Aspects for untangling cross-layer design and policy support

Djamel Sadok1,*,†, Luciana P. Oliveira1, Glauco Gonçalves2, Judith Kelner1, Nelson Rosa1 and Eduardo Souto3

1 Center de Informatica, Federal University of Pernambuco, Recife, Pernambuco, Brazil
2 UFPE, Recife, PE, Brazil
3 Departamento de Informatica, Universidade Federal de Amazonas, Manaus, Amazonas, Brazil

SUMMARY

Programming language concepts have inspired some networking design decisions. For example, concepts such as object encapsulation and interface invocation have been borrowed, at the time of their adoption, from an already well established object oriented programming paradigm. The authors suggest in this paper that it may be time again to revisit emerging software engineering programming paradigms to learn from them. More specifically, this paper discusses the practical tangling problem, embedded in conventional layer-coupling (linking) network software design and highlighted by recent research proposals for cross layer design. The adopted solution is based on the aspect-oriented programming paradigm. We show its programming efficiency, limitations and role in the seamless enforcement of multiple policy scenarios while emphasizing little design changes. Copyright © 2012 John Wiley & Sons, Ltd.

Received 10 March 2011; Revised 3 October 2011; Accepted 1 December 2011

KEY WORDS: network design; cross-layer design; aspect-oriented paradigm; policy-based management

1. INTRODUCTION

Existing layered communication stacks have been successful for many decades. They are modeled using clear design principles and constraints. As such, they establish that communications are only allowed among adjacent layers through well defined interfaces and service primitives. Though early discussions in the 1970s and 1980s led to suggestions of protocol stacks with varying numbers of layers and functionalities, the layering concept was never questioned. Recent initiatives such as the research into the next generation of the Internet (NGI) and the emergence of wireless networks led many to reconsider the wisdom of this particular principle. A myriad of new and perhaps ‘unconventional’ architectures have been suggested in the context of what the next generation of networks (NGNs) should look like. Some of these, such as the four years long Hagle architectural project [1], have entirely abandoned the traditional layered model present above the data-link layer. A radical departure from the existing transmission control protocol/Internet protocol (TCP/IP) suite was then taken by allowing mobile application level opportunistic message forwarding instead of using a network layer. More detrimental to the existing communication stack is the extensive wireless research suggesting the use of a loosely coupled layering model to obtain performance gains [2]. This model requires a minimal interaction between the layers. Therefore, any imperfect decisions at one layer should not be susceptible of affecting dramatically the entire architecture. Nonetheless, this model stops short of claiming that such negative effects can be entirely mitigated.

*Correspondence to: Djamel Sadok, Informatics Center, Federal University of Pernambuco, Pernambuco, Brazil.
†E-mail: jamel@gprt.ufpe.br

Copyright © 2012 John Wiley & Sons, Ltd.
1.1. The role of wireless networks in emerging cross layer paradigm

Without a shadow of doubt, it is mainly research in wireless networks that first led the way towards new stack programming models. With the popularity of wireless networks, devices with multiple interfaces, sensors and ad hoc networks and other highly context sensitive and dynamic networking technologies, there was an overwhelming need for new and often more complex network software design, one that takes into consideration new concerns such as energy consumption, failure and disconnection handling, capacity maximization, link state and quality of service at many levels of the stack project. The overall goal is a justified need to squeeze every bit out of the available radio spectrum while working under different and sometimes harsh radio signal constraints. Hence, many works have identified the urge to somehow venture away from the accepted stringent structured layered communication model by allowing some layers to sniff around others, sharing information and coordinating or synchronizing some of their actions. These schemes have been dubbed cross-layer (CL) and appear in many forms and shapes although essentially offering limited scope well targeted information sharing for coordination among layers. This layer infringement is of course overlooked, all in the name of doing a better job and optimizing many of the common concerns such as those earlier mentioned. The next section gives examples of the context where CL design has been applied.

2. CROSS LAYER MOTIVATION

Many standardization bodies such as those responsible for the development of successful telecommunications standards including UMTS (Universal Mobile Telecommunication System), Bluetooth, and B-ISDN (Broadband Integrated Services Digital Network) may sometimes give the impression that they might have adopted too much layering, driven by the presence of convergence and technology-dependent sublayers and the use of encapsulation. Although ‘divide and conquer’ is known for its manageability and problem structuring benefits, an increase in the number of layers and sublayers may sometimes become harmful and confusing from the software designer’s practical perspective. As an example, let us consider a network with performance and security related policies or rules. A layer designer may find that their enforcement may require replicating related information to these over several layers. In other words, the effect these policies have on the networking software can hardly be contained within a single layer. It becomes consequently cumbersome to add, remove or update such policies without introducing changes to all concerned layers. This obviously involves a lot of undesirable software design changes to the actual implementation at different places and the replication overhead. Instead of replicating information, a more efficient optimization approach would require per-context, for example, intelligent information sharing among such layers. This sharing is by no means a new practice and has been with us in fact since the last decade, for example, within solutions proposed for IP mobility where snooping on the network layer by the transport was suggested [3].

Research in CL design has been very actively pursued in recent years while leading to important gains in a number of application domains, especially in wireless networks as mentioned earlier. Nonetheless, when inadequately designed, a CL solution may result in unpredicted behavior in addition to the loss of architectural design control or neatness, because of the newly created links (couplings) among layers, as feared by many researchers.

