5. Applications, Results and Analysis

In this chapter the global view mechanism of CrossTalk is thoroughly analyzed as it serves as the basis for many cross-layer adaptations and optimizations. The following subsections furthermore demonstrate that the CrossTalk architecture provides a framework which can be used to achieve the goals of this thesis as stated in section 1.2. Additionally, several applications of the CrossTalk architecture are described and analyzed, to demonstrate how it can be utilized to increase ad hoc network protocol performance and how to alleviate some of the issues that arise in such networks and networks in general. The exemplary applications of our framework also show how the metric generation works in detail and how the Global View is utilized to change protocol behavior. Also, a novel application or network service for ad hoc networks is presented, that can easily be realized using CrossTalk's unique feature set.

5.1. Load Balancing – Solving Conventional Networking Issues Using CrossTalk

This chapter is a very good application example of CrossTalk's Global View mechanism as it nearly covers all aspects of it [50]. It furthermore covers many goals of this thesis with the exemplary load balancing application. One of the goals for example is to find a new way to deal with conventional networking issues in ad hoc networks. One of these issues, which in nature is not restricted to ad hoc networks only, is load balancing and this chapter describes a way how the Global View mechanism can be used to alleviate it. The solution presented here goes even a step further and also weakens broadcast storms [61], a common problem found in ad hoc networks causing congestion of the media and resulting in a high collision probability. Furthermore, this chapter contains a good example how the metric generator works also complying with the goals of this thesis. In addition, a way of protocol signaling is introduced as a means of a fallback mechanism.

5.1.1. Related Work

Load balancing and the reduction of the problems caused by the broadcast storm problem remain an investigated research area. Balancing the load in ad hoc

networks is important since nodes with a high load burden deplete their batteries quickly, thereby increasing the probability of disconnecting or partitioning the network. Additionally, if the network is more balanced, the spatial reuse of the spectrum allows for a higher throughput and relieves the center of the network [56].

One of the earliest pieces of work that addressed load balancing in ad hoc networks is part of the Associativity-Based Routing (ABR) protocol [57]. But the load, in their case relay load, is only regarded as a secondary metric inside ABR and a route's stability is the primary metric used for route selection. Load balancing itself is therefore not in detail evaluated but the load as a metric was identified.

Hassanein and Zhou presented the on-demand Load-Balanced Ad hoc Routing (LBAR) in [58]. The metric they present is rather coarse grained. The basis of their metric was termed *node activity* which is the sum of active paths through a node and therefore only reflecting the actual load indirectly as the actual load per path is left unconsidered. Furthermore, the node activities of neighboring nodes are taken into account called *traffic interference*. A destination waits for some predefined time to let multiple route requests arrive and then chooses the most suitable route according to a cost metric which is simply the sum of all the node activities and traffic interferences along the respective path. This general principle applies to most routing protocols presented here. The wait time has therefore to be chosen carefully and always adds some delay to the route discovery process.

In [62] the Traffic-Size Aware (TSA) Routing protocol based on the Virtual Path Routing (VPR) Protocol [63] is presented. For all virtual paths, which essentially are routes from some source to some destination, a load metric called *Entry Load*, based on the size of the traffic in bytes that go through the path is calculated. In addition the number of packets and the MAC header size are also taken into account to address the MAC contention overhead. This per path load metric is calculated by averaging the accumulated values mentioned above over the total lifetime of the virtual path. The sum of all Entry Loads is called the *Local Load* of a node. Furthermore, the *Regional Load* of some node X is the sum of all Local Loads of X's neighbors. Finally, the *Total Load* as defined in [62] is the sum of a node's Local Load and its Regional Load which is the basis of their load balancing algorithm. To compare different paths the Total Load of all nodes belonging to a path are summed up yielding the *Path Load*. Their metric is a little more fine-grained as other metrics as they take into account the size of the traffic in bytes and not the number of packets alone but as the metric is based on the average of an entries lifetime, fluctuations over the lifetime of a virtual path are averaged out. As an example, a path that is highly loaded at the beginning of its usage and later on is only lightly loaded will appear heavily loaded and only slowly decreasing over time. A node could also try to send many packets on its own behalf to make sure that it does not have to forward packets on behalf of other nodes. In addition, the question arises whether the accumulated Path Load is a good metric for load balancing. Longer paths around the edge of the network might be less loaded but since more nodes are involved in the overall path the Path Load will appear to be high.

The load balancing extension to VPR itself uses different headers making it none-compatible with the previous version. In addition, a static parameter is used at

the source to determine how long a node has to wait for incoming route replies. This can result in long delays or possibly missing the optimal path depending on the selected wait time. Interestingly, the performance of TSA is not evaluated against VPR which is the basis of TSA, which appears strange as the authors are also the developers of VPR. Instead some other shortest path routing algorithm, namely Implicit Source Routing (ISR) is used. Therefore, it cannot be said if this load balancing is more efficient than VPR and to what extend.

Dynamic Load-Aware Routing (DLAR) [66] is another on-demand routing algorithm designed explicitly for load balancing purposes. The load metric is simply the amount of packets buffered for a node's interface. It therefore only represents the current load without also taking into account the recent load situation. A path is chosen according to 3 different schemes. One is the path load, which is the sum of loads on a given path, the average nodal load on a path or the amount of nodes on a path above a certain load threshold. During the usage of a route the load metric is constantly recalculated and in case of a threshold violation a new path is chosen. DLAR's performance obviously heavily depends on a good choice of the various thresholds, especially the one that forces a new route lookup. As not being a cross-layer protocol the load information also remains inside DLAR and cannot be used for further adaptations as it is the case for most protocols presented here. In lightly loaded networks the question arises if a good distinction between routes can be made, as the buffers are emptied quickly and in general are empty most of the time.

All routing schemes presented so far are all very similar in the way they behave and the Busy Node Avoidance Routing (BNAR) protocol [67] is no exception. In fact it is extremely similar to the previously described TSA algorithm. The load metric calculated, called *busy rate*, is the sum of the times a node either receives or sends packets divided by the total observation window time. Again, as with TSA, the choice of this window is critical as for example there is no weighting inside this timeframe which could account for recent load changes. The authors recognize this shortcoming and restrict their scheme to scenarios with long-lasting routes with continuous traffic. As in many other protocols the sum of the individual loads along a path is the route selection metric. The destination chooses the route after some wait time having the drawback already mentioned before. More interestingly BNAR is extended in cross-layer fashion in [68] by including the Network Allocation Vector (NAV), employed by the Distributed Coordination Function (DCF) of the IEEE 802.11 standard into the busy rate calculation. In other words, their new scheme BNAR_with_NAV combines MAC layer information with routing layer functionality. The algorithm stays the same only the busy rate calculation includes the time a node defers channel access due to the setting of the NAV.

In [69] Wu and Harms describe their Load-Sensitive Routing (LSR) method. Their load metric is similar to the one of DLAR only that it also accounts for the amount of packets in the buffers of neighboring nodes. They define a path comparison function where the path load of a currently used path needs to exceed the path load of an alternative path by some threshold to be regarded as preferable. Or, the standard deviation of the path loads must differ by some other threshold value. These values of these two thresholds are vital for the functionality of LSR and difficult to pre-asses. One advantage of LSR is that the destination does not wait to send a route reply back to the source. But LSR sends

multiple replies back and the source can later switch the route. This paper indirectly confirms our assumptions about the load metric of DLAR. In low traffic scenarios LSR does not differ very much from the in principle similar DSR as the buffer state based metric cannot reflect load well in such scenarios.
Yuan et al. [70] present a scheme where a threshold simply decides whether a node forwards a route request or not, this way making sure that overloaded nodes will not have to carry more forwarding load. Obviously, the threshold value plays here once more a vital role. Therefore, they calculate the *average queue occupancy* of each node, which is the average of its own queue level one of its neighbors. Each node during the route discovery phase adds its average queue occupancy to the route request before forwarding it. The calculated threshold is simply the average queue occupancy of the nodes on the backward path including the current node. The problem with this approach is that a route request is not guaranteed any more to be successfully delivered.
Basically the same scheme is employed in [71]. Route requests are simply dropped on the violation of a threshold. One difference is that a predefined upper and a lower bound on the threshold exist whereas in [70] only a minimum bound is set. The maximum bound has to be chosen carefully though, as a low bound prevents a good load balancing efficiency and might actually ender routing impossible as all route request will be dropped. Also a flag is introduced which forces nodes to forward a packet which can alleviate the problem of the scheme in [70]. Their calculation of the load also works on the queue lengths but has some coarse grained time base and works with different static parameters and thresholds. Therefore, the whole calculation might not be very accurate depending on the overall network conditions.
In [72] the above load-balancing technique is combined with a caching enhancement method to increase the energy efficiency of the overall approach.
Zheng et al. [73] introduce their dynamic load-aware based load-balanced routing (DLBL) algorithm. Their metric introduces some slightly new aspects as they also include nodal propagation delay into their load metric but the actual load calculation is not explicitly addressed. They utilize it but do not state the way they calculate it. The route reply is used to create multiple routes which includes the total path load as a sum of individual nodal loads along a path.
The Simple Load-balancing Approach (SLA) [75][76] is a module which according to the authors works with any existing routing protocol, although it depends on the availability of periodic `Hello` messages which have some none-utilized space in their packet headers. It also represents a cross-layer approach as it utilizes energy information at the routing layer. The space in the packet header is initially used to add the nodes number of buffered packets and its remaining energy. But consecutively, it will be used to include the averages of its neighbors, this way creating a notion of the network-wide load and energy, sharing some similarities to CrossTalk's Global View mechanism of establishing a network-wide view. Using the buffered packets metric has the obvious disadvantages as stated before which can potentially also have a negative impact on their calculation of the network-wide view, which itself, in addition, has no time base. The paper defines three possible load metrics of which two also account for the remaining energy level. The first one is simply the ratio between a node's load and the network-wide load. The second metric is the ratio between a node's load to power ratio and the network-wide load to power ratio. The last metric is the

ratio between the energy left after all network-wide packets have been send to the energy left after all packets transmitted by the local node are dispatched successfully. The energy-based metrics clearly are a tradeoff between pure load balancing and the extension of the overall network lifetime.
The algorithm itself is two-staged, operating between two predefined thresholds. Crossing the lower load threshold SLA enters a passive load balancing state where all incoming route requests are dropped. Here it becomes obvious that SLA is designed for only on-demand protocols. Dropping route requests has been discussed before. After the second threshold has been violated SLA sends out packets to force a route discovery process circumventing the overloaded node which requires more substantial changes to the routing protocol or fundamental routing capabilities inside of SLA.
Zhou et al. designed [21] a cross-layer framework (CRDF) specifically for route discovery. It consists of two parts which are the priority-based route discovery strategy (PRDS) [82] and the virtual device information manager (VDIM). The overall goal is to reduce redundant broadcasts and to solve the next hop racing problem at the same time. The VDIM is responsible to manage cross-layer information from all layers and devices and provides an API to access that information. This information is utilized by PRDS, which uses it for *routing strategy automation* (RoSAuto). RoSAuto lets a source node choose the general routing strategy, such as least delay, which is further refined by nodes on a path.
PRDS works in a way that it uses some priority index to determine whether it is a "good" or a "bad" candidate for a route which can be derived from the information provided by the VDIM. Load is only one possible metric, therefore no actual load calculation is described but the priority index calculated is normalized to lie in the interval [0,1]. According to the priority index the forwarding of an incoming route request is delayed. In addition route requests are dropped after receiving the same route request more than n times, where n is a fixed threshold, to solve the rebroadcast redundancy problem.
In [83] the issue of shortest path routing and the core of the network is analyzed and the authors therefore propose a center-relieving forwarding scheme which essentially is a load balancing scheme. As shortest path routing only utilizes the distance towards the destination as a route selection metric they introduce a secondary metric which is the distance to the center of the network. This way the distance towards the destination is reduced in each step but at the same time the distance to the center of the network is maximized. The obvious problem with this approach that it will need to utilize some positioning system or be rather coarse grained on a distance metric based on hops. In addition, a notion of the core of the network is needed as every node needs to know how far the center is away. The actual determination of the core and the distance measurements are not described in detail.
There exists also some remotely related work which is only mentioned for completeness without going into detail. There are some special solutions to load balancing in ad hoc networks using node clusters with the goal to distribute the load evenly [64][65]. These solutions differ significantly with our own solution and no applicable load metrics are presented therein. Also multi-path routing solutions are not presented here in detail, as they are not directly related to our work and represent a very special solution to the problem. But, as the load is distributed over many paths in these approaches, they theoretically can achieve

load balancing to some extend although route coupling can render such an approach only marginally useful [78]. Even worse, Ganjali and Keshavarzian [79] show by analysis and simulation, that multi-path routing in ad hoc networks does not achieve any significant gain in load balancing compared to single path routing for a realistic number of paths. One multi-path approach is worth mentioning though as it uses the packet delay based metric for load balancing. Another approach [74] is based on the ability to have Internet connections over cellular technology and therefore represents no pure ad hoc network solution. The focus is on load balancing towards gateway nodes. The load balancing approach found in [77] works only in ad hoc networks with directional antennas. In [80] some load metric in conjunction with an obscure maximum throughput value and some arbitrary coefficients are presented in a very vague way which also includes the actual routing process. Therefore the details are omitted here.
In the load-balancing experiments carried out AODV was used as basis for our load balancing scheme. It has to be noted that we strongly believe that other routing schemes are as suitable for load balancing adaptations using CrossTalk. Nevertheless, there are very good reasons for the choice of AODV. On the one hand some cross-layer architectures such as MobileMAN make certain assumptions about the suitability of protocols. They strongly suggest the use of proactive routing protocols as they maintain routing state actively which is in consequence available at all times. Therefore, AODV serves very well as an exemplary case that the CrossTalk architecture does not make such assumptions per se. Furthermore, AODV is a very well analyzed and a well advanced routing protocol that is even one of only a very few protocols in discussion for standardization by the IETF. Many papers are based on AODV and many comparative studies exist, making it a highly suitable protocol base to incorporate adaptations as AODV itself was shown to be a well performing protocol already.
AODV's general operating mode is quite simple and is described only very briefly here. For a comprehensive functional description please refer to [60] and [84].
Whenever an application wants to send a packet and the destination is unknown, AODV issues a route request. What this translates to is flooding the network to find the destination node. When the destination node or, depending on the implementation, an intermediate node that has the destination in its route cache is reached a route reply message is send back to the origin of the route request. As the route request message is used to set-up the reverse route, the reply can follow this path without having to flood the network again. Once the reply reaches the source node, the transfer of the data packets begins. After the route is not actively used any more for a certain time the route is purged from the route caches on the nodes along the route instead of maintaining it proactively.
AODV comprises a neighborhood maintenance and detection mechanism based on periodic beacons called `Hello` messages. `Hello` messages are one hop broadcast messages that notify surrounding nodes of another node's presence. An absence of some consecutive `Hello` messages, after at least once being received indicates the departure of a former neighbor node.

5.1.2. Metric Generation and the Load Balancing Extension to AODV

As already mentioned the protocols that run inside the CrossTalk framework should not be completely redesigned. The general principle of the protocol should remain intact, i.e. should remain basically the same as in the purely layered version of the protocol. Minor changes to the protocols should suffice, keeping the design effort and the complexity low. The load balancing enhanced AODV is a very good example for this. To create a load balanced AODV variant using the Global View mechanism, two enabling mechanisms have to be implemented:

- A metric has to be derived that reflects the current load of a node.
- AODV has to be changed slightly to make use of the information provided by the Global View in comparison with the Local View, i.e. its relative state.

Let's first look at the generation of the load metric. Many different ways have been proposed to actually calculate the nodal load as presented in the related work section. We do not simply adopt any of the calculations presented for several reasons. Many metrics are based on a simple time window which we believe is hard to choose. If it is to small, bursty traffic patterns might not be evaluated correctly, as a load burst my be undetected or dominate the load metric. On the other hand if it is chosen too large then longer term changes are not accounted for in a reasonably timely manner. If a node for example stops transmitting packets all together then this should be reflected in a load metric timely as it clearly has capacity to route data traffic. We also do not choose the packet buffer fill degree due to the drawbacks described in the previous chapter. For our proposed metric of a node's local load, the number of packets sent by a node on behalf of other nodes during a relatively small time frame or slot t is used as a basis. We do not include our own packets to provide a degree of fairness and to enforce cooperation to some extend. If a node's own packets would be accounted for it could simply transmit many packets generated by its own applications and be rewarded by not having to forward packets on behalf of other nodes. This is an aspect that has so far not been regarded. Since ad hoc networks rely on the concept of cooperation every node has to perform tasks on behalf of others and should behave selfishly, which also applies to the consummation of resources such as bandwidth. A protocol should not be able to be exploited easily by such nodes, i.e. rewarding selfish nodes. As simple metrics can drive protocol behavior, giving indirect control over these metrics at application level to a user will certainly be exploited.
As already mentioned choosing a fixed timeframe might turn out to be critical. Therefore, we chose a method to make the choice of the actual time window less critical. When choosing the duration of a time window too small, as already mentioned then strong fluctuations that occur during one observed time period will make the load metric very unstable. To eliminate such fluctuations, we use n of the previously mentioned time slots to calculate the actual load over a time period of $n * t$. The choice of t should be made reasonable which is not very difficult. For example the `Hello` message interval of AODV can be used as a

basis together with a small multiplication factor *f*. The choice of n can also be derived from certain metrics used in the routing protocol. When a route expires for example a new route request needs to be issued and the route is likely not in active use any more by the previous application. Therefore, n should be chosen to have a total window size which is at least bigger than the route expiration time. This way short period bursty traffic will be accounted for over the lifetime of the route. To account for long term, rather constant changes in the load during that period, we calculate the weighted moving average of all slots, except the current slot, with a weight being equal to the slot number as shown in Fig. 5.1. The formula for this calculation looks like this:

$$l = \frac{\sum_{i=1}^{i=n-1} w_i s_i}{\sum_{i=1}^{i=n-1} w_i},$$

whereas l is our load metric, n denotes the number of slots used and s is the slot value. The weight w in the experiments was simply set to i but other functions are imaginable.

Using the above formula, the recent past will have a higher weight and will therefore influence the load metric in a stronger way. In other words, if a node stops forwarding packets for example, i.e. no active routes go through this node any more, this will be reflected in the load metric faster as opposed to only use a simple time window. Also if the load increases at a node this will be reflected in a timely manner with the past load's influence fading out over time. Breaking up the time window and using the function above, the actual choice of t and n become far less critical. Also there are some good indicators for the two values making the choice easy, which has not been addressed in this way so far.

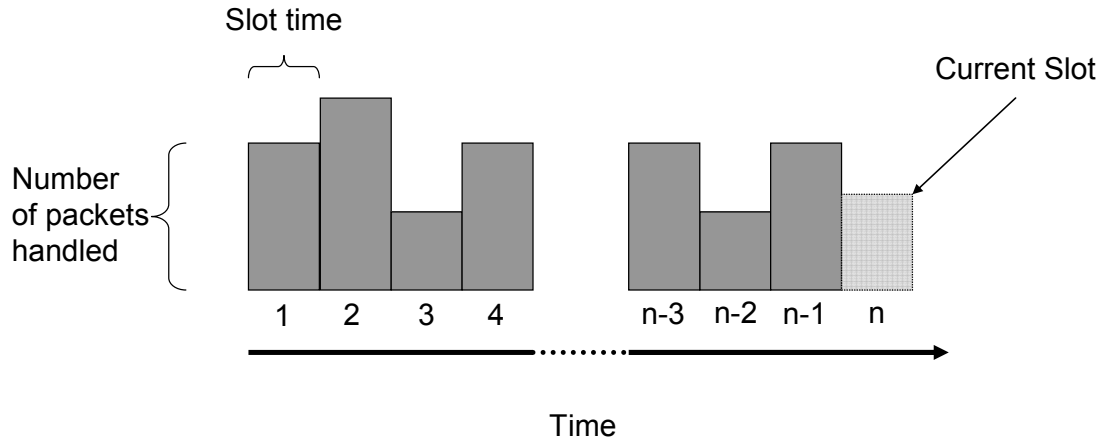

Fig. 5.1 Load metric calculation

The actual calculation of the load metric is done inside the metric generator who only needs to be informed by the routing protocol (or potentially the MAC layer) that a packet was sent.

Having a local load metric to disseminate the only thing left to do is to change AODV's algorithm slightly by making use of the global load information. Many approaches work at the destination of a route request as presented in the previous section. The destination waits for all route requests that arrive during a certain wait time and selects the most appropriate path. With our approach we

do not need to wait since a node already locally can do estimations about its suitability during the route detection phase. The advantage is that the delay can be reduced in comparison to these approaches and at the same time the likelihood for having found the optimal route is increased since when utilizing a wait time it might be chosen too small. In other words there is no route selection algorithm necessary.
Having a global view enables us to influence the route request phase of AODV, i.e. the route discovery. Dropping route request packets through overloaded nodes being the only mechanism to do load balancing as proposed by some approaches has the clear drawback that route requests are not guaranteed to reach the destination any more. Also having a fixed threshold has the disadvantage that it is very coarse grained and beyond the threshold no distinction between nodes is done any more. The advantage on the other hand is that the rebroadcast redundancy is implicitly reduced as well as the broadcast storm problem since fewer broadcast packets are sent.
Preferably, a load balancing scheme differentiates overloaded nodes according to their overload degree but at the same time makes sure that nodes that are overloaded beyond a certain point drop route request to make sure that no more routes will go through them. This also has a QoS aspect as such an upper bound guarantees that a route circumvents such hot spots. But it is very important that this upper bound is not a fixed threshold but is dependant on the current load situation in the network.
Therefore, our load balancing algorithm is a three-phase mechanism as illustrated in Fig. 5.2 and works as follows:
Whenever an AODV route request reaches a node, it calculates the global view and compares its own local load against it, i.e. it evaluates its relative load state. This comparison yields as a result if and to what extend the node is overloaded. If the node is not overloaded compared to the calculated global view, it resumes "normal" operation in terms of AODV. In this case the shortest path objective of AODV is pursued.
When the node finds itself overloaded, it calculates the *overload degree*, that is the ratio of the own local load and the global view of the load. From the point of being overloaded, which translates to an overload degree > 1, up to a predefined relative threshold, the *delay bound*, the node will hold back the route request for a certain amount of time before forwarding it. This threshold is relative as it represents a specific overload degree and is therefore dynamic. In other words, the higher the network load is, the higher the absolute load value the delay bound represents. This forwarding delay grows proportionally with the overload degree up to the delay bound, where the delay reaches its maximum. By delaying the route request, the probability increases that an alternative route (through other nodes) will be established circumventing the overloaded area, which is likely to be the core of the network as already mentioned. This way the following data packets will not have to be forwarded by the overloaded node, which would have even further increased the load and at the same time increased the collision probability with other packets in the overloaded area. Using this mechanism, newly formed routes and the subsequent data traffic are being pushed towards the edge of the network. With the delay being proportional to the overload degree, a route will be established balanced between being short in terms of the hop count and carrying mild load.

Once its delay bound is reached, a node holds back packets with a maximum delay d. Even if the load is beyond the delay bound, packets will not be delayed longer than d.

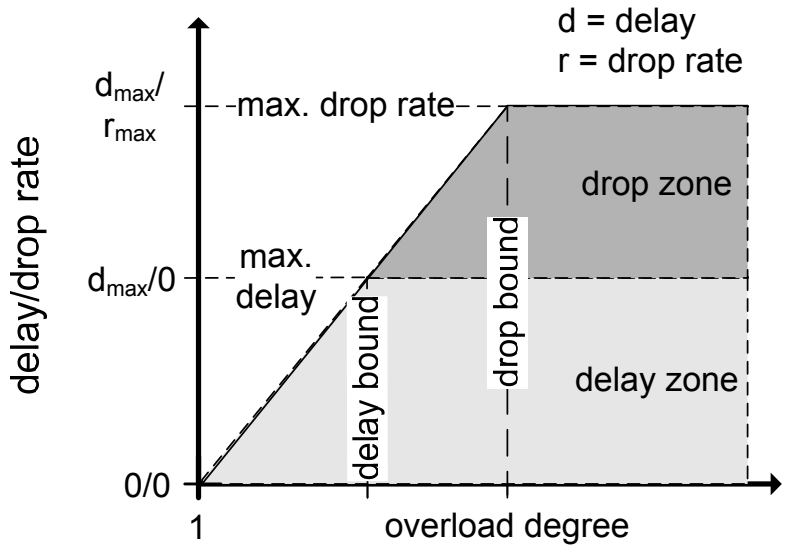

Fig. 5.2 The load balancing algorithm

The second phase of the algorithm covers the load reduction functionality. When a node has an overload degree beyond the delay bound, it starts dropping route requests with a certain probability. If it is not dropped the forwarding of the route request is delayed by the maximum delay d. The drop rate also grows proportionally with the overload degree up to another relative threshold, the *drop bound*. Reaching the drop bound the drop probability or drop rate reaches a predefined maximum value.

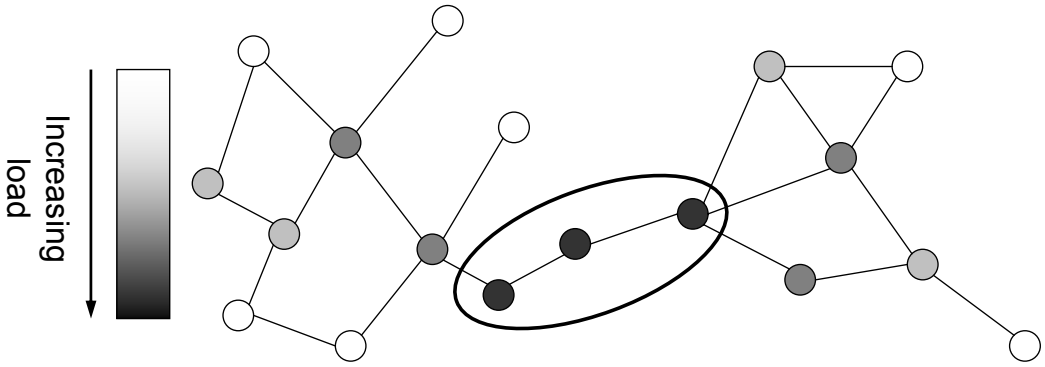

Fig. 5.3 Bottleneck node between network partitions

Beyond the drop bound the algorithm is in its third phase. A node that is overloaded heavily enough to be in this phase will drop route requests at the maximum drop rate or delay it by d. By dropping route requests, these heavily overloaded nodes make sure that the path towards the requested destination will not lead through highly overloaded nodes as already mentioned. Additionally, the broadcast storm problem in highly overloaded zones is reduced. The maximum drop rate should never reach 100% though since an extremely overloaded node might be so for the simple reason that it is the only node connecting parts of a network as depicted in Fig. 5.3. Since it does not accept every route request, the

node would keep up a certain path quality for existing paths through it, by limiting the amount of paths that lead through it.

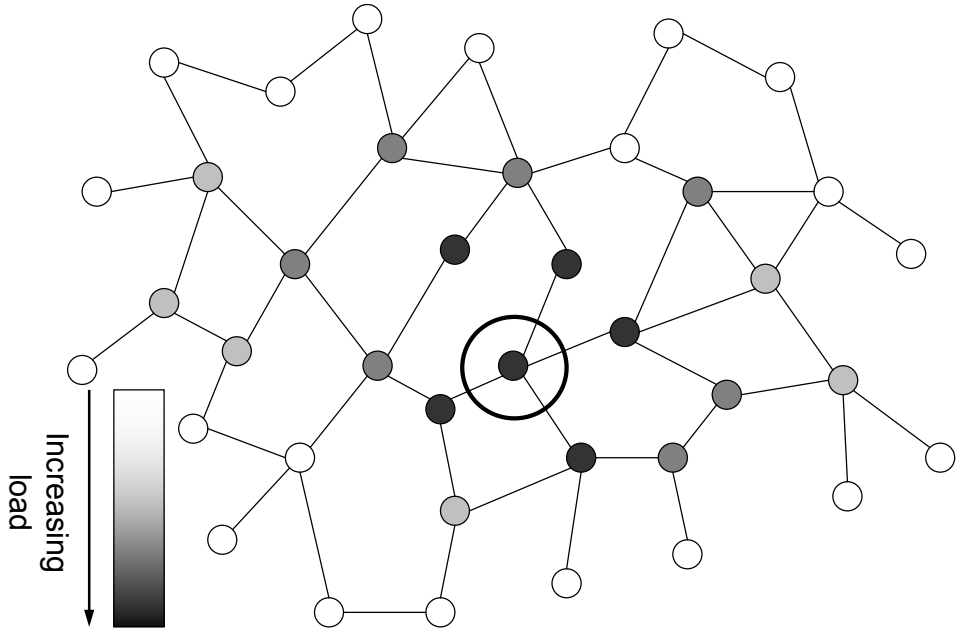

Fig. 5.4 The network core problem

The load balancing example is also a good showcase for the usefulness of the protocol signaling mechanism as described in section 4.4. Consider the node in the center of the exemplary network presented in Fig. 5.4. It is clearly overloaded as can be expected using a simple shortest path routing algorithm. The figure also displays that the neighboring nodes are also overloaded. Let us assume that all these nodes are operating beyond the drop bound, i.e. they reject most of the route request. The center node would therefore have many difficulties to establish routes successfully. It could determine by querying its Global View that it is in this unfortunate position under the assumption that the Global View of its neighboring nodes is virtually the same. Therefore, it could use the signaling mechanism described earlier to be able to successfully establish routes.
What is important to note is also that the changes to standard AODV do not influence the interoperability with our scheme. As the headers are not changed and the general algorithm is not altered in its core functionality nodes running the standard AODV will be able to communicate with nodes running our cross-layer enhanced ADOV. When mixing different versions of AODV the overall load balancing properties will deteriorate though.

