4. The CrossTalk Architecture

This chapter introduces the CrossTalk [50] cross-layer architecture and its components. It describes the unique features CrossTalk exposes in conformance with the goals of this thesis such as its mechanism to establish a global view of the network or its way to generate optimization metrics. Where appropriate, CrossTalk's features and components are briefly compared with the architectures presented in section 3.2.

4.1. Motivation & Architectural Overview

Naturally, CrossTalk shares some similarities with the architectures and design thoughts from chapter 3 as the general idea of cross-layer information sharing and adaptation is the basis of all the work presented in this document. When looking at the two tiered CrossTalk architecture consisting of two components in juxtaposition to the protocol stack, as depicted in Fig. 4.1, the protocol stack together with the components left of it look similar to the MobileMAN architecture as found in Fig. 3.2. And in deed, functional similarities exist. The component found on the left part of the protocol stack is called *Local View*. The Local View contains local knowledge about the network and networking conditions provided by the protocol stack as well as other state information obtained from system components such as the operating system. Such information could comprise data about the neighbor degree (number of one-hop neighbors) as it could be provided by the routing protocol, the bit error rate from the MAC layer or the battery status from a hardware driver or the operating system. It therefore represents the local state of the network node and its local view, hence the name, onto the network similar to the function the NetSt has in the MobileMAN architecture. Looking *only* at this part of the architecture without going into the details found in section 4.2 similar optimization and adaptation capabilities of CrossTalk and MobileMAN can be assumed.
Looking at the right side of the network stack in Fig. 4.1 a component called *Global View* can be seen that is one of the unique and novel aspects of this architecture. The Global View contains the same type of information that can be found in the Local View as mentioned before but represents not only the local state but network-wide state information. For the aforementioned neighbor

degree, this would mean that the network wide neighbor degree can be obtained from the Global View which directly translates to the network density. In other words the Global View allows a node to evaluate the network-wide status according to the parameters found in the Local View.

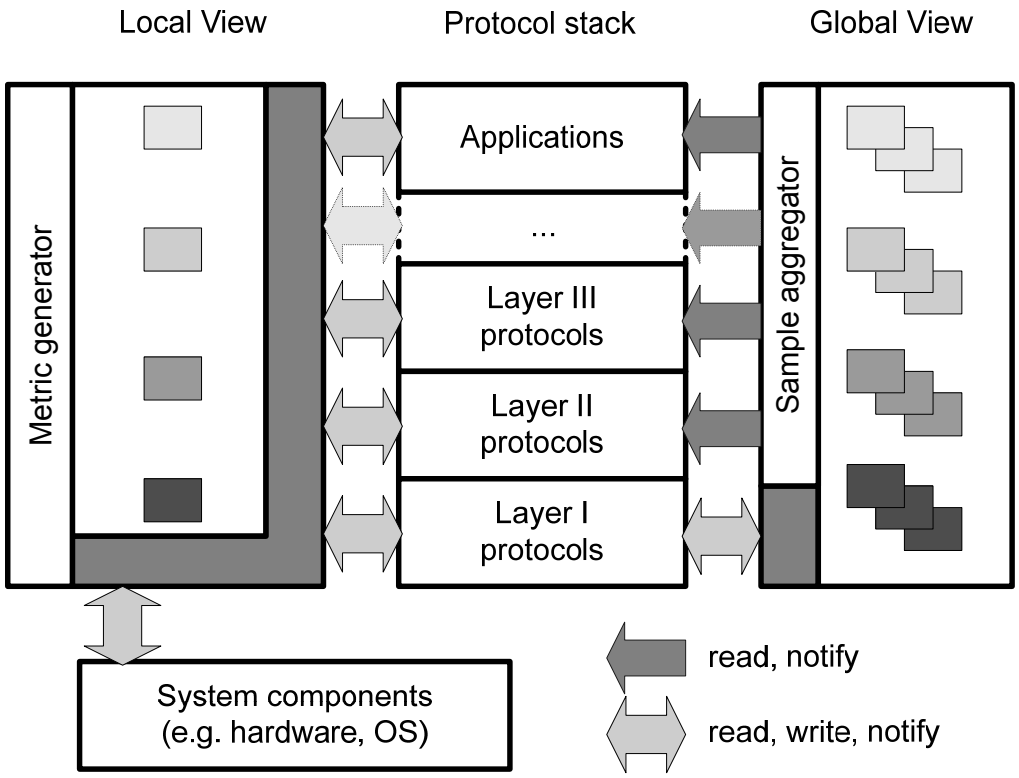

Fig. 4.1 The CrossTalk architecture

The motivation for this two tier architecture, without going into the details of how the two views are constructed, which can be found in later chapters, is quite intuitive. Having a network-wide or global view of the network in addition to having a local view allows a node to evaluate its own status against the average status within the network at any time. This comparison of local and global view yields the relative state of a node, i.e. a node can for example evaluate whether it has more or less energy left than the average node in the network, or if it carries more or less load than the network average and how much it is overloaded or not. Consider the following example. A node knows from its Local View that it runs at 60% of its capacity. This value on its own has only little meaning. The local knowledge the node has in this case does not necessarily allow for optimizations or protocol adaptations. But compared against a global capacity utilization of only 10% for example, the 60% imply that the node is clearly overloaded compared to the rest of the network. That might lead to a higher collision probability at that node and quicker depletion of its batteries, causing routes to break. The knowledge of its relative state allows the node to act accordingly and lower its communication burden. More generally speaking, the idea behind CrossTalk's Global View mechanism is to base local actions of a network node on global knowledge.

Local actions as opposed to global actions are lightweight in terms of resource utilization. The clear disadvantage is, of course, the potential lack of accuracy and efficiency. Global actions on the other hand are very expensive and they are

wasteful in terms of resource utilization such as bandwidth or energy but can achieve network-wide optimal results. Consider a reactive routing protocol and a route request as the global action. A request is flooded through the network to find the destination. Every node in the network might participate in finding the destination node. Ultimately, only a small amount of nodes will, after establishing the route, participate in the forwarding of data packets. But by involving a huge amount of nodes, possibly every node, the route could be found and it is quite likely that an optimal route was found. Local actions do not involve other nodes, making them lightweight. But they lack information beyond the node's scope which ultimately might lead to inefficiencies. For example, if a node is able to increase its performance locally by some means (e.g. by boosting its output power), it might at the same time significantly increase the interference with its neighbors. On a multi-hop path, that might effectively lead to a lower overall performance and also affect other routes in the network increasing the harmful effect of the local action.
CrossTalk's approach ultimately combines the advantages of both approaches to achieve global objectives at a low overall cost by following one basic principle: Act locally considering the global network status [51] ("Act locally, think globally"), this way simple local actions achieve global objectives [52].

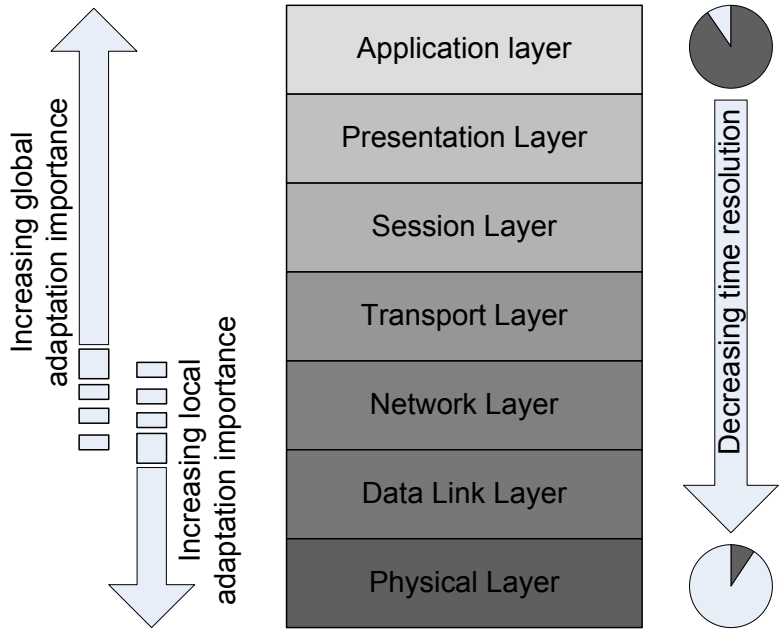

Fig. 4.2 Importance of adaptation type vs. timescale

It is wrong to believe that all optimization and adaptation processes should be based on a comparison between information from the Local View and the Global View. Some adaptations should solely be based on the local view. The reason for this are the different timescales adaptations are based on. Some phenomena vary based on timescales that are only within the range of seconds and below. This is especially true for lower layer system dynamics such as the time-varying channel characteristics. The establishment of a global view of the network must therefore be much faster than any significant variations of the phenomena. Considering this example, a global adaptation scheme is unable to compensate for the fast changing channel conditions. But following the protocol stack up to higher layers the importance of global adaptations and optimizations increases. Such global

