3. Related Work

This chapter introduces related work on cross-layer design, cross-layer optimizations and adaptations. The main focus is on cross-layer architectures and on single adaptation mechanisms that were introduced without specific architectural considerations. Additionally, some criticism on cross-layer design in general that was recently expressed and the ability of the proposed architectures to withstand that criticism is analyzed. Most importantly, proposed architectures are compared against each other and are discussed in detail. Related work that is closely connected to the adaptation mechanisms introduced later in this document will be discussed in the corresponding chapter.

3.1. Cross-layer Adaptations and Optimizations

Cross-layer adaptations are the fundamental protocol enhancements that a good cross-layer architecture enables and supports by providing basic services such as data exchange and accessibility. In other words, the adaptations are the mechanisms that utilize the cross-layer architecture to improve protocol efficiency.

Many such adaptations have been proposed in the literature for a very broad spectrum of possible application and protocol shortcomings. At the same time most of these adaptations leave out the actual process of cross-layer communication and interaction. Architectural considerations are rarely found but the sheer amount of proposed adaptations suggests that there is a definite need for an architectural framework to base adaptations on. These single adaptations also reveal the plethora of possible cross-layer adaptations and optimizations that can significantly increase the performance of an ad hoc networking stack. This section describes some of the existing cross-layer adaptation mechanisms that have been developed.

Starting at the topmost layer, the application layer, several cross-layer optimizations have been suggested and described. For example CrossROAD, a peer-to-peer system based on Pastry, was proposed in [17]. CrossROAD utilizes the network topology information provided by a proactive routing protocol that is the core requirement for CrossROAD. It also introduces some interaction between CrossROAD and the routing protocol which requires a joint design. The general idea is that by utilizing the routing state information the overlay network construction and maintenance can be optimized as the routing protocol already maintains complete topology information. CrossROAD does not address further ad hoc network specific in-layer optimizations though, such as proposed by DynaMO [5] and is restricted to proactive protocols. This restriction to a class of protocols in general is not desirable and a good architectural design might help to weaken these very strict requirements.

Multimedia applications are very demanding from a networking perspective as they have very strict bounds for QoS parameters such as jitter and throughput. On the other hand there is some flexibility and adaptability intrinsic to multimedia applications such as streaming video as, for example, video encoding, video resolution and others can be tuned. That is why many cross-layer optimizations target explicitly this group of applications. An example for such an optimization can be found in [18] where an error protection scheme for video transmissions is described. The general idea is that channel state information is utilized to adapt the video encoding scheme to provide the most appropriate error protection for a given packet loss ratio. What this scheme achieves in comparison to conventional forward error correction (FEC) is that the video quality degrades graceful with an increasing packet loss ratio.

When it comes to QoS ad hoc networks are probably the most challenging network variants that exist since they can be very dynamic in many respects which makes guaranteed availability of network resources and performance highly difficult. Hard-QoS, i.e. guaranteed bounds on delay, jitter or bandwidth in such environments is nearly impossible to provide as many factors that affect these metrics cannot be controlled. In [19] a QoS system was designed that supports soft-QoS for 4 different service classes by incorporating cross-layer interaction between the MAC, network and application layer. At the MAC layer many statistical metrics are collected and provided to upper layers such as the routing layer which performs load balancing by choosing the least busy route as measured by the MAC. The metrics are further used for traffic shaping and all system components together provide QoS in a DiffServ manner.

Other optimizations try to increase the performance of ad hoc routing protocols as they are one major source of inefficiencies in ad hoc networks. Several cross-layer optimizations work on the problem of node mobility. For example in [20] a link stability metric based on an entropy concept is calculated at the MAC layer and used for the route selection process. Some routing protocol messages have to be altered for this and the whole approach is tailored towards on-demand protocols which would violate two of our own system constrains as regular and cross-layer protocols are not interoperable in one common network any more due to the changed message format and as there are constrains on the choice of the protocols which can be used with the developed approach. With such an approach on cross-layering this would need to be done on a per-protocol basis which would be a very complex task. In addition, since the message formats are changed, to keep the network operational the protocols would need to be updated on every participating node simultaneously. This example illustrates the importance of a sound architectural design. Another cross-layer routing approach combines hardware driver information such as GPS location information to derive a priority index that is used inside the Cross-layer Route Discovery Framework

(CRDF) [21]. This framework was designed to solve two problems of on-demand routing protocols which are the “rebroadcast redundancy” and the “next-hop racing” problem. What this approach clearly shows is that information outside the traditional network stack (GPS, energy etc.) should also be made available to network protocols.

As energy is a primary concern in ad hoc networks, several cross-layer optimizations have been proposed to conserve energy. Several power control protocols have been proposed in [22] which connect physical and network layer in a way where in principal several proactive routing protocols run in parallel each at a different power level. Based on the routing table information, the power level is chosen for data traffic that fully connects the network while using the least energy.