In a wider sense, CL can be seen as a project design where there are important shared information and functionality flowing among the layers for use by algorithms, protocols, decision, and adaptation processes as they see fit [4]. From a software engineering point of view, this would be similar to allowing access to the inner variables or partial code of a software object as opposed to having to go through its well-defined interface in an object-oriented programming (OOP) context for example. This practice in OOP is understandably prohibited. Such sniffing among layers may render the design more tangled, complex to control, and even unstable when not properly and carefully handled. It is this risk that needs mitigation in this paper.

According to Ref. [5], there are three main reasons singled out as the driving factors towards the violation by CL of the existing orderly and simpler design. These are:
a) the performance problems specific to wireless communications: research has confirmed that important performance gains may be obtained by sharing information across the physical, data-link, routing, transport, and application services. This is somehow intuitive as the more information the different algorithms and mechanisms present at a given layer have, the better are their decisions and the more coordinated their actions become. For example, layer three routing benefits the network and its nodes when taking into consideration the physical layer information such as signal strength to increase its throughput and node energy levels to extend the overall network’s lifetime.

b) The emergence of opportunistic communications where wireless physical presence and proximity may dictate the upper layers’ behavior and decisions. As an example, one may consider emerging delay tolerant networks (DTN) that operate in an opportunistic store and forward mode. Here all the layers must coordinate their actions to detect and take advantage of nearby nodes through a new interworking layer known as the bundle layer. The DTN bundle protocol works on the basis of ‘bundle’ consignation, that is, by passing responsibility for the bundles being transported between nodes. The bundle layer is for DTN networks what IP is for the Internet, a technology independent glue. Without the presence of cross layer information coming from the underlying physical and transport layers, the bundle layer may not take full advantage of the presence of a nearby node, which can be harmful for the success of DTNs.

c) The emergence of new communication models such as cognitive networks [6] where software radios are used to search for and use opportunistically ‘white spaces’ (unused spectrum) to increase spectrum efficiency and mitigate its shortage. This model relies on the use of: (i) highly sensitive and agile transceivers capable of quickly detecting and returning the spectrum to its primary users, that is, the rightful main owners. This is known as the primary users protection requirement; (ii) advanced medium access control (MAC) and coordination schemes to access and share the unused spectrum among secondary opportunistic users; and (iii) the adaptation to the application load and traffic requirements.

A further reason for justifying CL software layer design results from the need to coordinate layers developed independently usually by different standards institutions. It is often the case that different technologies implement similar mechanisms to achieve the same goal. Nonetheless, when these are put together, some unforeseen side effects or conflicts may result in disastrous results because of incompatibility between the two layers. A classical example is the use of TCP in combination with ATM’s available bit rate class congestion control mechanisms. Here, we have two transport solutions that were developed, more than a decade ago, by two separate standardization entities, the IETF (Internet Engineering Task Force) and the ATM Forum, to deal with the same issue, that of network congestion. Because none of the layers coordinates its work with the other one and use different granularity levels, they end up reacting to each other’s cures and leading to a suboptimum solution. A CL coordination among these two layers would have avoided this problem.

2.1. Cross layer applications

A number of solutions have adopted CL design principles for network optimization [7, 8]. In wireless network design, many proposals made MAC performance design tightly dependent on physical layer conditions such as signal-to-noise ratio and power control, among others [9]. The achieved performance increase comes as no surprise. The authors in [8] investigated the effects of the BER and power control over the packet size and showed that a coordinated approach between power control algorithms and the MAC layer may reduce energy consumption. Song and Li examined the use of CL design in the context of OFDM networks [10]. OFDM’s use is very popular today in most emerging wireless technologies such as the WiMAX (Worldwide Interoperability for Microwave Access) (802.16), Wi-Fi (802.11), UMTS, and LTE (Long Term Evolution) 4G cellular systems, Wireless Regional Networks (802.22) because it mitigates both intersymbol interference (ISI) and the effect of delay spread in multipath wireless environments. The main idea behind OFDM is a simple one, very narrowband symbols are transmitted long enough to mitigate their possible interference with other symbols (which is known as ISI) because of the different paths these signals may take with varying delays. In their research, the authors developed a utility function that bridges the
physical and MAC layers while being responsible for balancing network resource allocation efficiency and offering fairness. The results, as expected, also showed a performance increase allowing the support of more additional nodes into the examined network.

2.2. Examples of cross layer goals

Next, we give some other simple examples to show network design issues that cannot be framed within a specific layer. Some of these have been presented in the section above when discussing current work on CL. Let us take the following design policies or goals:

1) Maximize wireless network lifetime: this is obviously a goal that no single layer software can claim to satisfy it on its own. Next, we show how some cross layer considerations or subgoals may contribute to reaching this overall goal.

(a) At the physical layer, parameters such as the battery lifetime, transmission power, type of modulation, and so on, impact the lifetime of a wireless node and the whole network. For example, when there is no activity at the physical layer, the whole software stack may enter a standby energy conserving mode.

(b) At the link layer, nodes with better radio signal conditions and low noise levels may be given more access to the radio channel because they stand a better chance to deliver the information and avoid retransmissions that waste more energy.

(c) At the network layer, routing should know about nodes with low energy levels and avoid these as much as possible. Special aggregation roles may be given to more stable nodes. Furthermore, the physical layer should trigger new route calculations as soon it detects that it has lost a neighbor for example, as this has moved or simply was switched off.

2) Maximize packet delivery rate: this goal seeks to achieve a maximum level of packet delivery, that is, until reaching the 100% threshold. This goal can be fine-grained into a number of subgoals at the physical, link, routing and transport layers.