optimizations could include load balancing at the network layer which could be in the range of several minutes to decisions at the application layer about where to place objects and services which could have a timescale of many hours. In general, the longer lasting the impact of an optimization should be the more likely a global optimization process is suitable to fulfill the objective. Due to the nature of the conventional protocol stack the importance of global adaptations grows from lower to upper layer protocols as shown in Fig. 4.2.

4.2. Local View

As already mentioned in the previous section, the local view contains information provided by the protocols of the networks stack, applications or system components, therefore representing the local state of a network node. This chapter describes how this information and what kind of information can and should be stored inside the Local View. The interactions between the involved components are explained and the optimization metric generation, a general goal as stated in section 1.2, is discussed in detail.
Recalling one of the factors that, according to the authors of [49], made some popular architectures so successful and helped them to remain the blueprint for modern systems was their simplicity. A cross-layer architecture should be no exception to this rule, especially if such an architecture is intended to be used for a broad spectrum of possible scenarios. A very complex system for example would most probably not be used in sensor networks as only light-weight systems are applicable in such environments. Furthermore, the more functionality is placed inside the architecture, the more it looses its generic character.
The Local View component has therefore only two very basic functionalities:

- **Data Management**: Structured, controlled access to the information provided by the applications, system components and protocols.

- **Metric Generation**: Generation of more complex optimization metrics from very basic information provided by system components and network protocols.

4.2.1. Data Management

The data management functionality can be regarded as the core component of every cross-layer architecture since it is the enabling entity for the information exchange which is the basis of every cross-layer adaptation and optimization. Representing data in a fashion that is highly generic and extendible, and very expressive and easily accessible at the same time is difficult to achieve. For example the authors of [45] suggest to develop and to use a specification language that describes the data types that components provide and require. They have not designed the language yet, but their goal seems to be, by using a specification language, that the data management framework remains highly flexible, even if new protocols are added that come along with new types of information, represented in arbitrary data types. Such an approach has several drawbacks that have to be considered. Using a language to describe stored parameters, values, objects and data structures has the disadvantage that

descriptions and queries to the data management entity have to be processed in a rather complex form. For example XML descriptions would need to be parsed, requiring not only a parser for the language, possibly part of the architecture, making it more complex, but every interaction would consume significantly more processing power and storage. On powerful network nodes that might not impose a challenge but on low power devices such as sensor nodes, cell phones or PDAs such additional overhead is certainly not negligible. The MobileMAN group would also like to base their data management component, the NetSt, on more complex mechanisms such as introspection, XML descriptions and profiles [37]. They note though that it is important that the data management entity remains responsive at all times to guarantee an appropriate level of real-time behavior. Since they make use of call-backs in their design, which influences protocol performance in a non-deterministic way they suggest pre-fetching and caching mechanisms, further complicating the overall design.

The reason to allow network protocols to store arbitrary data inside the data management entity and at the same time let arbitrary protocols allow to access and use it seems very desirable at first. This would allow for protocols to use new types of data once they are provided by a component and protocols could interpret them by means of a specification language and/or introspection or some other technique. But thinking about real-world systems the question arises whether such powerful mechanisms are really suitable and necessary. We assume that this is not the case for several reasons.

First of all, it is highly unlikely that there is an arbitrary amount of parameters and metrics that make sense to be used for optimizations and adaptations which would justify such complexity. Instead there exist a small number of parameters that can be used or somehow transformed to be used for a broad spectrum of different adaptations at different layers of the protocol stack. The frequent reoccurrence of parameters such as the signal strength in the literature suggest that the most basic parameters seem to work extremely well for many purposes. Even if the simple signal strength is not used as such, it is used as the basis for the generation of probably more expressive metrics (compare section 4.2.2). The most significant and important parameters that are needed are the ones that quantify the effect which protocols have to adapt to. This are either more traditional parameters such as load to solve conventional network issues such as load balancing and QoS or parameters that reflect the unique characteristics of ad hoc networks such as a metric for the degree of node mobility. Some information can be placed in predefined data containers such as routing state information. Most routing protocols for example maintain overlapping information anyway. The way most protocols differ is the routing strategy not the routing state information. A routing state data container should hold information such as the address, the distance in hops, and the next hop towards that destination. If a protocol cannot provide such data, it will simply not fill out these parts of the data container.

Another reason why complex descriptions and data type query mechanisms most probably are not necessary is the fact that old protocols, even if they are able to interpret new information types correctly, will algorithmically not be able to apply that information for adaptation processes. In other words, if new parameters are available and older protocols should use that information they have to be updated or redesigned. In such a case, automatic parameter

interpretation is not useful as the protocol designer needs to change to protocol accordingly anyway. It should be mentioned here that with the experience gathered during the application of the CrossTalk framework it often appeared that new metrics added to the Local View and used for network-wide adaptations using the Global View mechanism described later, had its origin in the same layer or protocol as it was used for adaptations later on. In such a case interpretation is of no concern as the generating protocol will not need to interpret data it can provide itself. This of course holds only true for certain global adaptations as it can not be considered cross-layer communication locally on a network node. Examples of such cases can be found in section 5.1 and 5.2 with the theoretical background found in section 4.3.
Furthermore, there will not be an arbitrary amount of protocols that need to be highly adaptable. As on the Internet some very basic protocols will be used more than 90% of the time for a certain purpose. Take transport protocols as an example. Mostly TCP and UDP are used as a transport mechanism on the Internet. Parameters such protocols can provide should be included as predefined parameters inside the Local View. Once they are included, other protocols will very likely be able to provide the same parameters and other protocols will rely on their presence to carry out optimizations.
Finally, history has shown that usually only a small subset of architectural functionality is used in real-world systems. A good example for this is the ISO/OSI layered protocol stack (compare Fig. 1.2). The intended seven layers are usually not found in real world systems such as the TCP/IP stack where session and presentation layer for example are missing.

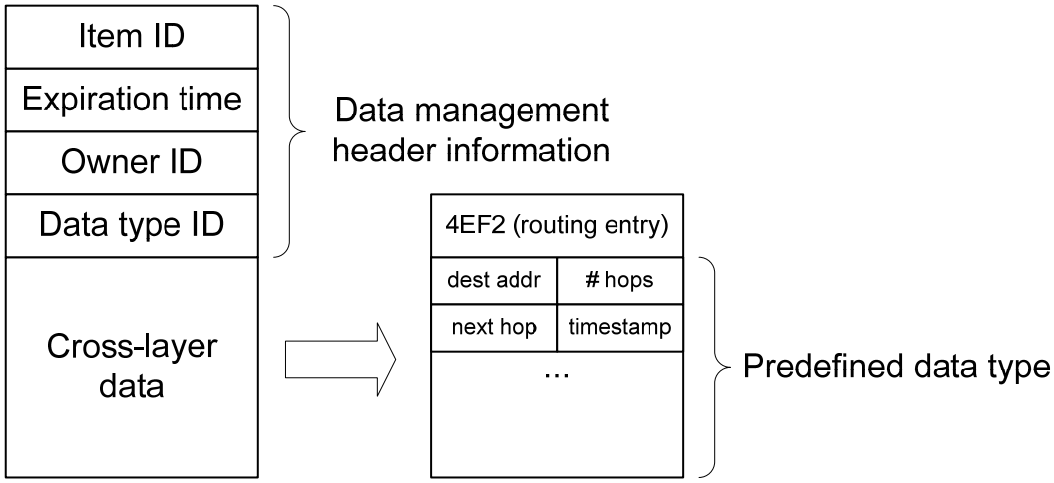

Fig. 4.3 Local View data container

As a conclusion, it can be said that the Local View should mainly hold predefined data types. This way interpretation is unnecessary and the complexity of the overall architecture is greatly reduced. Still it should be possible to store arbitrary objects inside the Local View. The reason for this is that the functionality the Local View offers, as described later in this chapter, can be used for software reuse reason and for certain global adaptations as further explained in section 4.3 and demonstrated in section 5.1 and 5.2.
Now that the general functionality is explained let's look at how the data management is realized and at the interactions with the Local View in more detail. Fig. 4.3 displays the structure of a data container that is used by the Local

