to evaluate the developed solution hence to demonstrate that this work is a major step ahead with respect to existing software-only solutions.

2 Motivation and Scope of Work

The intrinsic dynamism of Internet protocols has increased the demand for flexible monitoring frameworks designed to speed up the development of efficient and cost effective applications capable to analyze modern network protocols. Nowadays, most network monitoring infrastructures are built around hybrid frameworks combining the flexibility of software and the performance of hardware accelerators designed to offload network probes from selected computationally expensive tasks. The design of hybrid frameworks requires expertise in software, firmware and hardware development, as well substantial investments that have a negative impact on end-user prices. In fact, since the target of these devices is a niche market, their price is in order of magnitudes higher than commodity off-the-shelf network interfaces.

Packet capture accelerators are the most cost effective solution for improving software based traffic monitoring applications. As packet capture is the cornerstone of many passive monitoring application, capture accelerators have been able to provide substantial speedups to traffic monitoring applications by allowing incoming traffic to be copied directly into the address space of the analysis process without any CPU assistance.

In our past research, we focused on pure-software traffic analysis frameworks. In particular, we proposed filtering solutions that are capable to overcome the limitations of the popular Berkley Packet Filter (BPF) [8], a rule-based traffic filtering mechanisms provided by the majority of the operating systems. In [9] we describe a traffic filtering mechanism that, contrary to BPF, can be reconfigured in real-time and scale in terms of number of traffic filtering rules. In [10] we present a traffic filtering and analysis framework named RTC-Mon that substantially simplifies the development of modular and efficient traffic monitoring applications. The core of the framework is a rule-based infrastructure that allows traffic analysis components to be enabled over the traffic matching rules. By introducing services for IP de-fragmentation, packet parsing and maintenance of flow state statistics, the development

efforts for implementing monitoring applications are substantially reduced. The framework is useful for implementing traffic analysis applications, such as VoIP and IPTV monitoring software, where traffic filters must be added/removed in real-time.

In our previous works, we decided not to leverage any specific monitoring device in order to reduce costs and simplify the deployment. In this work instead, we evaluate the opportunity of accelerating our framework by exploiting mainstream NICs. Unlike special purpose monitoring hardware, off-the-shelf network interfaces target the mainstream market and therefore come at low end-customer prices. Even if these NICs are not designed for accelerating monitoring software but rather tasks as virtualization, some of their features can be successfully exploited for increasing the performance of traffic analysis applications.

Modern off-the-shelf adapters provide several independent RX/TX queues and hardware-based mechanisms such as Receive-Side-Scaling (RSS) that balance network flows among RX queues mapped on processor cores. By splitting the traffic among queues, the workload, both in terms of packet processing and interrupts can be balanced across cores for better exploiting the intrinsic parallelism of modern computing architectures. As of today, the majority of server class adapters in the market are multi-queue enabled and support RSS for splitting the traffic across queues. The main limitation of RSS is that the balancing policy is static hence it cannot be adapted to changing traffic conditions. This represents a serious limitation as workload unbalances correspond to scalability penalties. Even if it is possible to augment RSS with software based traffic balancing policies, this approach is, in practice, unfeasible for high-speed networks as the performance penalty is severe. Therefore, NIC manufacturers are introducing the second generation traffic balancing hardware mechanisms that are dynamically configurable in order to adapt traffic balancing policies to every traffic condition. Although these mechanisms have been introduced for enhancing general purpose networking, we believe that packet filtering will also benefit from these breakthrough balancing technologies, and therefore, the performance gap between special purpose monitoring devices and off-the-shelf network adapters would be reduced.

In this work we present an advanced and yet easy to use open source software framework that leverages the customizable hardware assisted traffic balancing and filtering features introduced in modern NICs. As these filtering features will likely be available in future NIC cards manufactured by various vendors just as happened with RSS, we believe that this work is not limited only to the specific NIC we considered in this paper, but it paves the way to supporting a new family of cheap 10 Gbit (and 40 Gbit in the future) network adapters.

3 Related Work

The industry followed three paths for accelerating software applications by means of specialized hardware while preserving the software flexibility:

- Accelerate the capture process via packet capture accelerators [3, 4] that allow incoming packets to be copied directly to the address space of monitoring applications without any CPU intervention.
- Split the monitoring workload among different network probes using smart traffic balancers [5] so that each probe receives and analyzes a portion of the traffic.
- Run traffic analysis software on programmable network cards based on network processors [6] or massive parallel architectures [7]. Programmable network cards are massive parallel architectures on a NIC. Monitoring applications are implemented in C and executed on these device [7] that run a modified version of Linux which simplifies the porting of existing applications on top of this special purpose architecture. However, even if they have been able to substantially simplify the development compared to network processors based cards, the porting is still not trivial.

Most general purpose operating systems support rule based filtering mechanisms such as the BPF where filtering expressions are compiled into an intermediate language and interpreted by a virtual machine running at the kernel layer. PF_RING [11] is an advanced network monitoring framework enhancing Linux with more flexible filtering mechanisms implemented in software by means of kernel modules. NetVM [15, 16] is a virtual machine designed to simplify the development and maintenance of complex and yet efficient packet processing applications running on top of heterogeneous network devices. FFPF [17] is an extensible and high-performance packet capture and filtering architecture based on Linux. Contrary to our work, FFPF does not leverage modern multi-core or multi-queue interfaces. The SCAMPI project [18] provides a feature rich monitoring API but it has been designed to run on top of specialized monitoring devices and therefore it yields poor performance when running over commodity hardware. [22] describes a framework for high-speed networks monitoring that provides features such as IP defragmentation and flow reassembly, that relies on a pure-software implementation of a packet scheduling algorithm proposed in [23]. Our work instead, exposes to the software layers an API to design hardware assisted packet schedulers.

Capture accelerators based on FPGA, implement filtering mechanisms at the network layer by means of rule sets (usually limited to 32 or 64) similar to BPF. Filtering runs at wire-speed. As the rule set is not meant to be changed at runtime, its scope of application is drastically limited. Often traffic filtering is used to mark packets and balance them across DMA engines. Traffic balancing policies are similar to RSS and are usually implemented at the

FPGA layer and allow the traffic to be split among cores within a multi-core processor. As for traffic filtering, dynamically updating the traffic balancing policies at run-time is in practice unfeasible as a card reconfiguration may require seconds if not minutes.

4 Framework Design

In our past research, we developed an extensible traffic analysis framework implemented under the Linux Kernel called PF_RING [11] which accelerates packet capture and implements packet parsing and filtering by means of dynamically loadable kernel plugins. A user space library called libpfring provides an easy to use API that allows user space applications to interact with the framework.

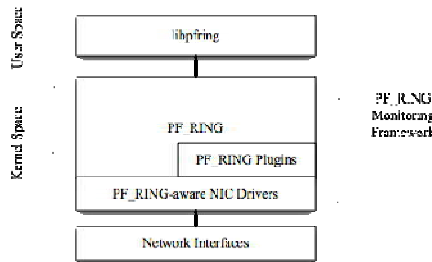


Fig. 1. PF_RING Monitoring Framework.