The major transport protocol found on the Internet, the TCP (Transmission Control Protocol), has been found to perform very poorly in mobile ad hoc networks. This is due to the assumptions standard TCP makes about the underlying network. For example, if a packet does not make it to its destination congestion is assumed since the Internet is a very reliable, wired medium and packet loss due to other reasons is unlikely. TCP would reduce the packet send rate to alleviate the congestion. This assumption does not hold true in wireless networks as packet loss is likely to be due to the interference-prone, shared wireless medium and reducing the send rate on every lost packet degrades the performance significantly. Many cross-layer schemes to improve the performance of TCP in dynamic, heterogeneous networks have been proposed. One exemplary scheme is called A-TCP which was developed in [23] using MAC layer statistics to fine-tune the transport layer. More specifically, the MAC layer collects statistics about the unsuccessful transmissions of RTS (Request To Send) packets. These statistics are used to adjust the size of the maximum window size at the sender side to prevent a sender from causing congestion at forwarding nodes. Another TCP improvement can be found in [24] where an additional software module generates and drops TCP acknowledgements locally to prevent unnecessary acknowledgement exchange. The actual cross-layer interaction is here also between the transport and MAC layer where ARQ (Automatic Repeat-reQuest) messages already indicate the successful delivery of a packet. To add an additional software component might not be a good design choice for the general case as every adaptation and optimization process might require such a component making the overall design of the communication system highly complex.

There exist many more cross-layer optimizations and adaptations on all traditional layers of the protocols stack, too many to describe them all in detail. As this thesis’ focus is not on single adaptation mechanisms but on cross-layer architectural design this section should only give an idea how many possible adaptations exist and how such adaptations can look like. The vast amount of single optimization mechanism itself is a key motivation for a good architectural framework. A good general overview can be found in [25].

3.2. Cross-layer Architectures for Ad Hoc Networks

Compared to the vast amount of single cross-layer adaptations only a few cross-layer architectures have been proposed so far. All architectures share some common features as the general cross-layer idea is very straight forward. What they significantly differ in is the way the cross-layer principle is implemented, what kind of application focus and scope the architecture has and where the actual adaptation intelligence is situated. The following sub-sections present proposed architectures in detail and compare the goals of these architectures with the goals of this work.

3.2.1. Cross-layer Approach To Self-healing (CATS)

CATS [27], as the name implies, is tailored towards self-healing properties across all layers of the network stack. The approach itself is tailored towards sensor nodes and battlefield applications, as self-healing is crucial in those scenarios. CATS does not require a change of the underlying infrastructure, namely the routing protocol and the medium access control to be able to accommodate the protocols provided by the military. The authors of [27] introduce a component they call *Management Plane* containing the *Cross-layer Platform* which contains protocol information and is visible across all layers. The Management Plane itself is an active component which carries out necessary self-healing functions and actively influences packets as they traverse the protocol stack. It can even actively influence protocol behavior. As an example, the Management Plane can actively change the destination of a packet or make the routing protocol stop responding to route requests. One of the core components of the CATS approach is that each node maintains a table containing interchangeable nodes. Interchangeable nodes are those nodes which are in close proximity and provide the same sensing and computational abilities. In case one of the nodes fails, an interchangeable node can take over the responsibilities of the failed node.
The disadvantage of CATS is that the framework itself represents an active component, i.e. the framework itself influences protocol behavior and state. This means that for every protocol a mechanism must be developed to do this and protocol designers must cater for service points where the framework can actually influence protocol behavior and data. That means that as new protocols and applications are introduced the framework has to be updated or replaced too. That in general is not a good architectural design. In addition designing a protocol is not independent any more from the underlying framework and the design complexity increases significantly. The question that arises is how the authors want to leave the routing protocol and MAC protocol unchanged and still achieve their goals as they need to access the protocol data and in addition change the protocol's behavior. CATS is also a very powerful component. Failure of CATS can result in a total failure of the network stack. Bugs, unforeseen circumstances and attacks to CATS can easily disrupt a functioning network.
Compared to the goals of this thesis CATS also does address the issue of unreliability in ad hoc network scenarios. Since it mainly targets at reactive self-healing strategies and contains active functionality instead of merely providing a generic architectural framework it does not cover per se a broader spectrum of ad hoc networking issues. Traditional networking issues are left an open issue and will most probably be left to in-layer adaptations. Novel application support is not mentioned explicitly as well as metric generation and provisioning as protocols will remain unchanged and therefore will not make use of additional information. Also static parameterization is not addressed. Instead some

parameters mentioned are static for CATS like the number of nodes in the table of interchangeable nodes. Also network-wide optimizations are not dealt with or envisioned. In fact none of the architectures presented in this chapter do. Up to now, CATS was never exemplary evaluated, i.e. it was never actually used to demonstrate its operation.