5.1.3. Experimental Setup

All experiments [53] to evaluate the Global View mechanism were carried out using the ns-2 network simulator [59]. Due to its non-conformance with the Internet draft, the standard AODV implementation was replaced by the implementation of the Uppsala University (AODV-UU) [85]. Every simulation run carried out was running for 600 simulated seconds. To isolate the effects of various parameters such as network size, network density, mobility and more on the Global View mechanism and our AODV reference application, we keep all simulation parameters fixed except the one we want to evaluate. The standard or

base simulation network was a static, 200 node network on a 2000x2000 meter square plain. Each node has a transmission range of 250 meters. The nodes were placed randomly onto the plain, each running a traffic pattern agent that searches for a new destination every 10 seconds and sends one packet per second afterwards. On the basis of this configuration one parameter was changed for our simulations. For the reference application, the delay bound was chosen to be 1.8 and the drop bound 2.5 using a maximum delay of 100ms and a maximum drop rate of 67%. For every data point in the following graphs, 20 simulation runs were averaged to compensate for marginal phenomena.

For our experimental evaluation, we added some thresholds for the calculation of the global view as a security measure. Whenever the Global View component contains an insufficient number of samples, our CrossTalk architecture would not calculate a network-wide average on request forcing "normal" mode. This way the generation of a highly incorrect network-wide view is prevented. Also the composition of the samples was evaluated. If more than two-thirds of the samples were from direct physical neighbors, the cross-layer entity would not calculate a global view for that metric preventing cross-layer operation, either.

When a node joins the network, we added an initialization procedure to bootstrap the node's Local View as mentioned in section 4.4. The joining node would broadcast the initialization request to its neighbors. The neighbors reply with a message containing their global view, which in turn is used at the newly joined node to fill the first slot of the load calculation algorithm. This way a node directly disseminates relatively correct load information which is adjusted over time by its own observations. In these experiments we also exploit the fact that AODV is sending periodic `Hello` messages. Instead of only using this kind of message to distribute our own local information, we also enrich `Hello` messages alternatingly with information of our one hop neighbors. This way, we use the one-hop broadcast `Hello` to disseminate recent and close-by information more effectively. In later experiments we do not do this any more to have a comparison if such diversification mechanisms are useful. During our follow-up experiments we found out that there is no significant gain.

We identified several parameters to test in our experiments. Since the global view's correctness is based on the amount, quality and diversity of the samples, the network structure has to be considered, as it has an impact on it. For this purpose, we evaluated the parameter network density. The denser a network is the more recipients a single message has which can have an impact on the global view. We also varied the network size in terms of participating nodes. The more nodes participate, the longer paths become and border effects might influence the global view differently. Another parameter that has to be considered is node mobility. In our case that is mobility according to the random waypoint mobility model. Mobile nodes can potentially help disseminating the local view data more efficiently. Finally, as a network structure parameter the aspect ratio of the topology geometry was varied over a broad range, which in our experiments is the ratio of the topology's geographical width and height. A narrow topology with longer path lengths might have an advantage concerning the correctness of the global view, since along a long stretched path in a narrow topology many nodes receive the same disseminated data.

Since we do not generate any messages, the Global View mechanism is also dependant on the traffic pattern and on the load. Therefore, we tested our

CrossTalk architecture under different load scenarios. The traffic pattern we apply distinguishes between local and distant communication. Within a certain radius (in hops), a node considers the communication local, whereas beyond that radius the traffic is considered distant traffic. In our simulations, we evaluate different ratios of local and distant traffic. Finally, we analyzed the global view under churn conditions for the simple reason that the global view will become more accurate over time since it most probably aggregated more samples. With churn, there are constantly nodes joining the network with an empty global view influencing the correctness of the local evaluation process.
The experiments described in this chapter made no assumptions about memory limitations on the devices. Therefore, the stateful dissemination approach was not applied. The analysis of memory requirements and communication overhead can be found in chapter 5.4.

Table 5.1 Load balancing experiment parameters

Simulation parameter	Parameter range
Network density	50 – 100 nodes/km²
Network size	50 – 400 nodes
Mobility	1.4m/s
Topology aspect ratio	1:1 – 1:9
Traffic generation	0.5 – 2 pkts/s
Traffic pattern	25% - 100% within 3 hops
Churn	each node fails every 60 – 250s

5.1.4. Experimental Results

The results we were aiming at were twofold. First, the correctness of the global view provided by CrossTalk has to be analyzed. All protocols that make decisions on the global view data rely on a certain degree of correctness. Due to the dynamic nature of the metric itself, the network and the dissemination process the global view, as already mentioned can never be 100% correct, but a correctness degree within certain bounds is essential. Furthermore, for some application is might also be very important that the global view is not only relatively correct locally but also uniform within certain bounds across all nodes. In other words, it might be important that all nodes have virtually the same global view. Therefore, before showing the results of our AODV reference application of CrossTalk, we present an analysis of the quality of the Global View mechanism. We show the quality of the global view using three different metrics which have been analyzed for all the scenarios described above. The first metric to show the global view's accuracy is the average value of all global views in the network, i.e. the average of all global views locally calculated at a node. This

value is compared with the average value of all local views in the network. Ideally the two values should match. Of course, the average alone does not reflect the quality of the global view accurately enough. The second metric reflects the global view's uniformity across nodes, which is the standard deviation of the global view at each node in the network. This value is compared against the standard deviation of the local view at each node in the network as a measure of the uniformity of the data samples collected by the Global View. Ideally, the global views' standard deviation is zero, independently from the local views' standard deviation.

The third metric we termed correctness. We calculate the average local view artificially representing a perfect Global View and then compare the local view of each individual node against it. If this comparison and the comparison of local and global view at each node yield the same result (overloaded vs. not overloaded), the node evaluates its status correctly, and otherwise it fails to do so. The correctness is the percentage of nodes in the network that evaluate their status correctly.

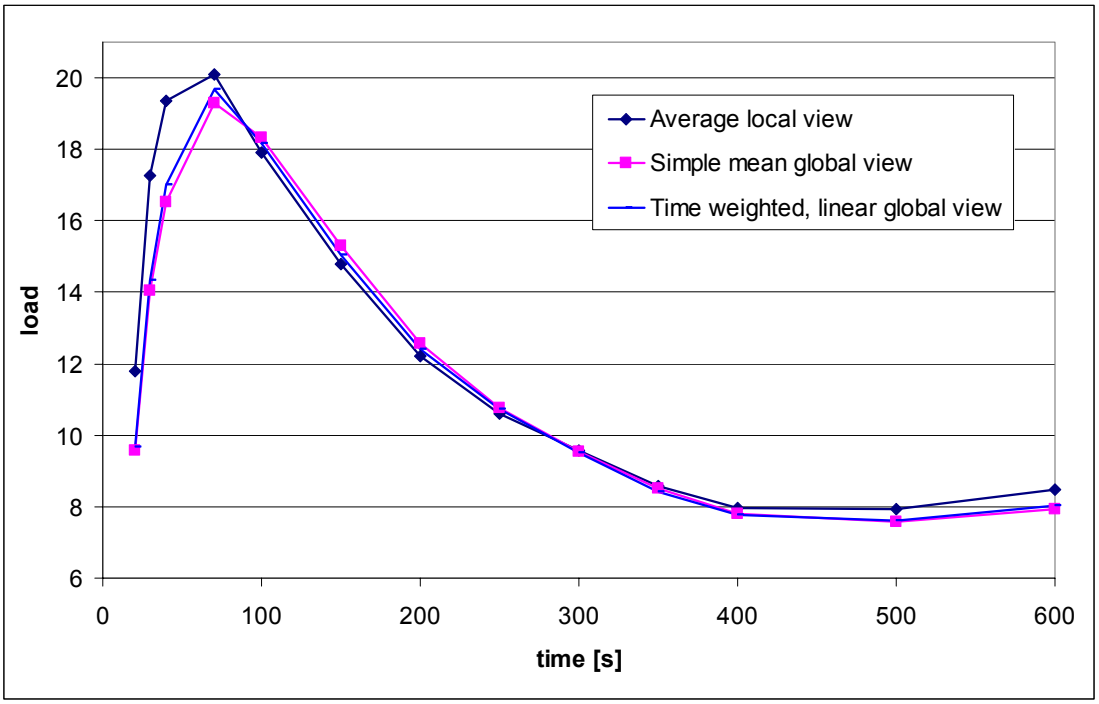

Fig. 5.5 Comparison of the average global and local view over 600 simulated seconds in a typical mobile scenario

Let us look at the load metric behavior over a simulation run, i.e. over 600 simulated seconds as illustrated in Fig. 5.5. The figure depicts the average local load metric in comparison of the average global view produced by the network nodes for two selected global view algorithms in a mobile network. The reason why only two global view algorithms are shown is the similarity of the resulting average global views.

At the beginning of the simulation all nodes start out having no state at all. When the traffic generators start to create their data, all the network nodes issue route requests congesting the network. This can be seen by the steep increase in the network load. Over some time nodes will be able to establish routes without having to query the network as some routes are already present and in use. As a

consequence the load is dropping to some relatively stable level. The key statement of Fig. 5.5 is that the Global View mechanism of CrossTalk can adapt to such sudden increases in the network load extremely well. The average network-wide view is at all times very close to the average local view across all nodes, i.e. the theoretically perfect global view, nearly independent of the actual algorithm to compute it. When the network load slowly levels in over time the global view adapts virtually synchronously. The time based weighting algorithms of CrossTalk perform slightly better as they account for the increase by putting more weight on recent data samples. In all other tested scenarios CrossTalk's behavior is very similar and for reasons of clarity these additional graphs are omitted here. In reality, a network will most likely not bootstrap in such an extreme way. The network would gradually evolve and nodes would not come up all at the same time. But this simulation can be seen as a worst case scenario and shows very well that the Global View very early is able to produce a high quality network-wide and that sudden increases of the metric in question do not affect its quality.

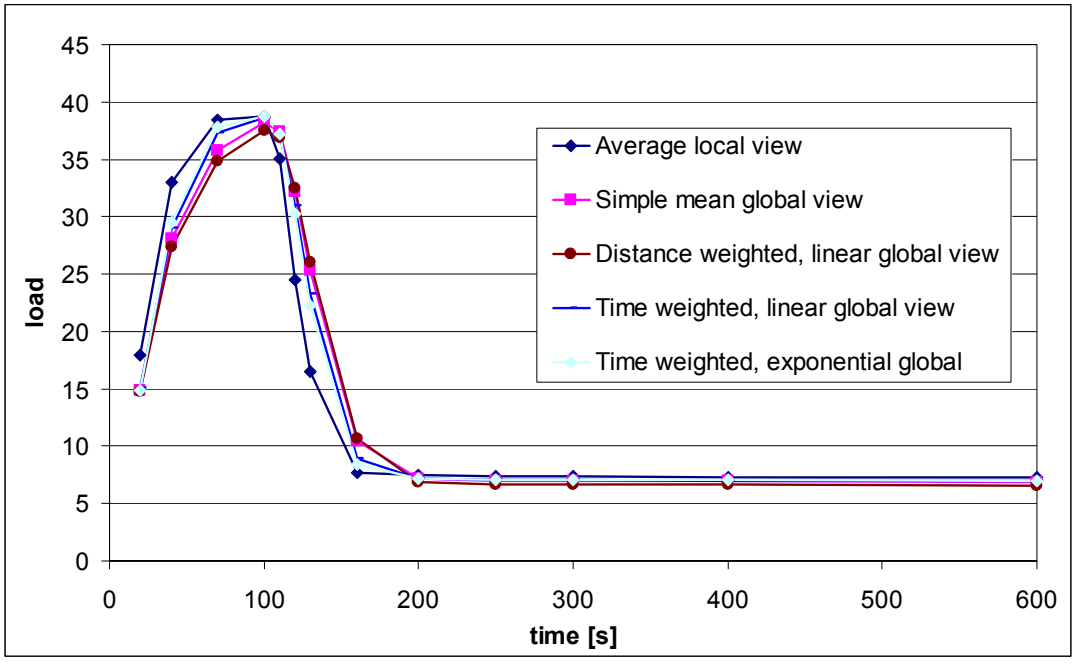

Fig. 5.6 Behavior of the global view with a sudden metric decrease

Another issue is the sudden decrease of an optimization metric due to a sudden decrease of the respective network parameter. The question is if the Global View is able to reflect those very well. We therefore evaluated the reactivity to extreme drops in traffic. We let the traffic generator run at each node for the first 100 seconds of a simulation run and then reduced the packet rate by a factor of 5 and the lookup rate for new destinations by a factor of 2. The results are displayed in Fig. 5.6 clearly showing the ability to react to extreme metric changes. Again, we left out some global view calculation algorithms for the sake of clarity due to the very similar results. The sudden load drop is again reflected the most accurately by the time-base weighted moving average slightly out performing the other algorithms. The graph shows how extreme the change in the load of the network is, still the global views timely reflect the change. Although such a scenario might appear unlikely in the real world, this can again be seen as a worst case

scenario. If the Global View mechanism can compensate such sudden, network-wide and extreme changes, minor fluctuation will surely be compensated without difficulties.

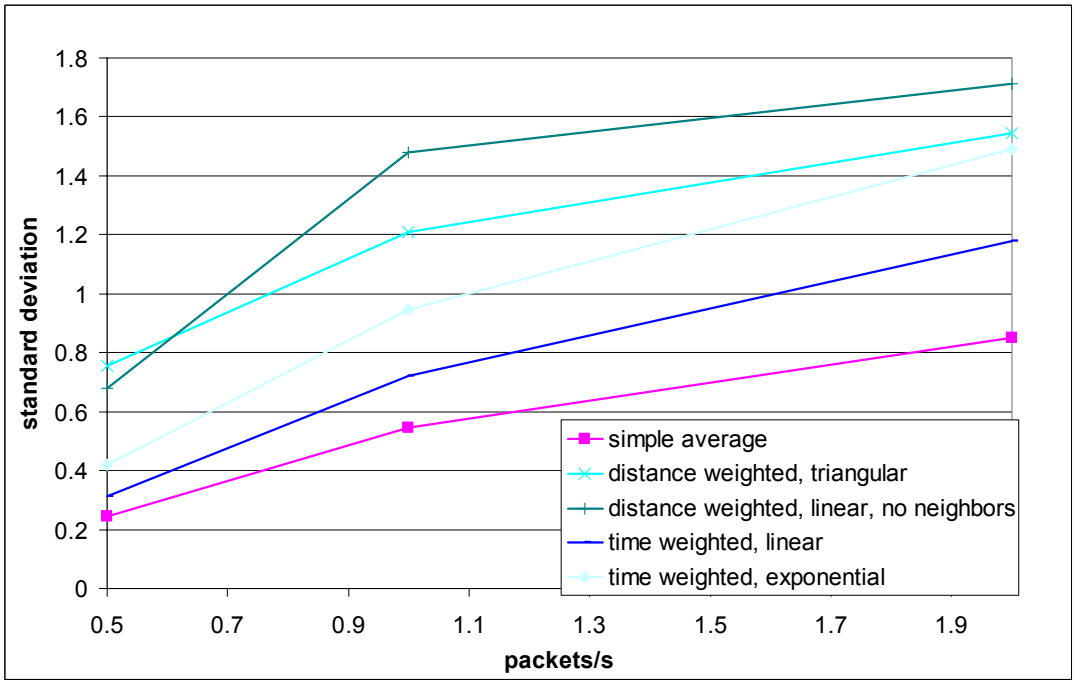

Fig. 5.7 Global view standard deviations in different load scenarios

Let us now focus on the three core metrics for the quality of the global view as described before. We will now stop focusing on the behavior of the global view over time but rather concentrate on a range of different network parameters. Having seen that the averages of the global views are very close to the actual average of the local views, we now need to analyze the uniformity of the global views found in the network. As a measure, as described before, we use the standard deviation. In Fig. 5.7 the standard deviations of selected global view algorithms are shown, one for each weighting category. As can be seen, in terms of uniformity the simple average outperforms all other algorithms. Before going into more detail some extra information has to be presented to actually give some significance to the data presented. The standard deviation of the local views, which is a measure of the uniformity or vice versa for the degree of discrepancy of the data samples, has to be compared against the standard deviations found in Fig. 5.7. This standard deviation of the local views ranges from 15 to 30 increasing over the observed load. That the standard deviation increases makes sense of course as the absolute amount of traffic increases. But also the core of the network will non-proportionally attract far more load than the border nodes of the network as the overall traffic increases which leads to an even stronger load imbalance. As can be seen in Fig. 5.7, CrossTalk's Global View mechanism can very effectively create a uniform network-wide view across all nodes. Although the simple average global view calculation outperforms the other approaches by up to a factor of three, the absolute difference is only marginal. The distance based algorithms expose the biggest standard deviation as the load in the network highly depends on the topological locality. The time based algorithms perform a little better but as the likelihood of having more up-to-date information from close-by nodes is relatively high, therefore having a stronger

impact on the global view. Again, for all tested network parameters the results are similar; therefore these graphs do not contain any additional information and are omitted here, which is also true for the following graphs.

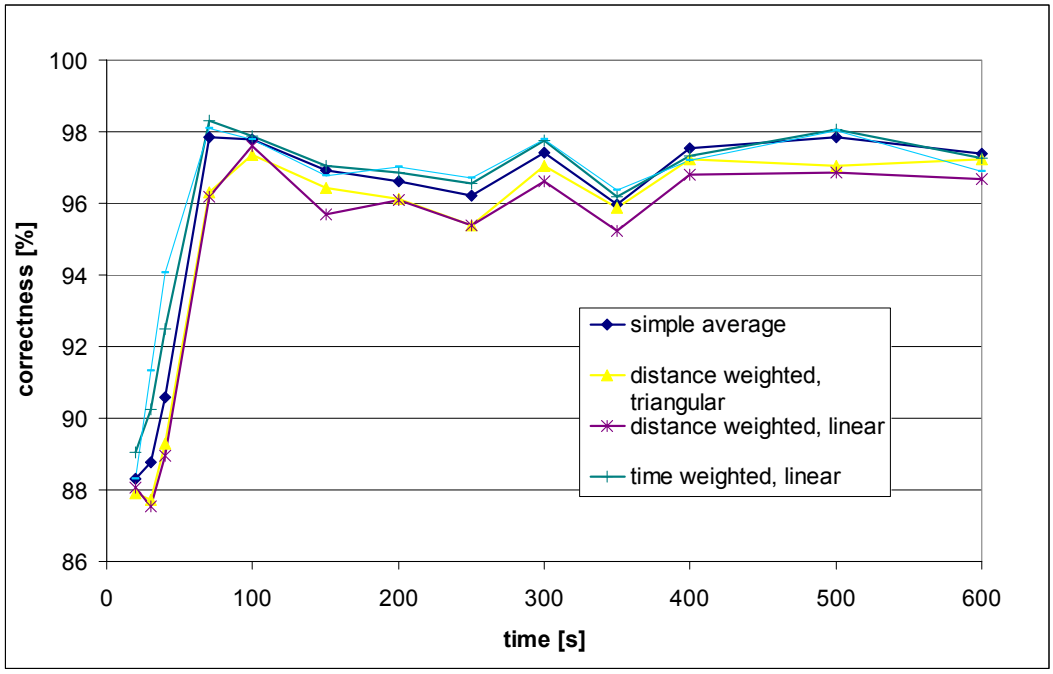

Fig. 5.8 Global view correctness in scenarios with high churn rates

We have analyzed so far the uniformity and the accuracy of the average global view. The last metric to reflect the quality of the global view creation is our correctness metric. In Fig. 5.8 the correctness of various selected global algorithms over the duration of a simulation run is presented. Especially in churn scenarios the global view can severely suffer as nodes constantly join the network having no state at all. We believe that we evaluated severe churn rates with a node having a lifetime between only 60 and 250 seconds with a failing probability uniformly distributed over the remaining 190 seconds once a node existed for a minute. At the beginning of the simulation the correctness is slightly below 90% due to the fact that all nodes just start to build up global knowledge and the steep increase in network load. Here the time based algorithms work the best as they account for the rapid changes the most. Very quickly the correctness degree grows to up to 98% reflecting a highly correct global view inside the network. Even the subsequent node failures and joins do not significantly impact the correctness degree over time, partially due to our initialization procedure.
Fig. 5.9 illustrates the global view correctness over a broad range of network sizes. Here again the simple average outperforms the other algorithms with all algorithms being well above 90% correct for all tested network sizes. The main advantage of this algorithm though can be seen in small networks. With increasing network size, the distance based algorithms gain. The reason is that a distance based algorithm such as the triangular weighted moving average stronger accounts for the network core. In small networks the ratio between border nodes to core nodes is much bigger than in larger networks. That means that with increasing network size, the local values of core nodes influence the network-wide average stronger and stronger therefore increasing the accuracy of the distance based algorithms.

With the analysis so far, we could show that the global view produced locally at a node is of very high quality reflecting the actual network-wide situation very precise in all tested scenarios. Having carried out the global view analysis, we can base our load balancing scheme as described above on the data provided by CrossTalk with quite some confidence. We tested the AODV load balancing extension in all the scenarios described in section 5.1.3.

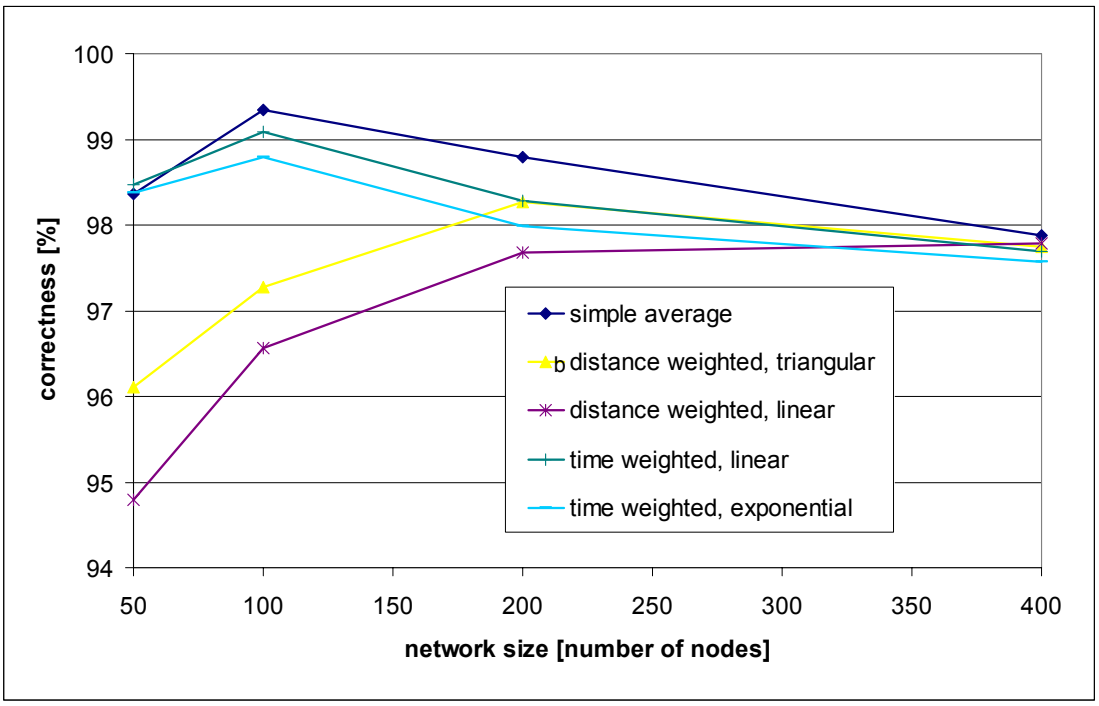

Fig. 5.9 Global view correctness in networks of different sizes

The performance of our scheme was compared against the purely layered AODV. The performance itself was measured according to multiple metrics. The average number of messages sent per node during a simulation run reflects whether there is an actual load reduction effect in the network or not. This metric shows how well our algorithm protects severely overloaded nodes and at the same time reduces the packet collision and retransmission problem by circumventing congested areas. The on average most overloaded node reflects the reduction of the bottlenecks within the network. The hot spot nodes, i.e. those nodes that are loaded the most in a network will have a poor performance for their own data traffic and also, as they are burdened the most, will deplete their batteries quickly from which point on other surrounding nodes will follow and the network starts to break apart. By relieving these nodes the network lifetime is also increased. The coefficient of variance, which is the ratio of the standard deviation to the mean multiplied by 100, shows the actual load balancing effect amongst nodes when applied to the average number of packets send per node. It is a measure for how well our algorithm is able to push data traffic towards the edge of the network. Additionally, we measured the average delay per hop, which is influenced by both effects (load reduction and load balancing) since both reduce the likelihood of collisions. Also, since we delay the route request it might happen that the route request itself takes longer than in standard AODV. Therefore, in our delay metric, we include delay caused by the route discovery process to have a fair comparison. Since we drop packets, there is the potential problem that our

packet delivery ratio (PDR) declines as routes might not be established and packets get dropped. Also our paths might become longer and the collision and interference probability increases. Therefore, we also monitored the PDR. To calculate the global view we followed the KISS (Keep It Simple, Stupid) principle and applied the simple average algorithm, which performed very well as shown before.