(a) At the physical layer, the signal strength is important to avoid radio links subjected to noise and interference. Power control may then be applied to avoid interference between the concurrent transmission paths.

(b) At the link level, more specifically the MAC sublayer, nodes need to calculate the residual available bandwidth to avoid network congestion and the load at a node given in terms of its data size in buffers. Furthermore, error recovery may be in the form of costly frame retransmissions or based on the use of forward error correction schemes. The availability of radio information from the physical level would help establish a balance between costly retransmission and effective use of forward correction. When subject to large interference and noise, the use of forward correction of errors is not suitable and should give way to less aggressive modulation techniques and frame retransmission, for example, using the automatic repeat request (ARQ) scheme. Furthermore, the MAC protocol may be power controlled, where a receiver announces its interference tolerance. The transmitter then determines its power bound according to the receiver’s interference tolerance along a path.

(c) At the routing layer: the network layer may opt to use a source routing protocol to better balance the load distribution across the nodes. The routing metrics could include the signal strength and interference information collected at the physical layer to increase throughput. Alternatively, a cross-layer routing protocol that coordinates transmission power control and intraflow and interflow interference as routing metrics may be used. Here, each radio interface calculates the potential tolerable–additional transmission interference at the physical layer. When doing route discovery, the appropriate power level to evaluate each interface quality along paths is considered.

Table I summarizes the CL layer goals for each parent goal and their CL attributes. One may imagine a scenario whereby the goals are changed and the inner working decisions of the network are consequently modified to attend the new targets. In this work, we show how such changes are
Table I. CL goals, subgoals and attributes.

<table>
<thead>
<tr>
<th>Subgoals</th>
<th>Goals</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximize wireless network lifetime</td>
<td>Maximize packet delivery rate</td>
</tr>
<tr>
<td>Physical Layer</td>
<td>Minimize battery Consumption; Enter standby mode when possible; Monitor signal quality.</td>
<td>Monitor signal quality; Apply power control to reduce interference; Lower modulation rate in the presence of noise.</td>
</tr>
<tr>
<td>Link Layer</td>
<td>Give radios that consume less energy higher access priority; Transmit to closer nodes.</td>
<td>Avoid nodes with low signal quality; Monitor link buffer size; Use FEC when moderate errors; Retransmit when operating with high levels of errors.</td>
</tr>
<tr>
<td>Network Layer</td>
<td>Route avoiding nodes with less battery power; Give preference to routes with minimum required transmit power; Reconstruct routes on signal loss;</td>
<td>Reconstruct routes on signal loss events; Give preference to routes over nodes with higher residual available bandwidth to avoid congestion and packet loss; Avoid congested nodes where congestion at a node given in terms of its data size in buffers.</td>
</tr>
</tbody>
</table>

managed using the aspect-oriented paradigm in such a way that new goals do not radically affect an existing design.

2.3. Cross layer versus clean design

A CL solution is said to be conservative, in this work, if it maintains the layering division and merely introduces one or more new components responsible for coordinating their interactions. Examples of such architectures include the CrossTalk [4] and MobileMan Project [11]. On the other hand, nonconservative cross layer approaches are those that radically depart from the layering structure. A case in point is the role-based architecture [12], which represents a nonlayered architecture to the design of network protocols and organizes communication in functional units referred to as ‘roles’. A role is seen as an abstract entity with a functional definition of an element that is important for communication and performs a specific forwarding and/or processing service. For instance, a processing element may be a security or a media compression role where packets that need such service should be forwarded to.

although there are many new architectures that departed from the layered model, we will only discuss the conservative ones. Many works continue emphasizing the importance of a well-structured architecture to ensure that the obtained product remains as much modular as possible, easy to understand, develop, and maintain for an expected long life cycle. Despite this conservatism, the existing layered model has been violated in a number of ways as listed next [7]:

a) Vertical parameter calibration: this is seemingly the most commonly used approach. Here, parameters from different layers are jointly processed to help decision making improve the performance of the overall stack. Note that the optimization is not necessarily that of an application as one may be optimizing one or more specific stack functions such as routing, radio resource allocation, or transmission power.

b) The creation of new interfaces: alternatively, new interfaces between nonadjacent layers may be defined to show cooperation between these layers in the upward, downward, or even in the two directions simultaneously. This may be seen as extending the existing stack model. Nonetheless, creating new cross layer interfaces may increase stack design complexity and lead to a loss of modularity.
c) The creation of a new coupling layer: as another form of extension, this layer would keep in mind the functionality and working details of one or more other layers. In this case, there is no need to establish new interfaces as in the previous approach. The creation of such layer has the advantage of making the network designer concentrate on the performance and other issues at such layer instead of having these spread all over the other layers.

d) Layer merging: it is perhaps a natural tendency to merge layers that exhibit strong interlacing. This way, the existing model is maintained whereas the number of layers decreases. This has been observed, for example, in the evolution of the traditional seven-layer reference model in recent years. Session and presentation functionalities have been merged with those from the transport and application layers, resulting in what some refer to as the reduced reference model. As a result of layer merging, new optimizations may be made without breaking the structure of the new software stack. Obviously, this can only be done a limited number of times because it may lead to ‘fat’ layers with overloaded responsibilities.