3.2.2. ECLAIR

In [28] ECLAIR is presented, which is loosely based on previous work described in [29]. ECLAIR is a two tier architecture consisting of an *Optimization SubSystem (OSS)* and so called *Tuning Layers (TL)*. A Tuning Layer provides access to the data structures held in a specific protocol. For example, the Network TL or more specific the IP TL would have access to the state of the IP protocol and by altering that state the protocol behavior can be changed. A TL is split into two parts: a generic part containing common protocol layer interfaces that are implementation independent and an implementation specific part. That means that for every single implementation there must be a TL that is aware of the implementation details. TLs also provide an event notification mechanism to notify registered components, so called *Protocol Optimizers (PO)*, when observed data structures change. There is also a TL for user feedback. This way the user can directly influence the behavior of the protocol stack. In [28] an example application of the User TL is given, where a user prioritizes file downloads and TCP parameters are set accordingly.
The OSS is the cross-layer engine containing the optimization algorithms and corresponding data structures. The containers for the optimization algorithms are the aforementioned Protocol Optimizers. They take optimization actions on events they registered for or on protocol state that they have access to.
ECLAIR shares some similarities with CATS as it splits the data part and the active cross-layer optimization part. Both CATS and ECLAIR actively influence protocol behavior with the result that there is no well defined separation between framework and functionality any more. The reason behind integrating the cross-layer optimization algorithms into the framework is that this way there are only minor alterations necessary within the existing protocol stack which is one of the design goals of ECLAIR. The complexity of ECLAIR is increased by this approach as many components reside inside the architecture exhibiting complex interactions and relationships with each other. Security and stability might become an issue over time as the system will grow with every protocol, application and update added to the protocol stack.
ECLAIR addresses some of the goals of this thesis. Since it actively alters protocol state it addresses the issue of static parameterization. The User TL is a good example of catering for novel applications. What ECLAIR leaves out completely though are network-wide optimizations and optimization metric generation. A handling of the more conventional network issues is not explicitly addressed in [28] but it appears to be possible to some extend.

3.2.3. Global Resource Adaptation through CoopEration (GRACE)

The main focus of GRACE [30][31] is on mobile multimedia terminals and QoS provisioning and at the same time resource conservation. Primarily, GRACE tries to conserve energy as it is rated a first-class resource. Instead of restricting the cross-layer framework to the network stack the whole device is considered the optimization domain. Therefore, the authors of [30] use a slightly different terminology when talking about layers. A layer, as defined by GRACE, corresponds with a device layer which can be the operating system, the hardware, the network stack or applications as shown in Fig. 3.1. GRACE itself coordinates the cooperation amongst the system layers by providing interfaces to a central *resource manager* where the coordination functionality is situated. The resource manager mediates between the layers to find a suitable combination of possible configurations for each layer to achieve near optimal global results. It has to be stressed that in GRACE terminology, global refers to the device and not to the network. The way GRACE coordinates application demands and available resources is a two stage process, one involving global adaptations and one involving local adaptations. Global adaptations are only triggered rarely, i.e. the system normally operates in the local adaptation mode. Global adaptations only occur when either application demands grow significantly, new applications come into the system or resource availability changes, violating certain thresholds. Global adaptations work as follows. Applications need to specify their resource requirements together with the corresponding associated utility at certain operation points the application provides. To be more specific each application must provide a set of different requirement/utility configurations to the resource manager. Utility can be some application specific metric such as frame rate for a multimedia application and resource requirements are metrics such as CPU utilization. To acquire resource requirements at certain operation points, applications query the system layers through resource monitors. Having received all possible system configuration sets from all applications, the resource manager picks the configurations that meet the resource constrains at the highest possible utility and assigns the resources to the applications. In other words the resource manager mediates between the available resources and application demands guaranteeing fairness to the applications and preventing over-utilization of system resources.

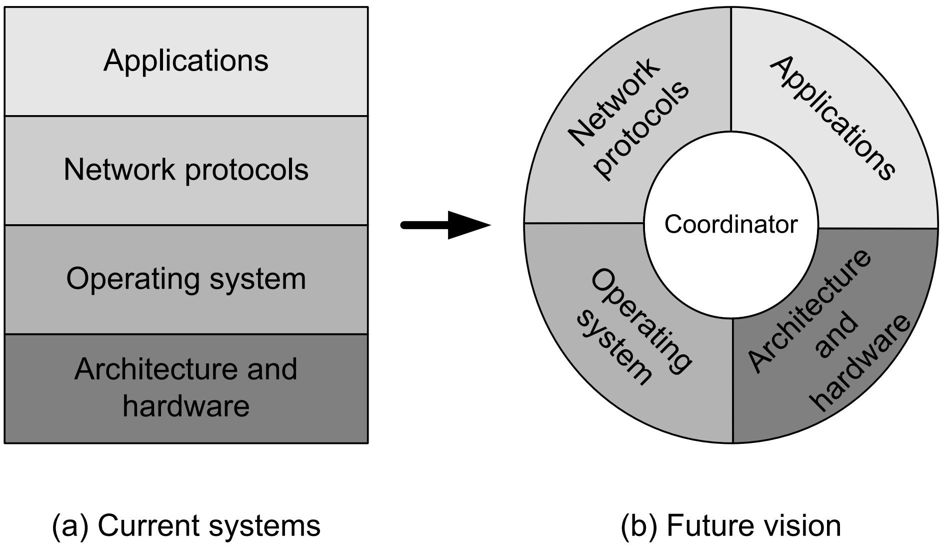

Fig. 3.1 GRACE's view of a device's system components [30]

As already mentioned the expected steady state mode is GRACE's local adaptation mode. Local adaptation does not involve the central resource manager.