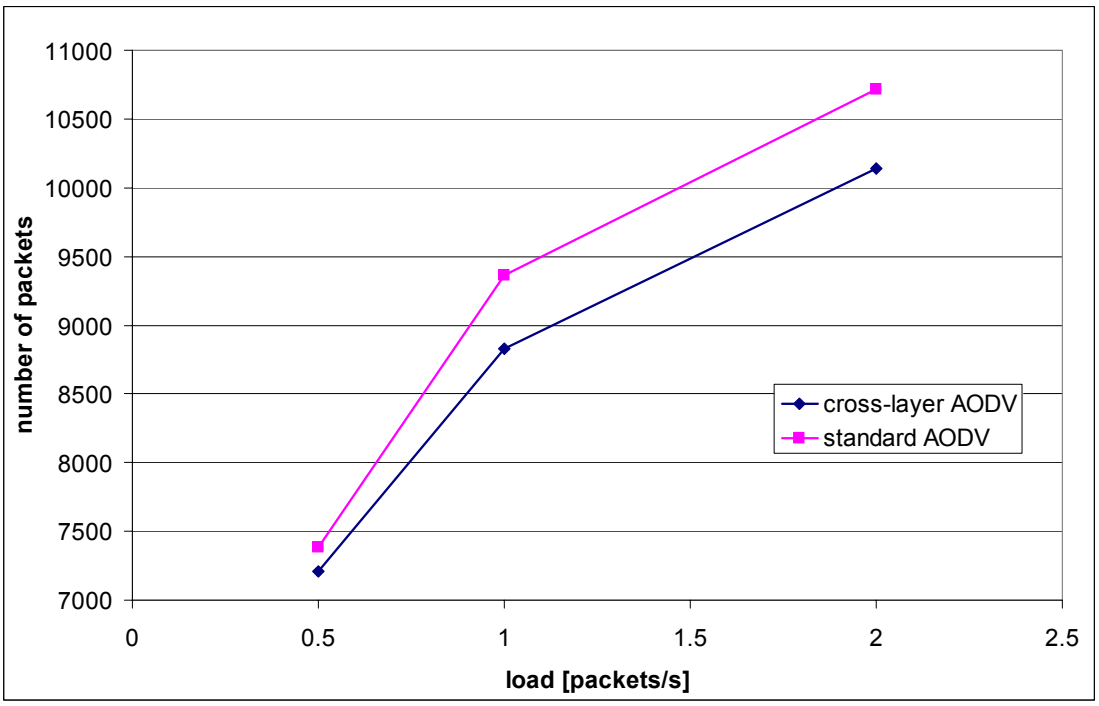

Fig. 5.10 Load reduction – Average number of packets send per node in differently loaded networks

Let us first consider the load reduction effect. Fig. 5.10 shows the degree of load reduction of cross-layer AODV by comparing its average number of packets sent per node against standard AODV's. The measurements displayed in this figure were taken from simulation scenarios of different amount of load as created by the traffic generator. As can be seen, as the load increases the load reduction effect grows. This is due to the reason that the load concentration problem in the core of the network also grows and more nodes will operate in phase three of the algorithm causing more nodes to disregard route requests. In this particular case as displayed in Fig. 5.10 we only have an average saving of up to 5%. The maximum saved amount was 15% for the topology geometry scenarios discussed later on. This shows that, although not the primary goal of the load balancing algorithm, our cross-layer AODV reference application is able to reduce the load.
In Fig. 5.11 the improvements of the per hop delay is presented, which is the time from a data packet being received at one hop till it is received at the next hop. As already mentioned the delay incurred by the route discovery phase is accounted for. This is done by including the time the data packet remains in the send buffer of the source while the route request phase establishes a path to the destination. In almost all tested scenarios the delay performance of our cross-layer approach was better than with the standard, layered approach, the only exception being highly dense networks (however, there was still a significant load balancing effect). The maximum delay improvement achieved in the tested

scenario parameter configurations measured was 65%. The figure shows the per hop link delays in networks of different topology geometry aspect ratios. An aspect ratio of 9 would translate to a network that occupies a plane which is rectangular that is 9 times as long as it is high. The reason why the per hop delay is dropping with an increasing topology aspect ratio can be explained by the fact that routes are on average obviously longer and therefore the route discovery delay does not that much influence the overall per hop delay. Additionally, the network gets a little less dense as the network borders become dominating resulting in less collision and congestion since the network core is thinned out. As can be seen, although the route request can potentially add delay at the route discovery phase the load balancing properties of our modified AODV influences the delay performance positively. This in principle reflects two things. One the one hand it shows that the delay potentially added by overloaded nodes during the route discovery phase does not influence the overall delay gain, as nodes on the routes that will eventually be utilized for the data traffic did not add any delay since they are not overloaded. In addition to that, it shows that the paths established by our cross-layer enhanced AODV are actually less loaded then standard AODV's paths where packets stay longer in nodes' send buffers, this way increasing the delay.

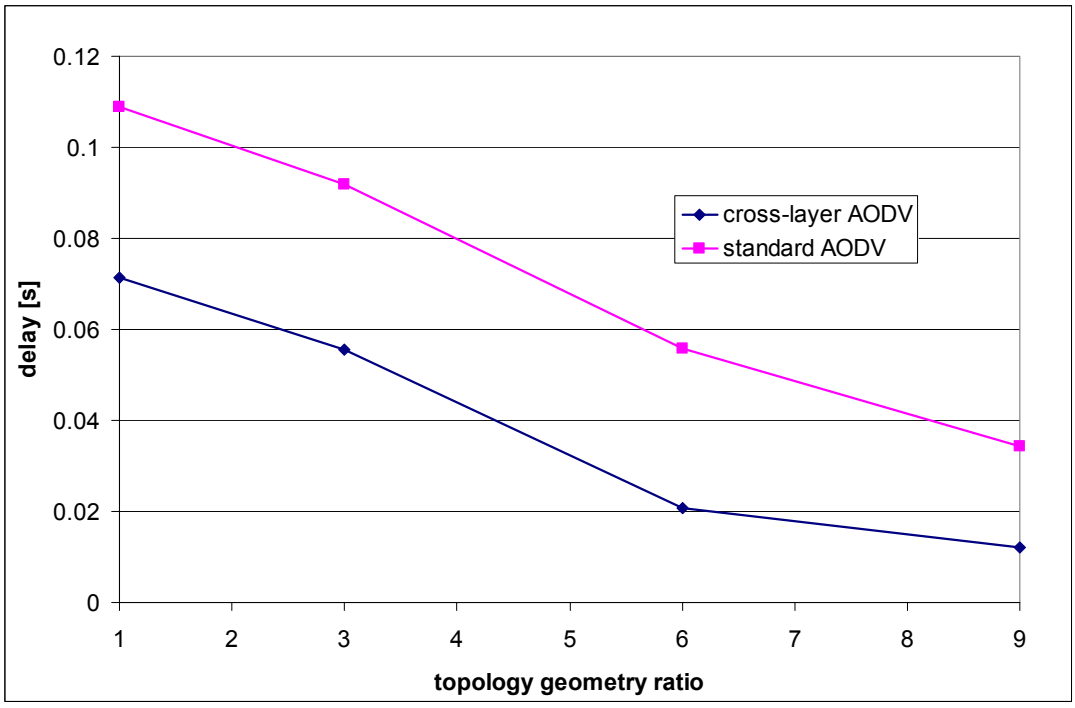

Fig. 5.11 Average per hop link delay in networks with different topology aspect ratios

One of the biggest problems of simple shortest path routing in ad hoc networks is that the core of the network is loaded extremely high. Some bottleneck nodes will carry a significantly higher load burden than any other nodes in the network. With our load-balancing enhanced AODV these hot spots experience a load relief as depicted in Fig. 5.12. The graph shows the load burden in number of packets sent during a simulation run in differently dense networks ranging from 50 to 150 nodes per km². As the figure shows the hot spot load reduction seems to be independent of the network density. In this particular scenario the reduction at

the most overloaded node goes up to 25%. This load relief effect at highly overloaded nodes was observed in all tested scenarios.

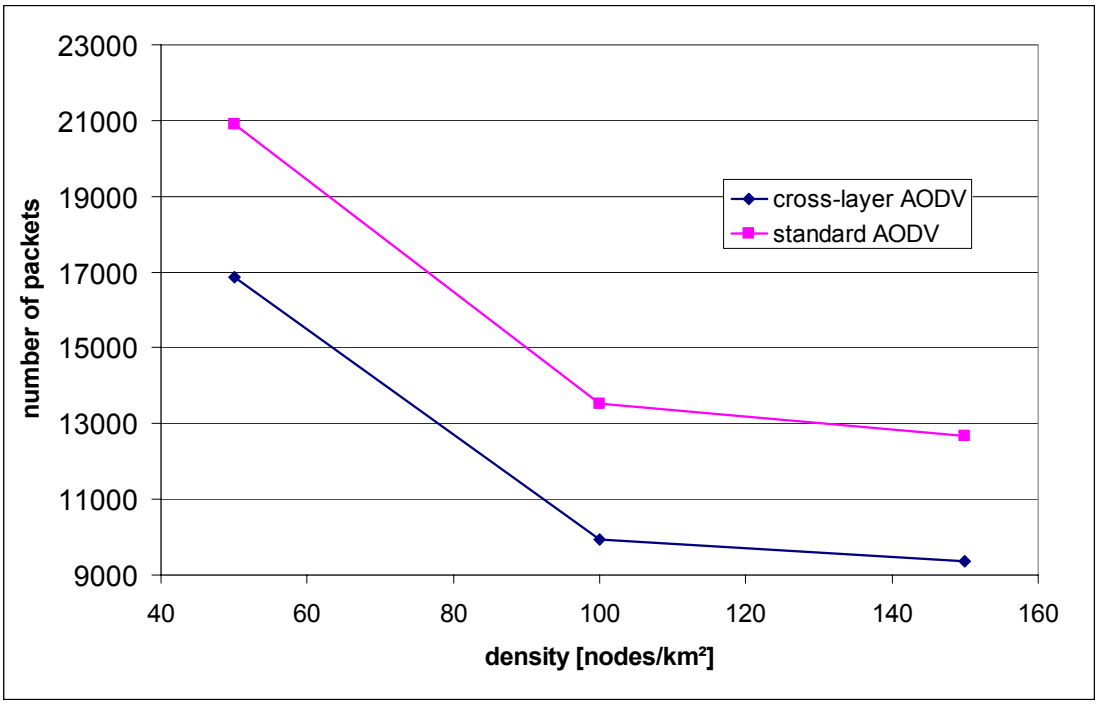

Fig. 5.12 Hotspot relief – Maximum packets sent per node in networks of different density

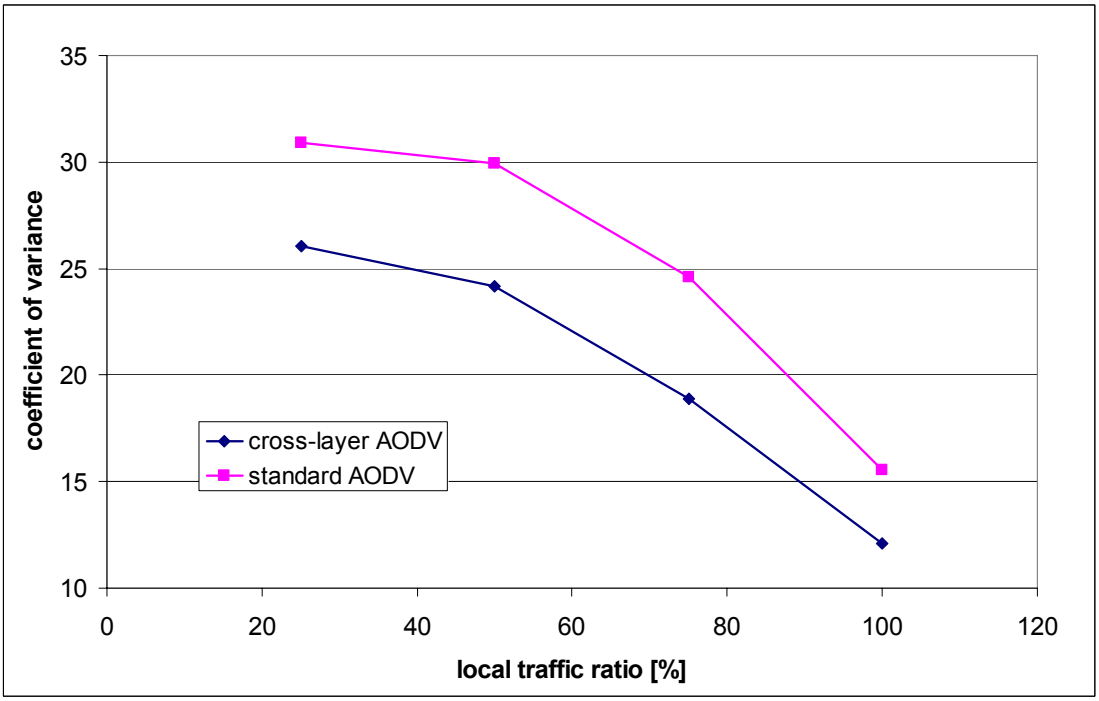

Fig. 5.13 Network wide load balancing effect expressed as the coefficient of variance of the individual nodal loads in scenarios with different traffic patterns (3 hop local)

The last performance metric for the evaluation of our load balancing reference application is the coefficient of variance of the average packets sent per node during a simulation run. This metric reflects the network wide load balancing effect, whereas the previous figure only had a focus on the worst network

bottleneck. We can not use the simpler standard deviation here, since the cross-layer AODV algorithm sends out fewer packets per node, which makes a direct comparison of the standard deviations impossible. In Fig. 5.13 the coefficient of variance is show dependent on the local traffic ratio. A local traffic ratio of 20% would simply mean that 20% of the packets a node sends are destined for nodes 3 hops or less away. In the tested scenarios the coefficient is up to 20% smaller using our cross-layer approach and can be observed in all scenarios.

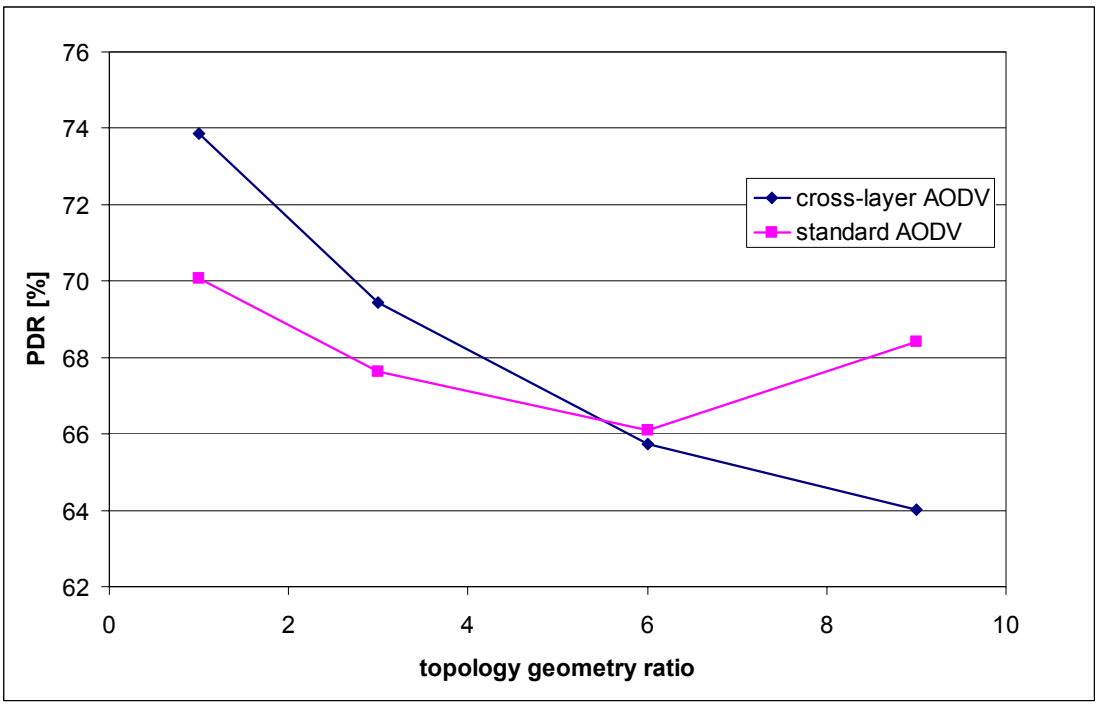

Fig. 5.14 PDR comparison in networks with different topology aspect ratios

In our tested mobility scenario our cross-layer architecture performed the worst in terms of load balancing compared to standard AODV. This is due to the fact that with total random mobility as generated by the random waypoint mobility model load balancing comes as a side effect since nodes traverse differently loaded areas constantly. Still, our algorithm achieves a coefficient of variance of 7.92 compared to 8.61 with the layered architecture and an average maximum load of 8941.5 compared to 9564.55 both improvements of slightly less than 10%. In this respect, the random waypoint mobility model can be seen as a worst case scenario for our load balancing scheme.

As already mentioned we also monitored the packet delivery ratio (PDR) achieved to make sure our scheme does still perform its core functionality efficiently. In only one single scenario the PDR dropped slightly below the reference PDR of the layered protocol stack. The PDR of the cross-layer approach was 64 compared to 68.4 in the topology with an aspect-ratio of 1:9 (see Fig. 5.14). This phenomenon can be explained by the fact that in topologies with such an aspect ratio, or an even higher one, there are only few paths through the network. Those are extremely overloaded compared to nodes at the edges of the network, for example. Nodes on these paths will sooner operate beyond the drop bound. Therefore preventing a route to be established, causing the source to drop the data packets after some unsuccessful route discovery attempts. On the other

hand our scheme guarantees therefore a higher quality for data packets through established routes as seen before.

5.1.5. Conclusion

The contribution described in this chapter was twofold. The first one being a novel load metric, which can be used for cross-layer load balancing, the second one was an exemplary application of this metric using our CrossTalk architecture. The load metric itself is adapting timely to load changes and reflects the actual load in the network very well. Additionally, it cannot be exploited easily by selfish nodes to be relieved from cooperation duties such as packet forwarding.
The load balancing adaptation of AODV showed that the protocol adaptation process to make a protocol utilize cross-layer information is straight forward and requires only little changes. The analysis of this approach revealed that CrossTalk's Global View mechanism delivers a high quality network-wide view and that protocol adaptations on its basis are therefore feasible. The load balancing algorithm itself performed very well and the per hop packet delay could be reduced by up to 65%. In addition bottleneck nodes were relieved by up to 25% of their load burden and the coefficient of variance of the packets sent per node was reduced by up to 20%.

5.2. Mobility Adaptations – Solving Ad Hoc Networking Issues Using CrossTalk

Node mobility is one of the unique system dynamics an ad hoc network exposes. The performance of the routing protocol heavily depends on the mobility degree and static parameterization of such protocols fails to make them operate efficiently in a broad range of network conditions ([87],[88] and [89]). Other networks suffer from node mobility too, such as cellular networks, but the problem's complexity is greatly reduced by having base stations. This way, mobility is only a one hop issue which can be solved by hand-off mechanisms. Mobility in ad hoc networks is much more severe as on a multi-hop path, arbitrary nodes can move causing routes to break constantly in a non-deterministic fashion. A source node can therefore not always determine in a timely manner that mobility has caused a route to break whereas in cellular networks this issue is more trivial. In an ad hoc network there simply is no infrastructure that can assist in solving mobility issues. This makes mobility a case for distributed, network-wide adaptations and therefore CrossTalk's Global View mechanism can be used to alleviate the problems caused by mobility [90].
Again two fundamental things have to be done. First of all, metric that represents node mobility accurately has to be found. Preferably this metric has its origin inside the protocol stack and reflects the topological change rate well. Secondly, this metric has to be utilized by the routing protocol as it has to compensate for the effects of node mobility.

5.2.1. Related Work

There has been extensive work on mobility models, mobility metrics and mobility adaptive protocols which clearly shows the need to solve and study mobility

related problems. Mobility models are designed to be used in simulations as a way to mimic or represent real world node movement. They can be categorized as either trace-based or synthetic models [86]. Trace based models are mobility patterns recorded from observations whereas synthetic models are a mathematical representation of node movement which tries to approximate real-life mobility patterns. Clearly, trace based models are more realistic as they were captured by observing a real-world environment but at the same time they are difficult to generate as it takes a significant amount of effort just for the observations and also a huge amount of data has to be collected. Therefore synthetic models are used nearly exclusively as they reduce the complexity of mobility modeling significantly and can be changed easily to simulate many different mobility scenarios.

The mobility pattern can have a significant impact on the network performance and therefore several mobility models with different characteristics should be tested to evaluate a routing protocol [86].

Mobility metrics on the other hand try to capture and express the degree of mobility independent of an underlying mobility model. They can be categorized into real-world applicable and artificial. Artificial are those metrics that are based on information not available in real-world scenarios or parameters from mobility models. Those metrics are only useful to compare simulation results but cannot be applied to enhance a protocol. Such metrics for example utilize the number of nodes in the network, the transmission range, relative velocity between nodes or the pause time of a mobility model just to name a few. Mobility metrics will be described in detail as we need to base our algorithm on a real-world applicable and accurate mobility metric. Mobility models are not the primary focus of this section. A good overview of many commonly used and referenced models can be found in [86].

In [91] Kwak et al. derive a mobility metric to be able to compare simulation results against each other, in the case that different mobility models or differently parameterized models were used. As the performance of a routing protocol and the degree of mobility are very closely related such a metric would enable a direct comparison. As they are only concerned with simulation environments their mobility metric is not real-world applicable at all. It operates on a remoteness function which is a function of the distance between two nodes. Furthermore, it works on a network-wide scale knowing the distance from each node to every other node and the transmission ranges.

Tsumochi et al. [92] analyze several mobility metrics many of which are not very accurate or widely applicable such as the pause time of a mobility model. They even propose to use the number of neighbors which seems odd. The models they propose that reflect mobility more accurately have in common that they are not real-world applicable as they utilize information such as velocity and accurate distance knowledge, i.e. some positioning system and the transmission range. The only exception is the frequency of link state changes but they do not describe a way how to calculate it at a node. So it can be assumed they focus on simulations settings only.

Some important work has been done by the authors of [93] and [94]. They propose various mobility metrics but again many of which utilize information realistically not available at a network node such as the relative speed between nodes. Some of those metrics are based on the connectivity graph of the network.

These therefore express the topological mobility. As they use the simulation time or the number of nodes in their calculations they can not be used in real world scenarios as such. But the general principle can be applied in real-world settings on a per node basis. Their results show that some of these metrics do not reflect the mobility degree very well and interestingly they find that the link duration metric is able to capture the effects of mobility very well.
In [95] a simplified mobility model is presented, mainly designed to analyze the suitability of two different mobility metrics, the link change rate and the link duration. The study also tries to derive a relationship between the link duration and the mean residual duration of a path. The authors come to the same conclusion as the authors of [93] and [94] that the link duration is a good metric which reflects the degree of mobility very well and also it is suitable to derive the residual lifetime of a path independent of the underlying mobility model. In other words they show that the lifetime of the route is in fact a function of the link duration. This is a very important aspect. Also in accordance with the previous paper they showed that the link change rate fails to have the above mentioned properties.
[96] has a strong focus on the real-world applicability of a mobility metric. They therefore define five core requirements a good mobility metric should fulfill which are:

- "Computable in a distributed environment without global network knowledge"
- "A good indicator of protocol performance"
- "Feasible to compute (in terms of node resources)"
- "Independent of any specific protocol"
- "Computable in real network implementations"

Using CrossTalk, the first requirement can be disregarded to some extend, i.e. for the network-wide mobility. It is valid though for the computation of local load at a node. These requirements serve as a good guideline for a mobility metric in general. The main contribution of [96] is the analysis of two mobility metrics which are the link change rate and the link duration. The results show that the link duration is the better metric and reflects the routing protocol's performance much more accurately. The problem with the calculation of the link duration metric is the determination of the time window size in which to observe the link duration. This issue is left open by the authors who recognize that the choice of the time window is critical but only mention that this issue is under investigation.
In [97] the previously mentioned link duration metric is combined with the link change rate as the authors believe this to be a more precise mobility metric. In their simulations they show the correlation between the metrics link duration, link change rate, their own metric and the maximum speed of the random waypoint mobility model and the transmission range. What they do not show though is if the mobility metric is able to reflect the routing protocol performance very well as in [93]. Also other mobility models were not evaluated which would have been important.
A mobility metric based on the received signal strength of two successive packets is proposed in [98]. Such a metric suffers from the problem that the received signal strength is not a good indicator of mobility as it can vary drastically in

short time intervals [99] even without mobility. The authors of [98] apply their mobility metric to choose appropriate cluster heads.
Tan et al. [100] basically use the same mobility metric for their simulations which has the aforementioned problems in real-world settings. The adaptation they propose is based on ADOV enhanced by two additional mechanisms. The first one is the limited forwarding of route requests. When a route request arrives at a node it checks if the mobility metric towards that node is above a certain fixed threshold. If this is the case, it will drop the route request. This makes the threshold vital and a strongly fluctuating mobility metric can cause many packets to be dropped increasing the likelihood of a route never to be established. The second mechanism limits the amount of intermediate nodes sending a route reply. If an intermediate node receives a route request and knows a route towards the destination it first checks if the mobility metric towards the next hop is beyond a certain threshold. In case this threshold is violated the route request is further propagated and no reply is sent.
An estimation of the residual link lifetime is attempted in [101] by observing past link lifetimes. They propose to have a finite array where each array element represents a certain link lifetime interval. A link that was up for a certain amount of time would increase the array element by one which interval it lies in. This is a rather coarse grained approach and the array dimensions need to be chosen carefully which in the real-world would be difficult. Also if the mobility changes the old values inside the array are never purged and they will always keep on influencing the calculations. Based on this array link lifetimes are estimated which can be used for routing decisions.
The Route Fragility Coefficient is a metric proposed by Tickoo et al. [102]. It is also utilizes the received signal strength as a basis and in addition assumes the free space propagation model and some constant to be known which depends on antenna gains and the wavelength of the transmission. By knowing these parameters they calculate the relative velocity between nodes and from that they derive whether nodes are moving closer or whether they move apart. The combination of those two metrics on a given path is termed Route Fragility Coefficient, obviously suffering from the simplified model assumptions as already stated before. A subsequent simulation study utilizes their metric in a modified version of DSR. Since they have to alter DSR's control packet headers their version will not be compatible with standard DSR. Their scheme also demands a change in the MAC layer as it has to keep track of successive received signal strengths and is therefore a cross-layer scheme as this information is passed to the routing layer. During the route discovery phase every nodes adds their information about the relative expansion or contraction towards the previous hop. The destination waits for a certain amount of time and selects to most suitable path as proposed by many other protocols before having the aforementioned issues.
In [103] a metric for the quality of a node is presented which also incorporates the stability of a node, i.e. an indirect measure for its degree of mobility. It is very simple the ratio of the number of neighbor nodes that remain in its vicinity in a certain time frame and the total amount of neighbors during that timeframe. This metric is relatively coarse grained and relies on a time window which might be difficult to choose. In combination with the buffer and power level of a node it

was shown to improve the selection of nodes that form a forest, i.e. a backbone inside an ad hoc network.
Many protocols have been proposed to use location information and as this is not directly related we do not in detail describe these approaches but one case has to be noticed which is Adaptive Location Aided Routing from Mines (ALARM) [104]. It not only makes use of location information, as for example provided by GPS, but it also takes the previously mentioned link duration into account. ALARM is a hybrid protocol. Based on a link duration threshold on the source route it switches between the Location Aided Routing (LAR) [105] protocol or utilizes a directed flooding algorithm when link durations are shorter than the predefined threshold. The authors state that there are some critical threshold values in their system that are difficult to choose. Again the selection of the window size for the calculation of the link duration is not addressed properly.
The mobility adaptive routing scheme MARio is presented in [106]. It is thought as a component in juxtaposition to the protocol stack that can provide link or path duration statistics. The statistical information about the route lifetimes can be used to establish a new route prior to the expected breakage of a route in use or it can be used to invalidate a route prior to its expected breakage. In the presented application example of MARio, it gathers statistics about every route constructed by the DSR protocol and uses an exponentially weighted moving average to calculate a path duration metric. From the description in [106] it becomes not clear whether the actual path lengths are differentiated. In case they are not the thresholds chosen for a route invalidation and for route pre-fetching might not be accurate. Their simulation study also reveals that for the Reference Point Group Mobility model, their approach does not perform well, which is bad as it is the more realistic model compared to the other tested model, the Random Waypoint mobility model.
Hu and Johnson [107] evaluate different mobility metrics in terms of how well they reflect the routing difficulty, i.e. in terms of routing overhead and route errors. They find, that from the tested metrics the minimal route change metric correlates the most accurately with the routing effort. They do not test the link duration metric though.
The Adapting to Route-demand and Mobility (ARM) control mechanism presented in [108] can be utilized by only proactive routing protocols. ARM consists of two mechanisms the *update-period control* and the *update-content control*. The update-period control is the interesting mechanism as it is dependant on a mobility metric. It determines the interval in which routing state updates are sent. The mobility metric itself is based on the one-hop neighborhood fluctuations during a routing update period. The performance of ARM was exemplary shown using the DSDV routing protocol.
Also designed to work with proactive protocols is the algorithm proposed in [109]. The utilized metric is the one already mentioned in [97]. The metric is utilized as an input into a feedback controller which adapts the frequency of routing state updates. In their exemplary application of their component inside the DSDV routing protocol they could not always outperform statically parameterized versions of DSDV.
There is a lot more work on mobility in ad hoc networks such as mobility predictions and more approaches based on location information. We do not

describe those here as they are not directly related to the work presented in the following chapters.

5.2.2. Metric Generation and the Mobility Adaptation Extension to AODV

The previous chapter introduced several possible mobility metrics but only few of which satisfy the criteria of [96]. The most promising metric appears to be the link duration as it was shown by several independent studies to reflect protocol performance metrics very well in addition to satisfying the previously mentioned criteria for a good mobility metric. The link duration is defined as the average time a link to a neighboring node exists (a formal definition can be found in [93]). In other words the link duration is a metric which quantifies the time two nodes are in each others transmission range on average during a certain time period.
The question that arises is how to exactly calculate the link duration metric locally at a node. First of all, a node must be capable of identifying nodes that arrive and depart from its transmission range. A lot of protocols such as AODV have a mechanism to deal with this issue. As already stated, AODV sends out `Hello` messages periodically to maintain accurate information about a node's network neighborhood. By sending out these beacons, nodes are aware of each other when they come into each others transmission range. On the other hand, if these messages are not received anymore for some time, the link between the two nodes is interpreted as being down. But even without an extra mechanism, passively gathering link duration times as described in [96] can be used making the mechanism to measure link durations somehow protocol independent. The only problem identified with the calculation of the link duration is the choice of an appropriate window size in which to observe the link durations. When chosen too small, long lived links are not accounted for correctly. As a result the calculated link duration would be too small indicating a higher mobility than actually present. Since the mobility is a time-varying measure, the link durations metric must reflect the current mobility accurately. Therefore, when the window size is chosen too large, the link duration metric fails to account for changes in the mobility as old data influences the metric calculation. This means that the size of the time window must be dependant on the mobility degree itself and its relative change rate.
The problem of determining an appropriate window size is the reason why we chose to use a different mechanism to organize the individual link duration measurements. Since an observation period should be dependant on the degree of mobility and at the same time it is used to derive the mobility metric itself there is an intrinsic conflict which has to be solved differently. To calculate the local average link duration at a node, we use two separate lists. Both lists are used to calculate the average link duration experienced locally at a node. One list contains all currently active links (links up list – LU), i.e. it contains all nodes that are currently in the local nodes network neighborhood. Whenever a node comes into transmission range, it is added to LU. The mechanisms to identify a node that intersects with a node's transmission range were described before. The second list contains links that are already down (link down list – LD), i.e. links to nodes that have entered and already left a local node's transmission range.