PF_RING runs on top of commodity network interface cards and can use both standard NIC drivers or PF_RING optimized drivers. These drivers, available for popular 1 and 10 Gbit adapters produced by vendors such as Intel and Broadcom, push incoming packets directly to PF_RING without passing through the standard kernel mechanisms hence accelerating capture speed.
PF_RING provides a flexible rule-based mechanism that allows users to assign packets to kernel plugins which are then responsible to dissect, order in flows, and compute flow metrics (e.g. voice quality) directly at the kernel layer without copying packets to user space. For example, it is possible to configure PF_RING to dispatch TCP packets on port 80 to the HTTP plugin, and UDP packets on port 5060 to the SIP plugin. The same rule-based mechanism can be used for filtering out from PF_RING analysis unwanted packets (e.g. discard packets coming from a specific host or port) similar to what the firewalling layer does at an operating system level.

With the advent of multi-core systems and multi-queue adapters, PF_RING has been extended with support of virtual RX queues [12], that enable specific plugins/user space applications to receive traffic from specific RX queues. The PF_RING kernel infrastructure is responsible to exploit facilities such as RSS for balancing and assigning packets to cores while queue information is preserved in received packets.

In summary, PF_RING has become an advanced framework that thanks to its rule-based mechanism, has been capable to simplify the engineering of modular applications and not just accelerate packet capture. However, the rule-based mechanism has been completely implemented in software, and therefore, it is inefficient at very high speed such as 10 Gbit.

Last year Intel introduced X520, a 10 Gbit card based on the new 82599 ethernet controller [13]. What makes this adapter interesting for PF_RING, is the ability to support in hardware dynamically configurable flow affinity filters for classifying, load balancing and dispatching traffic flows to processor cores. The filtering mechanisms introduced by 82599 can be seen as a fine-grained RSS that allows selected flows to be classified and dispatched towards specific cores based on configurable packet filters and not on RSS hashing.

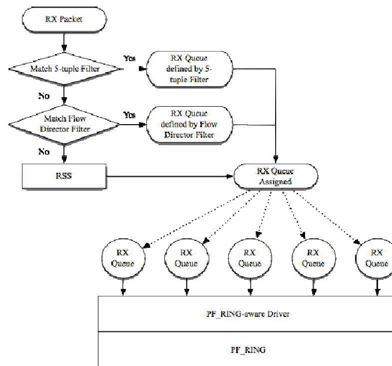


Fig. 2. Integrating 82599 with PF_RING.

The availability of this affinity facility in commodity adapters has been the natural solution to address performance issues of PF_RING at 10 Gbit. Exploiting the flow affinity filters is indeed attractive for:

- leveraging hardware facilities for dispatching packets across PF_RING plugins enabled on selected RX queues;
- dropping unwanted packets in hardware inside the NIC before they hit the driver.

In a nutshell, flow affinity filters introduce new opportunities, not yet exploited by operating systems and monitoring applications, for the implementation of hardware assisted packet schedulers capable to accelerate traffic analysis applications by fully exploiting the parallelism offered by multi-core architectures.
As we believe that 82599 is just the "first of a kind" and similar flow affinity filters mechanisms will soon be introduced by other vendors, PF_RING has been extended not only to exploit these features as implemented by 82599 but also to support future NICs providing similar capabilities. For this reason we introduced a new hardware-neutral software layer that is responsible for setting up specific flow affinity filtering rules in hardware. This layer has not been designed for natively supporting the 82599 controller in PF_RING, but rather as a foundation layer for offloading selected filtering tasks to those NICs that feature flow affinity filters. This means that:

- not all facilities offered by 82599 have been supported yet (e.g. IEEE 1588 time synchronization), but only those (i.e. flow affinity filters) that can be currently exploited by PF_RING for accelerating its operations (i.e. we have not added support of 82599 in PF_RING, but rather exploited those 82599 features that can accelerate PF_RING);
- adding support in PF_RING for flow affinity filters-like features in future NICs, will not require PF_RING redesign but it will just require the implementation of new extensions into PF_RING-enabled NIC drivers;
- existing applications such as RTC-Mon will not need to be recoded (but just slightly modified) in order to exploit flow affinity filters, as PF_RING transparently sets in hardware the appropriate flow affinity filters.

PF_RING supports two families of filters: precise filters where the whole `<vlan, protocol, ip/port src, ip/port dst>` tuple needs to be specified, and wild card filters where some filter parameters can be unspecified (e.g. `tcp and port 80`). When a packet is received, PF_RING uses the "best match first" policy, so it will first try to match the packet against configured precise filters, and in case of no match against wild card filters. Packets matching a filter will be passed to the specified plugin or action, if configured. Hardware flow affinity filters support has been added into PF_RING as follows:

- PF_RING-aware drivers notify (when the driver is loaded inside the kernel) the PF_RING engine whenever a given NIC supports flow affinity filters.
- PF_RING has been extended with a new function named `handle_hw_filtering_rule()` that allow precise and wild carded filters to be added/removed inside NICs.
- For each NIC supporting flow affinity filters, PF_RING adds a virtual file

whose path is `/proc/net/pf_ring/ethX/rules` that network administrators, and not just monitoring applications, can use for adding/removing filters by means of a simple echo of a string on it. For instance echo “`+(1,-1,tcp,192.168.0.10,25,0.0.0.0,0)” > /proc/net/pf_ring/eth3/rules`, instructs PF_RING to add in the eth3 device a new filtering affinity rule with id 1 and that sends all TCP packets from 192.168.0.10:25 to the core id -1. Since the identifier -1 does not correspond to a physical processor core, this rule allows packets matching the filter to be dropped at the NIC layer. Using another existing queue id would simply advise the filtering mechanism to direct the packets to the appropriate queue and hence through the SMP affinity mechanism in the Linux kernel into the desired core.

In order not to modify the existing driver structure by introducing new hooks for adding and removing filters, we decided to jeopardize some existing driver hooks. The advantage is that all current drivers do not need to be changed, and this gives us a way to migrate towards packet filtering integration when supported in Linux[1]. The data structure used to pass filter specifications to drivers is generic and does not rely on 82599 specific data types. In this way, the efforts for supporting future network adapters providing similar features will be substantially reduced. 82599 provides several types of filters including layer 2 and FCoE (Fibre Channel over Ethernet), but as PF_RING supports only precise and wild card filters, we focus only on 5-tuple and flow director filters that are very close to PF_RING filters:

- 5-tuple filters (up to 128 filters can be defined in 82599) allow packets belonging to flows identified by the 5-tuple <protocol, ip source, port source, ip destination, port destination> to be forwarded to a specific RX queue. 5-tuple filters are defined as `<id, protocol, ip/port src, ip/port dst, target RX queue id>`. Some of the fields specified in a 5-tuple filter can be “masked” (i.e. wild carded) in order to avoid comparing them against incoming packets.
- Flow Director (FD) filters can be specified as precise (i.e. the filter members are matched precisely against incoming packets) or hash (i.e. the packet hash is compared against the filter hash, conceptually similar to bloom filters [14]) filters. 82599 supports up to 32k precise filters. The number of distinct hash filters is not limited by design. However, the adoption of excessive hash filtering rules may lead to false positives. FD filters are expressed as `<slot id, VLAN, protocol, ip netmask/port src, ip netmask/port dst, target RX`

[1] In kernel 2.6.34 the ethtool, not the kernel itself, introduced limited support for EFD thanks to patches we submitted to Linux kernel maintainers.