Once resources are allocated, the systems layers have some degree of freedom to locally adapt to minor changing conditions and fluctuations as long as they do not excess the previously allocated resource allowance or reduce the overall system utility.
GRACE is a fairly complex framework as it addresses the whole device and its resources. The global adaptation process is by far the most complex task and at the same time the most important one. Every application that wants to make use of the GRACE framework must include some kind of cost model and should allow for multiple operation points. Without multiple operation points GRACE would not make much sense since there is no coordination possible. The cost model itself can become quite complex and difficult to predict since many applications vary drastically in resource consumption over time. This results in a pre-assessed operation point to be violated which in turn would trigger many expensive global adaptations, disrupting an unobstructed operation of the applications. Local adaptations are also very difficult since very often it is not possible to assess the effect of an adaptation exactly prior to applying it. This might not apply that much to the hardware, for example, as the effect of stepping the processor speed up or down is well known and easily quantifiable. But it is especially true for the network layer since the communication medium is shared and a node only has partial control over it. This is why network-related utility violations can be expected more frequently making global adaptations necessary. The question arises whether it is practicably possible to provide several network stack pre-configurations exposing well predictable performance measures for the complex global adaptation process.
Since GRACE does not focus on the network stack alone, actual cross-layer network optimizations are not a primary concern. That especially includes network-wide optimizations and also more traditional networking issues. What it does though is providing a framework for novel applications that are adaptable to the dynamics found in ad hoc networks and that run on typical devices found in such environments. Very positively to be mentioned is that there is a limited functionality GRACE prototype implementation called GRACE-1 [32]. The authors' application focus is very narrow and important for the framework. It is on multimedia applications that have periodic tasks such as video decoding has, as it can operate on a per-frame basis. GRACE-1 only adapts inside the operating system layer, the application layer and the hardware layer. The obvious problematic layer, the network layer, is not part of the prototype. The overall overhead of GRACE-1's global adaptations in terms of CPU stress is low compared to the computation of a MPEG frame but increases linearly with the number of applications running as can be expected. Additional performance gains can be achieved for process groups as described in [33]. Unfortunately, GRACE-1 does not utilize the local adaptations in most of the experiments as the ones that are applicable to their demonstrator application are considered to be to heavyweight by the authors. Therefore, GRACE-1 was further refined in [34] and renamed GRACE-2. GRACE-2's main focus is to show the effectiveness of another introduced local adaptation mechanism, the per-application mechanism, but still has the same application focus. The network layer is still non-adaptive and the network itself is a static simulated one. It was shown though that those per-application adaptations are much more lightweight compared to global ones as

can be expected. They also yield relatively good energy savings in scenarios with restricted network bandwidth.

3.2.4. The MobileMAN (Mobile Metropolitan Ad hoc Network) architecture

The MobileMAN [35] project itself is a European research effort funded under the 5th Information Society Technologies (IST) Framework Programme of the European Community. Being a very big project, MobileMAN tries to explore the various aspects and potentialities of a wireless, self-organizing metropolitan area ad hoc network including social and economical aspects. Building a cross-layer architecture is only one of their many objectives though fundamental to the overall project and therefore a primary concern. They very strictly follow the cross-layer idea as their main design objective is to let protocols share data freely but at the same time conserve layer separation. MobileMAN, as well as the other proposed approaches tries to solve the performance problems found in ad hoc environments and also has a focus on connecting this new technology with the existing Internet.