The amount of links stored in LD should match the amount of links in LU if possible. To achieve this, LD's size is adjusted in a way that it never exceeds LU's size. In other words, LD's size is determined by the amount of links that are currently active. When a link goes down, it is moved from LU to LD. If LD is smaller than LU, the entry is simply added. Otherwise the entry replaces the oldest entry in LD. If necessary, LD's size is further adjusted by removing the oldest entries in LD until the sizes of both lists match.
The intuition behind this mechanism is as follows. First of all, the overall amount of links stored for the average link duration calculation is determined by the amount of one-hop neighbors, i.e. by the density of the network. Ideally there are twice as many link duration entries in both lists together as there are one-hop neighbors. This way excessive storage consumption due to a badly chosen window size is prevented. The second reason for this mechanism is that something like a window size intrinsically exists. The higher the mobility the faster links move from LU to LD. So the higher the mobility, the faster "old" links are removed from LD, which is equivalent to a shorter window size. With less mobility links stay longer in LD, i.e. the intrinsic window size is bigger. In other words, the window size is controlled by the mobility degree itself.
Each entry in LU comprises the node's address or unique ID and the time the link was established. The entries in LD additionally comprise the time when the link went down. Using both lists, our mobility metric is calculated as follows. In a first step, the average link duration of the links in LD is calculated, which gives us a snapshot of the average links duration of the recent past. Afterwards, in a second step, all links found in LU that exist longer than the average link duration of the links in LD are also averaged. Both values are combined to give the average link duration which we use as our mobility metric. Let's look at a mathematical representation of our algorithm:

$$d_{LD} = \sum_{n=1}^{s_{LD}} L_n \,,\ \{L \in LD\} \qquad (1)$$

$$d_{LU} = \sum_{n=1}^{s_{LU}} L_n \,,\ \left\{ L \in LU \middle| uptime > \frac{d_{LD}}{s_{LD}} \right\} \qquad (2)$$

$$\bar{d} = \frac{d_{LD} + d_{LU}}{s_{LD} + s_{LU}} \qquad (3)$$

Formula (1) denotes d_{LD}, which is the sum of link durations found in LD, where s_{LD} represents the size of LD. Formula (2) calculates d_{LU}, which is the sum of link durations in LU satisfying the condition that they were up longer than d_{LD}/s_{LD}. The calculation to obtain the average local link duration finally is show in formula (3). It is sum of (1) and (2) divided by the size of LD plus the amount of link durations found in LU satisfying formula (2)'s condition.
The reason why we developed this algorithm is that is can accurately reflect the current mobility. If the mobility is slowing down, the links in LU over time will gain more weight than the links in LD. This automatically adjusts the link duration metric. If we only considered links that are already down, the link duration metric would start to adjust much later. The longer lived links would

need to expire before the mobility metric reacts to the decreasing mobility. As a worst case scenario consider a node that moves from a mobile into a static network area. It would still determine that it is in the mobile part of the network since the links around it would not expire. This is prevented using our approach. But generally, there is a potential problem with also taking the link durations of active links into account. Very recently established links can significantly influence the average link duration metric since their link duration is very low. This is why we chose the average link duration of LD as our dynamic threshold for the inclusion of links in LU. If the mobility starts to increase links will start to move faster from LU to LD. That means that the average link duration of past links will decrease, i.e. the threshold for the inclusion of existing links is lowered. So the mobility automatically adjusts the calculation of the link duration metric to reflect the increased mobility.

With the way we calculate our mobility metric, we eliminated static parameterization, in conformance with one of the goals as stated in section 1.2. The mobility degree and change rate themselves steer the way the average link duration is calculated. In addition, the network structure determines the choice of link measurements used for the mobility metric calculation. Our metric is also in conformance with the characteristics of a good mobility metric as described in [96].

To establish a network-wide link duration metric, CrossTalk's Global View mechanism is used. As an exemplary protocol to incorporate a mobility adaptation mechanism we chose the again the reactive AODV routing protocol. The description of its general operational behavior can be found in section 5.1.1. It was not chosen because it is the most suitable protocol for mobility adaptations, but because it is one of the best studied ad hoc routing protocols. Additionally, AODV does not rely on any extra information such as location information, making it more generally applicable. AODV should only be regarded as an exemplary protocol. We strongly believe that enhancements using mobility information can be done for any ad hoc routing protocol either reactive, proactive or hybrid. Even protocols already enhanced with location information provided by GPS were shown to benefit from link duration measurements [104]. Let us look at some examples to show that every routing protocol in principle exposes mechanisms that would benefit from mobility adaptations. The main application area of a mobility metric is the dynamic adaptation of protocol parameters. A proactive routing protocol that periodically broadcasts routing state information could adapt the update interval. This update period is the key parameter for table-driven routing protocols. If the mobility is high the update interval must be short to keep the routing state of each node up-to-date. If the mobility is low, resources such as bandwidth and energy can be saved by reducing the amount of redundant and unnecessary update messages.

Hybrid routing protocols can also benefit immensely from mobility adaptive mechanism. The general idea of these protocols is that a proactive protocol is used in a radius around each node, which keeps the overhead of route maintenance to a strictly localized area but routing performance in these areas is high. Only for distant communication reactive routing strategies are employed as the maintenance overhead for distant destinations would be too high. A mobility metric could be the basis for the choice of the zone radius. The lower the mobility

the bigger the proactive zone should be and vice versa. For the proactive part of the hybrid protocol the above stated update interval adaptation also applies.
Reactive routing protocols such as the one used in our experiments have several possible mobility adaptation opportunities. One is the tuning of the neighborhood maintenance interval. For AODV this would translate to adapting the `Hello` message interval. In low mobility scenarios frequent `Hello` messages consume resources without any benefit. The route cache of reactive routing protocols can also be optimized by dynamically choosing an expiration time for the various routes. All of the above mentioned protocols can be optimized towards choosing the most stable route according to a mobility metric.
For our link duration metric used in the following experiments, we utilize the mechanisms provided by AODV, i.e. the link duration metric is generated according to the link duration feedback of the `Hello` message mechanism. But as mentioned earlier, other available mechanisms could be used such as layer-2 notifications comparable to the ones provided by IEEE 802.11. Whenever AODV recognizes a newly arrived one-hop neighbor by overhearing a `Hello` message for the first time, a corresponding entry is added to our links up list (LU). When AODV determines that a link went down by not having received three consecutive `Hellos`, this entry is moved to the links down list as described in the previous section.
We identified several possible enhancements to make AODV mobility-adaptive. They include:

- Tuning of the `Hello` message interval
- Dynamic calculation of the route cache timeouts
- Improved route discovery and route selection

Making the `Hello` message interval dependant on the mobility degree is intuitive at first. As already mentioned, if the mobility is low, there is no need to excessively send out `Hello` messages since the local connectivity changes slowly. But if local changes should be detected timely, the `Hello` mechanism should remain in a relatively short interval. Increasing the `Hello` interval also increases the time that a link change is detected for two reasons. First of all if a link fails at the beginning of an interval than the remaining time of that interval the link failure will be undetected independent of AODV's parameterization. The second reason for an increased failure detection time is that AODV normally waits until more than a single `Hello` message was not received to mark a link as being down. Additionally, fast moving nodes might be completely undetected if the `Hello` interval is altered as they might not send a beacon while in the transmission range of some nodes. The `Hello` mechanism must also be synchronized amongst all nodes. That means that a node has to know in which interval another node sends out `Hello` messages to make sure that it can correctly identify when a node is not in its transmission range any more. This adds a degree of complexity to the alteration of the `Hello` interval. In our case, there is the additional dependence of our mobility metric on a timely detection of topological changes in the direct transmission range. Therefore, we chose not to alter the `Hello` mechanism of AODV.

The route cache has the advantage that it provides an accurate routing state over some time. Its accuracy clearly depends on the assigned route lifetime, i.e. on the time routes remain as being active inside the cache. Having a static route lifetime has one major disadvantage. If chosen to be too large, the node will start using stale routes, very often resulting in errors, which in turn results in increased routing overhead. If chosen too small, routes are purged from the route cache although they are still alive, possibly forcing unnecessary route requests. That is why we chose to make the route timeouts adaptive. To do this, we employ a heuristic. Consider two nodes. On average they are in each others transmission range for the amount of time equal to our average link duration metric provided by the Global View component, assuming of course that it is correct. Looking at each node individually and assuming a uniform distribution of the times the links became active in a interval of the average link duration, the expectation value of the remaining lifetime of a link is:

$$E(\bar{t}_{ld}) = \frac{\bar{t}_{ld}}{2}$$

whereas $\bar{t}_{ld}$ is the average link duration provided by CrossTalk.

The farther a destination is away the smaller the overall lifetime of the path will be as more intermediate mobile nodes can cause the failure of the route. We therefore calculate our route lifetime for a newly found path as:

$$t_r = \frac{E(\bar{t}_{ld})}{\#hops}.$$

For our scheme we need to slightly modify the entry format of AODV's route cache. We add the time a link or route becomes active to the information stored in the cache. Whenever AODV wants to update the timeout of a route, it will use this information to see if the link has been active for t_r or longer. If it has been active longer, there is an increased probability that this link will fail in the near future. From this time on, the route lifetime is only extended by the value defined by standard AODV to make sure the route is purged fast from the cache once it is not used any more.

The second mechanism of AODV that was modified is the route discovery and selection process. It shares some similarities with the mechanism developed by Tan and Seah [100]. A major difference is that our mechanism does not rely on heuristics to discard route requests, which makes it possible that no route is established at all. When a data packet has to be sent, our mobility-adaptive AODV tries to find a route in the route cache, which up to this point is also what standard AODV would do. If it finds a route to the destination it first checks if the remaining lifetime to the next hop on that route exceeds $E(\bar{t}_{ld})$. If it does, it will start to use that route; otherwise it will issue a route request to find a more stable route. Standard AODV would simply use this route. A failure would then cause AODV's route error mechanism to become activated and an additional route request would be issued. During the route discovery phase, intermediate nodes which have a route to the destination will perform the exact same test. This way, old routes will be avoided in favor of routes with a higher life

expectancy. In addition, we apply a second mechanism to establish the most appropriate routes. When a route request arrives at an intermediate node and it does not have a route to the required destination in its route cache, it will evaluate the remaining lifetime to the previous hop, i.e. to the node that it received the route request from. If the remaining lifetime exceeds $E(\bar{t}_{ld})$, it will simply forward the route request. If not, i.e. if the link was up already for more than half of the average link duration, an intermediate node will delay the forwarding of the route request. The maximum delay is given by the `Hello` interval of AODV. It decreases proportionally with an increasing remaining lifetime. This way the probability is decreased that a route is established which leads through links which will statistically fail soon. In other words, this mobility-adaptive scheme automatically favors links on a route with statistically long remaining lifetime. This way, data packets are routed on more stable links. Of course the mechanisms presented are all based on a probabilistic approach.
Again, what has to be noted is that the mobility adaptive AODV is still interoperable with its standard version counterpart. Neither message headers are altered nor is the general routing algorithm changed in a way that standard AODV would fail to make correct routing decisions.

5.2.3. Experimental Setup

For the experimental evaluation of our mobility metric and our mobility-adaptive version of AODV, we used again the network simulator ns-2. For the reasons stated in section 5.1.3 we chose the AODV implementation of the Uppsala University. We evaluated 200 node networks in a variety of mobility scenarios. It is important to choose a variety of models with different characteristics as interdependencies between models and metrics have to be avoided. By choosing different models interdependencies can be identified. Many previous studies only rely on a single mobility model, predominantly the Random Waypoint Mobility model. We chose 4 mobility models which comprise Random Waypoint, Reference Point Group Mobility (RPGM), Manhattan and Freeway [86]. They were chosen because they all represent very different scenarios and differ strongly in terms of nodal dependencies, directional restrictions and geographic constraints. They also vary in the amount of necessary parameters and interpret and use common parameters differently. For example, the speed metric used in the RPGM model is interpreted as the mean speed whereas in the Random Waypoint model the speed metric defines the maximum allowed speed. Due to these differences, they are very difficult to be compared directly but at the same time cover a broad spectrum of mobility patterns and are therefore a good subset of mobility models to test our metric and adaptation mechanisms.
For the RPGM model, we chose 20 groups with 10 members each and a speed mean between 2 and 20 ms/s. The simulation area had a size of 2400x600 meters. For the Random Waypoint model, we always set the maximum speed to be also the minimum speed to prevent speed decay as described in [110]. We used maximum speeds between 1.4 and 20 m/s and always a pause time of 10 seconds. The network area was 2828x707 meters. For the Manhattan and Freeway mobility model, we chose maps similar to the ones found in [93]. The maps for the Manhattan mobility model and the Freeway model are displayed in Fig. 5.15 and

Fig. 5.16 respectively. The minimum speed was always set to be half of the maximum speed which varied between 2 and 20 m/s for both models.

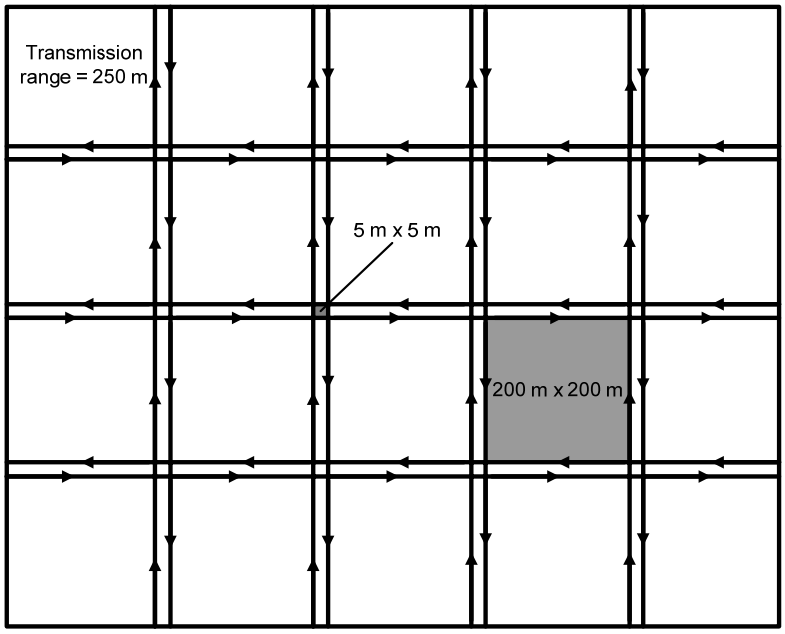

Fig. 5.15 Manhattan mobility map

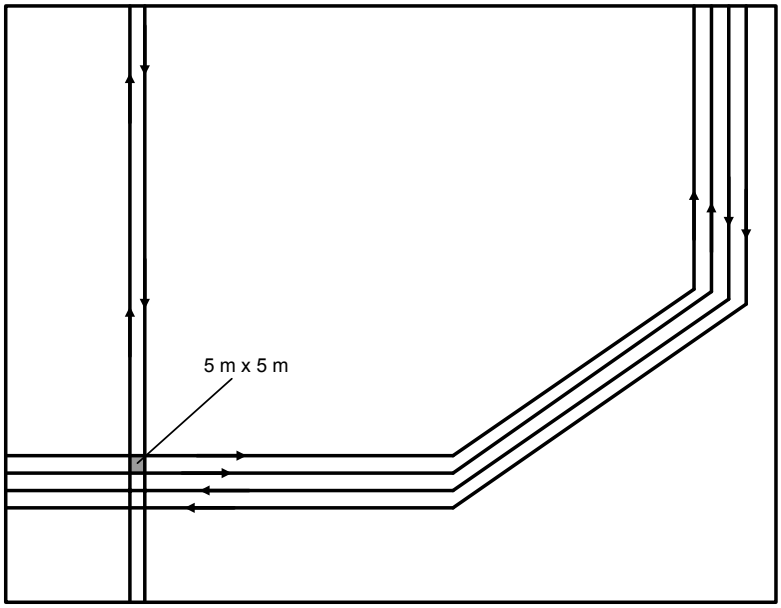

Fig. 5.16 Freeway mobility map

The traffic pattern applied defines 3 classes of nodes. Class 1 represents the high load nodes. These nodes generate a packet once every 1.5 seconds and switch destinations randomly every 20 seconds. Only 15 of those nodes existed per simulation run. Furthermore, there were 30 second class nodes generating one packet every 3 seconds and which change destinations every 40 seconds. 60 nodes in the network belonged to the 3rd class issuing a packet in an interval of 6 seconds and changing destinations every 60 seconds. So in total roughly 50% of the nodes actively generated data packets. A simulation run lasted 1000 seconds, of which the first 200 seconds were used to settle the network. All statistics and measurements were gathered in the remaining 800 seconds.

Table 5.2 Mobility-adaptations experiment parameters

Mobility model	Parameter range
RWP	1.4 m/s – 20 m/s, pause time 10 s
RPGM	20 groups, 10 member per group, 2 m/s – 20 m/s
Manhattan	2 m/s – 20 m/s according to map
Freeway	2 m/s – 20 m/s according to map

5.2.4. Experimental Results

An analysis of the quality of the global view is largely omitted here as it yielded similar results to the ones found in section 5.1.4. Only a brief description of the findings is given later for completeness sake. For our mobility metric though we have to make sure that it actually is able to reflect protocol performance accurately as one of the requirements from section 5.2.1 defines a good mobility metric as a good indicator for protocol performance. We use 3 protocol performance metrics to prove that our mobility metric reflects the protocol performance very well. We use the packet delivery ratio as a measure for the success rate, the amount of protocol packet transmissions per data packet delivery as a measure for the protocol overhead and the average end-to-end delay including the route discovery process as a measure for the temporal efficiency.

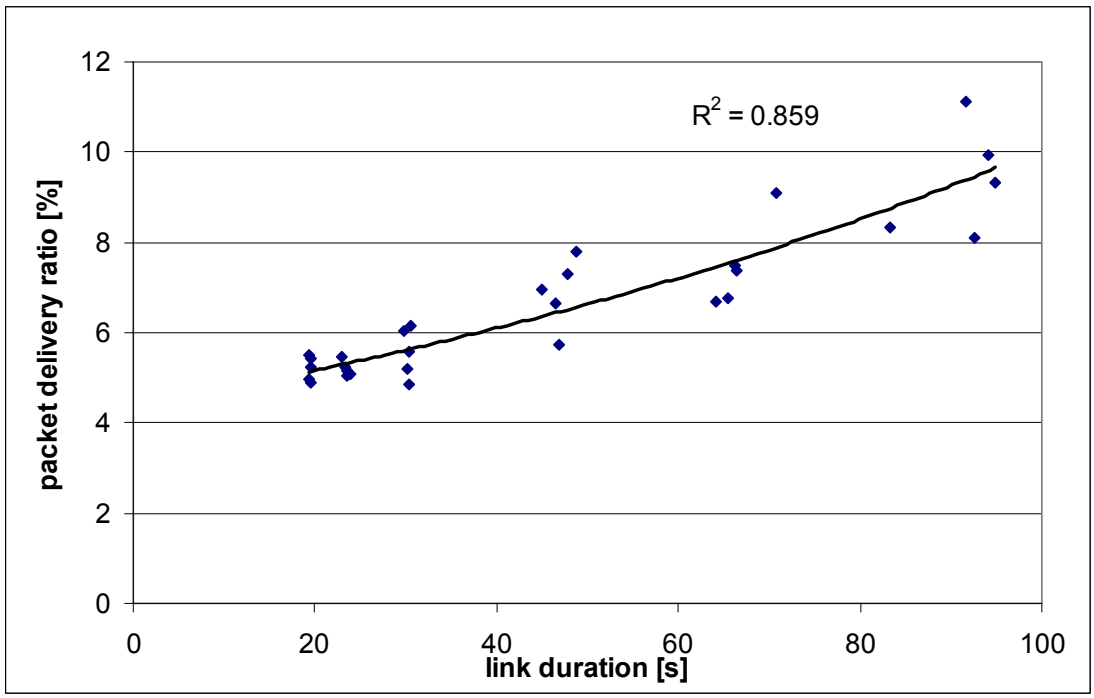

Fig. 5.17 Correlation between link duration and PDR in Manhattan mobility scenarios

To show that the link duration metric reliably indicates standard AODV's performance, the coefficient of determination between the three performance metrics and the mobility metric were calculated for all tested mobility models

and scenarios. The link duration used to construct the graphs below is the average link duration provided by CrossTalk's Global View component of each node in the network.

Fig. 5.17 shows the relationship between the link duration and the packet delivery ratio (PDR) for standard AODV in Manhattan mobility scenarios. Each data point represents a simulation run. As can be seen, the PDR increases with increasing link durations, i.e. with less mobile nodes, as can be expected. The fitted curve has a coefficient of determination of nearly 86%. In all tested scenarios the coefficient of determination for the PDR yielded similar results with a maximum of 87% for the tested Freeway mobility scenario. In other words the link duration metrics has a strong correlation with the success rate of the routing protocol.

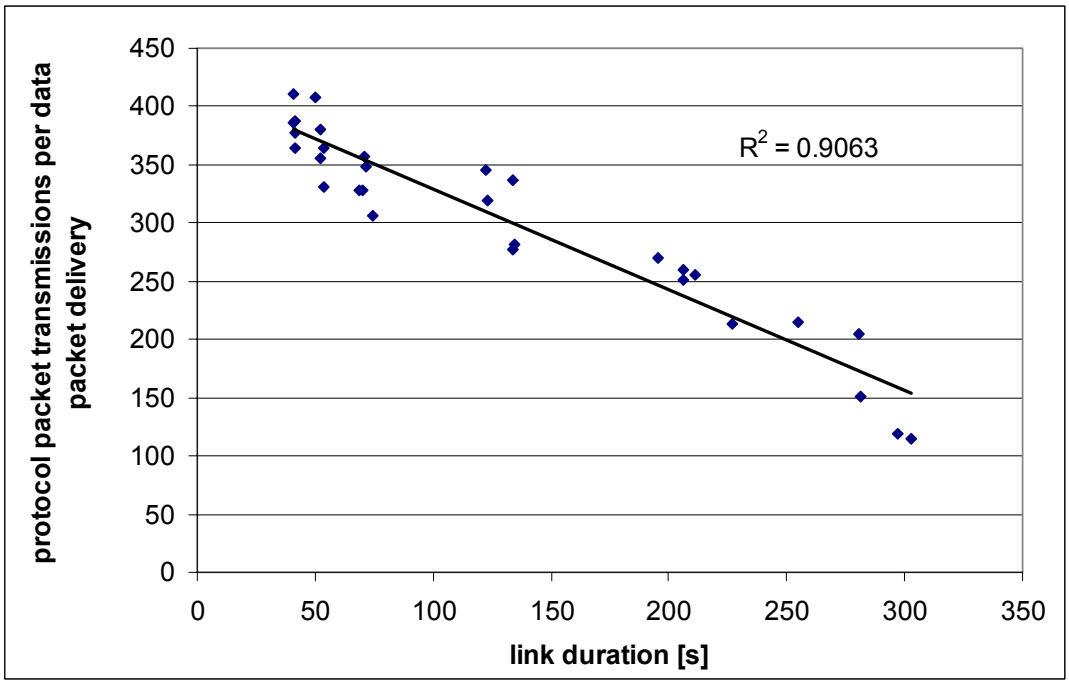

Fig. 5.18 Correlation between link duration and routing overhead in Freeway mobility scenarios

Fig. 5.18 displays the protocol packet transmissions per data packet delivery versus the link duration in the tested Freeway mobility scenarios. For shorter link durations, the protocol of course generates a higher overhead to deliver a packet successfully as routes tend to break more frequently. Again, all tested scenarios showed similar values for the coefficient of determination of the fitted curves which is very important as it shows the metrics independence of the mobility pattern. For this performance metric the fitted curve even exceeded a value of 90%, clearly showing that the link duration metric is able to reflect protocol performance in terms of routing overhead very well.

The last performance metric evaluated was the end-to-end delay. In Fig. 5.19 an exemplary simulation series is displayed showing the end-to-end delay in Random Waypoint settings with a broad range of mobility degrees. Here as well, with all tested mobility models, the coefficient of determination is very high showing that the link duration is a very good measure for the performance of the underlying routing protocol. In the figure below we see a coefficient of determination of the fitted curve of over 91%.

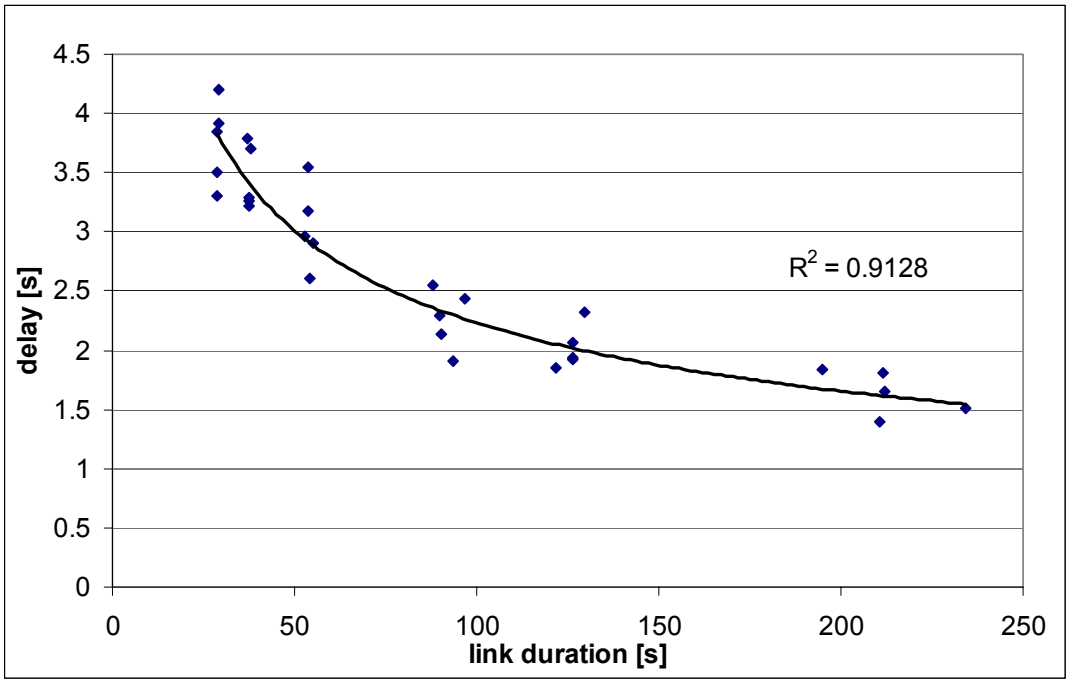

Fig. 5.19 Correlation between link duration and end-to-end delay in RWP mobility scenarios

The analysis of the quality with which the link duration metric reflects the routing protocol performance must be the basis of following work. A metric which fails to be an indicator for the routing performance might not yield good results when utilized for protocol adaptations. In the figures above we used the average link duration provided by CrossTalk's global view of each node in the network. Before we can use the metric provided by CrossTalk we must make sure that the global view itself has a high quality. That means that all nodes in the network must have a similar network-wide view and that the relative error from the real average of the global views must be relatively small. The above only represents the average which gives no information about the uniformity of the global view across nodes and also does not indicate the average relative error per node from the real average that exists in the network. Therefore, we need to do some further analysis before utilizing the link duration metric similar to the one found in section 5.1.4.

In no tested scenario, the coefficient of variance of the global views at each node, which can be regarded as a measure for the uniformity of the global view across nodes, exceeded 3.5% at the end of a simulation run. That shows that the distributed global view's uniformity is very high. The average relative error at the end of a simulation run never exceeded 10% and was in nearly all cases well below that, showing that the global view provides a good accuracy, i.e. reflects the network-wide average very well independent of the underlying mobility model.

