On the Modular Composition of Scalable Application-Layer Multicast Services for Mobile Ad-hoc Networks

Peter Baumung
Institute of Telematics
Universität Karlsruhe (TH)
eMail: baumung@tm.uka.de

***Abstract*— Mobile ad-hoc networks allow wireless devices to freely communicate without the need of any fixed infrastructure. Although many of the emerging applications share the need for multicast communication, their requirements regarding reliable data delivery strongly differ. One single multicast service will thus not provide an acceptable solution for different application and network scenarios. Instead, a multicast service's flexibility and adaptivity is required in both respects, in order to yield good performance. In this context, application-layer multicast services appear promising: As they rely on so-called overlay networks for data dissemination, they do not require network-wide support and can thus easily be optimized for specific scenarios. We in this contribution propose a novel architecture for the flexible composition of scalable application-layer multicast services. To do so, we subdivide the latter into different modules, such as transport and overlay routing. By making modules arbitrarily interchangeable, we increase a service's adaptability and facilitate its development. The service's scalability is generically ensured, by including our approved technique of Local Broadcast Clustering, which is applicable to arbitrary overlay-multicast algorithms, inside the architecture. As we additionally abstract from a specific network access, developed services can easily be operated and evaluated on top of different network technologies, comprising event-based network simulation software as well as true WLAN-capable devices.**

I. Introduction

Mobile ad-hoc networks (MANETs) consist of mobile devices, that communicate via the wireless medium without any fixed infrastructure. While two devices located in one another's transmission range communicate directly, intermediate devices bridge distances between farther nodes. As multi-hop communication is thus enabled, a complex and potentially highly dynamic wireless network arises. Here, many application share their need for multicast communication. Indeed, cooperating and thus communicating groups will be present in the majority of educational, touristic, rescuing or military scenarios.

From a multicast protocol's point of view, high-level applications mostly differ in their emitted network traffic as well as in their requirements regarding the efficient and reliable delivery of data. A protocol's task is to map these requirements on the communicational capabilities available in a specific network scenario. To maximize performance, multicast services will thus need to be tuned in the context of specific applications on the one hand, and network scenarios on the other hand. Whenever this context changes, protocol mechanisms will need to adapt in order to maintain performance. The integration of multicast services on the network layer of wireless devices (as e.g. proposed in [1], [2]) thus shows to be complicated, as the desired degree of adaptability can hardly be provided network-wide. A more convenient solution can be found in application-layer approaches. These link a multicast group's members using unicast tunnels, resulting in a logical network topology called *overlay network*. While the latter is used for forwarding multicast packets between group members using ordinary unicast schemes, protocol mechanisms and computational effort is entirely left to group members, and can thus easily be optimized for single applications. Intermediate (routing) devices may remain unaltered, as they are required to provide support for unicast routing only.

In the past years, application-layer multicast has been a growing research area, in the fixed Internet as well as in MANETs. Various proposals employing different overlay topologies and routing strategies (e.g. shortest-path-based [3], hierarchical [4], [5], ring-based [6] or source-routing [7]) for data dissemination have been made. Protocol evaluation in most cases is restrained to the measurement of common values within simulation environments, such as link stress or control flow overhead. Although these factors are fundamental for the proper characterization of application-layer protocols, two essential issues often remain unaddressed. On the one hand, the evaluation of data delivery is mostly done in the context of simulated CBR streams using different loads. The protocol's efficient functioning, when opposed to traffic emitted by *true* applications in real-world environments, thus remains questionable. On the other hand, evaluations of application-layer protocols usually leave reliability to TCP [5]. As the latter is known to achieve unsatisfactory results over MANETs in its standard implementation [8], various proposals have been made to increase its efficiency over MANETs [9]. The achievement of optimal performance using *one single* transport protocol for different multicast applications and network scenarios however is uncertain. We argue that especially transport mechanisms as well as routing algorithms must be tuned, in order to maximize performance in the context of specific application and network scenarios.

When developing efficient application-layer multicast ser-

This work arose from the MAMAS project which is funded by the German Research Foundation (DFG) within the SPP 1140.

vices, it thus seems inappropriate to fix the developed service to one specific overlay-multicast algorithm or reliability mechanism. Keeping the service's flexibility in every possible respect becomes a key factor for achieving good performance in the context of different applications. We in the following section thus propose a novel software architecture: Because of its design, the architecture allows the modular composition of scalable application-layer multicast services for mobile ad-hoc networks. Modules hereby comprise optimized reliability mechanisms, arbitrary overlay-multicast algorithms and generic performance extensions for overlay topologies. By fixing interfaces, modules become interchangeable, leading to flexible, scalable and highly adaptable multicast services. Additionally, the architecture enables the operation of services inside network simulation software as well as on real WLAN-capable devices without any changes of source code. We thus not only support the simulative evaluation and the respective tuning of single protocol mechanisms. Instead, we also facilitate the service's migration between simulation and real-world environments, and thus enable its evaluation in the context of true MANET applications.