2.4. Discussion of some cross-layer issues

Any software project requires a prior cost-benefit analysis. Is it ultimately worth losing the convenience of the layered protocol and service design to often seek limited gains? What is the cost of maintaining such new complex software? Such questions are yet to be formally taken in the context of CL design. If we adopt CL design into future next generation network architectures, we need to understand and measure the impact that such design may have on disturbing other elements of the network layers even for those that apparently seem not to be concerned. Furthermore, the optimization of some objective may interfere negatively on the performance of others. Work in [13] pointed to this problem and even created a name for it as the law of unintended consequences. This kind of unpredicted behavior contributes to hesitation of people in taking up this paradigm beyond research. With current initiatives for a new Internet architecture, it is perhaps the right time to include CL design as part of this agenda. Similarly, it may also be the time to revisit emerging software engineering programming paradigms. A special attention is given next to the emerging aspect oriented programming (AOP) paradigm [14].

3. NEW CHALLENGES

The designer of CL software may have to walk along a thin line. Existing approaches actually show a tradeoff between the amount of performance gain achieved and the level of committed architectural violations. Studies such as the one from [13] pointed out to potentially long term undesirable and unintended consequences as opposed to short term performance benefits that unbridled CL design may engender. Because current CL implementations of cooperation include the use of both inter-layer and intralayer entities to achieve centralized and distributed coordination, they not only violate the current layered model, but more preoccupying is their lack of concern with long-term software issues such as ease of understanding and maintenance.

The actual implementation of CL design has also seen a number of facets. Such concepts have been often implemented through a diversity of mechanisms and communication styles including: (i) the use of complex shared data structures and sometimes databases; (ii) the introduction of completely new abstractions where the layering concept is broken giving place to a new architectural design and even; (iii) allowing direct communication among layers to take place with no intermediary entity.

None of the above CL techniques can guarantee long-term code modularity and ease of maintenance because they rely on the additional sharing of variables, procedures, functions, etc. These may be accessed simultaneously from software at different layers hence turning information flow among them potentially harder to understand, follow, control, and maintain. There is therefore an urgent need for solutions that maintain the benefits of current good architectural design while still capitalizing on possible performance enhancements brought by CL. To such end, we suggest the use of AOP as a design and implementation style that turns out to be a suitable proposal to pursue.
when dealing with CL concerns. Next, we introduce a new programming approach based on the use of AOP, as a solution to the design problem at hand.

4. THE APPROACH

We describe next the programming approach proposed to capitalize on code simplification as a result of using AOP while still gaining from CL design. Our approach to the problem is divided into the following three parts:

a) First, we need to identify which cross layer concerns exist in our specific problem and then select the ones we would like to deal with using the AOP paradigm. Obviously, the object may not be to identify exhaustively all of these but the most important ones that are likely to help optimize the communication software, as shown in Table I.

b) For each of the cross layer concerns, a number of policies or goals may be defined to govern their execution: these policies are often subject to the nature of the network (used for business, military, emergency communications, etc.). Often such policies govern nonfunctional and functional areas of a network such as bandwidth, delay, security, load balancing, access control, network availability, congestion management, resource allocation, etc. The two goals shown in Table I are considered in the rest of this paper.

c) The cross layer concerns found in a) and policies from b) are then fed into AOP.

By following the steps described next, we show how Table I is obtained using our proposed strategy.

4.1. Policy specification

Our example, as shown in Table I, has two distinct policies for a wireless ad hoc network. Whereas the first one targets the maximization of the network life time, the second one seeks a configuration whereby the packet delivery rate is increased.

The proposed methodology is formally presented in the algorithm below whose steps are written in italics.

1. Step A.1 – Establish (M > 0) clear software design Goals \( G = \{G_i, i=1,M\} \);
   - The software designer may ask questions such as what is there to achieve, to preserve, avoid or eliminate. These questions should lead to identifying one or more system design goals. This step is similar to determining software requirements in the area of software engineering.

2. Step A.2 – Detect and remove possible goal conflicts;
   - With the increase in the number of goals and associated subgoals, it is likely that conflicts may arise. The software designer must ensure that such goals cannot conflict using approaches such as in [15] for their detection and removal. The actual details of this step are outside the scope of this document.

4.2. Finding cross layer concerns

The network designer should next identify the domain where cross layer design may apply. In this case, we chose the physical, data link and network layers for a wireless network as target layers as shown in step B.1.

1. Step B.1 – Selection and informal description of the system domain
   - This software domain consists of the (P)hysical, (D)ata Link and (N)etwork layers:
     \( \{L\} \leftarrow \{P,D,N\} \)

2. Step B.2 – Identification of the functional areas needed to implement the above target system software
   - In this step, we are looking for functional areas that have parameters that may influence possibly other layers, within the set \( \{L\} \). In other words, we identify those with a cross layer nature.

\( \{FA\} \leftarrow \{F_{A,1},F_{A,2},F_{A,3},\ldots,F_{A,k}\} \)
For each layer $l_j \in \{L\}$ do $\{\text{CL}_F A(l_j)\} \leftarrow 0$ (there are initially no cross layer elements in this FA at this layer $l_j$)

For each functional area $F_{Ai} \in \{FA\}$ do

Begin

For each layer $l_j \in \{L\}$ do

Begin

If $(F_{Ai} \times l_j)$ // this functional area can in any way be affected

// at this layer $L$

$(F_{Ai}) + \rightarrow p_{Lj}$; // Mark this function as begin affected

// by layer $L$, parameters $p_{Lj}$

End

End

The following output is obtained in our example as shown in Table II. Note that such Table may be built once and reused when needed as part of a library. Figure 2 shows the relationship of each of the goals $G1$ and $G2$ with these functions and the relationship among such functions.

3. Step B.3 – Extract the relevant cross layer concerns

As a result of the previous processing, we obtain a set of cross layer concerns $\{\text{CL}_F A,i(l_j)\}$ indexed according to the layers affected by them. The cross concerns and their relationships are depicted in the diagrams shown in Table III.