View. The protocol information, in whatever form it is represented, is the content of the container displayed at the bottom of Fig. 4.3. Here exemplary a routing state data element is displayed just next to it on the right side of the figure. For managing the data inside the Local View four additional pieces of information are required. First of all there is the unique item ID. This is generated using a hash function provided by CrossTalk. A protocol has to know a unique set of information for each entry. In this example the destination address of the routing entry could be used as input for the hash function and the resulting hash value will uniquely identify this data entry amongst all entries of the same type. Using this unique ID and a set of hash functions a bloom filter can for example be used to check the existence of data elements. The second piece of information is an expiration time given by the protocol or system component. Once the data is stale, out of date and therefore useless it will be removed from the Local View. Additionally there is the owner ID. This ID is assigned to a protocol or component once it registers with the CrossTalk architecture. Using this ID all interactions with CrossTalk are carried out. Using the Owner ID, access rights can be defined. Finally, the data container header contains a data type ID. This ID simply specifies what kind of data the specific container is holding. All header information can be used to query the Local View. For example, the data type ID can be used to retrieve all routing entries.

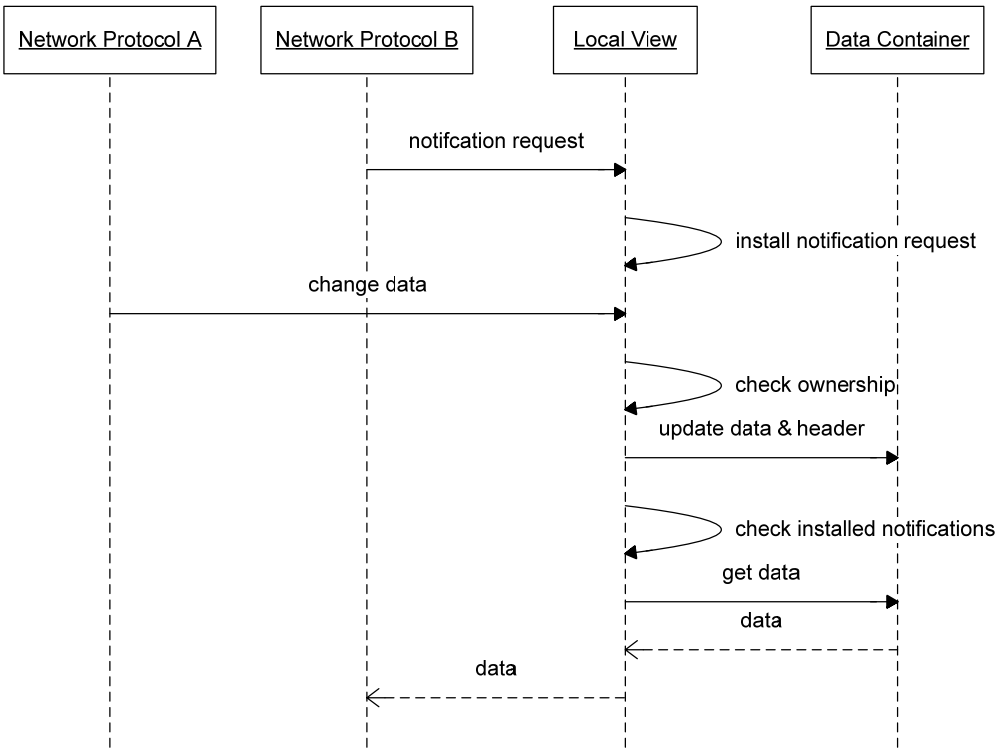

Fig. 4.4 Interactions between the involved components for a notification

Other than simple read and write operations, the local view offers one more interaction which is notification. Sometimes, it is desirable for a protocol to adapt only when certain thresholds are violated. For example, consider a power control algorithm that is intended to keep the output power of the radio at a minimum to locally save some energy. On the other hand it should not reduce the output power too much as another important aspect is network connectivity. The algorithm could work in a way that it increases its output power as soon as the

amount of neighbors in its direct radio range decreases below a certain number or decreases it if it grows beyond a certain threshold. The protocol would need to subscribe to the Local View with a query that if the amount of entries of data type 4EF2 (routing entry as in Fig. 4.3) with # hops equal to one is smaller than the threshold, it has to be notified. The amount of notifications should be kept to an absolute minimum as each change to an observed data type would trigger a threshold violation check inside the Local View. The flow of interactions for the case of a notification is illustrated in Fig. 4.4. Protocol B subscribes for a notification on some data provided by protocol A. After A changes the data, the thresholds of all to be considered registered notifications are checked and any violations are reported to the respective protocols, in this case protocol B (for more selected interaction flows please refer to appendix C).
All possible interactions with the Local View component are depicted in Fig. 4.5. *A* is a data container inside the local view. *A* in the diagram represents data provided by an application, a system component or a network protocol. The owner can write and overwrite *A,* whereas A can be read by any component that has access to the Local View including the metric generator. The same applies to notifications concerning *A* as displayed by interaction 1 in Fig. 4.5. The other interactions will be further discussed in section 4.2.2.

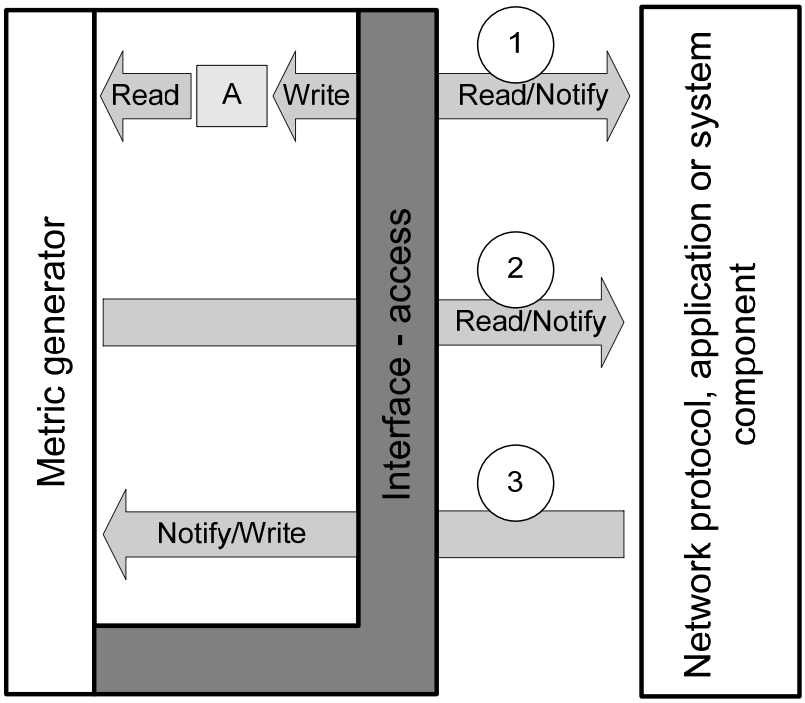

Fig. 4.5 Local View interactions

The way the Local View is designed and the nature of some of the information stored in it imposes certain requirements on adaptive network protocols running inside the CrossTalk architecture. Consider a protocol that utilizes an adaptation mechanism based on position information provided by a GPS receiver. Upon entering a building or tunnel for example that information might not be available any more. At some time the stored GPS coordinates will expire and the Local View will purge them. At this point the adaptation mechanism will not work any more. Or consider a routing protocol in use that does not provide certain information such as the next hop which another protocol needs for some

optimization. The fact that certain information is permanently or temporarily not available should not render a protocol dysfunctional. In other words, when designing a protocol only “layered” information flow should be assumed for the basic functionality. That means that without the presence of any additional cross-layer functionality the protocol must be able to provide its basic service. Only mechanisms to improve the performance of a protocol should be based on the information provided by CrossTalk. The absence of that information should therefore only result in a worse performance. This also guarantees that protocols not using the cross-layer framework, e.g. purely layered protocols, still function and are interoperable with new, cross-layer optimized protocols. The only effect these layered protocols have is that they on the one hand do not provide any information to the Local View which could be used by other protocols and they do not optimize their own algorithms by utilizing cross-layer information on the other hand.