In the previous paragraphs we showed that the way the link duration mobility metric is calculated in fact is highly suitable to reflect the degree of mobility and that it is a good measure for the performance of the routing protocol. Therefore, it fulfills all criteria of a good mobility metric as defined before. In the following paragraphs, we show the analysis of the applied mobility metric in the mobility adaptive AODV as described in section 5.2.2.

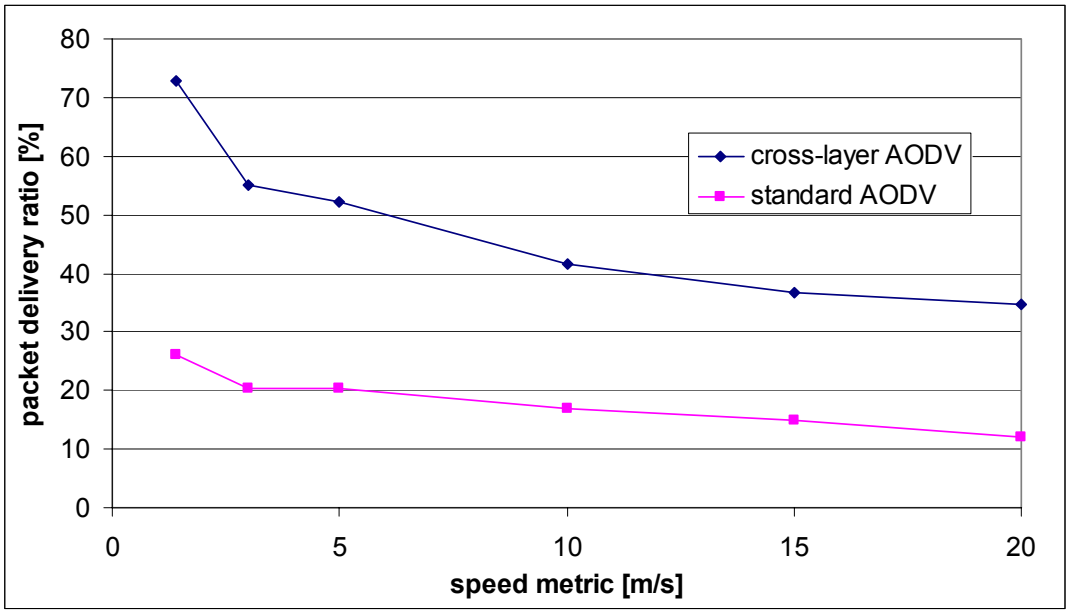

Fig. 5.20 PDR comparison in RWP mobility scenarios

In the following graphs we display the speed metric of the respective mobility model versus the performance metrics introduced in the previous section. We observed the most significant performance gain of our mobility-adaptive AODV in scenarios using the Random Waypoint mobility model. As can be seen in Fig. 5.20, the packet delivery ratio of our mobility-adaptive scheme exceeded standard AODV's PDR significantly. In low mobility of around 2 m/s we were able to increase the PDR by a factor of nearly 3 to roughly 75%. This factor could approximately be maintained over the whole range of tested mobility rates. The PDR of our scheme even never fell below the best achieved PDR of AODV in low mobility. In other words, our scheme maintained a significantly better PDR at 20 m/s as compared to the PDR of standard AODV at 2 m/s.

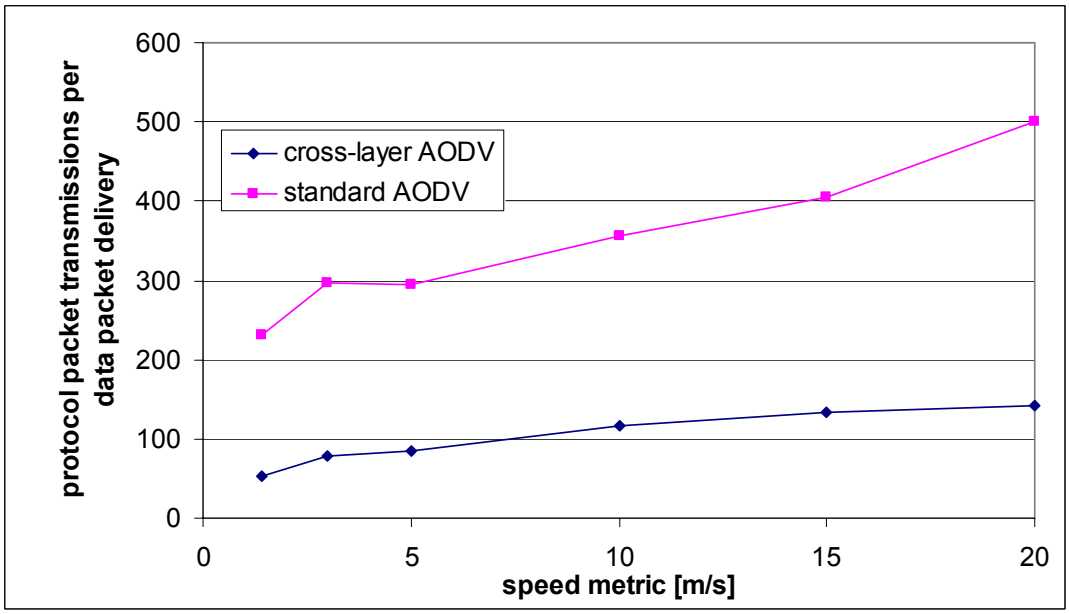

Fig. 5.21 Routing overhead comparison in RWP mobility scenarios

At the same time the routing overhead was reduced by up to 78% as can be seen in Fig. 5.21. The end-to-end delay was also improved, but the Random Waypoint mobility model was the only tested mobility pattern that clearly favors our scheme in terms of end-to-end delay. This is not necessarily a drawback of our algorithm but due to differing assumptions for the direct comparison of standard AODV and our modified version. The reason for this will be explained shortly.
In scenarios using the RPGM mobility model our scheme performed similar to the results obtained with the Random Waypoint mobility model. The only significant difference can be found in the end-to-end delay. Using the RPGM model, the delay performance between standard AODV and our scheme did not significantly differ.

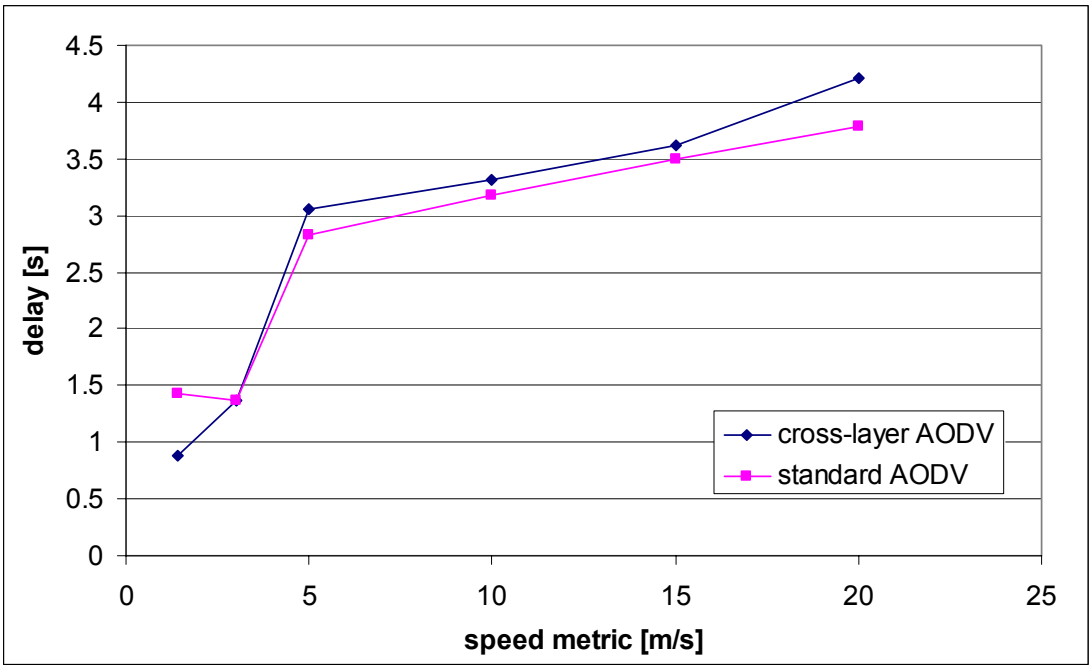

Fig. 5.22 End-to-end delay comparison in RPGM mobility scenarios

Comparing the delay performance of both approaches directly is not a fair comparison per se. One thing has to be mentioned when the delay of both approaches are compared. Since our mobility adaptive scheme has a much higher PDR, more packets are actually delivered. That is especially true for packets that are delivered to far away destinations, i.e. packets that have to be forwarded much more often since they travel on much longer paths. The simulations showed that the average path length of successfully delivered data packets was much higher in our adaptive scheme. On the one hand this clearly shows that our scheme is choosing much more stable and suitable routes. On the other hand this is one reason why the delay is higher since those packets are less accounted for in the standard AODV measurements. This makes a direct comparison very difficult if, in terms of fairness. Additionally, to be completely fair, when the PDR is lower, an application would need to retransmit the lost packets. This would add to the contention in the network, further reducing the PDR and increasing the delay and routing overhead. Such a retransmission scheme was not implemented in our simulations. A fair comparison of the end-to-end delay would probably be one where the PDR of both approaches are the same. This could be done by

experimentally approximating equal PDRs in differently mobile scenarios using the same mobility model. This would be a very time-consuming process though.
An interesting fact to notice is that, when looking at the protocol overhead of standard AODV as displayed in Fig. 5.21, it appears that delivering one data packet successfully generates, depending on the mobility rate, more control messages than there are nodes in the network. Naïvely, that implies that when not taking into account the size difference between data and control packets that broadcasting might be much more efficient above a certain mobility rate in certain scenarios. With the scheme we propose, utilizing a more sophisticated routing protocol still makes sense in scenarios with a much higher degree of mobility.
For the random waypoint mobility model, even at a maximum speed of 20 m/s, the control overhead per successful data packet delivery is well below the number of nodes in the network. As already mentioned the Random Waypoint mobility model yielded the best results. But under all other tested mobility models, our adaptive-AODV performs significantly better than standard AODV as well with the exception of the end-to-end delay for the previously stated reasons. The reason why scenarios using the Random Waypoint pattern inflict less stress on the routing protocol is because there are no bounds on the way nodes move. The more realistic patterns do not allow a node to move purely randomly as there are regions defined where nodes are allowed to move, nodal dependencies exist and the changes in speed and direction can be restricted.

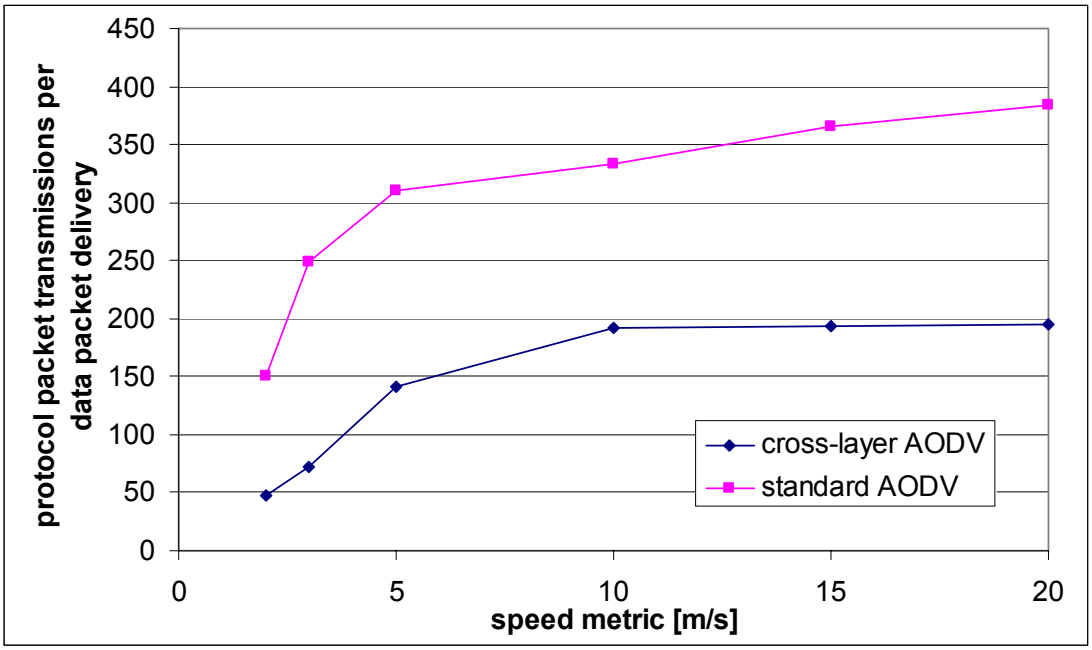

Fig. 5.23 Routing overhead comparison in Freeway mobility scenarios

Fig. 5.23 displays the protocol overhead of standard AODV compared to our approach in Freeway mobility scenarios. In low mobility we again generate three times less overhead per successfully delivered data packet which quickly stabilizes at a factor of around 2 for higher mobility rates. So for this performance metric, we clearly outperform standard AODV. As for the PDR, the gain lies between a factor of 1.5 and 2 with a maximum PDR of 75% for our scheme and 35% for standard AODV.

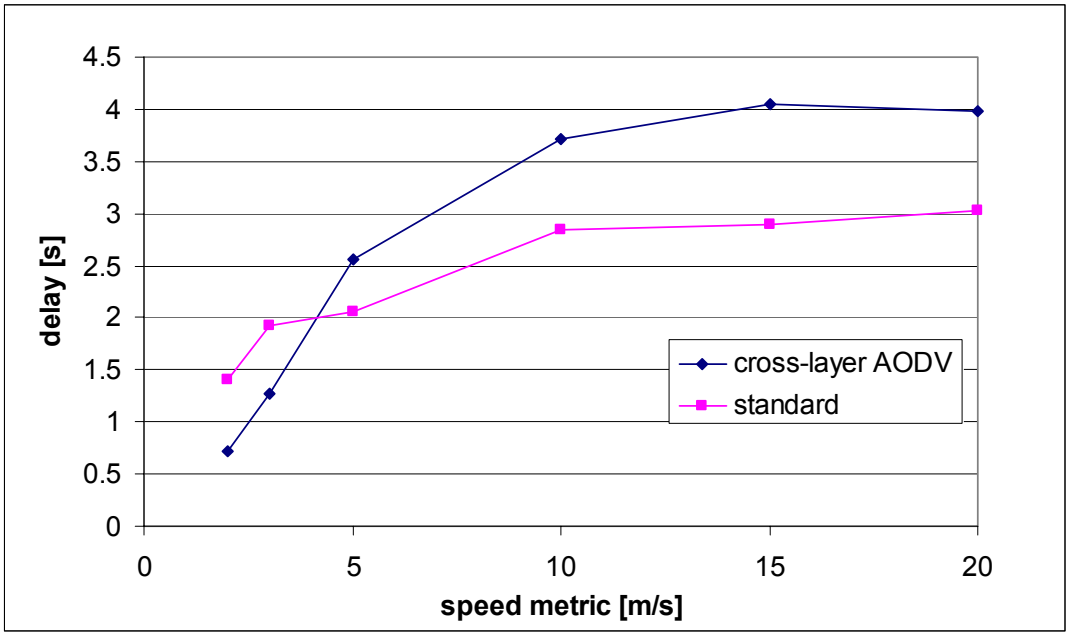

Fig. 5.24 End-to-end delay comparison in Freeway mobility scenarios

In terms of end-to-end delay our approach starts to have a higher delay above 4 m/s for the Freeway mobility model as shown in Fig. 5.24. Up to this point our approach clearly outperforms standard AODV in terms of delay. After this point a direct performance comparison is difficult for the previously mentioned reasons and it cannot be said which approach performs best. The worst increase in delay is roughly 40% for the mobility degrees analyzed.

Of all tested mobility models, the Manhattan model put the most stress on the routing protocol. In all tested scenarios AODV performed much worse when Manhattan mobility was applied. Still our version of AODV achieved a PDR that was between 2 to 2.6 times higher in comparison to standard AODV.

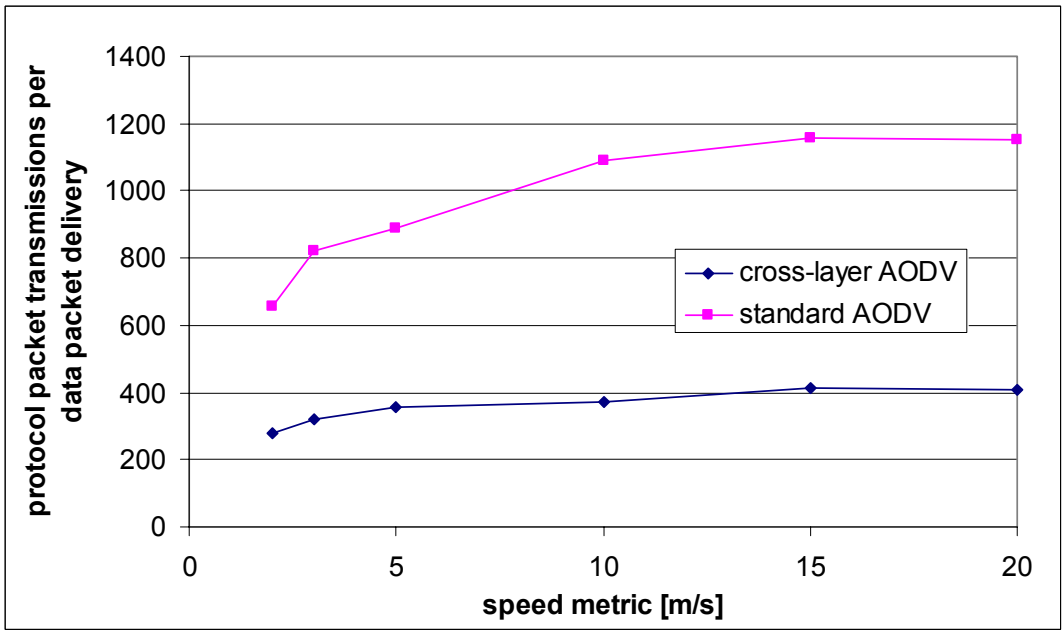

Fig. 5.25 Routing overhead comparison in Manhattan mobility scenarios

Fig. 5.25 presents the enormous routing overhead generated in Manhattan mobility scenarios. With high mobility rates, using standard AODV can hardly be justified under the tested conditions. At a speed of 20 m/s, nearly 1200 packets are sent per successful data packet delivery. The PDR itself is at that point at an alarming 5.2%. Perhaps broadcasting every single data packet would make more sense in such a situation. In contrast, the adaptive-AODV has a slower increase in protocol overhead over the range of mobility degrees and was reduced in total by up to 65% as compared to standard AODV.

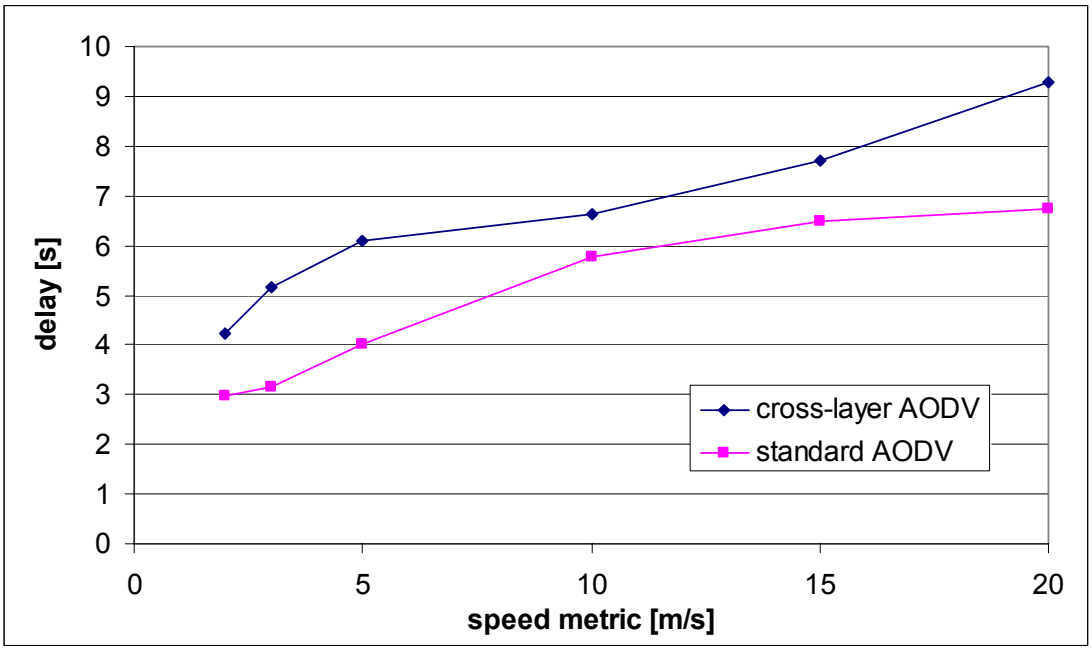

Fig. 5.26 End-to-end delay comparison in Manhattan mobility scenarios

The Manhattan mobility pattern is also the only tested model that lets our scheme perform worse in terms of end-to-end delay over the whole range of mobility rates. The extend of the increase in delay can be seen in Fig. 9. Although less pronounced than the increase in PDR and the overhead savings there is a clear increase in the end-to-end delay of up to 65% in low mobility which is due to the reasons stated before.

We now attempt a comparison of the mobility models we used. When evaluating mobility metrics and mobility adaptations the underlying models used are very important. Most studies though restrict themselves to the rather unrealistic Random Waypoint mobility model. The reason might be that it is very easy to implement and to parameterize. And additional reason might be that it is the only model that comes with the standard ns-2 package. Fig. 5.27 depicts all mobility models used in our experiments. The routing overhead performance metric displayed are the ones taken from our adaptive AODV. The x-axis represents the speed metric of the corresponding model. Although the speed metric does not have the exact some influence in the tested scenarios it can be used as a rough metric for comparison. What Fig. 5.27 quite obviously illustrates is that the Manhattan mobility model is by far the most demanding model. On the other hand the Random Waypoint model is the least demanding model. This is the case for all routing performance metrics evaluated. Therefore, when evaluating a mobility metric or mobility adaptive protocol more than a single

mobility model should be evaluated. When choosing only a single mobility model the Manhattan model is probably the best choice as it puts the mode stress on the protocol.

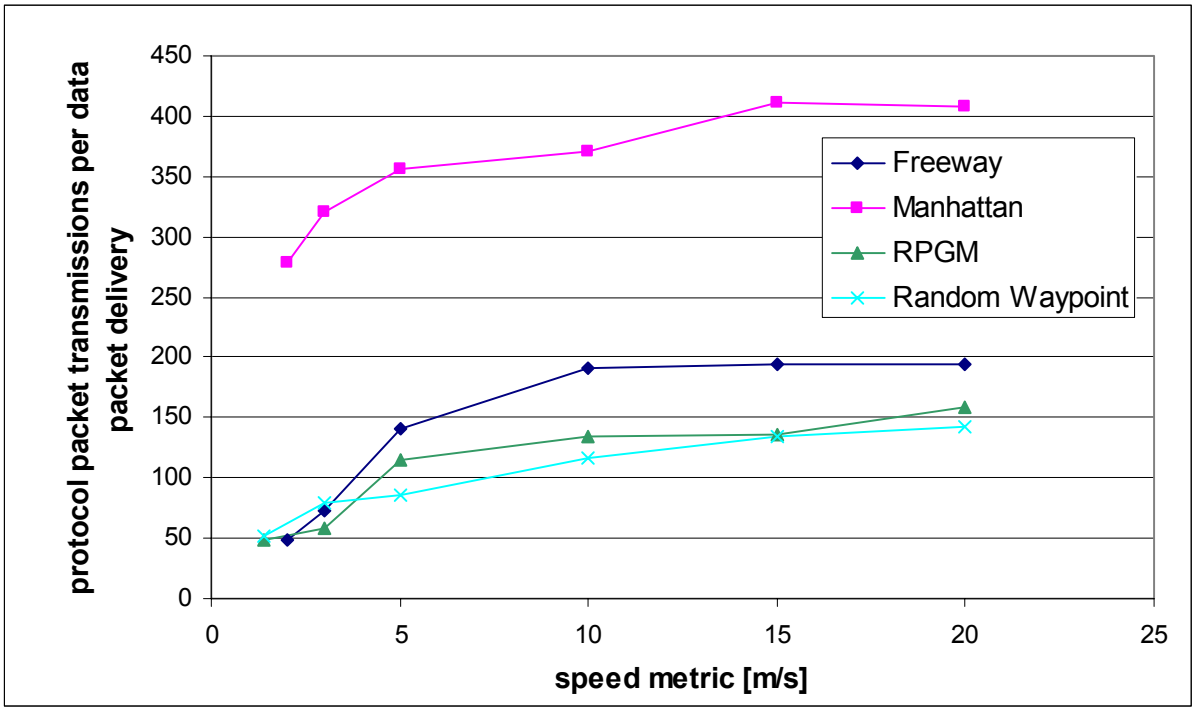

Fig. 5.27 Routing overhead across mobility models

5.2.5. Conclusion

In the previous sections we presented a novel algorithm to locally calculate the link duration from link statistics without predefined parameterization. This way, the algorithm adapts automatically to changing networking conditions and is therefore suitable over a wide range of possible network dynamics. We further analyzed its suitability to reflect the routing protocol performance.

The second main contribution was the application of the link duration metric to make an exemplary routing protocol (AODV) mobility-adaptive. We showed how and where mobility information can be utilized. The proposed adaptive AODV clearly outperformed standard AODV in terms of packet delivery ratio and protocol overhead by a factor of up to three in a vast amount of mobility scenarios. Only in some situations did the end-to-end delay increase, which can be explained by the increase in path lengths due to the increased PDR performance.

Furthermore, the tested scenarios showed that a wide range of mobility models should be tested, as they all have different characteristics. Although the principal conclusion remains the same in all scenarios, the performance varies. The simulations showed that the Random Waypoint mobility model seems to be the least stressful for the given routing protocol. Therefore, we believe that, if only one mobility model is tested, it should be a more demanding one such as the Manhattan model.

Another contribution was a second analysis of the quality of the Global View similar to the one found in section 5.1.4 only using a different metric. The findings again support our confidence in utilizing CrossTalk's Global View mechanism.

5.3. Partition Detection – A Novel Application Using CrossTalk

One of the goals of CrossTalk is to enable novel applications based on the mechanisms it provides [111][112]. One of the unique phenomena in ad hoc networks is network partitioning. A network partition occurs when a connected network topology breaks down into two or more separate, unconnected topologies. In fixed infrastructure based networks, the occurrence of a network partition is highly unlikely and only possible if big parts of the infrastructure fail simultaneously. In mobile ad hoc networks, the problem of partitioning can also occur due to the mobility of the participating nodes making it a more common phenomenon. Various applications and network services for mobile ad hoc networks could benefit from a reliable partition detection mechanism. As an application, distributed mobile games are especially prone to network partition since they are expected to be played on the go. They are played, for example, while waiting for a bus or subway. Here often significant amounts of players move in a group and therefore partition the network. In such a situation it would be beneficial to detect the partitioning and not interpret it as node failure. This way, two or more separate games can be established without having to restart a game. In general applications which have to perform expensive tasks in terms of network load on node failure would benefit from partition detection. If that node failure is due to a network partition, the application could choose to wait a certain amount of time expecting it to be a temporary phenomenon. This way, the application could resume the normal operation after the network partition problem is solved without loading the network unnecessarily. Also, applications based on swarm intelligence can act and try to re-establish a fully connected network or they can intelligently adapt to the situation when using a partition detection system. A type of network service which would benefit is auto-configuration. When network partition takes place, a possibly big part of the network address space becomes available again in separate partitions and can therefore be reused. A network based on a topology control mechanism could also benefit from such an approach. On detection of network partitioning, the nodes could extend their transmission range to reconnect the separate network topologies. In this chapter we compare two approaches to detect network partitioning, a centralized and a distributed approach. Both approaches have unique advantages which are compared and analyzed.