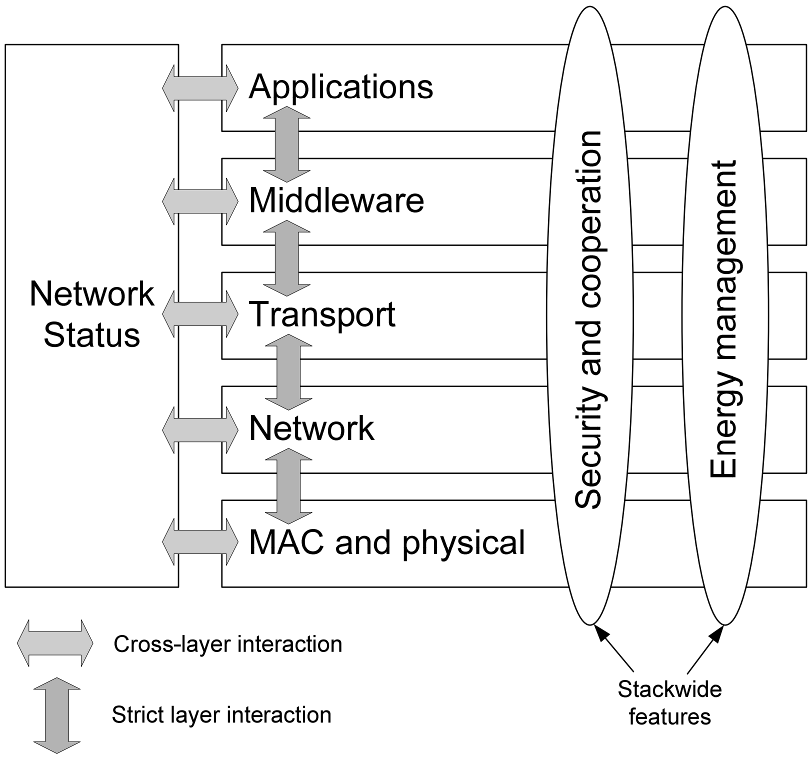

Fig. 3.2 The MobileMAN architecture [35]

[35] and [36] describe the conceptual MobileMAN cross-layer design and the general structure of the architecture as depicted in Fig. 3.2. They identify some general objectives which are cross-layer by nature, i.e. which are relevant at each layer of the network stack, such as security, energy management and cooperation. The core component, enabling the data sharing, is called *Network Status (NetSt).* Protocols of all network layers can store their collected data within the Network Status, acting as a general purpose data container. The access to the Network Status is well defined, regulating the way protocols actually access the stored information, i.e. write and read operations on the data are strictly governed by

the Network Status itself. By placing this component including the interaction with it in juxtaposition to the network stack, a normal operation is guaranteed. Or in other words, within a stack, purely layered network protocols and protocols making use of the Network Status can coexist. In [37] this characteristic is termed *loosely-coupled cross-layering* as the Network Status introduces a level of indirection and acts as a mediator between layers. The more purely layered protocols operate within the network stack the less optimization potentialities exist as the protocols do not make use of the additional information provided by the Network Status and at the same time do not provide any information. That means that in order to fully exploit the advantages of the MobileMAN architecture, network protocols must be redesigned to provide their information and also make use of information from other layers. Within the MobileMAN project also a peer-to-peer middleware, CrossROADS [17], was designed which makes use of the cross-layer framework. Interestingly, the authors of [35] and [36] already make certain assumptions about the suitability of certain protocols classes to make use of MobileMAN as a cross-layer architecture. Reactive routing protocols for example are seen as much less suitable since topology information which could potentially be shared is not necessarily available at all times or largely incomplete.

In the 10th deliverable [37] of the MobileMAN project the authors mention to continue working with a layered architecture and only explore as much as possible the cross-layer design in parallel. The design itself is further refined in subsequent deliverables [37][38][39]. Two types of interaction are supported by the NetSt, *synchronous* and *asynchronous*. Synchronous interaction simply means that a protocol requests data from the NetSt whereas asynchronous interaction is an event notification system the NetSt provides and protocols need to register for. Events are classified as *internal* and *external* where internal events refer to events generated inside a protocol and external events are triggered by the NetSt itself. The data inside the NetSt is abstracted to accommodate for the various protocols which internally represent their data differently. In other words, the abstraction the NetSt provides is an agreement about the common representation of data inside the NetSt. The transfer of protocol internal data into the NetSt is done by using *call-backs*. They are provided by the protocols and transform internal data into the abstracted form of NetSt, which actually invokes the call-backs.

The MobileMAN cross-layer framework fulfills some of the requirements that are the basis for the work described in detail later in this document. Seamless integration for example is guaranteed as layered and cross-layer protocols can coexist in the same protocol stack. Clearly, MobileMAN has a focus to solve the problems that are unique to ad hoc networks such as energy efficiency of which some are an issue across layers by nature. A problem of the MobileMAN architecture is that it already limits the amount of possible protocols running inside the framework. For example, a proactive routing protocol must exist that maintains complete topology information if CrossROAD should be used. Additionally, that protocol must be able to disseminate CrossROAD service information. This somehow violates the aforementioned goals of the MobileMAN architecture. Another difference is that more significant data is not generated but instead very complex data abstractions are provided by the architecture which potentially should accommodate every kind of information which can be produced by a protocol of the network stack. Also, the framework itself does also not cater for network-wide objectives and optimizations and per se.

3.2.5. WIDENS (WIreless DEployable Network System)

The WIDENS (WIreless DEployable Network System) project [40], also supported by the European Community's research fund under the Sixth Framework Programme, includes cross-layering as a fundamental principle into the system design. The WIDENS architecture is an ad hoc communication system for future public safety, emergency and disaster applications and is therefore intended for very well defined deployment scenarios. The advantage is that certain assumptions can be made such as the existence of a central server and a hierarchical organizational structure of public safety organizations. WIDENS makes use of these known environmental and organizational specifics in their systems design. In addition to that, specific protocols are already chosen for the architecture but the fundamental ideas are still in principle generic even if the chosen solution is not as there exist many protocol dependencies. The project itself comprises many objectives ranging from MAC/PHY layer co-design to network layer design.