4.2.2. Metric Generation

Cross-layer design is a very metric centric approach. The most fundamental part is to provide metrics to base decisions on. An important feature is to have metrics that are easy to generate, which virtually every network system should be able to provide. This way those metrics can be assumed to be present for adaptations and optimizations. Such a set of metrics should be provided by the architecture to guarantee uniformity, eliminating interpretation ambiguity. This set of metrics ideally covers large parts of possible protocol adaptations together with some fundamental and very basic metrics provided by protocols and system components such as the received signal strength, bit error rate and others.
The metric generator as depicted in Fig. 4.1 and Fig. 4.5 is another unique feature of the CrossTalk architecture. The MobileMAN architecture for example does explicitly not generate any data as stated in [37]. We have several reasons to do so.
When looking at the data container shown in Fig. 4.3 it becomes clear that there will be many single data containers that may have some form of relationship. The single routing entries for example might together reveal some information about the topology such as the network diameter. If there is no topology data type then this information would need to be generated and instead of having each protocol in need of that information to calculate that data, this functionality is placed at a central location, the metric generator. In addition, if this data is not required at all, the generator will simply never calculate it. This way, in comparison if this more complex data would be part of a data container, processing power is saved. Generating this kind of information inside of CrossTalk and not inside the protocols has also the advantage that the protocols remain more responsive as they can focus on the core service they provide.
Another reason for having an additional component that generates more complex optimization metrics is that some metrics have to be observed over time whereas the Local View only reflects the exact state of a node at a point in time. For example by observing the network density over time, a protocol could conclude whether the node is moving into a more dense area of the network or in a sparser one. A hybrid routing protocol could use that information to change its routing strategy. Furthermore, some pieces of information are most likely useless from

an adaptation standpoint when only being observed at a certain time instant. Take for example the information that a packet was just sent or received. That can hardly be used for any kind of optimization. But observed over a certain time period, the simple notification about incoming and outgoing packets without considering destination or other header information can be condensed to a load metric [53], for example. The same could also be done by a protocol registering for a notification but as already pointed out notifications are expensive in terms of processing overhead and the generation would not be done any more in a central location. Together, these are the responsibilities of the metric generator.
The interactions in Fig. 4.5 marked 2 and 3 refer to interactions with the metric generator. Interaction 3 illustrates the way external components such as network protocols or the operating system can feed data or simple notifications to the metric generator directly. An example for such a notification could be that a packet was just sent, which the metric generator will use for the previously mentioned load metric calculation. Interaction 2 depicts how external components can access the data produced inside the metric generator. Instead of placing it inside the local view it can only be accessed directly. This way the necessary data is collected constantly but the metric itself is only generated on demand, saving processing power.

4.3. Global View

When looking at the CrossTalk architecture and comparing it with other architectures, its most powerful and unique feature is the Global View. It allows a network node to evaluate its local state against the network-wide state according to the metrics the Local View provides. This comparison enables a node to change protocol behavior and parameterization to achieve global objectives such as increased overall network lifetime, load balancing and more.
This section describes how data is disseminated by the architecture to distribute the locally collected or generated data to allow each and every node in the network to estimate the network-wide status. It further discusses the mechanisms and algorithms that actually compute the network-wide value of a disseminated metric.

4.3.1. The Data Dissemination Process

The only way to get an exact network-wide view of some parameter or metric is to look at each node in the network, collect the data for that metric from all nodes and then aggregate that nodal information in some way. In other words each node would need to inform every other node about its locally measured or calculated metric so that every node can calculate the network-wide average. One way to do this would be to let every node broadcast that information periodically. This way each node would receive the necessary data to compute a global view. The problem is that this approach puts a lot of stress on the network as, depending on the interval, a large quantity of the available bandwidth is consumed by data dissemination packets. Choosing the dissemination period too small will save bandwidth but at the same time the global view will loose some accuracy. Actively broadcasting the information has also the disadvantage that some nodes might spend some energy and effort on data dissemination whereas

its optimization goal might be energy conservation. Therefore, the CrossTalk data dissemination procedure is not an active one, meaning that no control or data dissemination packets are actively sent by the framework.
CrossTalk leverages packets that are sent for any kind of purpose and enriches these packets with information from the Local View by piggybacking the information onto those packets. For example, onto outgoing application or routing packets the neighbor degree can be piggybacked to let the nodes in the network be able to estimate the node density of the network. Every CrossTalk node receiving a packet, either because it is the intended destination or next hop or it simply overhears the packet by being in radio range, extracts that information and adds it as a sample to its Global View. This way, numerous samples are collected at every node in the network. Special attention should be directed to overhearing messages. The wireless medium itself is one of the prime reasons for the dynamic nature, the various performance bottlenecks and networking challenges found in ad hoc networks. On the other hand it allows doing certain things which cannot be done in infrastructure-based wired networks. One of these things is overhearing messages that are not explicitly addressed to a network node. The broadcast nature of the wireless medium, on the one hand imposing challenges, can and should be exploited as it also opens up opportunities. CrossTalk exploits it by more efficiently disseminating the data from the Local View as message overhearing expands the number of recipients.
Fig. 4.6 illustrates the data dissemination procedure. Node A represents a node on which an application is generating data to be sent and node B being a forwarding node. As can be seen, as intermediate nodes only utilize the lower layers to perform the forwarding task, the data dissemination module has to be placed at the lower layers of the protocol stack. At node A in the figure, the module selects data to be disseminated from the Local View and piggybacks it onto a packet. This data is extracted at node B where the data dissemination module adds it as a sample to the B's Global View.