`queue id>`. Currently all configured filters must have the same mask defined in 82599.

The 82599 adapter is quite different from many FPGA-based NICs as it does not use a TCAM (Ternary Content Addressable Memory) for handling filters. This means that a filter is configured by setting up specific NIC registers and, therefore, that the last configured filter overwrites the previous register value. For this reason, it is not possible to read from the NIC all configured filters, and therefore the driver has to maintain the list of configured filters. The advantage of this approach is that, contrary to many FPGA-based NICs where setting a filter requires card reconfiguration, in 82599 setting a filter is extremely fast and from the application point of view it takes as long as the `setsockopt()` system call necessary to pass the filter specification to the kernel, making this NIC usable in environments where filter configuration has to be dynamically changed.

5 Use Cases and Validation

Validation has been performed using an IXIA XM12 10 Gbit traffic generator and a NUMA computer using a single 6-core Xeon® X5650 (Westmere) CPU at 2.67GHz. In all tests we have injected IPv4 UDP traffic with random payload at wire speed, and compared the number of packets sent by the traffic generator with those reported by *pfcount*, a simple packet-counting application running on top of PF_RING. pfcount spawns and binds a thread per core (i.e. thread X is bound to core X). The injected traffic contained 6 flows, each balanced to an individual core using hardware filtering rules. Packets have been captured using the standard NAPI-based 82599 driver enhanced with PF_RING and hardware filtering support.

Table 1. Hardware vs. Software Filtering Comparison

Frame Size (Bytes)	Test 1		Test 2	
	Software Filter (Capture Rate)	Hardware Filter (Capture Rate)	Software Filter (CPU Load)	Hardware Filter (CPU Load)
64	5.7%	6.3%	95.6%	None
128	10.0%	11.6%	95.4%	None
256	19.5%	23.2%	98.7%	None
512	37.4%	42.3%	3.5%	None
1024	99.8%	100%	3.3%	None
1518	99.6%	100%	< 0.1%	None

In the first test we compared hardware (i.e. 82599) vs. software (i.e. PF_RING) packet filtering using a single filtering rule that match for every incoming packet (i.e. the entire traffic is forwarded to the user space). In the second test we have injected traffic that does not match any configured filter, and verified that there is no load on the CPU whenever hardware filters are used. On the contrary, what we observed with software filters, is that for packets up to 256 bytes the CPU utilization was around 95%, and about 3% for larger packets. This leads us to the conclusion that in the hybrid model of software and hardware filtering we propose, it is recommended to use software filters only for medium to large packets.
In order to further improve packet capture, the authors have developed TNAPI [25], a multithreaded RX queue polling mechanism that significantly improves packet capture performance with respect to the standard Linux NAPI.

5.1 Realtime Multimedia Traffic Monitoring

As described earlier in this paper, RTC-Mon has been designed to efficiently handle VoIP calls and video-on-demand traffic analysis at 1 Gbit. In order to scale the solution to 10 Gbit, we have slightly modified the original RTC-Mon code as follows:

- A few 5-tuple filters have been configured:
  - All the SIP signaling packets go to core 0.
  - Non UDP (i.e. ICMP/TCP) packets are dropped.
  - UDP traffic on popular ports (e.g. port 53 used by DNS) is also dropped.
- Whenever a new VoIP call has been setup, such call is tracked by adding two FD filters (one per call direction) that send the voice traffic for the tracked call (i.e. RTP traffic) to the same RX queue where the RTP plugin is active. In order to evenly balance the traffic across queues, the queue ids used for voice traffic are selected in round robin so that all queues have almost the same amount of traffic.

This setup has allowed RTC-Mon to operate efficiently in 10 Gbit links where VoIP is only a portion of the overall traffic, thanks to 82599 filters used to discard packets not belonging to calls being tracked. Unfortunately, not all unwanted packets have been discarded and a small portion of them is still received by PF_RING. This is because 5-tuple filters are evaluated before FD filters, hence it is not possible to set 5-tuple rule that discards all the remaining traffic because this would also discard traffic that matched by FD filters. It is worth noting that the ability to setup thousands of flow affinity

filters with almost no latency is a key factor for using effectively 82599 in cases where filter setup latency is crucial as with RTC-Mon.

5.2 Network Troubleshooting

Troubleshooting a heavily loaded 10 Gbit link using popular tools such as tcpdump and wireshark [24] is almost impossible due to severe packet capture loss. Furthermore, most commercial tools are not distributed with source code, hence it is not possible to recompile them in order to take advantage of PF_RING flow affinity filters. In this case, we used PF_RING's `/proc` interface for setting a few traffic filtering rules that discard in hardware unwanted traffic, hence pass to the Linux kernel only those packets that must reach network monitoring applications. This solution has the advantage that existing applications do not need to be modified, and PF_RING is used just for allowing the network administrator to easily configure (e.g. using a shell script) flow affinity filters without having to code a C/C++ application sitting on top of libpfring.

5.3 Traffic Classification and Balancing

In case monitoring applications do not run on the same box where an 82599 based NIC is installed (e.g. because they run on a non-Linux OS such as Windows), it is possible to create a traffic filtering box using the *pfreflect* application part of PF_RING, that filters incoming packets and copies them onto one or more NICs based on the PF_RING filters configuration. As PF_RING filters (hence flow affinity filters) are evaluated before reflection (i.e. packet bridging in PF_RING parlance), this application can be used for creating an inexpensive traffic filtering box that can be used for reducing the amount of traffic to analyze. If the filtered traffic is less than one Gbit it can be forwarded onto a 1 Gbit card so that legacy measurements box do not need to be updated to 10 Gbit. Furthermore as PF_RING supports traffic balancing, it is possible to forward filtered traffic onto several output interfaces by balancing each RX queue of 82599 onto a different output interface. This solution allows high-speed links to be monitored and troubleshooted without having to purchase costly 10 Gbit measurement boxes.

5.4 Lawful Interception of Internet Traffic

Since the approval of the wiretapping in the US in 1984, lawful interception (LI) has become very popular. In LI a lawful authority requires to intercept and store specific traffic for the purpose of analysis or evidence. In IP

networks, this means that traffic originated/directed to specific IPs or flowing on specific ports need to analyzed. Doing this on a 10 Gbit link using software-based traffic filters can be inefficient as packet loss might prevent captured traffic from being analyzed properly. In order to implement a simple packet capture system driven by signaling protocols such as Radius or DHCP, it is possible to setup (e.g. via the PF_RING /proc filesystem interface) a few filtering rules that discard all traffic except signaling (similar to the setup used in 5.2) and traffic belonging to target IPs that need to be intercepted.

5.5 Firewalling at 10 Gbit

The Linux netfilter/iptables firewall is quite efficient but it cannot operate with no loss on heavily loaded 10 Gbit links. The use of 5-tuple filters can definitively help dropping unwanted traffic or tracking NAT sessions using FD filters. Unfortunately the Linux firewall is more flexible than 5-tuple filters, hence it is not possible to do a one-to-one mapping between iptables rules and 5-tuple filters. This means that 82599 can be used to discard a large portion of incoming traffic but not all, leaving to netfilter the duty of completing packet filtering. Nevertheless this hybrid, hardware plus software, filtering architecture allows to significantly boost the firewall performance in most situations. Currently we are add filters using the PF_RING /proc filesystem interface as we have not yet added native 82599 support into netfilter.