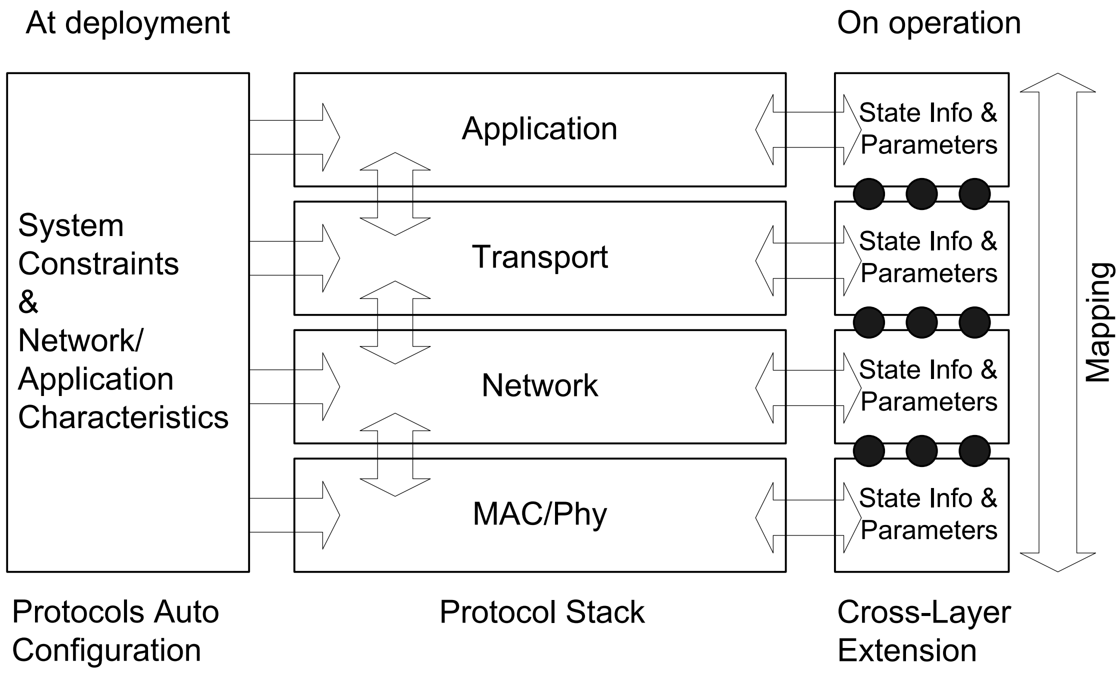

Fig. 3.3 The WIDENS architecture [40]

A special focus of the cross-layer functionality is on the routing problem in ad hoc networks under Quality of Service (QoS) constraints [41][42]. In [43] the cross-layer interactions are described which are realized through well defined interfaces between adjacent layers. These cross-layer extensions provide state information and parameter mapping between adjacent layers to increase protocol reconfigurability and adaptability [44] as can be seen in Fig. 3.3. The cross-layer information is utilized in the case that in-layer optimizations cannot prevent performance degradation. Potential adaptation loops are avoided by only allowing interactions between adjacent layers. Information from non-adjacent layers can only be accessed through the mapping functions of the layer above or below. This

can only be done if the cross-layer adaptations at the adjacent layer are insufficient and other layers need to adapt in addition to achieve a satisfactory networking performance. Furthermore, this way, unnecessary and unintended cross-layer operations can be avoided.
The scope of the WIDENS project is rather narrow as it targets a very specific application domain. Cross-layering is a little more restricted compared to other approaches such as MobileMAN since only adjacent layers perform cross-layer interactions at first and only in cases where these adaptations are not sufficient other layers gain access to lower layer information. This potentially leaves out possible performance gains. In addition, it is not necessarily true that neighboring layers are the optimal choice for adaptations if only one layer is chosen to adapt.
The general goals of the WIDENS are comparable with the MobileMAN project. As such, it has similar limitations compared to the design goals in section 1.2. These include that network-wide optimizations are not a primary design goal and mechanisms to achieve them explicitly are missing. Additionally, the architecture does not utilize basic protocol information to compute more relevant and meaningful metrics for protocol adaptations. On goal is reconfigurability, but basic protocol characteristics are set at deployment time. Protocol parameterization might therefore be rather static which might be a valid assumption for a system such as WIDENS as its deployment scenario is rather well defined.