5.3.1. Related Work

There has only been some effort to detect partitioning in ad hoc networks. We are not going to describe partition prediction or anticipation schemes here in great detail as their focus is different and often these schemes rely on trajectory information obtained by positioning information which we do not assume.
One of the few partition detection schemes is presented in [116] considering asynchronous distributed systems. They focus on processes rather than on network nodes. Their goal is to distinguish between network partition and process failure. Using signal strength measurements a node will send an alert message to inform other nodes, or more precise other processes about an upcoming partitioning event. The accuracy of signal strength measurements was discussed in previous sections. This clearly is a cross-layer solution by design. In

addition there are unreliable failure detectors in the system which monitors processes to observe if they crash. The name indicates that these detectors are not necessarily correct. The authors develop two ways to detect partitioning. The first approach tries to obtain knowledge about its neighbors and its neighbors' neighbors. Partition detection is triggered by neighbors if they believe they are the only node that can connect to certain unavailable processes. On the detection of a partition the still reachable processes are informed. The other partition detection scheme relies on total topological knowledge and the configuration of the distributed application at start time. In general their system is not described in great detail and it is not analyzed in any way.

A little more comprehensive approach is presented in [117] and [118]. Here partitioning is not concerned in a network-wide scale but between a client and a server, whereas the client should be enabled to detect the partition. For their system to work it is important that there are multiple disjoint paths between the source-destination pair that is going to be monitored. They also try to anticipate a partitioning event and derive two link robustness metrics where one utilizes the probability of link breakage. The authors fail to mention where they get this information from. Not surprisingly this metric does not perform very well. The other metric simply represents the longest suboptimal disjoint path found. They then introduce a threshold that, if violated for a certain amount of time raises a warning flag that has to be interpreted by the application. This makes the system somewhat inflexible and the general question is how to choose this threshold. Their analysis shows a high efficiency for the simple metric but the authors fail to mention how they obtain that value. Additionally, to obtain multiple disjoint paths a broadcast algorithm is used. The authors do not mention how these paths are maintained as the nodes move. In general, all their descriptions show some lack of detail.

Especially the academic interest in group communication pushed for a partition detection scheme. Babaoglu et al. [119] developed a partitionable group communication service which allows so called "partition-aware applications" to operate in separated network topologies and, after two or more partitions merge, reconfigure themselves. The partitioning problem is handled by a simple PING/ACK mechanism. A node sends a PING message to another node. If it does not receive an ACK in a certain amount of time, that node is added to a list of suspects. A dynamic timeout mechanism is used which leads to a reasonably accurate suspect list. This scheme lacks the ability to distinguish between node failure and partitioning which for most applications is desirable. Also it does not carefully choose the nodes that monitor the network to increase the detection probability. In general most of the schemes described here do not offer this important feature.

Killijian et al. [120] go a step further and try to anticipate a network partition. They only briefly describe their system, which consists of 3 entities: a failure anticipator, a movement planner and an environment evaluator. Obviously these components heavily rely on some form of sensing equipment and a huge amount of information exchange. Especially the sensing equipment cannot be assumed for all participating nodes. This approach appears to be quite heavy weight and the authors do not mention how they intend to detect that partitioning has taken place. They also do not show any facts and figures that demonstrate the accuracy of their anticipation scheme.

In [121] simple heartbeat messages are used. Partitioning is suspected in the absence of an expected heartbeat. Clearly, this scheme does not allow to distinguish between node failure and network partitioning.
Other research conducted in the field of group communication explicitly includes the network partition problem [122][123]. These approaches show that they are resilient against the problems arising after a network breaks apart and eventually merges back together, but none of the approaches deal with the detection of the event itself.

5.3.2. The Partition Detection System

Mobile ad hoc networks based on radio technology such as Bluetooth or wireless LAN are restricted in terms of bandwidth and their multi hop capability. That limits the amount of participating nodes and the size of the network topology. In such networks we have the notion of borders due to their geographic constraints and their wireless nature. To be able to exploit this characteristic is important for the network partition detection system. Border nodes are especially suitable to play a vital role in the detection process. The general idea is that border nodes exchange messages among each other. This way they monitor large parts of the network as the messages traverse extensive network regions. If the network breaks up into separate partitions the nodes cannot reach each other any more and therefore detect the partition. This is of course only true if nodes are unable to communicate due to the failure of one of the nodes. In order to be able to distinguish between failure of border nodes and partition, extra functionality is added. The network wide partition detection is complemented by a local monitoring mechanism to prevent false partition detection due to node failure.
Before going into the details of the partition detection systems developed it has to be noted that they do not rely on CrossTalk per se. One can argue that in the application layer a developer can easily implement CrossTalk's Global View functionality to at least some extend. Whether CrossTalk's Global View is used or not, the approaches developed are cross-layer in nature and the presented solutions could go through CrossTalk's local view to obtain routing protocol information. But even that we do not as even this potential cross-layer functionality can be implemented at the application layer.
The partition detection systems presented are therefore a good comparison against CrossTalk's Global View functionality against an application layer implementation that does basically the same and we will see that it is less precise. That means the presented solution should preferably be based on CrossTalk.
Before we go any further, the disadvantages of having an application layer mechanism are manifold. First of all each node would need to run the application. Having the functionality in a common framework would make sure that the functionality exists on each node. A second drawback is that only application layer packets can be enriched with additional data which reduces the efficiency of the data dissemination procedure. Also, every application that would rely on such a mechanism would have to implement it in addition to the aforementioned problems. As every application does the same thing individually, there are less possibilities of optimizing the data dissemination procedure in addition to the development overhead. Finally, the mechanisms that could be leveraged at lower

layers have to be re-implemented. That means that the same functionality has to be implemented twice. This unnecessary redundancy consumes resources that are scarce in ad hoc networks.
Let us now look at the partition detection systems in detail. Both approaches we developed distinguish two sets of nodes. One set consists of nodes not actively taking part in the network partition detection system. The other set of nodes actively probes the network as part of the system. The nodes probing the system have to be chosen carefully as mentioned above to ensure that most of the network topology is monitored. Their key property is that they have a relatively small amount of neighbors compared to the nodes not actively supporting the partition detection system. The prominent reason for this is because nodes with a relatively low neighbor count are most likely at the border of the network topology. The number of neighbors a node has allows us to select the most appropriate nodes for the active part of our system. In other words, it allows us to identify border nodes. This is again a good example for the application of global knowledge. The local one-hop neighbor count or neighbor degree does not imply whether a node is at the border of the network or not. Only by comparing its own neighbor degree with the average neighbor degree in the network a node can determine if it is in a sparse network area and therefore at a border with a high probability. But as stated before, in this experiment we try to solve the problem at the application layer.
Both approaches work in a way that nodes periodically send out beacon messages serving as keep-alive messages. Every active node monitors a certain amount of other active nodes (or just one in the centralized approach as will be explained later). If a beacon message from one of these monitored nodes is not overheard for a certain amount of time, network partition is suspected. Therefore, active nodes should be placed far apart so that the beacon messages travel through large parts of the network, thereby increasing the monitored network area. In order to distinguish whether the absence of the beacon is due to node failure or due to network partition, a local validation mechanism is used, as already mentioned. The nodes sending beacon messages elect a so called “buddy”. This buddy is an one-hop neighbor monitoring the node sending the beacon message. If the buddy node cannot hear its one-hop neighbor any more, it starts a route request for it. If it finds the node over a multi-hop path, it asks that node to elect a new buddy. If it does not find the node, it suspects node failure and notifies the other active nodes about that incidence. Using this buddy mechanism the probability for a false partition alarm is significantly reduced. It can be even further reduced by electing multiple buddies since a false partition alarm is only set off if all buddy nodes together with the active node fail simultaneously.
Since the active node (or border node) identification is so important, we evaluated two different approaches here as well, a static one and a dynamic one, which adapts well to topology changes. Since the criteria for becoming an active node is the neighbor count, all nodes in the network have to monitor their immediate network neighborhood periodically. Every node sends a broadcast message with a time-to-live (in hops) of one so that the message is only heard by direct neighbors, which in turn send an acknowledgement. This way, every node always has a relatively up-to-date view of its immediate network environment. This exactly is the mechanism which much more efficiently could be handled by cross-layer mechanisms. As an example, AODV could provide this information at no extra

cost. By not utilizing cross-layer mechanisms the exact same mechanism would be carried out twice, once at the network layer, once at the application layer. This clearly is a very wasteful approach in terms of resources.

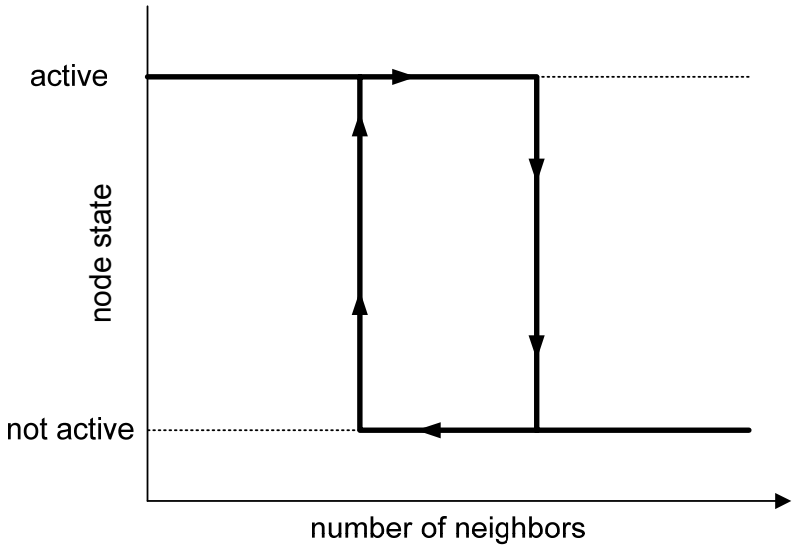

Fig. 5.28 Neighborhood hysteresis

The static approach uses a fixed threshold. Here a node becomes active if it has equal or less neighbors than the threshold. In the dynamic approach, a node constantly piggybacks its own neighbor count onto application data packets. The receiving node extracts that information and generates a threshold based on the last set of neighbor counts received. If its own neighbor count is below that value, it becomes active. This part, i.e. the dynamic approach could much more efficiently be realized using CrossTalk's Global View as previously mentioned.
Border nodes can also leave the active state again. When their neighbor count increases, they stop actively taking part in the system as it indicates that the node is moving away from the border making it less suitable to participate in partition detection. Since in mobile networks the network topology could constantly change and, therefore, the neighbor count could fluctuate, the system uses a kind of neighborhood hysteresis. That means that a node enters the active state at a certain neighbor count but returns to the inactive state at a neighbor count higher than the threshold to enter the active state as illustrated in Fig. 5.28. Without this mechanism border nodes could potentially change their state frequently causing a high overhead.
If an active node has only a few neighbors, there is a high probability that its neighboring nodes also have a relatively low neighbor count due to the nature of ad hoc networks. It would be inefficient to activate those nodes since that area of the network topology is already covered. Both developed approaches make sure that nodes adjacent to active nodes do not become active themselves through a neighbor inquiry mechanism when joining the network, or by overhearing activation messages. The inquiry mechanism also makes sure that newly switched on nodes learn about beacon sources.
Note, that the prerequisites for our system to work are that an efficient routing algorithm exists, which supports broadcast and unicast. Additionally, it should provide the ability to set a time to live field.

The first system that was developed takes a centralized approach. Only one of the active nodes sends out beacon messages periodically, with the other active nodes only being the endpoints for those messages. On start-up of the system, the first node that detects that it should become active sends out a broadcast message telling each node that it is the beacon source (or server). Conflict situations can be handled by an algorithm similar to the lowest ID algorithm [113] or by another clusterhead election procedure. Every subsequent node that becomes active sends a notification to that node causing it to periodically send out a beacon (unicast message) to the just activated border node. As long as all the border nodes receive the beacon periodically, the network between the border nodes and the beacon source is monitored and connected. This, again, is why the placement of the active nodes is so important.
If one of the border nodes does not hear the beacon any more, it would always interpret it as a result of network partition. That is why we introduce the local validation mechanism, which was already briefly mentioned. It works as follows. In the centralized approach only the beacon source elects a buddy on start-up. The buddy periodically sends out a PING message to the beacon source which responds to that message with an acknowledgement. It is important that the interval of this PING message is much smaller than the interval of the beacon message. This way node failure is detected much faster than partitioning and the monitoring border nodes can be notified before they suspect partition due to the absence of the beacon. When a new node within the network becomes active and registers at the server, the server sends out an updated list of all border nodes to the buddy. This way the buddy always has an up-to-date view of the partition detection system and knows which nodes it has to notify if the server fails and could also quickly replace the server on failure. Another approach would be to simply broadcast the failure message, instead of using unicast messages. We did not implement this approach, since we wanted to keep broadcasts to a minimum.
Failing nodes are not the only source of disruption for the partition detection system. In a wireless network, messages are much more likely to fail than in a wired network. Therefore, partition detection mechanisms must be robust against an increased message failure rate. The border nodes do not expect every beacon message to arrive at their destination. Every border node allows a certain number of messages to be lost before they suspect network partition. If the allowed beacon message loss is for example 3 and the beacon interval is 10 seconds, the system would suspect partitioning after 30 seconds have passed without having received a beacon message and also not having received a failure notification from the server buddy.
A similar mechanism is used for the keep-alive-messages exchanged between a buddy and the server. However, since they are one hop neighbors, message loss is considered to be far less significant.
The border nodes that monitor the server have to send a keep-alive message back to the server infrequently to tell the server that they are still participating since they have no fail notification mechanism like the server does. That ensures that the server does not load the network unnecessarily after a border node failed. If a node changes its state to be inactive it notifies the server for the same reason.
The centralized approach has one general disadvantage. The partition which contains the server does not detect the partition as such. This is because in the centralized approach active nodes do not exchange beacon messages amongst

each other. Detecting partitioning in the server partition might not be necessary though. If the applications in all but one partition have to suspend their operation, the centralized approach is sufficient. This situation is described in the primary partition model [114][115].
There is also one general case in which partitioning is not detected, neither in the centralized nor in the distributed approach. The algorithms will not detect a partitioning when a part of the network is separated that does not contain an active node. If the border node detection works properly, this can only occur if a very small amount of nodes are separated in a rather unlucky constellation. The question is whether someone would consider this a network partition.
The distributed approach is in its operating mode similar to the centralized approach. One key difference is that every active node sends out beacon messages. On becoming an active node, the node sends out a broadcast message telling every other node that it just became active. Every node in the network, even the inactive ones, store a certain number of border node addresses together with the distances to those nodes in hops. If there are more addresses to store, the node either replaces "old" entries or it replaces addresses of nodes that are closer than the new node. When a node now becomes active, it uses these nodes as partner nodes. It sends a request to send beacon messages at a certain interval to them. With the active nodes already stored while having been inactive, a node becoming active already knows of other active nodes in the network and can directly participate in the system. The replacement strategy mentioned above ensures that large parts of the network are covered by the system since only the nodes furthest away are kept. Since every active node sends out beacon messages, they all need a buddy node. The buddy acts a similar way as in the centralized approach. A difference is that the active node always informs the buddy node when either a new partner node is chosen, or if a beacon requestor fails or is added. This way all other nodes (a small number of active nodes) that in some form interact with the active node get notified if it fails. The partner nodes then stop sending beacons and the beacon requestors have to choose a new partner for themselves. If active nodes get notified that one of their partner nodes failed, it can either choose to wait until a new node becomes active, or if too many partners fail, it could start a lookup request for active nodes to fill the partner table. The latter is especially useful after a partition was detected.
There is one critical moment in the system on startup. When the detection system is initiated, the nodes start checking whether they should become active. That would lead to a broadcast storm shortly after startup. This is why nodes start a timer when they receive an activation broadcast. If that broadcast timer is not expired when they want to send their own activation broadcast, the nodes delay it until after the timer expires. This mechanism makes sure that the network is not loaded unnecessarily, keeps collisions to a minimum at startup and saves bandwidth for other packets such as application data packets.
The distributed approach has one possible weak point. If a buddy (or buddies if multiple exist) and its corresponding active node are separated by partitioning, the partition remains undetected in the case that the active node is the only active node in its partition.

5.3.3. Experimental Setup

To validate the two approaches and to analyze their behavior and performance we implemented them using the ns-2 network simulator. We evaluated four basic scenarios:

- Networks without partition and failing nodes
- Networks with partitioning but without failing nodes
- Networks with failing nodes but without partitioning
- Networks with both failing nodes and partitioning

Additionally we changed parameters for the border node detection. Every scenario was simulated 50 times with a given parameter combination. The networks we simulated consisted of 50 nodes, which we believe is a reasonable number for scenarios such as a mobile gaming scenario at a subway station or group communication at a construction site or in a company.
As already mentioned, the border node detection is a vital mechanism for our system. In our simulations we let the nodes move according to the random waypoint mobility model. In some scenarios the nodes are expected to move only little or extremely slowly for example while participating in a multiplayer game. We believe that there is a vast amount of other scenarios where this assumption cannot be made. Therefore, we let the nodes move at an average walking speed. This way we can evaluate the stability of our system and we see nodes changing their state due to changes in their immediate networking neighborhood. The mobility model itself has a certain characteristic [110] that we believe might mimic the behavior of ad-hoc application users. In the random waypoint model the node density in the center of the network area increases over time. Mobile gamers for example might in reality move together a little closer since mobile gaming and other applications in ad hoc networks also have a social implication.
For the dynamic border node identification approach, every node picked a node at random every 10 seconds and sent a packet with its own neighbor count piggybacked onto that packet to that node. Depending on the application, that might not be very much, but we wanted to evaluate whether the system still works with applications that generate only little traffic. The threshold itself was then generated out of the last 10 application data packets a node received.

Table 5.3 Partition detection experiment parameters

Simulation Parameter	Parameter range
Network size	50 nodes
Mobility	Random Waypoint 1.4 m/s
Traffic generation	0.1 pkts/s to random destination

5.3.4. Experimental Results

Let us first have a look at the border node identification process. This process is crucial for the efficiency of both partition detection schemes. Very important is that even in the face of node mobility, the active nodes are close to the border of the network.

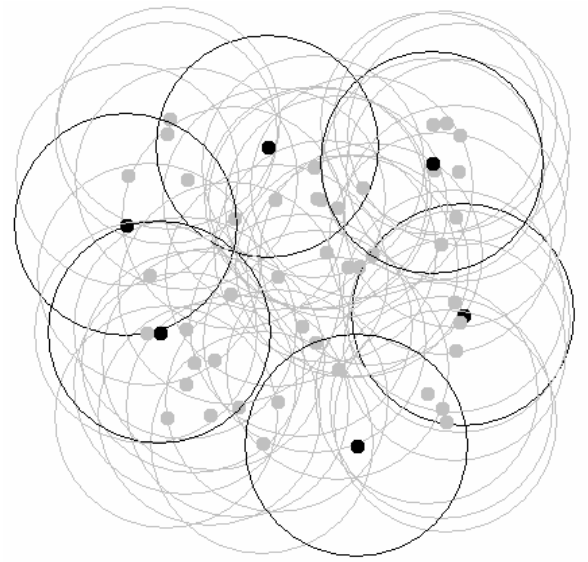

Fig. 5.29 Network topology after 50s of simulation

Fig. 5.29 shows a typical network topology after 50s of simulation. The system had plenty of time to perform border node identification but the nodes themselves did not move a significant amount. The figure shows the geographic positions of the nodes with their respective transmission ranges denoted by the dots and circles respectively. The dark nodes are the active nodes monitoring the network, i.e. they believe to be at the border of the network. The figure clearly shows that the border node detection scheme reliably identifies border nodes within the topology. It can be also seen, that the overall amount of nodes that activate themselves is low as neighboring nodes do not become active as well as previously described. This mechanism obviously works well too.

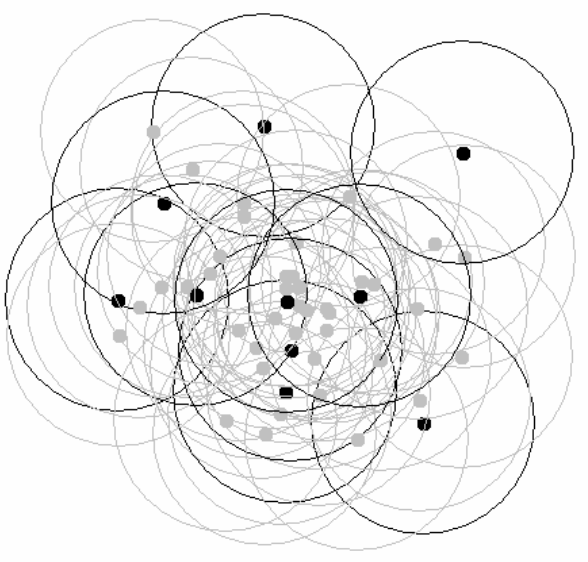

Fig. 5.30 Network topology after 500s of simulation

In Fig. 5.30 the same network is displayed again but after 500 seconds of simulation. The effect of the mobility model can be seen clearly. The node density is much higher in the center now, increasing the average neighbor count. The border node detection lost a slight bit of its accuracy. That can be explained by the fact that the nodes only check at certain intervals whether to become active or not. This could be much more efficiently be solved by CrossTalk as the interval itself would be much smaller due to other application packets and routing control packets for example. During the time interval the partition detection system is passive, the neighborhood might have changed and after the next check the node might change its state. Another reason could be that the last 10 application data packets received led to an unbalanced dynamic threshold at some of the nodes. That has no negative effect on the partition detection. Only the network is loaded a bit more until the nodes finally change to the inactive state. For example the node in the center of the network most likely will change its state soon after the displayed network snapshot. This inefficiency though could be reduced using CrossTalk as it was shown to provide a highly accurate global view. In general it can be said that the borders are still very well covered and that the partition detection system is fully operational. It is compared to a solution based on CrossTalk very wasteful in terms of resources though.

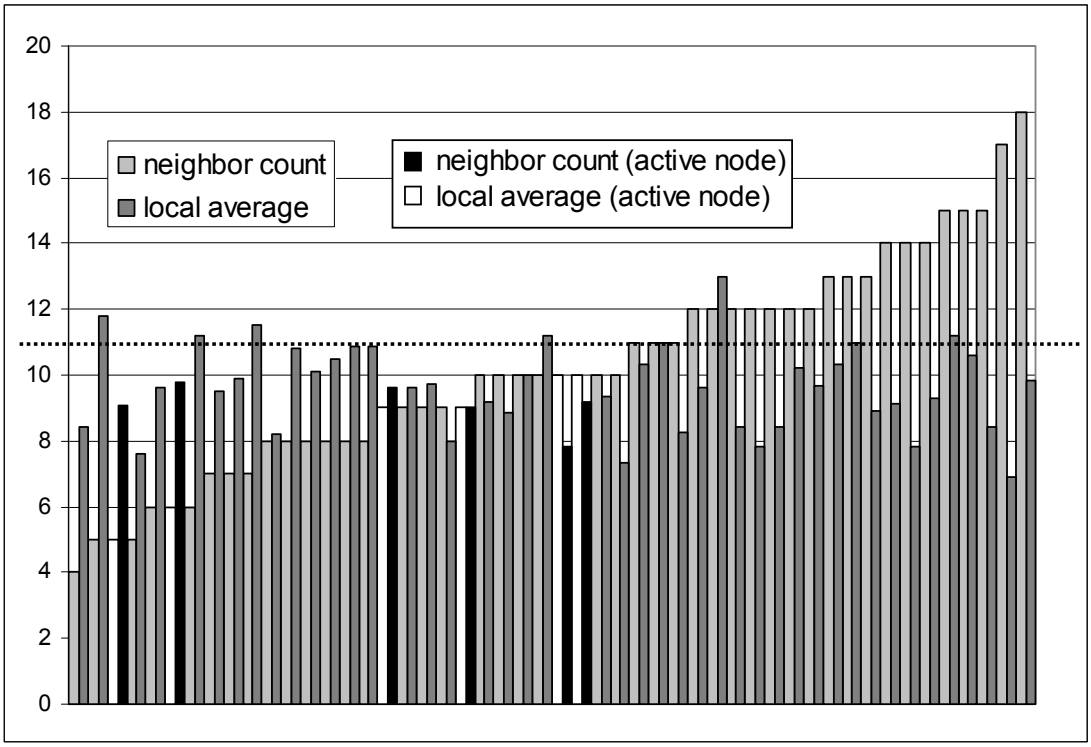

Fig. 5.31 Dynamic border node detection statistics after 50s simulation

Fig. 5.31 and Fig. 5.32 show the key parameters in more detail. In both graphs show statistics collected in the same situation depicted in the previous figures, i.e. after 50 and 500 simulated seconds. They show the neighbor count which is the actual number of neighbors of each individual node in ascending order together with the locally calculated neighbor averages which is the network-wide neighbor average calculated form the piggybacked data received by each individual node, i.e. the application layer global view. In other words, each two consecutive lines represent one node ordered by the number on neighbors a node has. The

probability for being a border node is therefore the highest on the left side of the graph. The dashed horizontal line is the network wide neighbor average calculated employing global knowledge. The closer the local average of an individual node is to that line, the more reliable it can identify itself as a border node. If more nodes have a local average above that line, the overall system might generate a large number of active nodes. If the local average is in general calculated too low too few border nodes could be the result. That of course is also dependant on the way the threshold for becoming an active node is calculated.
From the figures one can see that at the beginning of the simulations we have a global average of 11 neighbors per node, whereas that average increases to 18 neighbors after 500 seconds. That translates into an increase of 63% in neighbor density. In our case, this characteristic of the chosen mobility model is a good test scenario as the border node detection algorithms has to adapt to this change. From Fig. 5.30 it can be seen that this is the case.
Fig. 5.31 shows that the dynamically calculated neighbor average in general reasonably well reflects the real network-wide average. Most of the nodes, though, have a local neighbor average slightly lower than the artificially extracted average. Also the deviation of the different locally calculated averages is relatively high. Clearly, CrossTalk would perform much better. In the figure, the statistics of active nodes are shown in black (local average) and white (neighbor count) respectively. In Fig. 5.31 the values are all within the normal thresholds of the system i.e. activation threshold and neighborhood hysteresis. The first three active nodes from the left all have a lower neighbor count to the calculated average neighbor count. They clearly qualify to be active nodes. For the next active node both values are equal. The node might have gotten more neighbors and/or the calculation of the average neighbor count might have changed. The same applies for the next two nodes which should evaluate themselves differently. But the neighborhood hysteresis prevents them form becoming inactive to fast as the neighborhood might be in a state of frequent flux.

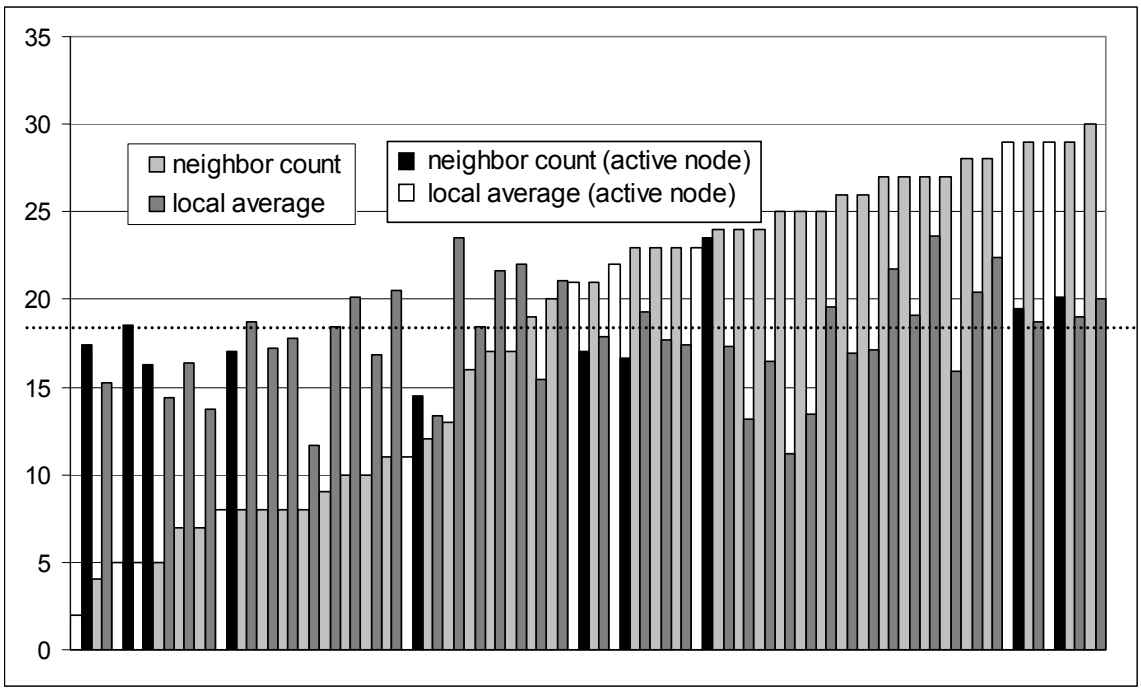

Fig. 5.32 Dynamic border node detection statistics after 500s simulation

In Fig. 5.32, i.e. after 500 simulated seconds, some of the values do not fully satisfy the system criteria due to the reasons already stated above. For example, the two rightmost border nodes in Fig. 5.32 should leave the active state next time they check their neighborhood. The reason why so many nodes have far less neighbors than the locally calculated threshold and are nonetheless not active is that they have a one hop neighbor which is already active. Still the approach is reasonable. As can be seen from the figures, the neighbor average is not constant, but the system adapts to the changing environment and the border node count is not decreasing. The system is still fully functional. But again it has to be stated that clearly CrossTalk would do a much better job in this case in terms of relative error and standard deviation compared to the application layer global view without the additional overhead.