6 Open Issues and Future Work

The main limitation of the current implementation is the lack of a compiler that transparently compiles BPF filters into PF_RING (hence flow affinity) filters. Due to this limitation, users must configure both BPF filters (e.g. on the command line while starting the monitoring tool) and flow affinity filters (e.g. using the PF_RING `/proc` filesystem). In future code releases we plan to implement such feature so that BPF-aware applications (e.g. Wireshark) can still use BPF for setting filters while the underlying kernel layers add automatically flow affinity filters in order to reduce the amount of packets that will hit the BPF filtering engine. In addition to 5-tuple and FD filters, 82599 also supports SYN filter that diverts to a specific core all incoming TCP packets with the SYN flag set. While its support would be trivial from the 82599 point of view, the PF_RING engine instead needs some extensions in order to add filters that can select packets based on TCP flags.
Finally we would like to use 82599 in the context of OpenFlow switching, for implementing efficient in-kernel switching across network applications

without requiring external switching equipment. From the hardware point of view, we envisage that future NICs will further enhance flow affinity filters number and expressiveness (e.g. adding the ability to filter tunneled traffic), add per-filter statistics (e.g. number of packets and bytes that matched each filter) so that developers could implement efficient NetFlow caches in hardware.

7 Conclusions

Monitoring the Internet is challenging as high-speed networks are becoming popular and traffic patterns more complex. In order to satisfy the increasing performance requirements and reduce deployment costs, modern network monitoring frameworks should leverage those features offered by mainstream NICs that are introduced for general-purpose networking and not fully exploited in the context of network monitoring. This paper has presented an evolution of PF_RING, a monitoring framework originally designed for accelerating packet capture, that exploits hardware-based filtering mechanisms offered by the Intel 82599 based NICs and likely future NICs. Thanks to flow affinity filters PF_RING can now fine-grain flow balance packets across cores, classify traffic and discard unwanted communication patterns directly into the NIC before packets hit the driver. The validation process has demonstrated that many network applications can benefit from this work, making it very general and usable also outside of the network monitoring domain. Not to mention that it is finally possible to combine the speed of hardware with the flexibility of software for effectively monitoring 10 Gbit networks using commodity network adapters.

Availability. This work is distributed under the GNU GPL license and is available at no cost form the PF_RING home page (http://www.ntop.org/PF_RING.html).

Acknowledgments. The authors would like to thank Intel and in particular Edward Clinton and Richard P. Kelly for their support during this research work.

References

1. W. John and others, Passive internet measurement: Overview and guidelines based on experiences, Computer Communications, vol. 33, issue 5 (2010).

2. G. Memik and W.H. Mangione-Smith, Specialized Hardware for Deep Network Packet Filtering, Proc. of FPL 2002, Montpellier, France, (2002).
3. S. Donnelly, DAG Packet Capture Performance, White Paper, (2006).
4. Napatech Inc, The Napatech Protocol and Traffic Analysis Network Adapter, White Paper, (2006).
5. cPacket Networks, cVu 320G: Aggregation, Complete Packet Inspection Filtering, Automatic Flow Balancing, (2010).
6. P. Crowley and others, Characterizing processor architectures for programmable network interfaces, Proc. of the 14th international conference on Supercomputing, Santa Fe, New Mexico, (2000).
7. A. Agarwal, The Tile processor: A 64-core multi-core for embedded processing, Proc. of HPEC Workshop, (2007)
8. S. McCanne and V. Jacobson, The BSD packet filter: A new architecture for user-level packet capture, Proc. of Winter '93 USENIX Conference, (1993).
9. L. Deri, High-Speed Dynamic Packet Filtering, Journal of Network and System Management, (2007).
10. F. Fusco and others, Enabling High-Speed and Extensible Real-Time Communications Monitoring, In Proc of IM 2009, (2009).
11. L. Deri, Improving Passive Packet Capture: Beyond Device Polling, Proc. of SANE 2004, (2004).
12. L. Deri, Towards 10 Gbit NetFlow Monitoring Using Commodity Hardware, Proc. Joint EMANICS/IRTF-NMRG Workshop, Munich, (2008).
13. Intel Corporation, 82599 10 GbE Controller Datasheet, Rev. 2.3, (2010).
14. B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, July 1970.
15. F. Risso and others, Extending the NetPDL language to support traffic classification, Proc. of IEEE Globecom, (2007).
16. L. Degioanni and others, Network virtual machine (NetVM): a new architecture for efficient and portable packet processing applications, Proc. of 8th International Conference on Telecommunications, (2005).
17. H. Bos and others, FFPF: Fairly fast packet filters, Proc. of OSDI ’04, (2004).
18. J. Coppens and others, SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks, Proc. of E2EMON Workshop, (2003).
19. L. Deri, nCap: Wire-speed Packet Capture and Transmission, Proc. of E2EMON, (2005).

20. L. Degioanni and G. Varenni, Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware, Proceedings of IMC '04, (2004).
21. M. Smith and others, Enabling High-Performance Internet-Wide Measurements on Windows, Proc. of PAM 2010, pp. 121-130, Zurich, Switzerland, (2010).
22. M. Dashtbozorgi and others, A scalable multi-core aware software architecture for high-performance network monitoring, Proc. of the 2nd international conference on Security of information and networks, pp. 117-122, Famagusta, Cyprus, (2009).
23. M. Dashtbozorgi and others, A high-performance software solution for packet capture and transmission, Proc. of 2nd IEEE International Conference on Computer Science and Information Technology, pp. 407-411, Beijing, China, (2009).
24. F. Fuentes and D. C. Kar, Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose, Journal of Computing Sciences in Colleges, Vol. 20, Issue 4, pp. 169 - 176 , (2005).
25. L. Deri and F. Fusco, Exploiting Commodity Multi-core Systems for Network Traffic Analysis, Technical Report, http://luca.ntop.org/MulticorePacketCapture.pdf, (2009).

Chapter 6
Embedded Rule-based Management for Content-based DTNs

Jorge Visca, Guillermo Apollonia, Matias Richart,
Javier Baliosian, and Eduardo Grampín

School of Engineering, University of the Republic, Uruguay.
{jvisca, gapollo, mrichart, javierba, grampin}@fing.edu.uy

Abstract. Several countries such as Uruguay and Brazil are implementing the well-known One Laptop Per Child Program (OLPC) by which every child that attend to primary school obtains in property a laptop with wireless capabilities. They carry their laptops from home to school and back every day and, as we observed in our research, they also carry their laptops to parks, community centers etc. That provides a wide platform for opportunistic, delay tolerant, networking applications. This paper presents a low-cost, delay-tolerant, network of sensors implemented embedding high-level decision-making capabilities inside consumer-grade wireless routers working together with the OLPC laptops. The sensors are deployed at the living premises of children in environmentally vulnerable neighborhoods as well as at their schools, parks, etc. The environmental data collected by the sensors is carried to the school by the laptops and from the school to monitoring stations over the Internet. In this system, all the entities in the network are publishers and subscribers of configuration commands, policy-rules and environmental data, building a flexible, self-management solution.