3.2.6. TinyCubus

TinyCubus [45][46][47][48] is a cross-layer framework especially targeted at a very specific type of ad hoc networks, sensor networks. More specifically TinyCubus targets sensor networks that consist of nodes running the TinyOS operating system narrowing down the application domain of TinyCubus greatly. One of the core motivations for a cross-layer architecture solely designed for sensor networks is their data-centric nature making them unique in some respects. Also some special requirements, not limited to but especially pronounced in sensor networks such as energy efficiency can justify a focus on such networks only.
TinyCubus consists of three major components which are the *Tiny Cross-layer Framework*, the *Tiny Configuration Engine* and the *Tiny Data Management Framework*. The cross-layer framework provides two basic functions. One is data management in the sense that a layer can provide information that is stored in a data repository that in turn can be accessed by any other layer. The other function allows the execution of custom code through callbacks. Application-specific code can be invoked this way at lower layers.
The configuration engine is responsible for distributing and installing code in the network as TinyCubus allows changing components running inside the framework to be replaced or added at run-time. The configuration engine contains a topology manger which beyond being responsible for the self-organization of the network handles the node’s role assignment. A role a node has can depend on factors like hardware capabilities, location and others. For example a node with a virtually unlimited energy source can act as a data sink, gateway or cluster head. The code distribution entity of the configuration engine

makes sure that updating components is done in an energy efficient and reliable manner.
The last component, the Tiny Data Management Framework, is where the name TinyCubus has its origin. The core component is called *Cubus* as it represents a three dimensional data management structure. Those three dimensions are: system parameters, application requirements, and optimization parameters. Algorithms are classified according to the three aforementioned dimensions. For example a routing algorithms is chosen that can be used in high mobility (system parameter), is reliable (application requirement) and energy efficient (optimization parameter). This way the most appropriate algorithm from the set of available algorithms is chosen according to the various requirements. This requires either that many algorithms of the same type (e.g. routing, time synchronization) are available or different parameterizations of one protocol are classified differently. Since sensors are tiny devices in the sense that they are very limited in terms of computational capabilities and storage only the code for a specific requirement set is installed on a device. If conditions change, code has to be downloaded onto the sensor to adapt to the changes.
As already pointed out the TinyCubus project has a very limited scope as it only targets sensor networks running the TinyOS operating system. The Cubus of the framework can grow extremely large as each dimension can already be very large itself. Defining the parameters of each dimension therefore becomes a very difficult task. For example, the question arises if mobility as a system parameter is enough or if it should rather be low, medium and high mobility or any other subdivision. Then the question arises how to classify if an algorithm is suitable in a particular situation. The authors suggest to do that either via simulation or real-world analysis. But then, the tested scenarios must exactly match to make results comparable and a vast amount of parameters must be tested to fully evaluate an algorithm. And how to choose between two algorithms that expose the same classification also remains an open question. Furthermore, with each added dimension holes in the Cubus might be created and adding dimensions as such after the system is deployed might be difficult. The idea of downloading code might also be a problem in fast changing environments as it is not very efficient to do so to adapt to changes and during installation of the code the network might be non-functional. If an algorithm is changed at a node in the network that might also mean that the code has to be changed network wide as two completely different protocols or algorithms are not necessarily compatible any more and that single node would not be able to communicate any more.
Due to the narrow focus many of the goals of TinyCubus cannot be directly compared to the goals of this thesis. The Tiny Cross Layer Framework represents the general cross-layer idea of inter-layer information exchange but does not seem to be the central component of TinyCubus and is not described in much detail. One goal is slightly overlapping as static parameterization in TinyCubus is indirectly addressed by having a different parameterization of an algorithm to be represented at a different position in the Cubus. The parameterization though is discrete and coarse grained depending on the granularity of the different dimensions.

3.3. Potential Weaknesses of Cross-layer Design

Cross-layer design is one way to meet the challenges of highly dynamic networked environments such as ad hoc networks. Considering the fact that some standard protocols such as TCP do not perform very well in such environments and that many protocols have to be reinvented completely such as routing and time synchronization protocols, introducing cross-layer optimizations seems to be feasible as protocols need to be created or adapted already. But besides feasibility of the introduction of cross-layer optimizations or even a cross-layer architecture the question arises whether cross-layer design exposes certain weaknesses as apposed to the layered approach.

In [49] the general cross-layer principle is analyzed and compared against the layered approach from a higher level view, i.e. in a more abstract way. The key advantage of a layered architecture is argued to be the modular nature and its simplicity which allows for future improvements of the business logic of the network protocols without tempering with the architectural framework itself. In [49] many well designed and in their nature simplistic architectures are presented as examples of good architectural design. What all of these architectures have in common is the impact they had on their respective application domain. For a technological development, good architectural design is the warrant for its long existence and its widespread use and application. On the other hand, architectural short-cuts might lead to significant performance improvements. Compared to the long-lasting impact of a well designed architecture the impact of these performance improvements might be short-termed. Therefore, cross-layer optimizations should always be embedded within or supported by an architectural framework.

The interaction between layers and independent optimization processes which act on information provided by other protocols might yield certain unintended consequences. As the principle of protocol independence and separation does not exist as such any more, at least not as strict as it is in a layered architecture, protocols influence each other. In a worst case scenario adaptation loops can be created between two or more protocols resulting in a dysfunctional or poorly performing network node. To counteract such unintended consequences, the authors of [49] suggest the usage of a dependency graph where each node is a parameter shared between protocols and the edges represent the respective dependency relations. To further improve the stability of the cross-layer protocol stack time scale separation can be introduced where different protocols can control one parameter but under different time scales. Only for closed loops in the dependency graph which act at similar time scales additional analysis is needed to proof system stability.