4. Step B.4 – Determine potential cross layer parameters

This is accomplished by looking at those functional areas or services that respond to more than a parameter affecting them in different layers of the software. Table III shows that both goals and concerns have a number of cross layer parameters that affect their enforcement. Some of these parameters, such as routing, affect both the network lifetime and packet delivery rate. Hence, these two goals should be carefully enforced to avoid conflicts. This is a typical example, where the level of conflict is not clear beforehand but that nonetheless may

<table>
<thead>
<tr>
<th>Layer</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical wireless</td>
<td>Battery consumption monitoring (BC)</td>
</tr>
<tr>
<td></td>
<td>Radio signal quality reporting (RS)</td>
</tr>
<tr>
<td></td>
<td>Power control</td>
</tr>
<tr>
<td>Network lifetime</td>
<td>Channel coding (MOD - Modulation)</td>
</tr>
<tr>
<td></td>
<td>Information coding</td>
</tr>
<tr>
<td></td>
<td>MIMO</td>
</tr>
<tr>
<td></td>
<td>Access control</td>
</tr>
<tr>
<td></td>
<td>Antenna diversity support</td>
</tr>
<tr>
<td></td>
<td>Error correction codes (EC)</td>
</tr>
<tr>
<td></td>
<td>Adaptive MAC</td>
</tr>
<tr>
<td></td>
<td>Automatic repeat request (ARQ);</td>
</tr>
<tr>
<td></td>
<td>Layer 2 broadcast</td>
</tr>
<tr>
<td></td>
<td>Cryptography</td>
</tr>
<tr>
<td></td>
<td>Addressing and routing</td>
</tr>
<tr>
<td>Network</td>
<td>Congestion control (CC)</td>
</tr>
<tr>
<td></td>
<td>Interworking</td>
</tr>
<tr>
<td></td>
<td>Resource discovery</td>
</tr>
</tbody>
</table>
Table III. Cross concerns for our example.

<table>
<thead>
<tr>
<th>Service</th>
<th>Maximize network lifetime</th>
<th>Maximize delivery rate</th>
<th>Cross layer parameters ($p_{ij}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery consumption monitoring (BC)</td>
<td>✓</td>
<td></td>
<td>battery-level estimated-lifetime</td>
</tr>
<tr>
<td>Radio signal quality reporting (RS); Power control (PC)</td>
<td>✓</td>
<td>✓</td>
<td>signal-level</td>
</tr>
<tr>
<td>Channel coding</td>
<td>✓</td>
<td>✓</td>
<td>power-control-step, max-power min-power</td>
</tr>
<tr>
<td>(MOD - Modulation) Access control (AC)</td>
<td>✓</td>
<td></td>
<td>Mod = QPSK, QAM-16, QAM64</td>
</tr>
<tr>
<td>Error correction codes (EC)</td>
<td>✓</td>
<td></td>
<td>none, best-signal, priority, service-type, window backoff parameter,..</td>
</tr>
<tr>
<td>Automatic repeat request (ARQ)</td>
<td>✓</td>
<td></td>
<td>Two</td>
</tr>
<tr>
<td>Routing (RT)</td>
<td>✓</td>
<td>✓</td>
<td>source-routing, on-demand,..</td>
</tr>
<tr>
<td>Congestion control (CG)</td>
<td>✓</td>
<td>✓</td>
<td>drop eligible bit (DE), drop tail, early discard,..</td>
</tr>
</tbody>
</table>

arise during the enforcement phase if both goals set the routing protocol parameters differently. Because conflict resolution is not our concern in this work, we will not come back to discuss it.

For each functional area $F_A \in \{FA\}$ do

Begin

If $|F_A| > 1$ then

// this functionality appears in more than a
// layer, hence it is a cross layer concern

$K \leftarrow |F_A|$ // $k$ represents the number of interaction

// parameters this function has

For each layer $l_j \in \{L\}$ do

For $i=1$ to $k$ do

// the parameter is added as

$\{CL_{F_A}(l_j)\} \leftarrow p_{ij}$

End

End

5. Step – B.5 Focusing on a subset of parameters

Furthermore, for each cross layer concern, we have raised a list of potential per-layer $\{l_j\}$ parameters that control their interaction, namely, $p_{ij}$. Next, the designer may optionally reduce the set of parameters that will be taken into the cross layer optimization. The aim is to concentrate on a subset of parameters that are more relevant to the cross layer concern and avoid a complex optimization strategy. Table IV shows a reduced set of parameters for each of our two goals.
Table IV. Cross concerns for our example.

<table>
<thead>
<tr>
<th>Concern</th>
<th>Maximize network lifetime</th>
<th>Maximize delivery rate</th>
<th>Layer</th>
<th>Cross layer parameters (p_{ij,l})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery consumption monitoring (BC)</td>
<td>✓</td>
<td></td>
<td>One</td>
<td>battery-level, estimated-lifetime</td>
</tr>
<tr>
<td>Radio signal quality reporting (RS);</td>
<td>✓</td>
<td>✓</td>
<td>One</td>
<td>signal-level</td>
</tr>
<tr>
<td>Automatic repeat request (ARQ)</td>
<td></td>
<td>✓</td>
<td>Two</td>
<td>On, off</td>
</tr>
<tr>
<td>Routing (RT)</td>
<td>✓</td>
<td>✓</td>
<td>Three</td>
<td>source-routing, on-demand,...</td>
</tr>
<tr>
<td>Congestion control (CG)</td>
<td></td>
<td>✓</td>
<td>Three</td>
<td>drop eligible bit (DE), drop tail, early discard,</td>
</tr>
</tbody>
</table>
all the sense, it may be less relevant for other scenarios such as a wired network. Such context dependency shows the importance of putting in place a simple, speedy, efficient, and flexible way to dynamically ‘hot swap’ and change policies when needed by the service or network designers. This should be achieved without the need to rewrite the applications, just the new policies. To achieve this, the aspect-oriented paradigm is used as part of a methodology that identifies the concerns, implements them, and creates the final system by combining them as shown next:

a) **Identifying system concerns** – In this step, we decompose the requirements to identify cross-cutting and core concerns. To clarify these concepts let us look at traditional network layered architecture where the problems are decomposed by layers with their respective core concerns. For example, in the OSI reference model the Network and Transport layers, have among their core concerns the functionalities of routing and congestion control, respectively. Hence, a layer 3 (see Java code in Figure 2) could be constituted of methods shown at lines 5–9, 10–13, and 14–15 expressing the functional modules \((FA_1, FA_{n-1}, ..., FA_n)\), and attributes at lines 2, 3, and 4. Nonetheless, this modular division is not respected by our two concerns, namely, maximize network lifetime and packet delivery as shown by the attributes into each method at lines 6 and 7 from the \(FA_1\) method.
b) **Concern implementation** — In this next step, we implement each concern independently. Let us continue using our example, where the concern of maximizing network lifetime (herein pointed as a policy) is crosscutting the physical, data-link, and network layer, as depicted in Figure 3.

The first step is to identify in each layer the functionalities or services that may be affected by this policy. Drawing a parallel with the approach presented in Section 4, this corresponds to the identification of the functional areas and its respective parameters that may contribute to increase the network lifetime. By identifying the functional modules, we can focus on each individual concern separately and reduce the overall complexity of design and implementation. To illustrate this, Figure 3 shows an abstract example of network layer functional modules where the implementation logic for maximizing network lifetime (MNL) policy resides inside the maximize network lifetime module (designed by AOP as an Aspect). AspectJ may then associate an aspect to each goal as shown by line 1 in Figure 4.

For each policy inside the goal we used the *pointcut* from AspectJ that references the context and local where the policy should be applied. For example, policy G1 in line 2 captures the execution of methods \( FA_1 \) and \( FA_{n-1} \). The reader is invited to think of *pointcuts* as those specifying the weaving rules and join points at situations satisfying those rules. The code to be executed at a joint point that has been selected by a *pointcut*, is defined in an *Advice* statement, and can execute before, after, or around the joint point. Using an *advice* statement, we can enforce the policy before executing the code at given *join* points that are spread across several modules. The body of *advice* is much like a method body — it encapsulates the logic to be

```java
1 public aspect PolicyMaximizeLifeTime {
2   pointcut G1() : call (void FA1() || FA_{n-1}());
3   pointcut G2() : call (void FA1());
4   pointcut Gn() : call (void FA_{n-1}());
5   before() : G1() {
6     //APPLY POLICY G1 [p11]
7   }
8   before() : G2() {
9     //APPLY POLICY G2 [p12]
10   }
11   before() : Gn() {
12     //APPLY POLICY Gn [p1n]
13   }
14  }
```

Figure 4. MNL aspect expressed in AspectJ.

executed upon reaching a join point. For example, in lines 6–8 in Figure 4, we write an advice that will apply the policy G1 using the attribute p11.

As shown in Figure 3 and 4, the crosscutting MNL requirements are now mapped directly to just one module — the MNL aspect. With such modularization, any changes to the crosscutting MNL requirements affect only the MNL aspect, isolating the clients completely. The result is that AOP modularizes the crosscutting concerns in a clear-cut fashion, yielding a system architecture that is easier to design, implement, and maintain.

c) Aspectual Recomposition – In this step, we specify the recomposition rules by creating modularization units, or aspects. The actual process of recomposition, also known as weaving or integrating, uses this information (i.e., implemented concerns) to compose the final system. For example, once the core part of the MNL concern has been implemented in a module it is then necessary to specify the points where the actions relative to maximization of network lifetime will be taken. The system then uses these rules to correctly invoke the MNL calls from the specified operations. In other words, in this step, we are able to refine the policies by removing, updating or adding new ones. For example, in Figure 5(a), we modified the policy in Figure 4 expressed in AspectJ by the addition of a check mechanism in line 7 before applying policy G1 in line 8. As a result, the system recognizes the code in Figure 5(b), because the Java framework is able to receive the AspectJ code and generate or update the previous Java class (here the L3 class). Then, the software designer only needs to change a policy in one location instead of searching for policy G1 modifying it wherever it appears such as in lines 6 and 12 in Figure 5(b).

5. USE CASE

In the earlier sections we presented the motivation for cross layer feedback and how the programming approach could be used to capitalize on code simplification as a result of using AOP while still gaining from CL design.
In this section, we present a simple use case to validate the concepts introduced in this paper. We use the MNL policy as a running example for a use case of implementation and validation. Because the cost of introducing changes into a protocol stack is relatively high we use TinyOS, an operating system developed for use on wireless sensor networks (WSNs). Currently, this is supported by many commercial products.

Wireless sensor networks are known to require light-weight protocol stacks. For instance, WSNs lack a transport layer because the end-to-end communication is handled by the applications. In this context, we propose modifications to the protocol stack considering a cross layer design between the link and routing layers. First, two relevant TinyOS WSN protocols, namely B-MAC and OPER are described.