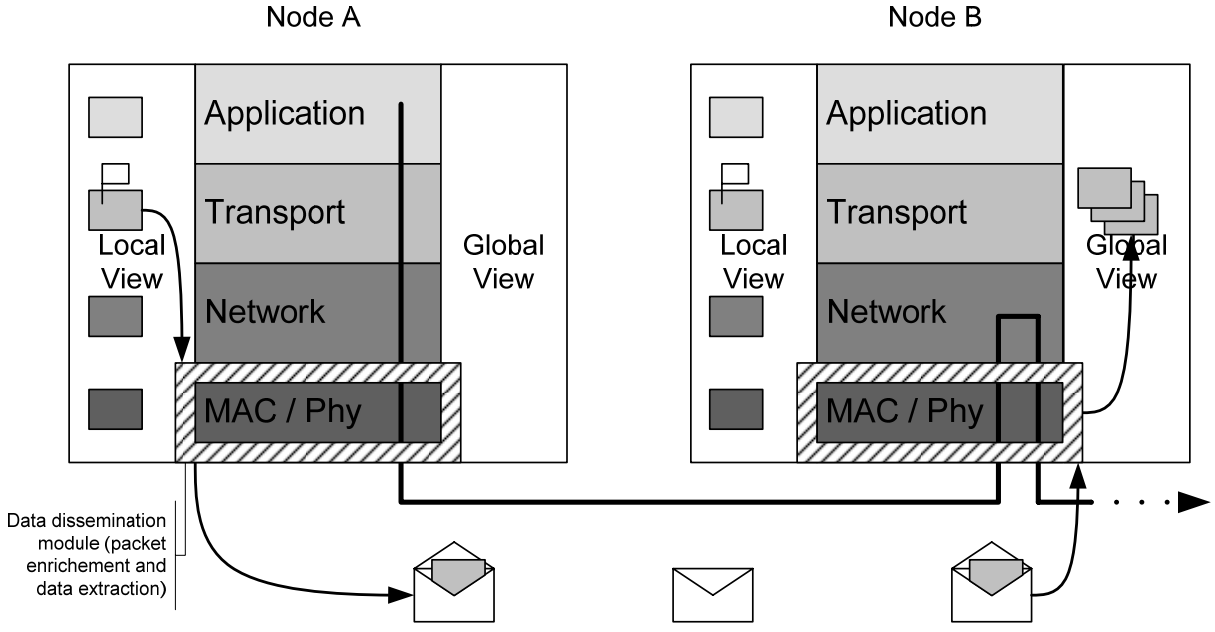

Fig. 4.6 CrossTalk's data dissemination procedure

By not generating data dissemination or control packets, the overall overhead of the dissemination procedure is minimized. Additionally, only the source of a packet is adding information, i.e. only the node that generates the packet is adding information form its Local View. Forwarding nodes only extract the origin's local information and do not add any additional overhead. This way, the packet sizes are only marginally increased, keeping the overall overhead low and making fragmentation due to increased packet sizes unlikely. When adding a sample to the Global View, additional information can augment the collected samples. Besides giving it a timestamp, for example distance information can be added if available. As an example, the topological distance in hops could be extracted from a packet header. The timestamp is used to remove stale samples form the Global View. Each information type is assigned a useful lifetime. Once this lifetime expires the sample is purged from the Global View.
By piggybacking the information the overall footprint of the system is kept low for two reasons. First of all the information that is propagated into the network is most likely a simple low precision number and no complex data structures which only increases the packet size marginally. It can be compared to adding an additional small header to the packet. Complex data structures exist in the Local View but they most likely will be less suitable for network-wide adaptations. Think of the individual routing entries, which only would be useful as a whole and in case they are really needed they should rather be obtained by the routing protocol itself, i.e. by a proactive routing protocol. The second reason is, that by using packets that have to be sent anyway the whole additional data for an individual data dissemination packet, such as the various headers is not only saved but additionally the whole additional medium access contention problem is avoided, which can potentially result in collisions and retransmissions.
The overall communication overhead can further be reduced by employing a stateful approach. So far the data dissemination method described is purely stateless as the dissemination module keeps no track of when, where or what was piggybacked onto outgoing packets. Introducing a state is a trade-off between spending memory and little extra computation time versus spending energy on additional bits that have to be transmitted. Since communication is, in terms of energy consumption, far more expensive than computation [55], the stateful approach helps conserving battery power. Depending on the device, the one or the other strategy might be favorable or perhaps not feasible as a sensor node might simply not have enough storage available for additional state information. The stateful approach saves energy and reduces the communication overhead by not adding Local View information to every outgoing packet. To reduce the amount of messages that have to be enriched, a dissemination state is added that basically is a small table which has four columns that every node maintains. One column contains the destination address of the enriched packets. The second column contains the type of data that has been disseminated and is only needed if more than one optimization metric is disseminated at all. Form a database perspective those two columns together can be regarded as the primary key as they can be used to uniquely identify an entry. In case only one metric is disseminated the destination address alone serves this purpose. The third column contains the value that was disseminated and the final one holds a timestamp of the time the packet was sent.

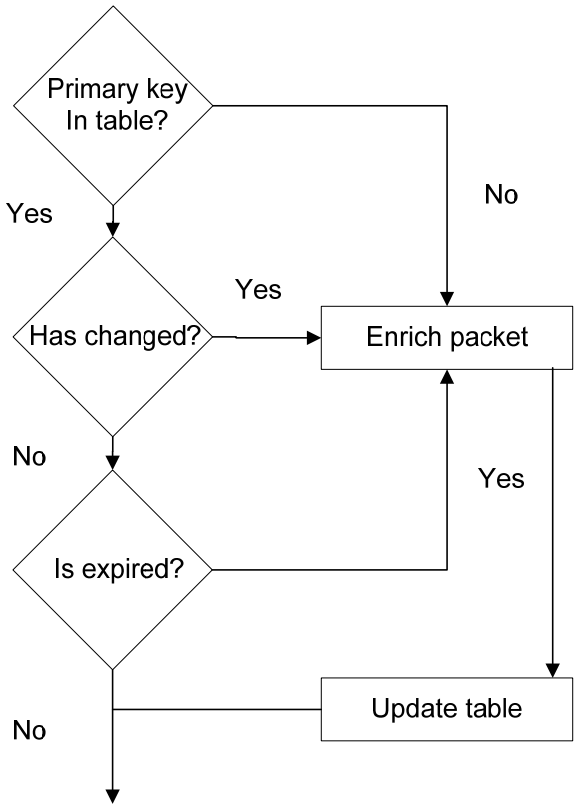

Fig. 4.7 Stateful dissemination algorithm

The stateful dissemination algorithm is illustrated in Fig. 4.7. Whenever a node is about to send a packet and is potentially able to enrich it, it first consults the dissemination state table. If there is no match for the primary key in the state table, i.e. the destination plus the type of data that is about to be piggybacked onto the packet, the packet is enriched with the information from the Local View. After a packet has been enriched of which the destination was not found in the table, it is updated according to a simple rule. Either the oldest entry in the table is purged, or if an empty row is available, that one is used. The absence from the dissemination state table implies that either the destination has never been contacted before or has been contacted a relatively long time ago. Therefore, in either case, the data should be disseminated as it will reach new nodes or refresh old Global View data samples on the path towards the destination.

If the primary key is found in the dissemination state table, it is checked whether the value has changed significantly. What "significantly" means here is very application dependant, i.e. it is dependant on the required precision of the Global View that is constructed from that data. If the value has changed beyond a certain threshold, the packet is enriched to update the new value at the nodes receiving the packet.

Finally, if there is an entry for the destination and the value has not changed beyond the threshold, the timestamp is checked. If one third of the useful lifetime of the local information has passed, the local information has to be disseminated again. This procedure makes sure that Global View entries in other nodes are refreshed before they are purged. Only if all conditions are met, will the packet not be enriched with local information.

As a special case, n rows are reserved for broadcast packets, with n being equal to the number of data types that have to be disseminated which is equal to the

number of possible primary keys for the broadcast address. Their state should never be purged from the dissemination table since broadcasts are the most expensive type of operation form a communication standpoint. With giving broadcasts persistent rows in the dissemination table network flooding is guaranteed to be most resource conserving when it comes to the data dissemination process.
There are two major advantages of the stateful approach. One, as already mentioned is of course that less total information is propagated using a dissemination state. Another advantage is that nodes which solely act as relays without generating messages on their own can now use non-enriched packets to disseminate their local information, this way contributing to the global knowledge.
Criticism to the overall dissemination procedure might be that there is no immediate control over the data dissemination process. If not many packets are sent not many data is disseminated resulting in a deteriorating Global View. The answer to that criticism would be that the timestamps on the samples collected will prevent the Global View from severe deterioration as it is used to purge old samples. If there are no samples or only a few inside the Global View no network-wide estimate will be produced, forcing the protocol to run in purely "layered" mode. Additionally, the fewer packets are sent the less potential there is for network-wide optimizations and in very lightly loaded networks global adaptations will not be necessary and local adaptations based on the Local View solely will suffice. An active dissemination technique would in such a case only waste resources whereas our approach would not. In other words, the more potential and need there is for adaptations and optimizations, i.e. the more traffic the network produces the more information is automatically disseminated by the CrossTalk framework.
Not all data elements in the local view should be constantly disseminated. A network protocol needs to flag the data type that should be disseminated as shown in Fig. 4.6. If a protocol is laid out do load balancing using the Global View then it would most likely only require the load metric to be disseminated. It on the other hand might be able to add much more to the Local View which can further be used for local adaptations. The amount of different metrics disseminated should be reduced to only a few. Depending on what the network should be optimized towards this preference can be set for example at deployment time. Imagine a sensor network, the probably most important optimization in such a network is the overall network lifetime, i.e. energy conservation which can be set at deployment time, as this requirement will most likely not change during the lifetime of the network.
The dissemination module itself could be realized with a mechanism similar to Libmac as implemented on the ORBIT testbed. Libmac allows appending measurements on a per-frame granularity level at the MAC layer.