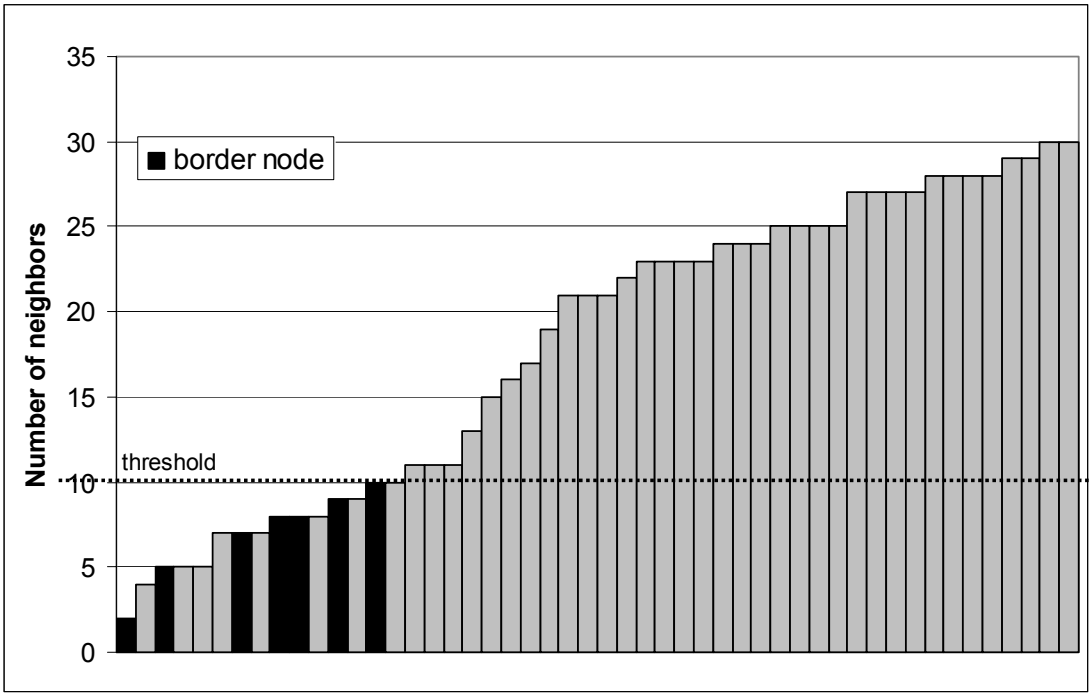

Fig. 5.33 Neighbor statistics after 500s for the simple threshold approach

In the previous figures we analyzed the dynamic approach. In comparison, the simple threshold approach is not able to adapt to changing environments. The first problem is that the threshold has to be chosen very carefully. If it is too low, no or too few nodes become active which leads to a dysfunctional partition detection system. A second problem arises when the environment changes and nodes leave the system. This is of course only the case when the topology changes in a way that the average neighbor count increases like it does in the scenarios we tested. For the exact same movement traces the simple threshold approach always had less successfully detected border nodes at the end of simulation runs, when starting out with equal thresholds. Fig. 5.33 shows the neighbor count of each individual node together with the threshold value (dashed line) for the simple threshold approach. The black marked data points are active nodes. What someone could do to increase the number of active nodes is to set the threshold relatively high. This way, active nodes will be guaranteed, but their placement does not have to be optimal any more. With such a high threshold, the number of active nodes would not excessively grow as nodes neighboring an active node

would stay inactive and the density of the network would not allow the number of active node to grow unreasonably. We simulated the threshold approach in such networks with a threshold of 50, which is equal to the total number of nodes. This way every node would activate itself without having an active node in its radio range. The result was that only 12 nodes were activated leaving the number of active nodes at a reasonable number. The positions of the nodes within the network were left to chance and therefore the border node detection lost its relative determinism and usability. Even worse would be the effect in networks based on a short-range radio technology resulting in networks that are far less dense than the one shown in Fig. 5.29. The number of active nodes would further increase if the threshold is set to a high value.

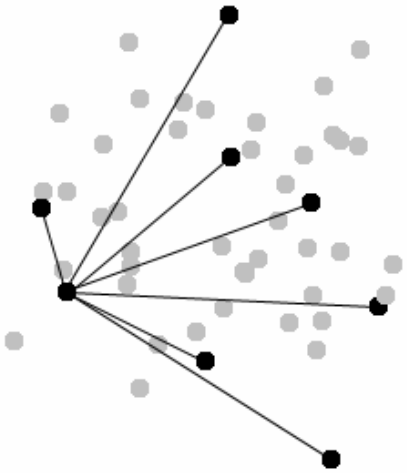

Fig. 5.34 Typical centralized partition detection system structure

Let us now have a look at the two network partition detection schemes. Fig. 5.34 shows a typical centralized partition detection system structure. Clearly the server is the most vulnerable element of successful partition detection. For our simulations we chose the first node that activates itself to become the server. In a real network we would have to use a mechanism similar to a cluster head election phase used in hierarchical routing algorithms as already mentioned or a node initiating the application would serve as the beacon source (e.g. the game initiator).

For the first set of simulations we measured the amount of packets sent by the partition detection system averaged over 50 simulation runs without partitioning. The simulations lasted 500s and the results show the cost of operating the system in terms of message exchanged in the state that it should be in most of the time, i.e. in an unpartitioned network.

Fig. 5.35 sums up the number of different messages exchanged. The parameters chosen in the simulation runs are the following. Every 15 seconds every node probed the neighborhood and then decided whether to change its state or not. The buddy sent a PING message every 5 seconds and the server sent a beacon message every 15 seconds to each of the border nodes. The figure does neither include the application data traffic nor the neighbor probes since they are equal in both approaches.

As can be seen, the buddy traffic dominates the overall load generated by the system with nearly 50% (220.2) of the message total of 447.5 messages in average. These messages are in most cases one hop messages and therefore do not load the overall network very heavily. Using CrossTalk these messages could be saved in addition to the neighbor probes not shown in the graph. The other half of the messages generated mainly consist of beacon messages (214.1) which most likely are multi-hop messages loading the network more than the buddy messages do. This is basically the business logic of the partition detection algorithm. Some of those messages might be saved using CrossTalk as the routing state could confirm the existence of the node. The rest of the sent packets can be neglected except for the one broadcast message from the server at start-up. The main load of the partition detection system is carried by two nodes: the server and the server buddy. The server has to send all beacon messages and also 50% of the buddy packets (acknowledgements). Since the buddy changes during the simulation due to node mobility the rest of the buddy load is distributed over multiple nodes.

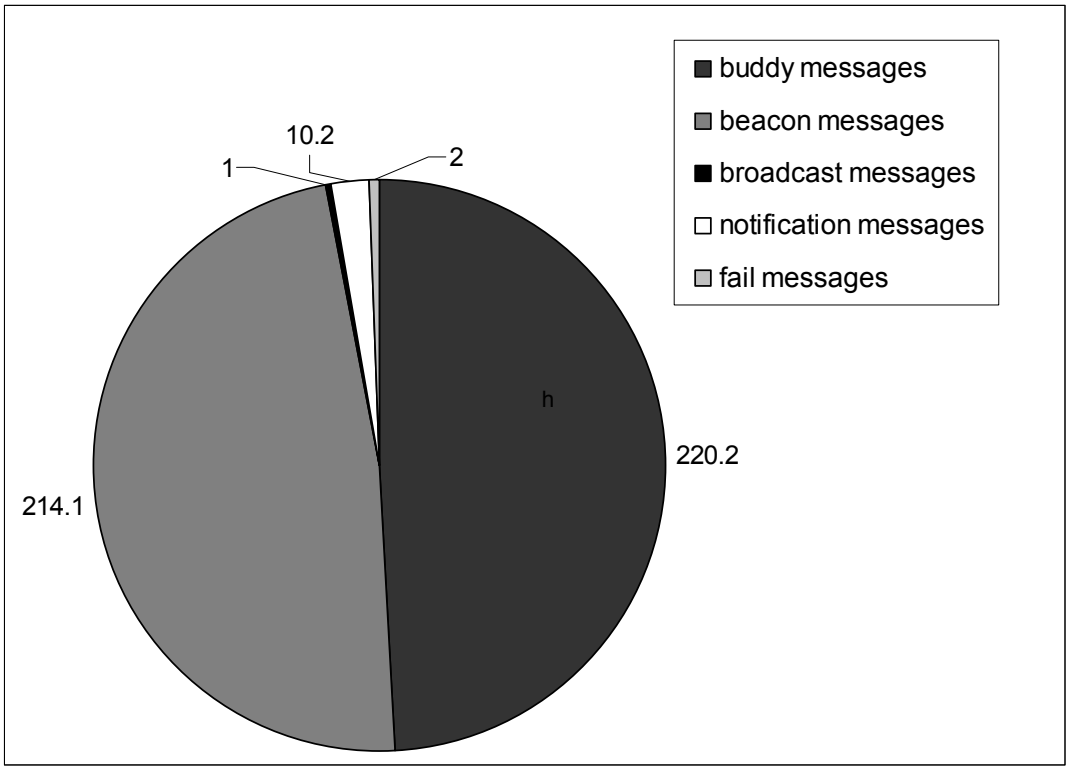

Fig. 5.35 Maintenance cost in number of messages sent for the centralized approach during a 500s simulation run

The total traffic generated (447.5 messages) averaged over all nodes in the network would roughly translate into one packet sent per node every 55 seconds. In mobile gaming scenarios that amount of traffic would only be a fraction of the overall load generated by the application. The relation between partition detection inflicted load and load caused by the application is also highly dependent on the parameter set of the partition detection system. If an application needs to detect the partitioning relatively fast more packets have to be exchanged compared to a relatively slow detection process.

In the simulations where we let nodes fail we concentrated on the server. The buddy always replaced the server and notified the other active nodes. The system could always handle server failure. The only time it was not able to handle the failure and a false partition alarm was initiated was when both, server and buddy, failed simultaneously. Such a situation should be fairly rare, but to overcome this problem, an active node could elect more than one buddy node. Simultaneously failing border nodes could have an effect on the partition detection system only if by chance all border nodes in one partition failed after the partition occurred. The extra messages sent after a partition was detected are negligible compared to the messages sent throughout a simulation run.

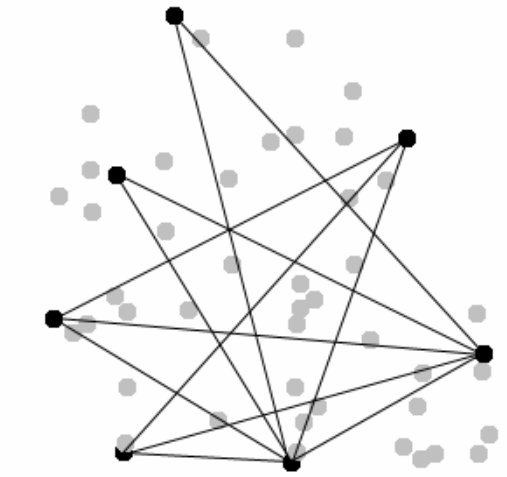

Fig. 5.36 Typical partnerships amongst nodes in the distributed approach

The distributed approach spans a mesh in the network topology as illustrated in Fig. 5.36. Every route from one active node to the other is sensitive to partition detection. The distributed approach of course generates much more traffic since much more beacon messages have to be exchanged and also every active node needs a buddy.
The load generated can be seen in Fig. 5.37. The settings chosen are the same as for the distributed approach. Additionally, every active node should communicate with 3 other active nodes, temporarily less when nodes fail. The average total message count sums up to an amount of 3121 messages. For the chosen parameter set that is roughly 7 times the amount of messages that have to be exchanged compared with the centralized approach. Again 50% or 1617 of those messages are one-hop buddy messages. 904 beacon messages are exchanged amongst the active nodes. The individual load of a single active node, though, is far less than the server load in the centralized approach. Now also a significant amount of fail packets have to be sent. This kind of message includes notifications to active nodes if another active node changes its state. The other message types are negligible except the request messages that in average sum up to 95.2 messages and describe the messages exchanged to build up a partnership between two active nodes.
The simulations also showed that the distributed approach is much more resilient against node failure. The problem of simultaneously failing buddy and

active nodes also persists here. If someone wanted to decrease the risk of such an unlikely event even more, multiple buddies would be the answer here as well.

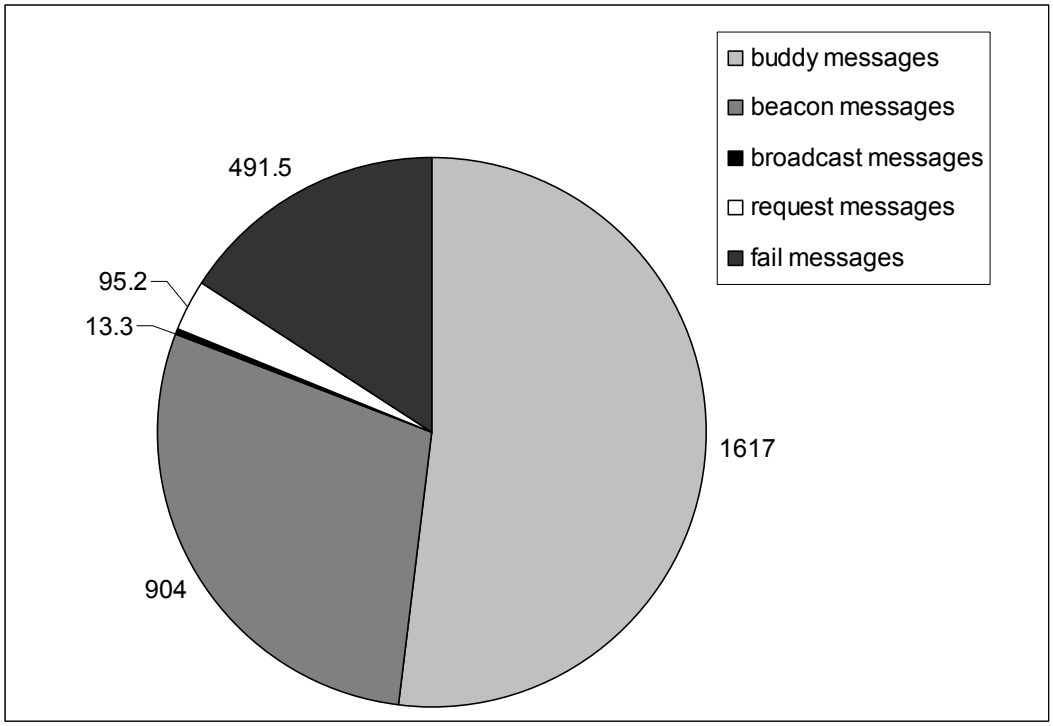

Fig. 5.37 Maintenance cost in number of messages sent for the distributed approach during a 500s simulation run

The effectiveness of the distributed system is illustrated in Fig. 5.38. It shows the different partnerships that exist among the active nodes in the network. The separate partitions both contain 5 active nodes. 15 partnerships span the partitioned region. That means that 15 partnerships have to be destroyed in order to disrupt a successful partition detection. In this particular case it would mean to let nearly all active nodes fail simultaneously without giving the system time to recover.

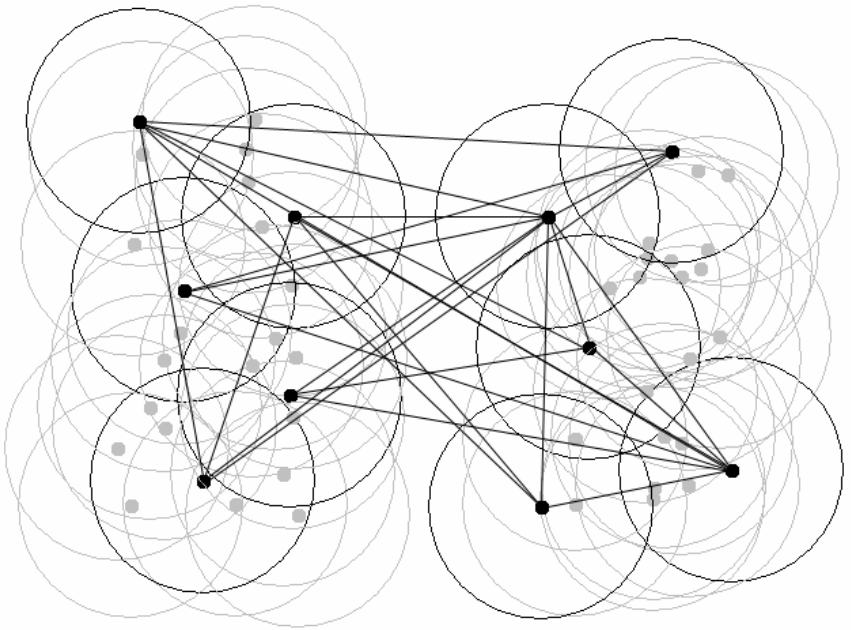

Fig. 5.38 Partnerships amongst nodes during partitioning

5.3.5. Conclusion

In this chapter two different approaches to detect network partitioning were evaluated. Both approaches are based on the notion of border nodes and their successful identification.
With both systems one is able to distinguish node failure from network partition, with both systems primarily differing in terms of resilience against failure and network load. Both approaches explicitly select the best suited nodes and are also able to distinguish node failure from network partitioning. Our distributed approach is still extremely lightweight, of course depending on the temporal granularity of the detection mechanism, resilient, and efficient.
Both our approaches have unique advantages. The centralized approach generates a by far lower message overhead compared to the distributed approach. It is in its structure much simpler, but burdens one single node far more than the rest of the nodes. The centralized approach also has some critical system states. For example during the time between server failure and the time when all active nodes registered at the new server the network is completely unmonitored. The same problem occurs during the time when a new server has to be elected in a separated partition. The server election phase itself could be a complex and costly task in terms of network load and system downtime.
The distributed approach is far more resilient against node failure. Multiple partnerships make sure that a single or more failing nodes only reduce the monitored area of the affected nodes temporarily. That also has the effect that there is no downtime in that system except if a large number of nodes fail or an extremely unfortunate combination of nodes fail simultaneously. That also makes the system more stable against malicious nodes trying to disrupt the system. These positive properties of course come at a cost. The distributed approach loads the network far more than the centralized approach.
The whole system was built in user space, i.e. in the application layer. It serves as a good example to compare against a CrossTalk-based equivalent as it can be argued that at the application layer, the same functionality can be implemented in this case. The business logic of the system itself, i.e. the exchange of monitoring messages can under some circumstances be made more efficient using CrossTalk. The main advantage would be though that the enabling algorithm, i.e. the border node identification can be made more efficient and precise and low cost. The algorithm itself is per se a global adaptation problem and therefore CrossTalk's Global View mechanism should be employed. Therefore, the partition detection system is a good example of the novel applications CrossTalk enables.

5.4. Global View Scalability Enhancements

Optimizations based on the relative state of a node inside a network, i.e. based on the comparison between Local View and Global View, must be based on reasonably correct data the both views provide. The Local View will no doubt be correct as the data provided origins at the local network protocols and they got this data through measurements or directly from their own data repository. The Global View though is constructed based on information gathered by the data dissemination process employed by CrossTalk. This section therefore solely deals with the quality of the Global View and the corresponding cost involved [54].

Furthermore, some the cache replacement algorithms for the case a device is unable to store all incoming Global View samples are evaluated. This analysis also answers the question if it is really necessary to store each and every sample to generate a reasonably correct network-wide view. Furthermore, the stateful approach as described in section 4.3.1 is analyzed according to the relative savings compared to the stateless approach.

5.4.1. Experimental Setup

A vast amount of experiments were carried out, again inside the ns-2 simulator. The metric that was disseminated in these experiments and on which the Global View data is based on was a load metric [53] generated as described in section 5.1. The routing algorithm used was again the Ad-hoc On-demand Distance Vector (AODV) algorithm where we only enriched route requests and route replies in addition to application packets. `Hello` messages were enriched with information of one-hop neighbors since we did not use the information of direct physical neighbors for the calculation of the global view.
Having learned from the impact of different network settings analyzed in section 5.1, we carefully chose our simulation parameters. Every simulation carried out included churn, i.e. the leaving and joining of nodes. Churn can have a significant impact on the global view as newly joined nodes start out stateless and have to establish their global view from scratch. Therefore, we evaluated, as we believe, a quite severe churn rate as we did before. Each node stays for at least 60 seconds in the network and stays for 250 seconds at maximum, failing within this 190 second timeframe according to a uniform distribution.
To help nodes to establish their global view faster, we utilized the previously mentioned initialization procedure. A node would send a packet with a time-to-live (TTL) of one hop requesting the global view information of its neighbors to initialize its own. For the case of AODV that could be coupled with the `Hello` messages.
The area of each simulation has an aspect ratio of 1:4 and a density of 50 nodes per km². Mobility, as shown in section 5.1 proved to be – if anything – marginally advantageous so we did not simulate mobility extensively. The high churn rate in fact can be seen as discrete mobility in a way with the additional penalty of loosing the obtained state information.
For the traffic generation, we chose relatively low load scenarios. The reason for this is that the higher the load is, the better CrossTalk will perform since it has a larger choice of packets to enrich. We evaluated two kinds of traffic patterns, one where every node generates the same amount of traffic. Every two seconds, a node would send a packet to some fixed destination and after 40 seconds it would change its destination. Sending packets at that rate is, as we believe, extremely low. The other traffic pattern employed distinguishes between three traffic classes as introduced in section 5.2. The first class is an extremely low traffic generator where only every four seconds a packet is sent to a destination and destinations are changed every 50 seconds. The second class sends a packet every 2 seconds and changes destinations every 30 seconds. The third class sends 2 packets per second and changes destinations every 10 seconds. The distribution of the classes is exponential, which means that there are roughly twice as many class-2 nodes as there are class-3 nodes and twice as many class-1 nodes as there

are class-2 nodes. In other words, there are only few "high" traffic nodes but many low and mid-range traffic nodes in the network.
As for the cost of storage, we evaluated several cache sizes in combination with several network sizes and load settings.
The size of the tested networks ranged from 50 to 400 nodes to be able to derive scalability trends. Each simulation had a duration of 600 seconds. The first 100 seconds were disregarded as a startup phase since the whole network started from scratch without any state at all (even without any routing state). Over the remaining 500 seconds the results were averaged.
The parameters to reflect the quality of the global view are equal to those found in section 5.1 with the exception of one parameter. We added the relative error of the global view to reflect its quality. The relative error in our case is calculated by looking at each node individually and by determining the relative error of the global view generated at each node as compared to the actual, correct global value, which is calculated artificially. The relative error in our analysis is the average of all relative errors in the network. The second metric is our correctness metrics as introduced in chapter 5.1. The third metric is the standard deviation of the global views across all nodes in the network. Since we only compare corresponding simulation parameters, we do not calculate the coefficient of variance as the standard deviation can be compared directly. It reflects the uniformity of the global view, i.e. the degree of dissimilarity of the individual global views within the network.

Table 5.4 Scalability experiment parameters

Simulation Parameter	Parameter range
Network density	50 nodes/km²
Network size	50 – 400 nodes
Topology aspect ratio	1:4
Traffic generation	0.5 pkts/s or exponential traffic pattern
Churn	each node fails every 60 – 250s

5.4.2. Experimental Results

The experiments had two objectives both in respect to scalability. On the one hand the necessary overhead in terms of cache size for the Global View component should be analyzed. The experiments described so far all assumed that memory for the Global View cache was of no concern. So the experiments should reveal if CrossTalk is feasible with very limited memory availability. But not the size alone was a focus but also finding the most suitable cache replacement strategy as described in section 4.3.2.
The second objective was to find out how well the stateful approach performs in direct comparison with the stateless approach, which was the approach employed

in the previous experiments. That comprises the actual overhead induced by both approaches on the one hand and the quality of the global view on the other hand.

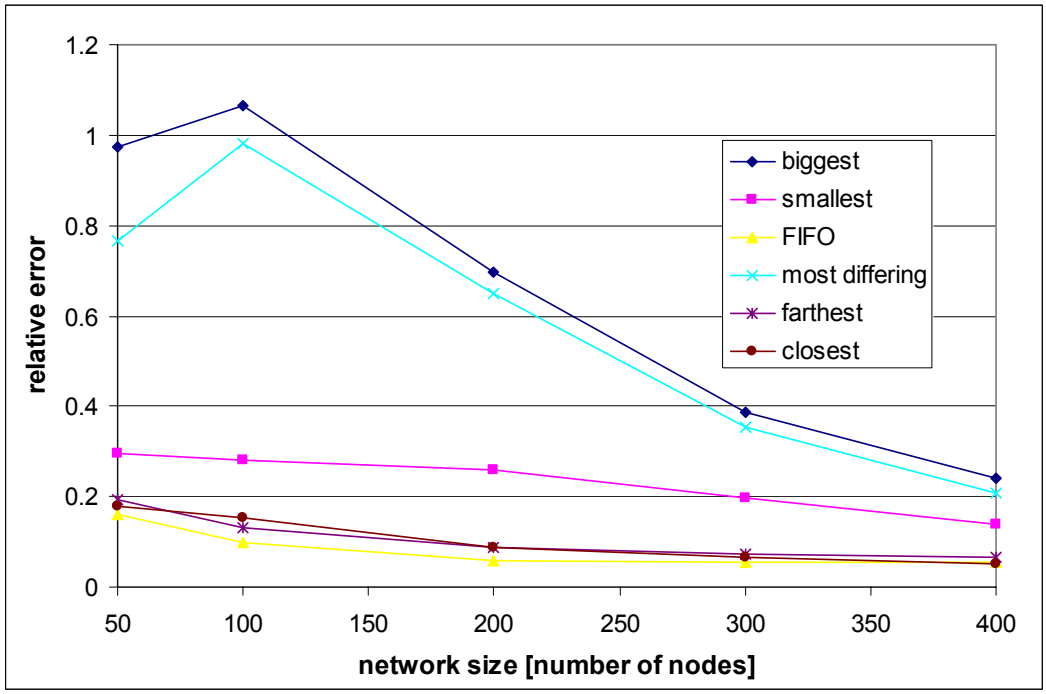

Fig. 5.39 Correlation between network size and the relative error at a fixed cache size of 50% of the number of nodes for different cache replacement strategies

In Fig. 5.39 the relative error of the Global Views for all caching strategies are displayed in increasingly large networks with caches that are able to store data samples from 50% of the participating nodes. In other words the relative cache size according to the number of nodes participating stays constant along the x-axis. The graph therefore represents scenarios in which a node cannot store all possible disseminated values and has to employ one of the cache replacement strategies. As can be clearly seen, the FIFO strategy performs best in all network sizes, closely followed by the strategies which replace entries according to the distance metric (closest/farthest). FIFO maintains a relative error well below 0.2 in all scenarios. It even displays a decreasing relative error with increasing network sizes. From the 50-node network to the 400-node network, the relative error is lowered by a factor of 3 to roughly 0.05 for the FIFO strategy. This trend hints at the scalability properties of the CrossTalk data dissemination technique since growing network sizes do not have a negative impact on the state of the global view.

The “most differing” and “biggest” replacement strategies perform worst. For small networks, the global view is only an extremely rough estimation of the actual situation. For adaptation or optimization processes or the alteration of protocol behaviour such a global view would be useless. With increasing network sizes, the performance increases here, too, but the general trend shows the importance to consider large, or the most differing values in the calculation of the global view. In general, the results imply not to replace cache entries on the basis of their value with the exception of the smallest values as the corresponding replacement strategy performs reasonably well. An explanation to this phenomenon has to deal with the fact that the disseminated value from which

the global view is constructed is the load metric presented in section 5.1.2. In small networks there is a relatively small core with only few nodes. These nodes though will be highly loaded as they are most likely on the shortest path. By replacing the values of those few nodes which have a big impact on the actual network-wide average the global view's quality suffers strongly. With an increasing network size the core of the network grows, the load inside the core will be slightly better distributed and there are relatively and of course absolutely more nodes that actually have to carry a higher load burden. By replacing the sample in the Global View that represents the highest load the overall effect on the calculated network-wide average is less strong. Therefore, the corresponding graphs decrease with much stronger than the others with growing network sizes. The reason why all graphs in Fig. 5.39 decrease with growing network sizes is that the strong discrepancy created by the border effects become less pronounced in larger networks and because there are absolutely more cache entries to choose from when a new sample arrives. Fig. 5.40 confirms these observations.