1 Introduction

Several countries such as Uruguay and Brazil are implementing the well-known One Laptop Per Child Program (Plan Ceibal in Uruguay [4] and UCA in Brazil [6]) by which every child that assists to the primary school obtains in property a laptop with wireless capabilities. Besides carrying the laptops to school everyday, children frequently keep the laptops with them when going to parks, community centers, etc. This provides a wide platform for opportunistic networking applications. The basic concept behind opportunistic networking is that, in the absence of a fixed connectivity infrastructure, some data of interest (in our case, domestic environmental data) is transferred between mobile devices using the "connection opportunity" that arise whenever mobile devices happens to come into the range of other devices because of the mobility of the devices' users.

The ongoing DEMOS project (Domestic Environment Monitoring with Opportunistic Sensor networks) is developing a low-cost platform for environmental sensors, such as air-quality sensors, at the living premises of children in environmentally

A. Clemm and R. Wolter (eds.), *Advances in Network-Embedded Management and Applications*,
DOI 10.1007/978-1-4419-7753-3_6, © Springer Science+Business Media, LLC 2011

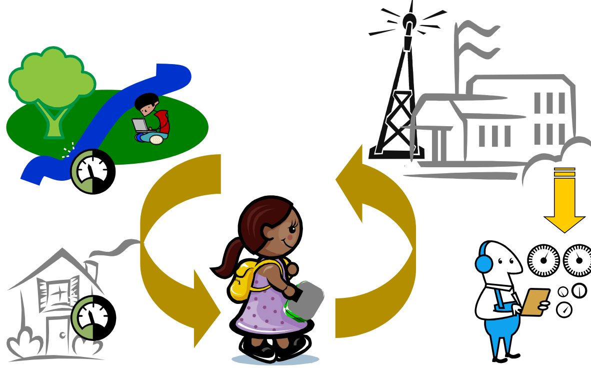


Fig. 1. The DEMOS Project

vulnerable neighborhoods as well as at their schools, parks, etc. The environmental data collected by those sensors are transmitted, using opportunistic networking techniques, to the children's laptops as they pass by during their daily life. Later, at school, using the same techniques, the data will be transmitted into the local school server and from there to an environment monitoring station using the Internet. This monitoring station may be operated by governmental or non-governmental organizations, including the same community that is being object of the monitoring.

Each time data moves from sensors to laptops, school servers and monitoring stations, the collected information is aggregated and summarized to guarantee the scalability of the solution. Additionally, since the information may include data about the position and environment of peoples' houses, the confidentially of that information has to be protected by encryption or anonymization techniques.

Opportunistic networks (ON), sometimes called Delay Tolerant Networks (DTN), as opposed to infrastructure networks, are built on-the-fly by mobile, intermittently connected ad-hoc nodes, allowing to run delay tolerant applications. Many ON routing algorithms have been proposed, see [8] for a comprehensive review, but the common idea is always to *store* the message, to *carry* it for some time, and to *forward* it when a suitable mobile node happens to be in range with the hope that, after some of these *store-and-forward* steps, the message will eventually arrive to its destination.

The optimum values for DTN algorithm's parameters are dependent on the network characteristics (number of nodes, movement patterns) and data flow characteristics (load, data patterns). Usually, those attributes are highly dynamic and hard to predict. Some analysis can be done through extensive simulation, but fact remains that to optimize the network performance, configuration parameters must be adjusted to the environment, at runtime.

We propose a mechanism that allows us to take configuration decisions in a distributed fashion, responding to high level rules.

The structure of the paper is as follows. The overall description of our ON protocol is presented in Section 2. Later, in Section 3, we shortly motivate the need of including self-management capabilities to ON nodes. The overall design of the system is presented in Section 4.1. The characteristics of our prototype and the main hardware characteristics and constraints that drive the implementation is presented in Section 5. Finally, in Section 6, the paper presents an evaluation of the *DEMOS system* that shows how it is possible to perform self-optimization decisions with a very reasonable overhead in terms of CPU and memory.

2 An Opportunistic Content-based Routing Protocol

The DEMOS architecture implies the existence of an opportunistic network between the sensor devices and the data collection points. In an opportunistic network, the existence of a connectivity path between any pair of nodes in a given moment is not guaranteed. For a message to reach its target, it may be necessary that some node or nodes keep the message in their own memory until they can deliver the message.

There are several methods and algorithms for Opportunistic Routing, but to support DEMOS operations we created RON, a new content-based opportunistic protocol. The reasons for creating a new protocol were:

– Usually, opportunistic network algorithms are intended for destination-based routing. On the other hand, Content-based routing provides us of several advantages: as specifying a data flow does not rely on having inventory on the network devices, the deployment is simpler; multicast and broadcast messaging is more naturally represented; the messages can be dynamically prioritized or routed based on message properties such as the type of issuing sensor, danger level, or geographical location. All this rich behavior can be changed at runtime without changing the configuration of any node of the network, providing great flexibility.
– Many gossip-based algorithms operate on a Peer-to-peer basis, where each link is considered isolated from the rest. On the other hand, our target are wireless networks, where when one node issues a message, all the nodes in its range receive the message independently of their address in the network layer (the “broadcast advantage”). Our routing algorithm uses this property to reduce the network traffic.

RON is a publish-subscribe protocol in which a node that is interested in some type of messages, issues a subscription that specifies that interest using a logic filter. This subscription is flooded through the network passing from one node to another each time those nodes are within range of each other. When a node wants to publish a notification, it broadcasts a message containing the notification to all the nodes in the neighborhood. Each of these nodes matches the received message against the subscriptions it is carrying, and decide whether to carry or ignore it according to routing rules. Eventually, the message will reach all the subscriber nodes. The algorithm is described in more detail below.

```
NOTIFICATION
notification_id=notif123
source=sensor_node1
message_type=trap
watcher_id=watch_temp
mib=temperature
value=36.5
END
```

(a) A Notification.

```
SUBSCRIBE
subscription_id=sid123
subscriptor_id=collector1
FILTER
mib=temperature
value > 35
END
```

(b) A Subscription.

Fig. 2. The Messages of the Notification Bus

2.1 RON Protocol

In our network, there are two entities: subscriptions and notifications. The associated messages are plain text multi-line strings.

Notification messages are the mean of distributing information in the network. A sample notification is shown in Figure 2(a). As seen, it is a simple list of key/value pairs. The only mandatory attribute is notification_id, a unique identifier. The user of the bus must define additional fields and their semantics. In the example we can see a source field (the identifier of the sensor node), and the remaining fields describe the sensor-reading being carried (36.5 from a temperature sensor).

A Subscription message signals the interest of its creating client on receiving certain notifications. A typical Subscription is shown in Figure 2(b). The subscription is composed of two parts, the header and the filter. The header contains general information, like a unique Subscription identifier and the identifier of the node subscribing. The filter specifies a set of conditions, which must be met by a notification for it to be delivered to the client. Each condition is of the form attribute-operator-value, where attribute is a Notification field. If a Notification does not contain that field, the given expression is considered satisfied. In other words, a Notification fails a filter only if it contains a field which fails some condition. In the example, the subscription will match any notification with a payload of a temperature over 35 (whatever sensor node originates it).