4.3.2. The Global View Calculation

The previous section had a focus on how the data to generate a network-wide view of a certain metric is disseminated whereas this section deals with the actual computation of such a global view. Clearly, with the previously described approach to disseminate our data there will never be a 100% correct network-

wide view on each node in the network and that is not what CrossTalk is aiming at. Even with an active data dissemination technique, depending on the dynamic character of the metric itself, the structure of the network and the dissemination period, establishing a 100% correct view might realistically not be possible. What we want is a reasonably correct and up-to-date view within certain bounds that will still be accurate enough to let a node evaluate its relative state in the network.
The key in establishing the global view besides disseminating the data is the way the collected samples are aggregated. Therefore, part of the Global View is a collection of algorithms to compute the network-wide view from the collected samples which reside in the sample aggregator as illustrated in Fig. 4.1. Intuitively, the simple mean value of the samples would be an appropriate way of estimation of the network-wide view but due to the special nature of ad hoc networks and probably due to relatively fast changing metrics other algorithms can be more beneficial.
But before going into the details of the specific algorithms provided, let us look at how the Global View can be seen in a more abstract way. This way, its overall value and significance can be more easily evaluated. More specifically, let us look at the information flow provided by the various models that have been proposed so far.

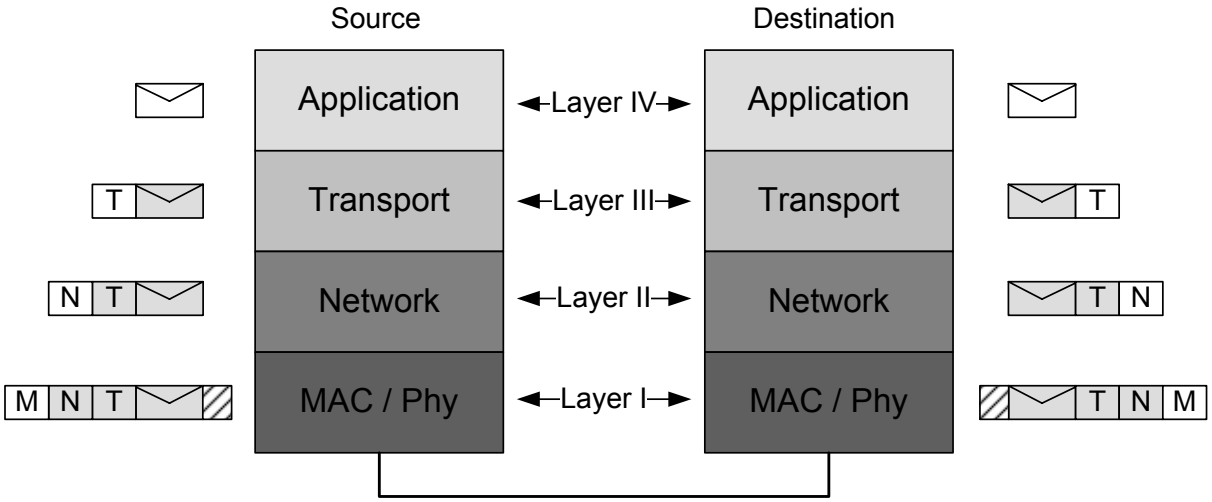

Fig. 4.8 Information flow in a layered architecture

When going back to the layered approach the information flow is very easily describable. The layered approach allows for a peer-layer information exchange and nothing more when only looking at the flow of information exchanged between layers. What is meant by peer-layer information exchange is depicted in Fig. 4.8. Every protocol of each layer adds some header and/or trailer information. The information is ignored locally at a node as it is simply treated as payload by lower layers. A peer-layer protocol, i.e. the exact same protocol on a remote node, will interpret the header and/or trailer information added by its remote counterpart and nothing else. That is the sole information flow that a layered architecture offers.
The cross-layer architectures developed so far add an additional information flow as protocols now can also locally share information more or less freely depending on the architecture. This information flow was already illustrated as for example

in Fig. 3.2 or Fig. 3.3 for some presented architectures. In Fig. 4.9 all models are compared directly in one single diagram. The layered approach is shown by the information flow arrows in the middle of Fig. 4.9. Furthermore, it illustrates the up-to-now proposed cross-layer information flow which can be found on the left side, labeled intra-node cross-layer communication as it all takes place locally on a network node.

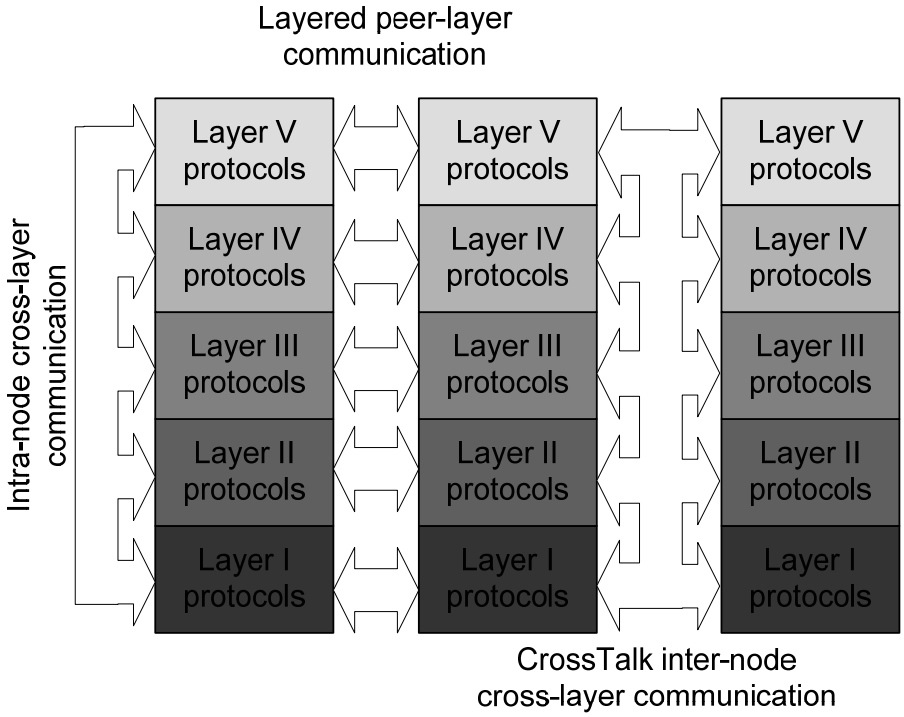

Fig. 4.9 Information flow in the various models

The CrossTalk architecture preserves these two kinds of information flow. The layered approach by its protocols directly and the intra-node cross-layer communication by means of the Local View only. In addition, CrossTalk goes one step further and introduces a new information flow by means of the Global View depicted on the right side of Fig. 4.9. This type of cross-layer communication is termed inter-node cross-layer communication. By disseminating data between nodes which is then shared amongst all layers on the remote nodes is in a way a combination of the two previously mentioned information flows. It provides a node with information about the local state of individual remote nodes and additionally enables a node to estimate network conditions on a global, i.e. network-wide scale. In other words, the Global View enables remote cross-layer communication.

The algorithms that compute the global view are all of one general type, namely weighted moving averages:

$$v_{global} = \frac{\sum_{i=1}^{i=n} w_i s_i}{\sum_{i=1}^{i=n} w_i}$$

The above formula denotes the general structure of a weighted moving average to calculate the global view v_{global} of a metric s, where n is the number of samples inside the Global View and w is a weighting factor based on some dimension such as time or distance. Therefore, s_i are the samples collected inside the Global View for that metric and w_i are the corresponding weights applied to s_i. It has to be noted that for a constant weight $c=w_i$ the simple mean value of the samples is calculated.
The difference between the available formulas inside the Global View is therefore only based on the calculation of the weighting factor. As already mentioned by keeping the factor a constant the mean of the samples, collected through our data dissemination process is calculated. The other weighting functions need some more explanation as they follow different intuitions.