Finally, the authors of [49] fear that the implementation of cross-layer optimizations might lead to, what they call, spaghetti-like code, i.e. code that is difficult to maintain. Consequences of such code include difficulties to update the network stack, slow development and an increase in cost which ultimately will end up on the consumer side. As a result, the cost and potential instabilities might hinder the widespread adoption of ad hoc networks.

Some of the criticism is addressed by the architectures in section 3.2. Consider MobileMAN for example. The interaction between protocols is handled by the NetSt through well defined interfaces. By only allowing protocols to interact

through this mediator the creation of hard-to-maintain spaghetti-like code should be prevented as modularity is preserved. Another example is the WIDENS architecture. Some adaptation loops can be prevented as the access to protocol parameters is regulated by adjacent layers. This way, if a lower layer is able to achieve satisfactory performance, the parameter is not passed on to higher layers, which in turn might prevent a possible adaptation loop but not necessarily all possible unintended consequences as some examples in [49] demonstrate. On the other hand, ECLAIR would be a negative example as the framework can actively influence protocol behavior. Co-design between protocols and the architecture are therefore necessary making the development of network protocols a more complex and time consuming task in addition to architectural alterations.
The goal of a cross-layer architecture must therefore be that it is simple in its design and modular in nature to guarantee the longevity of any technology implementing such a framework. Furthermore, it should allow identifying or detecting possible unintended consequences and as a result it should not utilize the cross-layer optimizations that affect the overall network performance negatively. And finally, the layer separation principle should be kept intact as much as possible by not directly having interactions between protocols but by providing a level of abstraction where protocols do interact through a mediator. This way implementation independent interfaces to the mediator can be designed which provides control over the cross-layer interactions and allows designing protocols in a clear, modular and structured way.

3.4. Assessment of the Presented Architectures

In this section the architectures presented in section 3.2 are compared against each other concerning selected criteria. Although it is sometimes difficult to directly compare the architectures, the table below is intended to give a rough overview of the complexity, applicability and optimization potential of the architectures presented before. These chosen representative criteria are the basis for the following comparison:

Application generality: Application generality comprises the aspect of the application domain of the architecture, i.e. in what kind of networks and for what kind of scenarios an architecture can be used. Or in other words, it is a measure of how generic an architecture is in terms of its applicability (Excellent/++ → very poor/--).

Functional complexity: The functional complexity comprises the amount of complexity introduced by the architecture that accounts for the cross-layer optimization processes and the amount of cross-layer functionality inside the architecture excluding the data and information side (Very high/-- → very low/++).

Data management complexity: This is a metric for the amount of effort put into holding, sharing, representing, evaluating and accessing data including mechanisms such as XML representation, data abstraction and cost models (Very high/-- → very low/++).

Network optimization potential: The network optimization potential expresses how many network optimizations can be supported by an architecture. This includes for example, local network adaptations and network-wide adaptations (Excellent/++ → very poor/--).

Protocol complexity: Represents a measure for the amount of effort that has to be put into creating, maintaining or updating a protocol for the given architecture and the degree of complexity for interactions with the architecture (Very high/-- → very low/++).

Protocol assumptions and preconditions: Is a measure of the assumptions and preconditions that protocols must or should fulfill to be able to work inside the respective architecture (many/-- → no/0).

The above described criteria are weighed using the following symbols in the table below:

++	: Excellent, very low
+	: Good, low, high
0	: N/a, medium, no
-	: Poor, high, some
--	: Very poor, very high, many

Table 3.1 Assessment of the architectures described in 3.2

Criterion / Architecture	Application generality	Functional complexity	Data management complexity	Network optimization potential	Protocol complexity	Assumptions and preconditions
CATS	-	-	-	+	0	-
ÉCLAIR	++	--	-	+	--	0
GRACE	+	-	--	+	-	-
MobileMan	++	+	0	+	+	-
WIDENS	0	0	-	0	-	-
TinyCubus	--	-	-	0	0	-

3.5. Summary

A vast amount of cross-layer adaptations and optimizations have been proposed in the recent past spanning all layers of popular protocol stacks. Most of these single mechanisms do not consider an architectural framework though. The actual cross-layer interactions are not the primary focus of most publications and are generally assumed to be somehow available. The sheer amount of cross-layer adaptations suggests though that a sound architectural design would not only greatly reduce the design and implementation complexity of new and existing approaches but it would also clearly be a necessity to have a common architectural framework to support these cross-layer optimizations efficiently. The architectures presented all try to support cross-layer protocols in different ways and they also differ in their application scenarios. They also differ in their ability to detect and prevent potential risks that exists when weakening the strict layer separation principle. Therefore, a direct comparison is not always a straight forward process but can be done using criteria that abstract from the architectural details.