To move those entities trough the network, each node maintains S_j, a table of accessible destination (Subscriptions in our case) with an associated quality for each. The quality of a destination represents how good is the node to reach that destination. Nodes periodically exchange their destination lists with their associated qualities, and update their own qualities in the process. Also, the nodes keep a set of messages N_j. The amount of messages that can be carried is limited, so a decision must be made on what message to carry. The destination quality information is used to select which messages are best carried by the node, which are the messages targeted at destination with higher qualities.

As usual for a gossiping algorithm, RON can be split in two threads: an active emitting main loop (the Control Cycle), and a passive event handler (the Message Handler) that receives, processes and issue messages.

The Control Cycle periodically triggers two actions. The first is the broadcasting of a *Views message*. This message contains a list of all subscriptions in S_j and their associated quality. The receivers of this node will use that data to maintain their own subscription qualities and trigger *Notification* broadcasts. The second Control Cycle's action is to periodically decrease the quality for all subscriptions (the aging process).

The Message Handler listens the network and can receive two types of messages:

– *Views message*. These messages come from other node's Control Cycle. Upon reception, for each subscription in S_j that also appears in in the *View message*, the algorithm will increase its quality. Furthermore, it will iterate trough all notifications in N_j, and for each that matches a recevied subscription will broadcast a *Notification message*.
– *Notification message*. Notifications proceed from the Message Handler of other nodes. While N_j is not full, all notifications are stored. When N_j becomes full, the algorithm will look for a Notification with a smaller accumulated quality Q to replace. The accumulated quality Q for a notification is implemented as the sum of all subscription qualities that the given notification matches.

To keep the broadcasts reduced to a minimum, special care is taken. In first place, when *Views message* are broadcast, only the subscription's identifiers and associated qualities are transmitted. If a node sees the identifier of a subscription it does not have, it requests it in a separated message. Second, nodes timestamp entries in N_j and S_j as they listen the medium, and will refrain from transmitting data already seen inside a (configurable) time frame. Thus, when a *Views* broadcast triggers a *Notification message* broadcast from one node, this message will reach other nodes in the network and will inhibit them from repeating it. To this purpose and to avoid synchronization problems every sending is delayed by a small random time.

When broadcasting a *Views message*, there is a chance other nodes will answer with notifications already carried. To reduce this chance, a bloom filter of carried notifications could be included in said *Views message*. This way, receiving nodes could easily check if a notification is a repeat and skip it. For an in-depth description of the RON algorithm, see [10].

3 Managing a Delay Tolerant Protocol

There are several parameters that have a significant influence on protocol's performance. The optimum values for these parameters depend on network characteristics, which can be stable through the network's lifetime, or change over time.

The main configuration parameters of the protocol are:

– *Buffer size*. This parameter controls the amount of carried notifications. A too low value can cause message loss, when a message is replaced on all nodes' buffers before reaching destination. Too high a value has an impact on performance and resource consumption, as messages take space to be carried, computing power

to be checked for transmission, and air time when transmitted. Optimum buffer size depends on such parameters as network latency (e.g., there is no benefit in keeping messages after a copy reaches destination) and network load (the rate of arrival of new messages).
- *Subscription quality management parameters.* These parameters control the reinforcement and aging behavior of the quality of subscriptions, and should be tuned to the real probabilities of encounter in the network, which in turn depend on the network density and movement patterns.
- *Views broadcast period.* This parameter controls the rate of gossiping messages, and must also match the network density and movement patterns. Too low a rate will miss encounter opportunities, and too high a rate imposes an unnecessary load on the network.

There are several other parameters that could be added to the basic protocol, like remaining battery or available bandwidth. An example of such extension applied to PRoPHET can be seen in [11]. Our management infrastructure uses all configuration parameters in a consistent way, and thus provides a method to easily integrate and take advantage of new variables.

The optimum values for those parameters can be hard to estimate in advance. For example, the movement patterns and number of nodes change: there is a week/weekend cycle, and an yearly cycle of vacations. Network patterns also change: a single new subscription or sensor installed can define a new flow of data that changes the load imposed on the network. At the same time, the sensor nodes emit data depending on local readings, and thus are hard to predict.

This leads to the need of a mechanism that would manipulate the configuration parameters autonomously, adjusting them to optimize the algorithm's performance. We implement this mechanism through the use of PDP (Policy Decision Point), a general purpose policy engine. The idea is that a state machine is used to recognize patterns that occur in a flow of events as described in [9]. A success in recognizing a pattern triggers a corresponding action or set of actions. Those actions can be configuration change commands, or abstract notifications that express the occurrence of a situation. The later can be consumed by other recognizers in a hierarchical way.

4 System Overview

The DEMOS system (see Figure 1) consists of a set of services deployed on the nodes, according to their function in the DEMOS network. The nodes fall under some of the following categories:

- *Sensor Nodes.* These are the nodes that collect environmental data. Those are usually fixed, and have attached sensing hardware. Different nodes can have different sensing hardware attached.
- *Collectors.* These nodes are usually placed in schools, and are the recipients of data generated by the Sensor Nodes. Data is collected and relayed to a central Management Station through the Internet.

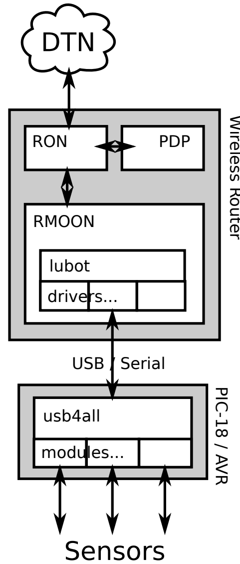


Fig. 3. Architecture of a DEMOS sensor node

- *Carriers.* These are the mobile nodes that relay information from the Sensor Nodes to Collector in an opportunistic fashion. Usually these are children's XO laptops.
- *Management Station.* All data is summarized and prepared for analysis here. Also, rules are prepared and deployed from here.

Sensor Nodes, Carriers, and Collectors participate in an Opportunistic Network, and thus run the supporting service, RON. Additionally, Sensor Nodes run a monitoring service attached to the sensing hardware.

4.1 Common Node Architecture

On Figure 3 the general architecture of DEMOS sensor node is shown. RON and associated PDP modules are responsible for the opportunistic routing of messages. They must be available on all edge and mobile nodes participating in the DTN. RON is the routing service, and it relies on PDP, a general purpose Policy Decision Point, to take decisions that affect the routing. The PDP service will be described with more detail in the Section 5.2.

Rmoon is the monitoring service, responsible for collecting and sending sensor data through the network. The internal architecture of the Rmoon module and associated sensor device will be described in more detail in Section 5.3.

Mobile nodes for DEMOS only have the RON and associated PDP modules installed. PDP module is available for other uses besides opportunistic routing.

It can be used, for example, to allow sensors deciding intelligently on what and when sensor data must be collected and sent, autonomously trigger actions on some conditions, etc.

During the development in place of real environmental sensing hardware a general purpose microprocessor board was used, the PICDEM FS USB[3] demoboard. This is a board for prototyping systems based on a Microchip PIC-18 microprocessor, and has a temperature sensor and a variable potentiometer on board that were used as stand-ins for real environmental sensors. Other sensors can be attached to the board through Digital Lines, I^2C bus, A/D converters, etc. This platform was selected on the grounds of low cost and availability on the local market. Nevertheless, our software has been also ported to the AVR family of microprocessors.