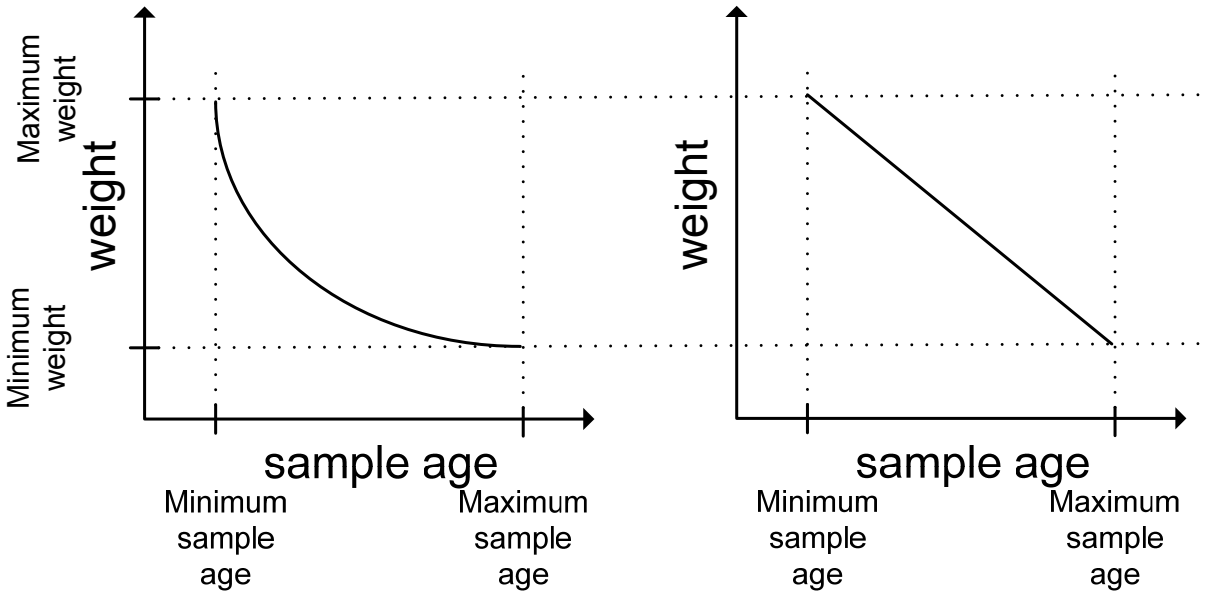

Fig. 4.10 Time based weighting functions

The first concern when calculating a value representing a global view from samples collected during a period of time is the question of accuracy as the disseminated values might already have changed without noticing. One mechanism takes already partially care of this. Upon entry into the Global View the samples are time stamped. After expiration of their useful lifetime they are purged from the Global View and will therefore not influence the calculation any more. But there are phenomena that could rapidly change such as the mobility in a network. Imagine a sensor network which monitors a geographic region for a phenomenon that only rarely occurs such as fire. On the observation of the phenomena, in our example the outbreak of a fire, the network has to react and perform its task. Before the outbreak the network was probably only very lightly loaded by some control traffic now on the other hand many network nodes might suddenly create data traffic to notify the data sink, query other sensor node to see if the sensed data is accurate or to calculate the extend of the fire. In such a situation the disseminated load information obtained during the period prior to the fire might still influence the global view calculation until these samples expire. In other words there is a certain timeframe in which the global view of the network might be less accurate. Therefore, the generation of a global view for these fast changing metrics should account for such possible significant

fluctuations. The key lies in the calculation of the weighting factor for the weighted moving average. For this purpose two functions are provided as depicted in Fig. 4.10, one giving proportionally more weight to recent samples and one favoring recent samples even more by giving them exponentially more weight, the latter one adapting to sudden changes of the observed metric even faster.

The second type of weighing function targets the peculiarities of ad hoc networks that exist due to their nature. The metric that is the basis for this type of function is the topological distance in hops. This information as stated before can be added to augment the samples from the Global View. The first function simply gives samples from far away nodes proportionally more weight than samples from nodes close by. The intuition behind this is very simple and has its roots in the fact that there are certain parameters which have a dependency based on the geographical and topological proximity of nodes. Nodes that are very close most likely will contribute more samples to the Global View than nodes that are far away. Therefore nodes in the close vicinity of each other can potentially dominate each others Global View. As an additional effect, some metrics will locally be similar amongst nodes due to the nature of ad hoc networks. For example the number of one-hop neighbors will very likely be similar amongst nodes in direct transmission range as they cover largely overlapping regions of the network topology, otherwise they would not be able to communicate with each other. These two effects can be to some degree alleviated using this function that linearly increases with the distance as shown on the right side of Fig. 4.11. For all functions available, samples from direct physical neighbors can be completely excluded from the Global View calculation as they are a special case in that they can always be overheard and due to the other reasons stated before.

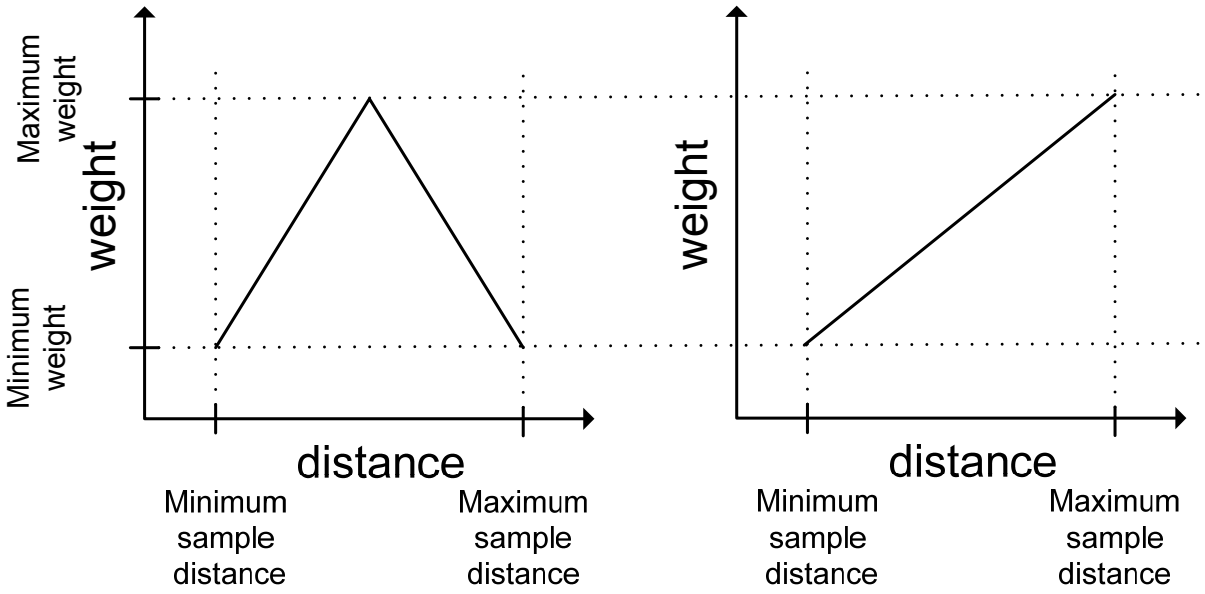

Fig. 4.11 Distance based weighting functions

There is another effect in ad hoc networks which is also due to the topological and geographical dimensions. As the network has geographical bounds formed by the transmission ranges of the nodes the border of the network is sparser than its core. Furthermore, the core of the network most likely will be under a higher load especially when using shortest path routing as not only more nodes are present

at the core but also most routes originating at the borders will lead into or through the core [56]. We will from now on refer to these effects as border effects as displayed in Fig. 4.12. If a metric is prone to these border effects and the Global View should instead rather reflect the situation inside the core of the network a triangular function is provided as illustrated on the left side of Fig. 4.11. The samples from nodes furthest away most likely have their origin at the border of the network and are therefore weighted less. Samples from nodes close by are also weighted less as they might yield similar values as found locally. The highest weight is assigned to nodes in a medium distance between the local node and the suspected border nodes most likely being at or close to the core of the network.