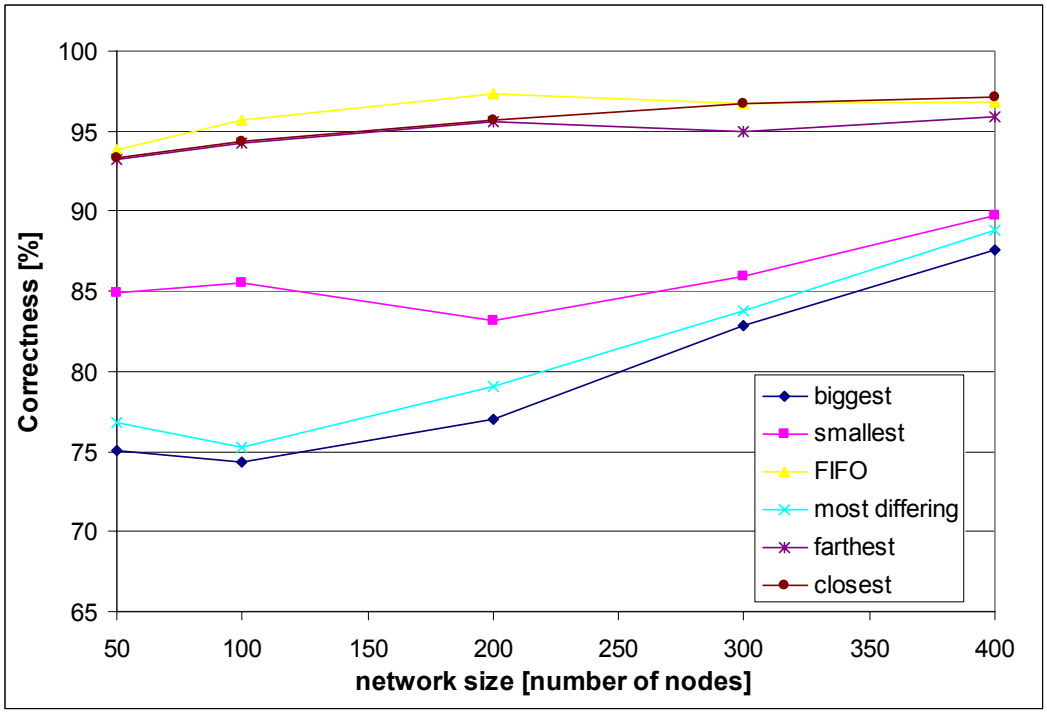

Fig. 5.40 Correlation between network size and the correctness at a fixed cache size of 50% of the number of nodes for different cache replacement strategies

Fig. 5.41 displays the dependency of the quality of the global view on the cache size used for the FIFO replacement strategy. As it turned out to be the best choice we continue to have a closer look at it. The cache size in Fig. 5.41 is expressed as the fraction of data samples it can hold over the number of nodes in the network. A cache size of 10% in a 100 node network implies that a node can hold 10 data samples and so forth. Here again the scalability properties can be seen. With growing network sizes, an increase in the cache size becomes more and more insignificant. That means that the quality improvements grow sub-linearly with increasing cache and network sizes. As an example let us look at the 50 node curve at 100% cache size. Here we have a cache able to hold 50 data samples and the relative error is at roughly 0.13. The same absolute cache size,

i.e. a cache that is able to hold 50 data samples, we find at 50% for the 100 node network curve, at 25% for the 200-node network and at 12.5% for the 400-node network. With the exact same absolute cache size in those networks, the relative error drops to 0.08 for the 400-node network. That clearly shows that it is not necessary to invest more memory for increasing network sizes. With a fixed cache size, the actual relative error is dropping when the network grows. Memory-wise, CrossTalk's global knowledge approach clearly scales and, with relative errors well below 0.1, it also provides a high quality of the global view.

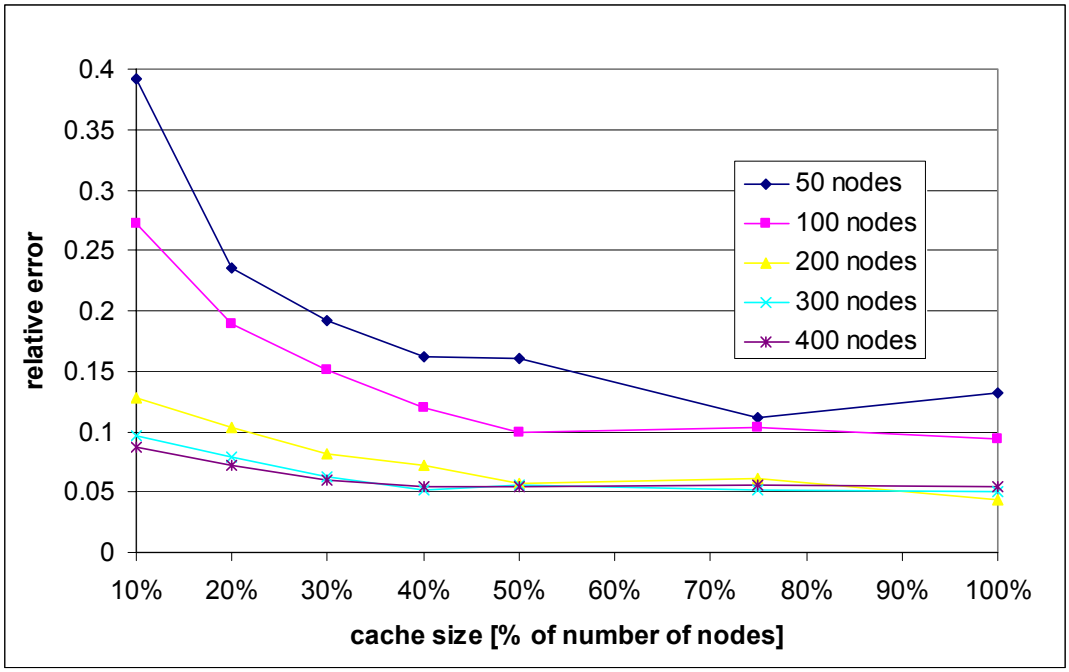

Fig. 5.41 Dependency of the quality of the global view on the index size when using a FIFO replacement strategy in networks of different sizes

Fig. 5.42 and Fig. 5.43 show the standard deviation of two selected cache replacement (FIFO/Biggest) strategies for differently sized caches. In small networks and with small cache sizes, the FIFO strategy does not perform as well in terms of uniformity. The standard deviation is relatively high in these networks under the previously mentioned conditions. The Biggest replacement strategy exposes roughly the same uniformity degree across all network sizes for caches with a relative size of 10% of the network size. But very quickly with growing network sizes and also cache sizes, FIFO again outperforms the other replacements strategies as the uniformity changes similar to the relative error. As can be expected, all approaches perform identically when approaching cache sizes that are able to hold all possible samples from throughout the network.
The presented standard deviations differ significantly in their overall behavior with growing cache sizes. For the FIFO strategy there is a clear trend that with a growing cache the standard deviation drops, at first very fast and then only slowly. The reason for this clear trend is very simple. Samples get replaced according to their time of entry into the cache. Therefore, for small caches, the actual value distribution inside the cache is quite random. As only few samples can be stored the overall standard deviation of the global views of all nodes in the network is high. The bigger the cache the less impact this randomness has as the sample base grows.

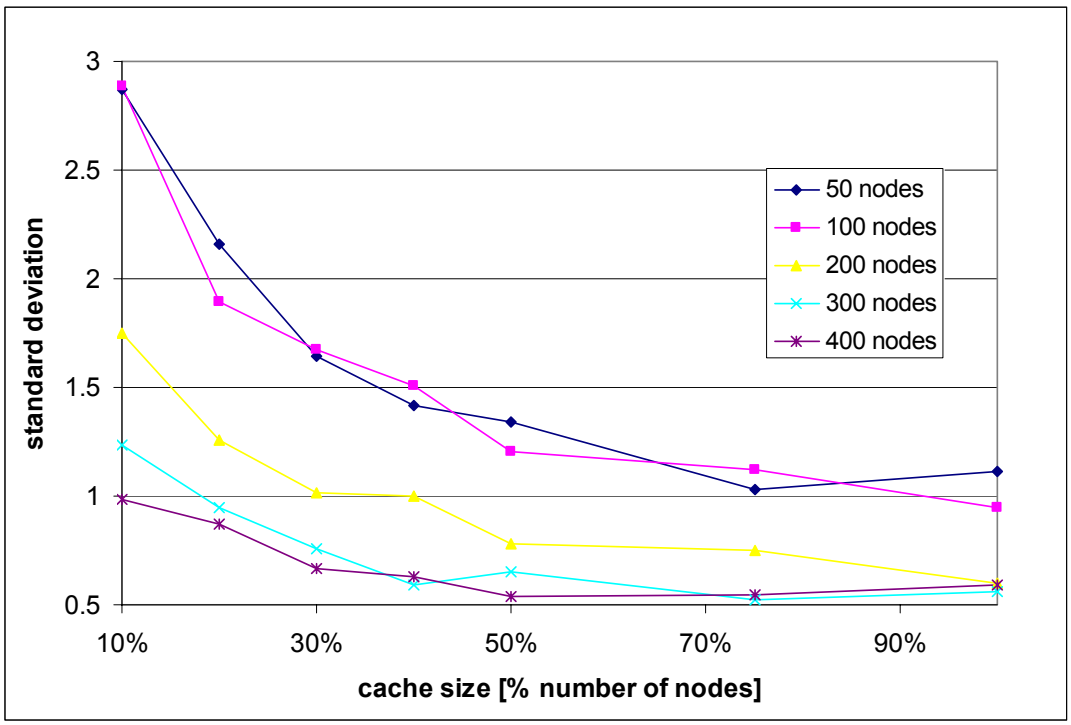

Fig. 5.42 Dependency of the uniformity of the global view on the cache size using the FIFO replacement strategy in networks of different sizes

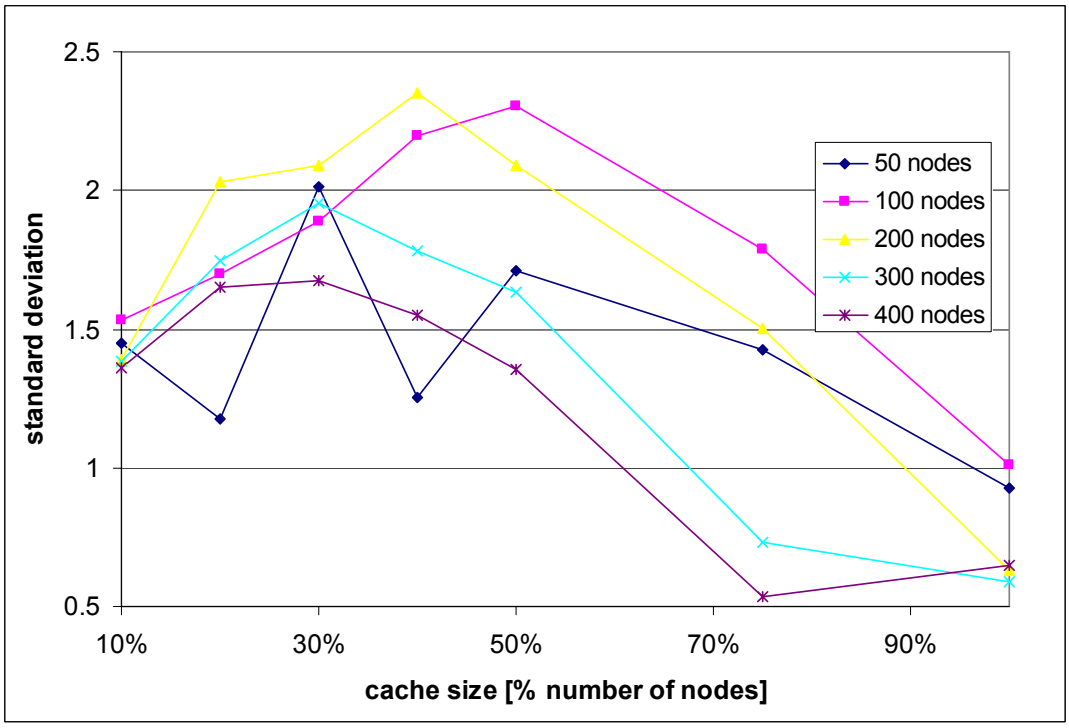

Fig. 5.43 Dependency of the uniformity of the global view on the cache size using the Biggest replacement strategy in networks of different sizes

On the other hand the Biggest replacement strategy has a direct impact on the actual values that reside in the cache. As the biggest samples are purged from the cache with a high probability in favor of a smaller value, the standard deviation for small networks with small cache sizes is smaller as compared to the FIFO strategy. The standard deviation grows at first as in some caches now a few

big values will remain whereas in others all will be replaced with smaller values. This creates less uniformity across nodes. This effect at some point, depending on the network size, fades with an increase in the cache size as can be seen in Fig. 5.43.

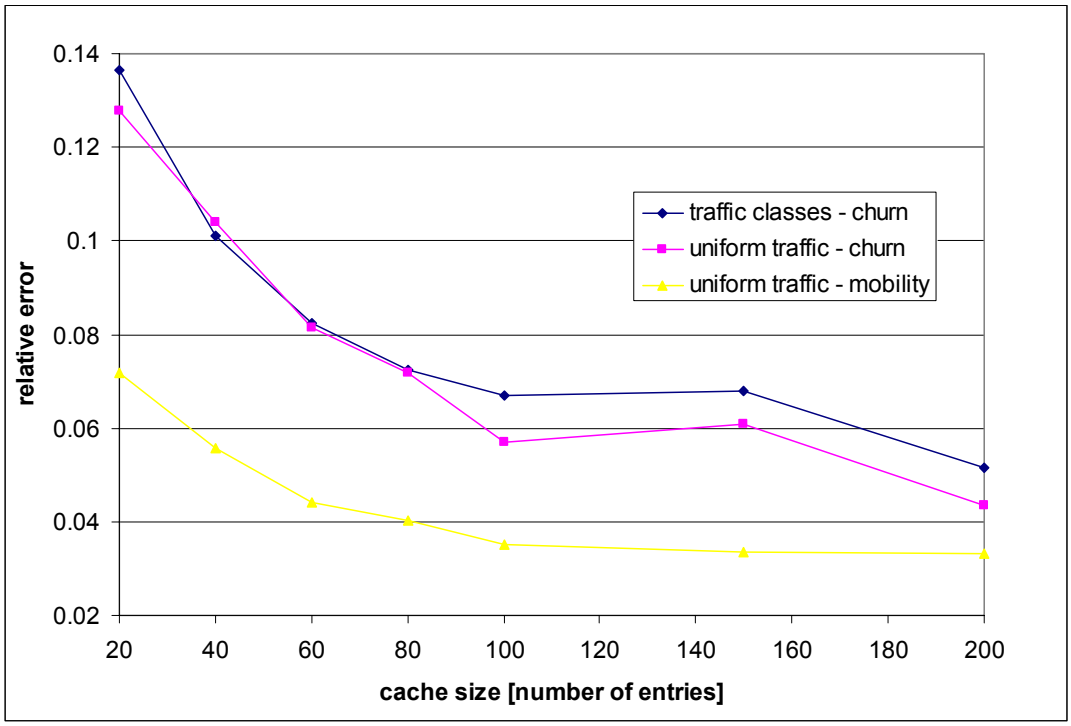

Fig. 5.44 The impact of churn and the traffic pattern using the FIFO replacement strategy in a 200 node network

One important aspect of all our simulations was the high churn rate. The effect of churn and different traffic patterns can be seen in Fig. 5.44. The curve with the lowest relative error denotes the simulation setting without churn, following the Random Waypoint mobility model at 1.4 m/s and a pause time of 10s. As can clearly be seen, the more stable in terms of node departure and arrival a network is the less caching space is needed to achieve a global view of a very high quality. In the tested scenarios, churn-less networks achieve twice as low a relative error at same cache sizes. The traffic pattern on the other hand does not seem to have a highly significant impact on the quality of the global view as the uniform traffic pattern and our simulation using exponentially distributed traffic classes perform similarly.

So far all, results utilize the stateless data dissemination technique of CrossTalk. The remainder of this section deals with the changes of the global view when utilizing our new stateful dissemination technique and with the respective cost reductions in terms of communication overhead. The threshold for new dissemination in case the disseminated value changed was chosen to be 10%, and the local information should also be newly disseminated after one third of its useful lifetime expired, i.e. one third of the time a sample is held inside the Global View component before being purged. This way not every packet needs to arrive at a node in order to keep the sample inside the Global View cache on a path.

By adding state information, we somehow decoupled the absolute amount of data that is generated by source nodes from the traffic pattern. As an example

consider an application that sends 20 packets per second to some other remote application. In the stateless approach we would piggyback local state information 20 times. If it would send only half that amount we would also only half as often add additional information, i.e. the overhead we produce is directly in correlation with the traffic or load pattern. With the stateful approach we would add the exact same amount in both scenarios, absolutely speaking of course. What we have not done is decoupling the overhead from the number of participating nodes in the network. This is partly due to the choice of our traffic pattern. Since we choose our destinations randomly, the average path lengths grow with the network size naturally. The absolute overhead we measure and display in the corresponding graphs is not the amount of packets that are enriched at each source node, but we also count every enriched packet that has to be forwarded.

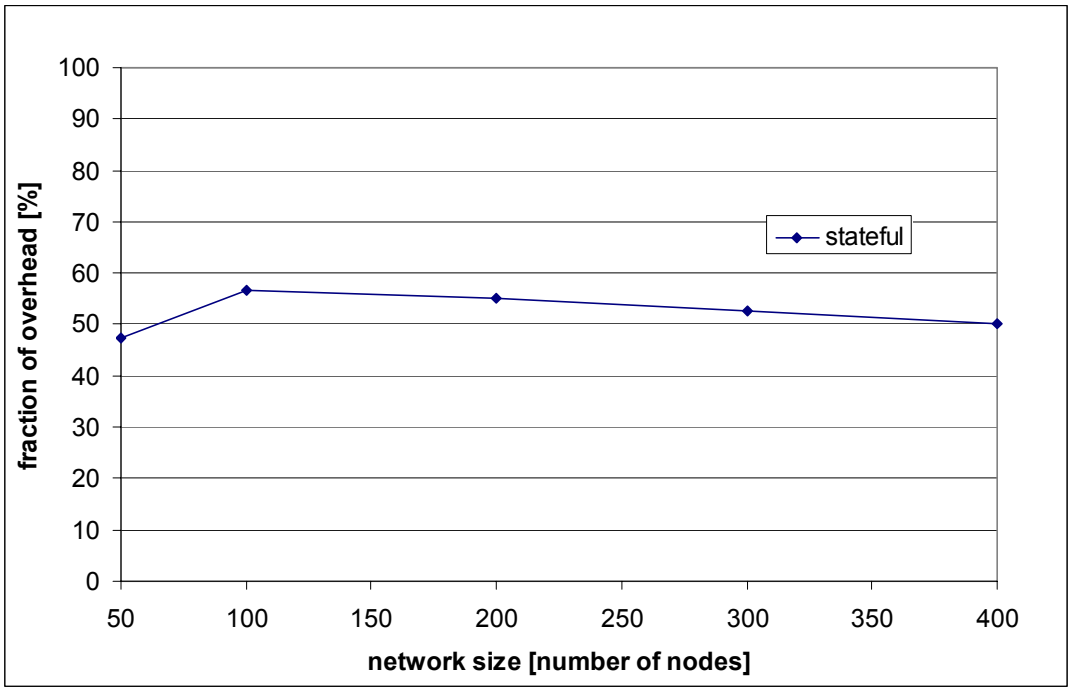

Fig. 5.45 Overhead reduction of the stateful data dissemination as opposed the stateless approach

In Fig. 5.45, the savings are displayed in reference to the stateless approach. Compared to the stateless alternative, we saved roughly 50% of the generated overhead in our tested scenarios. This value remains relatively constant with growing network sizes. Therefore, the relative savings remain constant or in other words with the chosen traffic pattern we constantly only generate half of the overhead. The absolute network-wide overhead of course grows as each node has to disseminate its local information and this information has to be forwarded more often. This effect can be seen in Fig. 5.46. The overhead, although small since only the packet size of enriched packets that have to be sent anyway, grows over-proportionally with growing network sizes. Only regarding source nodes, one would see a linear increase. The slight drop of the growth for large networks is due to the fact of an increased number of collisions, reducing the packet delivery ratio. Although the overhead increases, the benefits from applying CrossTalk can easily compensate for that increase as shown in the previous experiment described in sections 5.1 and 5.2.

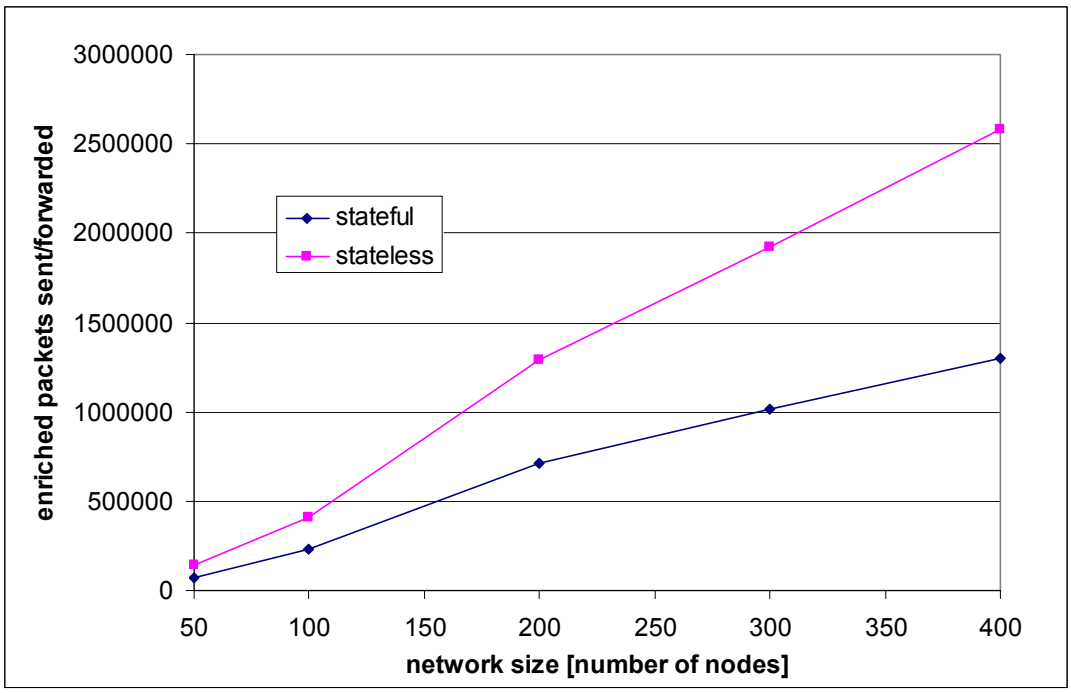

Fig. 5.46 Absolute overhead of both dissemination techniques

As shown, we saved a significant amount of the overhead generated which per se is already small. Since we disseminate data less frequently, there will of course be some effect on the quality of the global view. How strong this effect is, depends on the thresholds chosen for the re-dissemination of local data. For the threshold chosen above, the effect is displayed in Fig. 5.47. We can see, as expected, a slight degradation of the quality of the global view as the relative error increases marginally. Compared to the relative savings the degradation is miniscule and even degreases slightly with growing network sizes.

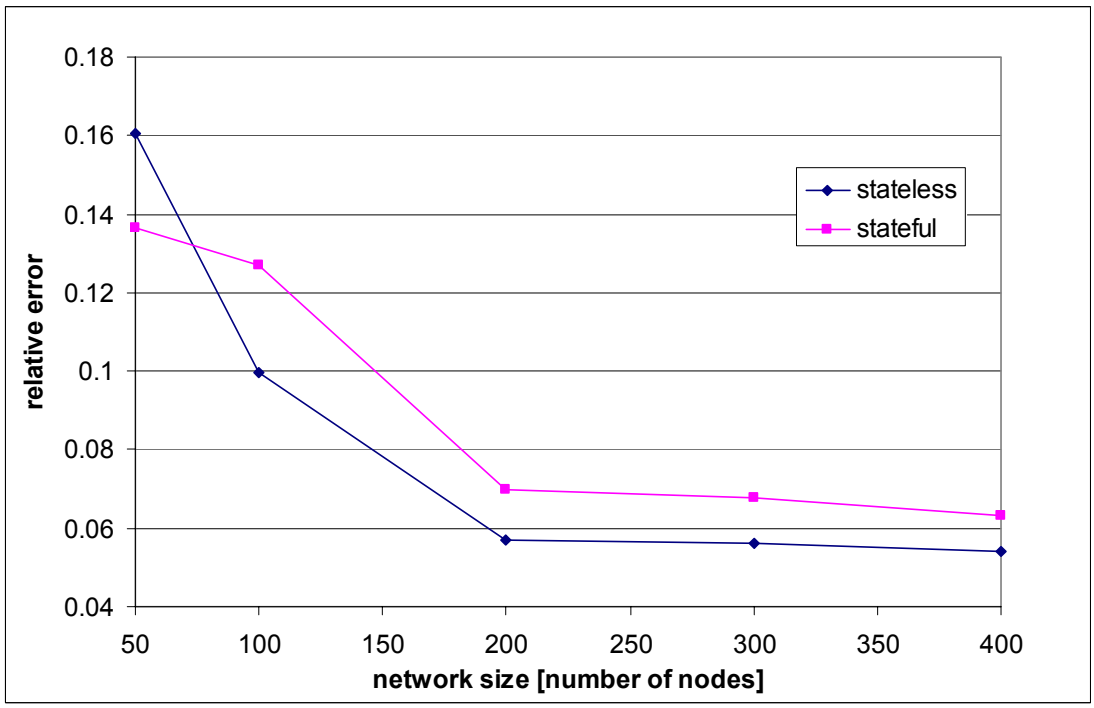

Fig. 5.47 Quality of the global view for stateless and stateful data dissemination

Our dissemination technique has another advantage which has not been mentioned so far. In networks with nodes that only relay packets but not actually generate traffic themselves, these nodes were not able to contribute to the global view.
Now, those nodes could potentially enrich those packets with their local information that were not enriched by the source or any intermediate node. Doing this would increase the amount of data that has to be piggybacked overall though as the intermediate node would need to add its address. This has to be done since otherwise the packet source's local value would be purged along the path from the Global Views of intermediate nodes.
As can be seen from the graphs, the overhead that we calculated increases super-linearly. That is because we include intermediate hops and route request, which are broadcast packets. This might in general seem as if the overall approach does not scale. But it has to be mentioned that the overall size of the data piggybacked is very small. Additionally, no extra control packets are sent keeping the overhead and the overall footprint lightweight. As previously mentioned, the data added to a packet can be seen as an additional header field. In this sense, CrossTalk's Global View mechanism is as scalable as having header information.

5.4.3. Conclusion

In this section, two aspects concerning CrossTalk's Global View scalability were analyzed. The first one concerned the memory side of the sample cache that CrossTalk maintains inside its Global View. The question that was answered was how many data samples should be stored to achieve a reasonably accurate Global View and which cache replacement strategy should be employed. It turned out that a reasonably fixed size cache performs well in all tested scenarios and adding additional storage did not significantly impact the Global View's quality. In other words, memory-wise CrossTalk's Global View is very much scalable. As for the replacement strategy, the FIFO algorithm largely performed all other tested strategies.
The second scalability aspect is the communication overhead introduced. The stateful data dissemination procedure was analyzed in direct comparison with its stateless counterpart. In the tested scenarios, it was observed that the stateful approach can significantly reduce the overall overhead by only slightly deteriorating the quality of the global view.

5.5. Summary

CrossTalk's Global View component is its most outstanding feature when compared to the capabilities of other related cross-layer architectures. It enables network-wide adaptations, global optimizations and novel applications. This chapter introduced several applications of the Global View mechanism including a load balancing scheme and mobility adaptations. The analysis of the quality of the global view CrossTalk provides showed that it reflects the network-wide status very well and it is therefore suitable to base cross-layer mechanisms on it.
The adaptation and optimization mechanisms presented that make use of CrossTalk significantly outperformed the standard approaches. A novel partition detection system was also presented which should be based on CrossTalk.