5 Implementation

A requirement for our solution was the need to be deployable on a wide spectrum of hardware and software platforms. It ranged from Desktop PCs on central data collecting nodes, to embedded platforms such as Linksys WRT54GL wireless routers. The intermediate nodes included such platforms as OLPC XO laptops and Intel's Classmate netbooks, running several flavors of Linux and Windows Operating Systems. At the same time, sensor nodes should be able to interface with external sensor hardware. Another point of interest was the ability to be run in a simulation platform, for better analysis and tuning of the Opportunistic Routing component.

5.1 Language selection

It was decided to develop in Lua[1]. Lua is a compact virtual-machine based dynamic language, with great emphasis on extensibility. It is weakly typed and has a garbage collector. It is written completely in ANSI-standard C99 and thus the core has minimal dependencies, and can be run in an extremely wide spectrum of platforms down to embedded microcontrollers. At the same time, it offers powerful facilities to programmers, like regular expression matching, hash-based tables, functions as first class members, upvalues, and lexical scoping.

Our implementation runs on the core Lua with the only dependency of LuaSocket[12] library. The only platform dependent code needed was for interfacing the sensor hardware.

The very small number of external dependencies of Lua virtual machine allowed us to run it inside an experimental ns3 [2] branch intended for running native code. This branch is at early stages of development, and the high level of isolation of Lua code allowed us to run production code inside it with only moderate effort. Being able to run the same code in physical platforms and in the simulator greatly simplifies the development and tuning of networking protocols. At the same time, the small size per network node of the runtime allows us to simulate reasonable large networks. Additionally, ns3 allows to connect a real device to a simulated network. Thus it is possible to deploy a testbed split between a simulation and

physical devices, allowing to simulate a large network while monitoring the resource consumption on a real device.

For the embedded platforms in our tests (consumer grade wireless routers such as Linksys WRT54GL and Asus 520gu), we have used OpenWRT [5], a distribution of Linux for embedded devices. This setup provides a profusion of standard tools for managing and configuring the behavior of the node. At the same time, in recent versions of OpenWRT Lua is used as a platform for the administration web-page, and thus most of the runtime needed for our programs is already in place by default.

As a result, we could develop a networking platform, a general purpose decision engine, and a monitoring software in a high level scripting language, which can run unmodified on a PC, a wireless router, or a network simulator.

5.2 PDP

PDP's task is to take autonomous decisions, based on the stream of Notifications it receives through the Notification Bus. To this purpose, a special policy file is pushed from a central server. This policy file is generated in the central server from the rules specified by an administrator, and delivered to the PDP agent in a command notification through the network. The policy file is a Lua program that implements a state machine, accessible through function calls. When a policy file is received, the PDP executes it in a special sandboxed environment, and starts triggering it with notifications. The state is kept internally in the policy script, and the PDP must not know how it is implemented, only that call hooks are respected.

The PDP maintains a list of recent notifications, which is shared into the scripts environment. This list contains a sliding window of the last arrived notifications. To simplify the state machine, the notifications are ordered in the window by a fixed ordering rule. In this way, the state machine does not have to handle every possible permutation of the notifications of interest.

The size of the sliding windows can be set at configuration time. For this purpose there are two parameters: a length of time in seconds, and a maximum number of notifications in the window. Also, there is a special category of notifications: “happening notifications”. Those are notifications that signal permanent change in some attribute, and thus must be kept in the window for it to be correctly processed. For example, a decision could depend on whether a given service is active or not, thus the hypothetical “going_up” and “going_down” notifications from that service could be marked as happening to keep them in the window. This way, the state machine could count on the presence of said notification no mater how old they are. Marking and unmarking notifications as “happening” is policy's responsibility, and the PDP will comply while maintaining the sliding window.

Beside accessing the window, the policy script must provide the following calls:

- *initialize ()* This is called once after the script is loaded.
- *process_window_add ()* This function is called whenever a new notification arrives. It gives the opportunity to advance the state machine.
- *process_window_move ()* This function is called whenever the sliding window moves, thus a notification is removed from the leading edge. This implies that

the state machine must be reset, and run from the beginning of the window again.

All three calls can return a list of notifications to be emitted to perform whatever action the state machine decides to be made. So, *initialize*() would return commands to set-up the notification sources needed, and whenever a pattern is recognized during either *process_window_add* or *process_window_move*, the corresponding actions will be returned.

By expressing the state machine as Lua code, there is no intermediate representation for the state machine and thus no special parsing. All the parsing work is done in the central server, and the PDP just executes it.

5.3 Rmoon

Rmoon is a general purpose monitoring service. It is controlled through the RAN bus: it receives commands to setup and remove "watchers", and emits notifications when the watchers get triggered.

Each watcher has a unique identifier used to tag all its notifications. A watcher observes a variable (called a mib), and specifies a condition the variable must met to trigger a notification. The condition is specified as a comparison operator ($>$, $<$ or $=$) and a reference value. When the reference value is numeric, an additional hysteresis value can be provided. The comparison can be specified to be made against the instantaneous value of the variable, the difference with the last reading, or the difference with the last value that triggered a notification. In other words, the conditions can be set on the value of the variable, its rate of change, or the deviation from last reported value. Additionally, a timeout can be specified, which triggers a notifications emission if a given time has expired without the notification being triggered otherwise.

When used to monitor the node's state, Rmoon uses shell scripts and it is easily extensible. For the DEMOS project we extended Rmoon with support for external sensors. For this purpose we interfaced with a microprocessor software framework developed in our research group, called USB4ALL[7]. USB4ALL is a modular firmware that provides a high level communication mechanism. It allows the controller to discover installed modules, load and unload them at runtime, and query them through an RPC-like mechanism. Originally developed for the Microchip PIC-18 series of microprocessors, it has been also ported to AVR based Arduino platform. To interface with USB4ALL, a library called Lubot (also written in Lua) has been developed. The only native code needed on the Rmoon side is a Lua - libusb binding or a small message oriented serial library (depending on how the microprocessor board is attached, through USB or serial link).

5.4 RON

As mentioned earlier, RON tries to take the "broadcast advantage". While in traditional gossiping algorithms is the emitter responsibility to select the recipients for his messages, in our implementation the only decision the emitter does is whether to

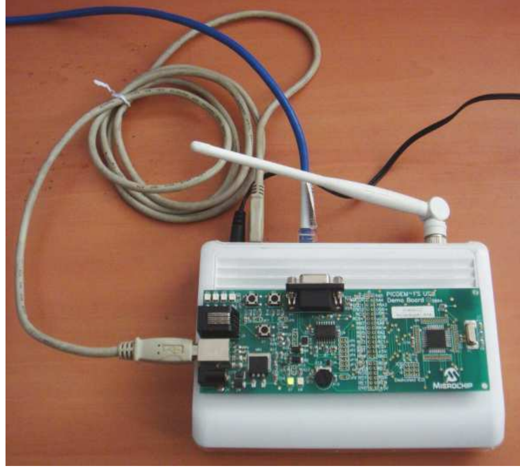


Fig. 4. The Prototype Sensor

emit. The listener responsibility is to decide whether they accept the messages, and thus become the receiver. To this effect, and to keep the traffic at the IP layer for flexibility, all the traffic for the opportunistic support is encapsulated in broadcast UDP packets (by default over port 8181).