Berkeley-MAC (B-MAC) [17] is a medium access protocol based on channel detection similar to CSMA (Carrier Sense Multiple Access). Its main task is the reduction of energy consumption while avoiding packet collision. It has been designed to facilitate its control from the high level layers. To this end it offers a number of adjustable interfaces. The main B-MAC services are: clear channel assessment used to avoid collisions, and low power listening used to save radio energy. Both help connectivity management by the nodes. At this data link layer, it is possible to obtain information important for optimizing energy consumption, delay, throughput, and reliability.

The OPER (on-demand power-efficient routing protocols) is a family of WSN on-demand reactive routing protocols [18, 19]. OPER disseminates the information captured by sensors, their addressing, naming, path quality estimates, link management, queue states, search scheduling while taking into account energy levels for nodes along a transmission path. More specifically, connectivity management is achieved through the exchange of the heartbeat HELLO message.

Figure 6 shows the elements for OPER and B-MAC that may be shared for the purpose of optimization. These may be encapsulated in aspects corresponding to transversal shared components.

Figure 6 reinforces the existence of common functionality between the network and link layers including connection management. Because of the delay and energy consumption resulting from layer three connection management function, the use of aspect MNL shown in Figure 7 is seen as a more suitable alternative design. This is because it allows the network layer to continuously obtain important connectivity information from the link layer. This is then used to help improve the WSN route maintenance.

The previously dispersed connection management functionality has become centralized into a single place or concern as shown in Figure 7. Bidirectional link or route testing, message acknowledgement, and routines for link problem detection are all examples of mechanisms that touch this

---

1http://www.tinyos.net/
functionality. This way, the OPER routing protocol may benefit from the aspect 'MNLAspect'. Our implementation of this simple common Aspect shows a better usage of bandwidth and lower energy consumption as described in the next section.

6. EVALUATION

To analyze the benefits of using an Aspect shared by the MAC and routing protocol, we adopted a simple metric: energy efficiency. It is defined as the ratio of the number of successfully exchanged messages, measured in bytes, by the total dissipated energy by the sensors in $\mu$Joules.

A number of simulations were carried out for sensor networks with 50, 100, 150, and 200 sensor nodes, randomly distributed in an area of $50 \times 50$ m$^2$ for the first two networks and $100 \times 100$ m$^2$ for the other two WSNs. Each node can transmit within a 10 m range.

We implemented OPER into the TinyOS protocol stack version 1.15. The total simulation time was set to 600 s, time sufficient enough for evaluating the behavior of the WSN network and its nodes. During the startup phase, the nodes are activated randomly and remain in this state during 10, 20, 30, and 40 s for the networks with 50, 100, 150, and 200 nodes, respectively.

Each initiated node started with a 3000 $\mu$Joules of stored energy. We purposely chose this limited amount to force the selection of new routes during the simulation. Energy consumption values were based on measurements taken from a real WSN network using the Mica2 sensors [20]. Table V summarizes the adopted simulation parameters.

Figure 8 shows the ratio between the successfully transmitted bytes per Joules for each of the WSNs with 50, 100, 150, and 200 sensors.

Table V. Simulation parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensing Area</td>
<td>$50 \times 50m^2 \text{ or } 100 \times 100m^2$</td>
</tr>
<tr>
<td>Network Area</td>
<td>50, 100, 150 or 200 nodes</td>
</tr>
<tr>
<td>Sink node localization</td>
<td>Center of Networks</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>600 seconds</td>
</tr>
<tr>
<td>Initial battery capacity of the sensor nodes</td>
<td>3000 $\mu$Joules</td>
</tr>
<tr>
<td>Startup time of the sensor nodes</td>
<td>Random(10, 20, 30 or 40) seconds</td>
</tr>
<tr>
<td>Message size</td>
<td>36 bytes</td>
</tr>
<tr>
<td>Message generation rate</td>
<td>random within the range from 0-30 seconds by packet</td>
</tr>
</tbody>
</table>
Figure 8. Ratio of transmitted messages (in bytes) per energy for WSNs with 50, 100, 150, and 200 nodes.

Figure 9. Number of active sensors for a 200 nodes WSN after 1500 s simulation time.

Figure 8 shows the energy saving of AOP design making use of cross layer information. The gains are 11.8%, 14.4%, 8.1%, and 10.2% for WSNs with 50, 100, 150, and 200 nodes, respectively. These gains have been achieved through two main optimizations. The cross layer information exchange through AOP design allows reducing the number of heartbeat network level hello messages. The performed simulations also take into consideration the number of neighbors. Hence, route selection is more efficient because of the updating of routing tables using cross layer information coming from the link layer.

The use of CL information as a result of the AOP design has also benefited the number of nodes that remained active during the simulation in addition to achieving higher delivery rates as shown in Figure 9.

The use of neighbor state information from B-MAC allowed the network protocol OPER to better react to changes in the WSN.

7. DISCUSSION

This paper shows the inherent properties of AOP making it a promising solution for untangling cross-layer design. First, AOP promotes clear design and reusability by enforcing the principles of abstractions and separation of concerns, unlike earlier development paradigms. This process is essential for the description of concerns that crosscut different functional modules in a protocol stack. Moreover, AOP addresses each concern separately with minimal coupling. This results in
modularized implementations and a greater flexibility for a network designer to introduce new functionalities. Second, because the implementation of each concern is separate, it also helps avoid code clutter, which is one of most important problems in cross-layer implementations. Third, the process of collecting scattered concerns into compact structure units, namely the aspects, provides cleaner assignment of responsibilities and should thus lead to software systems that are easier to understand and maintain. Fourth, the addition of a new cross-layer goal or policy is now a matter of including a new aspect and requires no change to the network architecture. Furthermore, when we add a new core module or extra-layer, for example secure IP, to the network system, the existing aspects crosscut it, helping to create a coherent evolution. The overall effect is a faster response to new requirements.