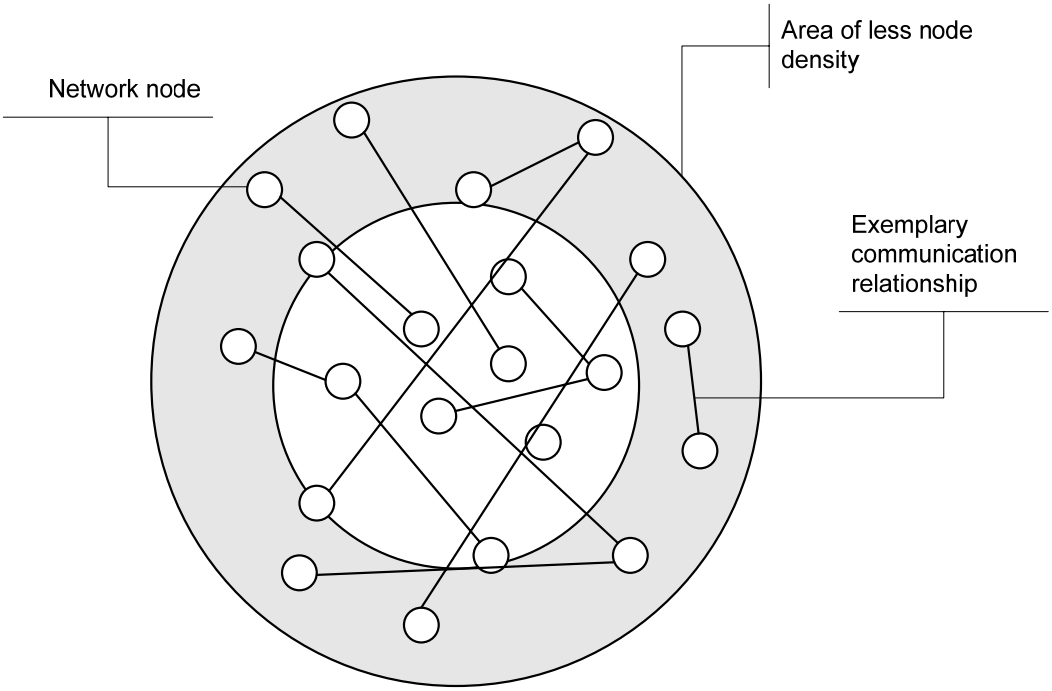

Fig. 4.12 Ad hoc network border effects

So far the algorithmic aspects of the global view computation have been described. As a second and very important concern there is the storage and data management aspect of the Global View. Collecting samples from throughout the network and keeping them for some time might not always be possible for all classes of devices and the question arises if that even is necessary. If storage becomes a concern then a suitable replacement strategy has to be found in case the Global View cache reached its capacity limit and new samples arrive. Different cache replacement strategies were implemented for this case. Probably the most intuitive strategy is to replace the oldest entry in the Global View, which corresponds to the simple first-in-first-out (FIFO) principle. Furthermore, other cache replacement strategies are available inside the Global View (for the experimental evaluation compare section 5.4) that do not consider the age of an entry, unless they are expired, but rather the value itself. One strategy replaces the biggest value, another one the smallest and a third one the sample that differs the most whether it is the biggest or smallest. The intuition behind these strategies is that the most extreme values inside the Global View will influence the computation the most. This can potentially lead to a highly incorrect network-wide view especially with very small cache sizes. The last two

replacement strategies consider only the distance of the collected samples. If spatial dependencies exist, these algorithms can be able to reduce such effects. A spatial dependency as already described can be due to border effects or due to the close proximity of nodes. The two corresponding replacement strategies replace the most distant or the closest data sample in case the Global View cache reached its maximum capacity.
In addition to providing the ability to construct a network-wide view, the Global View component can also be queried for the information regarding only a single sample and generally can be accessed in the same way as the Local View.

4.4. Node Bootstrap and Protocol Signaling

As the previous chapter described, the establishment of a global view of the network requires a node to collect a certain amount of data samples before a node is able to calculate a meaningful and reasonably correct view. When a node enters a network by switching it on, by moving into a networked environment or by roaming between two separate networks the Global View component will not be able to provide protocols with a network-wide view for optimizations till enough samples are collected. Therefore, a simple initialization procedure was designed, intended to bootstrap the Global View on entering a network. Depending on the nature of the ad hoc network in use, some are believed to suffer from high churn rates, i.e. the network experiences frequent node departures and constantly new nodes join the network. The initialization procedure will in such a case enable to node during the full lifetime of the node to carry out global adaptations using CrossTalk's Global View, even if that lifetime is only limited. Furthermore, it will in turn be able to assist in bootstrapping other nodes right from the start.
The procedure is quite simple. A joining node would broadcast an initialization request to its one-hop neighbors only, i.e. the broadcast would not be propagated into the rest of the network which can easily be achieved by assigning it a time-to-live (TTL) of only 1 hop. Nodes receiving such a request would send their calculated global view values back.
This procedure can now be used in two different ways which do not mutually exclude one another. One way is generally applicable as it initializes the Global View itself. The values received from the neighbors are added to the Global View this way not only having already received some samples but also some that are very close to the network-wide value. In other words the Global View is initialized with high quality information. These values are marked so that they are not replaced by the local values send by the neighbors in successive transmissions. They will be removed after their useful lifetime is expired. The node therefore has enough time to gather samples during that timeframe.
The second way the initialization information can be used is to initialize the metric generator. This procedure cannot be described in general as it is highly dependant on the metrics the initialization data is used for. An example can be found though in section 5.1. As described in section 4.2.2 the metric generator calculates metrics that have to be observed during a certain time window. On bootstrap no time has elapsed yet and the information provided by the initialization procedure can be used to initialize the generation of such a metric.

Another procedure which should be considered has to do with some of the criticism summed up in chapter 3.3. The criticism expressed in that chapter concern the issue that cross-layer optimizations can potentially influence protocol behavior negatively due to unintended consequences or adaptation loops. Since using network-wide adaptations there is no direct influence on the way other nodes behave, an extra monitoring and fallback mechanism should be used. As already described each protocol running inside of the CrossTalk framework has to be able to run in purely "layered" mode as the required cross-layer information might not be available at all times. Another reason for this is that the optimization process might not be satisfactory. Imagine a routing protocol based on the Global View mechanism. It might be that, due to unforeseen circumstances, the optimization in place might have a negative performance impact. A source node has no immediate influence on the behavior of a remote node. Therefore a protocol should have a signaling mechanism to force intermediate nodes to perform their functionality without the Global View optimization. This way, if Global View optimizations do have a negative impact, the layered mode is resumed and the network remains functional. This mechanism can also be used for monitoring purposes as a node might force the intermediate nodes not to use optimizations and can compare the cross-layer performance directly against the "layered" performance. This protocol signaling is not part of the CrossTalk architecture itself but would have to be implemented by every protocol making use of the Global View algorithm. Realizing this is not very complex though as it builds up on the fallback requirements of CrossTalk-enabled protocols. A good way to do this would be to have a flag inside the protocol header which indicates whether to use the Global View optimization or not. A good example for this can be found in chapter 5.1.
It has to be stressed that this procedure is not essential to the functionality of the Global View mechanism and does not have to be implemented. A reason not to do so could be that by changing a header, the protocol interoperability cannot be guaranteed. But many protocols allow expanding the header with arbitrary information or having some reserved bits for future use which could be utilized for a fallback mechanism flag.

4.5. Summary

The CrossTalk architecture exposes several unique characteristics and components not found in cross-layer architectures proposed so far. The most powerful and one of the central aspects of CrossTalk is the Global View component which is able to provide a network-wide view of metrics used for adaptations and optimizations of network protocol behavior. These optimizations are based on the comparison between the local node state and the corresponding network-wide state. This comparison yields the relative state of a node which can be used to base local decisions on it this way achieving globally near optimal results while employing lightweight mechanisms.