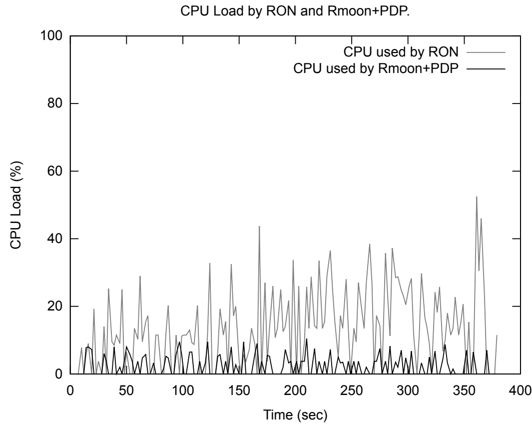
Another interesting problem is time-stamping of messages: as sensor and mobile nodes do not have reliable clocks, and the transit time through the network is highly variable, it is difficult to know when exactly the notification was generated. Our solution is to record the in-transit time of each message in a special field. Each time a node forwards a message, the time that the message spent stored in the node is added to the in-transit field. In this manner, the issuing time of a message is computed by the receiving node in reference to its local time. For the purposes of DEMOS project, the precision of this method is satisfactory.

6 Management Footprint

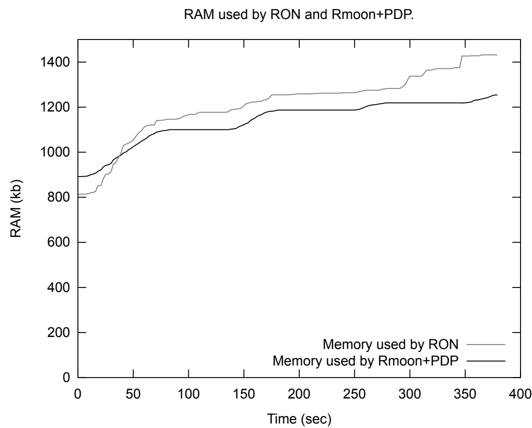
The focus of this paper is on the feasibility of embedding a high-level self-management system into the constrained devices of a network of sensors. With this goal we have set up a scenario with one sensor measuring the temperature of the air and publishing it, and a client producing an increasing number of different subscriptions.

6.1 Experiment

The objective of this experiment is to show that the footprint of the system presented above in Section 4.1 is small enough for the targeted constrained devices.



(a) CPU load at the Sensor



(b) RAM used at the Sensor

Fig. 5. System Footprint Study.

The setup of this experiment follows the scenario described above. The prototype sensor is an ASUS 520gu wireless router (as depicted in Figure 4) with 32MB of RAM and OpenWRT Kamikaze operating system, connected to a general purpose microprocessor board PICDEM FS USB. This device is running the notifications bus RON with the buffer size set to 40 messages (about 12KB), the monitoring service Rmoon and the Policy Decision Point described in Section 5.2. The measurements depicted bellow are made on this device. The client is a Core 2 Duo PC running RON.

The experiment runs during 380s. During the first 10s ten watchers are sent from the client arriving to the sensor during the first 50s. Each of those watchers

are configured to emit a notification every 5s. The increment on the memory load produced by the activation of the watchers can be seen at the beginning of the graphs in Figure 5(b). After the watchers have being submitted, the client start producing a different subscription every 10s. Every subscription matches with all the notifications emitted by the sensor. In order to simulate the load of the PDP reconfiguring the routing protocol, every time a subscription is received at the sensor, the PDP reconfigures RON's buffer-size to the same value of 40 messages.

Is worth noticing the evolution of the percentage of CPU consumed by the system in Figure 5(a) and its the memory load in Figure 5(b). The CPU consumed by Rmoon and the PDP combined is never beyond 10%. The CPU consumed by the notifications bus is quite spiky but with maximums of around 50%. This leaves plenty of CPU resources for other tasks. Regarding memory load, it grows during the arriving of the watchers up to about 1.2MB and then grows moderately as more subscriptions arrive. Considering that more than 30 different subscriptions matching the data of ten different watchers every 5s is an extremely high load for a environmental sensor, we conclude that the tested hardware can cope comfortably with the expected loads.

7 Conclusions and Future Work

In this paper we have presented the self-management part of an opportunistic network of in-house environmental sensors and a myriad of mobile devices that act as the carriers of the collected data. Our work shows that a content-based, opportunistic routing protocol benefits from some self-configuration features, such as the dynamic adaptation of the buffer-size of a node to the density of the network. We conclude that it is feasible to deploy rule-based self-management capabilities inside very-constrained devices such as the prototype sensor presented in Section 5. The footprint of the management components is not negligible but, from the initial experiments, we observe that it is small enough to keep us working on this path.

The system presented in this paper is in its developing stage, therefore much work remains to be made. The routing protocol presented has many aspects to be improved, but regarding the self-management functionality of the system, besides its feasibility, its correctness must be tested extensively. A whole set of management rules and scenarios are being experimented on the ns3 simulator.

Finally, despite the fact that DEMOS project takes advantage of the Uruguayan and Brazilian particularities, and proposes to use the platform of low-cost laptops of the OLPC program, the project idea is easily translatable to other communication platforms that are becoming ubiquitous in developing countries such as cellular phones with Bluetooth or other wireless capabilities.

Acknowledgments

This work was partially funded by the Research Funding Initiatives of the Latin American and Caribbean Collaborative ICT Research (LACCIR) virtual institute.

References

1. Lua, the programming language, http://www.lua.org
2. ns3 network simulator, http://www.nsnam.org/
3. PICDEM Full Speed USB, http://www.microchip.com/
4. Plan Ceibal, http://www.ceibal.edu.uy/
5. The OpenWrt Project, http://openwrt.org/
6. Um Computador por Aluno - UCA, http://wiki.laptop.org/go/OLPC_Brasil/Porto_Alegre
7. usb4all, http://www.fing.edu.uy/inco/grupos/mina/pGrado/pgusb/material.html
8. Balasubramanian, A., Levine, B., Venkataramani, A.: Dtn routing as a resource allocation problem. vol. 37, pp. 373–384. ACM, New York, NY, USA (2007), http://dx.doi.org/10.1145/1282427.1282422
9. Baliosian, J., Visca, J., Grampin, E., Vidal, L., Giachino, M.: A rule-based distributed system for self-optimization of constrained devices. In: Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on. pp. 41 –48 (1-5 2009)
10. Baliosian, J., Visca, J., Richart, M., Apollonia, G., Vidal, L., Giachino, M., Grampín, E.: Self-managed content-based routing for delay tolerant networks. Tech. rep., UDELAR (2010), http://www.fing.edu.uy/inco/proyectos/wan/documentos/ronreport.pdf
11. Huang, T.K., Lee, C.K., Chen, L.J.: Prophet+: An adaptive prophet-based routing protocol for opportunistic network. In: Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. pp. 112 –119 (20-23 2010)
12. Nehab, D.: Network support for the lua language. Web, http://www.tecgraf.puc-rio.br/˜diego/professional/luasocket/