On the other hand, AOP is a young programming methodology and so many open issues remain unaddressed. One such issue is its capability in providing tools that help to collect scattered concerns or nonfunctional requirements. For example, QoS resource management and the identification of points that crosscut a protocol stack and factoring out these into a separate aspect remains a hard undertaking. AOP is mostly suited for large-scale software development projects, especially as we want to emphasize in this article, distributed systems. However, it brings with it certain difficulties as far as testing and debugging are concerned. This is due to the side effects that stem from the dynamic (ad hoc) injection of runtime code and which could, in a worst case scenario, lead to semantic ambiguities in the control-flow of an aspect-oriented program if not steered in a well-controlled way.

Because of the lack of advanced tools, current aspect weaving processes have the following limitations: (i) because the aspect weaver is usually hard coded, programmers have no control over the aspect weaving process; (ii) if programmers need to modify the weaver, they cannot avoid the complexity of the target language, such as C++ or Java; and (iii) modifying the aspect weaver is difficult, tedious, time-consuming, and error prone.

In terms of execution, our use of the above examples has shown little to no performance overhead introduced by AOP. Such observations corroborate with the results from [21] and [22]. Their authors evaluated the CPU and memory overhead established by AOP when compared with OOP under Java and C++ programming languages, respectively. In some scenarios AOP outperformed pure Java code but overall the results remained very close with a difference below [21]. A similar behavior was observed in the case of pure C++ and AspectC++. AOP even outperformed C++ OOP when used in conjunction of a C++ compiler optimizer [22].

8. CONCLUSION

In this paper, we have initiated a discussion about the introduction of diverse software programming paradigms in the networking community. Particularly, we focused on the use of the AOP paradigm to develop CL solutions with different policies as part of a simple example. Moreover, we showed through this example that the use of aspect-oriented CL design can improve the modularity of existing and increasingly tangled CL code and avoid its usual scattering and repetition. Critics of CL design argue that the breakup of the traditional layer model often leads to unstable systems. Meanwhile, critics of AOP often talk about the difficulty of understanding it. However, perhaps the main reason behind these difficulties is simply the fact that we still remain at the early days of this paradigm. AOP remains a new field and further work to quantify the gains in terms of ease of design and performance is needed through actual implementations and more studies. The challenges remain numerous and more insights are needed to create working architectures.

REFERENCES


**AUTHORS’ BIOGRAPHIES**

**Djamel H. Sadok** received his Ph.D. degree in Computer Science in 1991 from the University of Kent at Canterbury, UK. He joined the Federal University of Pernambuco in 1993. He is now an Associate Professor after working for two years as a research fellow at Computer Science Department of University College London in the area of Security. Currently, his interests include traffic measurement, protocol design and validation, security and authentication, content delivery networks, and cloud computing. Professor Sadok has published more than a hundred technical papers and supervised the work of many Ph.D. and MS students.

**Luciana P. Oliveira** is a Ph.D. student at Federal University of Pernambuco and IT consultant at AT1 of the Brazilian government. Her special fields of interest are in delay tolerant networks; social routing; next generation networks and policy based management. She has research experience in the communication field, with emphasis on dynamic networks, routing protocols and P2P. She received her MS degree and obtained honors as the best student in the graduating class, both in Computer Science at the Federal University of Pernambuco. She has published some papers on International and Brazilian symposiums, conferences and workshops.
Glauco Gonçalves received his MS degree from UFPE. He is currently a Ph.D. candidate in science computing at the same university. He also works at GPRT, a research group in the areas of computer networks and telecommunications. His current research interests include systems performance evaluation, network management, and cloud computing.

Judith Kelner received her Ph.D. degree in Computer Science in 1993 from University of Kent at Canterbury, UK. She joined the Federal University of Pernambuco in 1979 where she is currently an Associate Professor and a senior researcher in the Network and Telecommunications Research Group (GPRT). She is also the head of the Virtual Reality and Multimedia Research Group. Currently, her interests include ambient network, virtual and augmented reality, advanced user interfaces, and multimedia systems. Professor Kelner has published almost a hundred technical papers and supervised the work of many Ph.D. and MS students.

Nelson Rosa received the BS, MS and Ph.D degrees in Computer Science from Federal University of Pernambuco, Recife, Brazil in 1992, 1996, 2001, respectively. Since 2001, he has been a lecturer at Federal University of Pernambuco. In this period he collaborated in projects related to service-oriented computing and cloud computing with HP Labs (Palo Alto). His research interests include development of quality-aware middleware systems and service-oriented applications.

Eduardo Souto received his Ph.D. degree in Computer Science in 2007 from the Federal University of Pernambuco in Brazil. Currently, Dr. Eduardo Souto is an Associate Professor at the Institute of Computing, Federal University of Amazonas, where he coordinates the undergraduate course in information systems. He is a leader of the Emerging Technologies and System Security (ETSS) research group. He has also been in the program committee and organizing committee of national and international conferences in Computer Science. His research interests are in the areas of Wireless Sensor Network, Network Security, Internet Traffic Measurement, and Distributed Systems.