Note that this contribution is on presenting the architecture only: Neither the detailed description of overlay-multicast algorithms and reliability mechanisms, nor extensive evaluations are subject of this paper.

II. The Modular Architecture for Application-Layer Multicast (MAAM)

In this section we present the *Modular Architecture for Application-Layer Multicast* (*MAAM*). Conforming to aspects of modular protocol design [10], we with this architecture enable the flexible composition as well as the easy adaptation of application-layer multicast software to different application and network scenarios. In Section II-A, we first give a brief overview on the architecture and list its most attractive features. As the MAAM defines modules following an object oriented design, Section II-B discusses the modules' responsibility and interchangeability by presenting the employed object classes and interfaces. Section II-C clarifies the architecture's event handling as well as the modules' interaction at the example of data forwarding.

A. Overview and Features

The MAAM is an object-oriented framework for the modular composition of scalable application-layer multicast services. The framework enables a protocol's adaptability by identifying different interchangeable protocol components. As can be seen from Figure 1, the framework thus decomposes a multicast software into different sub-layers. These comprise

- an application sub-layer implementing a user interface and emitting multicast traffic,
- a transport base sub-layer that integrates customized protocol mechanisms reflecting the application's requirements regarding reliability,
- a multicast base sub-layer that handles data forwarding and manages a member's connectivity inside the multicast group,

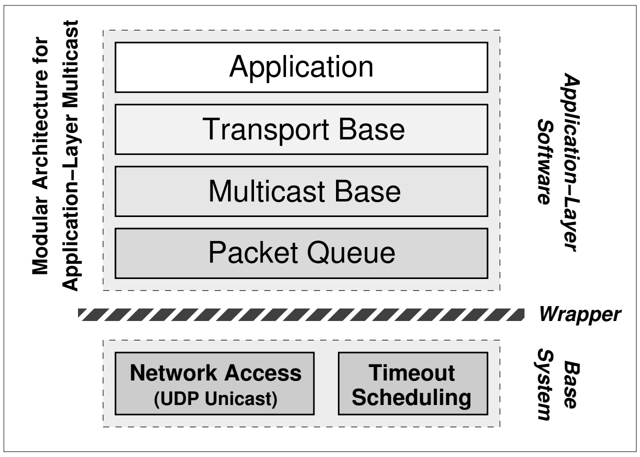
Modular Architecture for Application–Layer Multicast
Application
Transport Base
Multicast Base
Packet Queue
Application–Layer Software
Wrapper
Network Access (UDP Unicast)
Timeout Scheduling
Base System

Fig. 1. The MAAM's division into sub-layers.

- a packet queue sub-layer that integrates different packet managing strategies.

All sub-layers are designed as modules that by source code compilation are unified into one single executable application. One of the MAAM's main aims is in defining fixed interfaces between these modules, in order to make all of them arbitrarily interchangeable and thus easily reusable as well as comparable. By doing so, the MAAM facilitates a protocol's adaptation e.g. by allowing the combination of different acknowledgment strategies (implemented by different transport modules) with diverse overlay topologies (made available through different multicast base modules).

In previous studies, we presented our technique of *Local Broadcast Clustering* (short *LBCs*[1]). We showed that the use of LBCs avoids the most important scalability issues of overlay-multicast algorithms in MANETs, and thus leads to drastic performance increases [11], [12]. We thus decided to include generic LBC support in the MAAM. By doing so, we make the LBC extension available and applicable to *arbitrary overlay topologies*. Note that the LBC support is not directly visible in figure 1, as it transparently incorporates in transport and multicast bases.

As common for P2P software, the MAAM requires a base system providing network access. To achieve a high degree of portability, we in this work slightly extend a base system's responsibility. As can be seen from the bottom of Figure 1, a base system in our case not only provides network access, but also timeout scheduling. Our architecture thereby gets completely independent of a specific operating system, as it no longer contains any close system interaction. This however requires the architecture to be bound to a base system using a wrapper that encapsulates system specific issues, such as e.g. socket and timeout management. Summarizing, our architecture offers

- the modular integration of transport mechanisms tailored to a specific application's requirements and arbitrary overlay-multicast algorithms,

[1]A short summary of Local Broadcast Clusters and their effects on application-layer multicast services is given in the appendix at the end of this document.

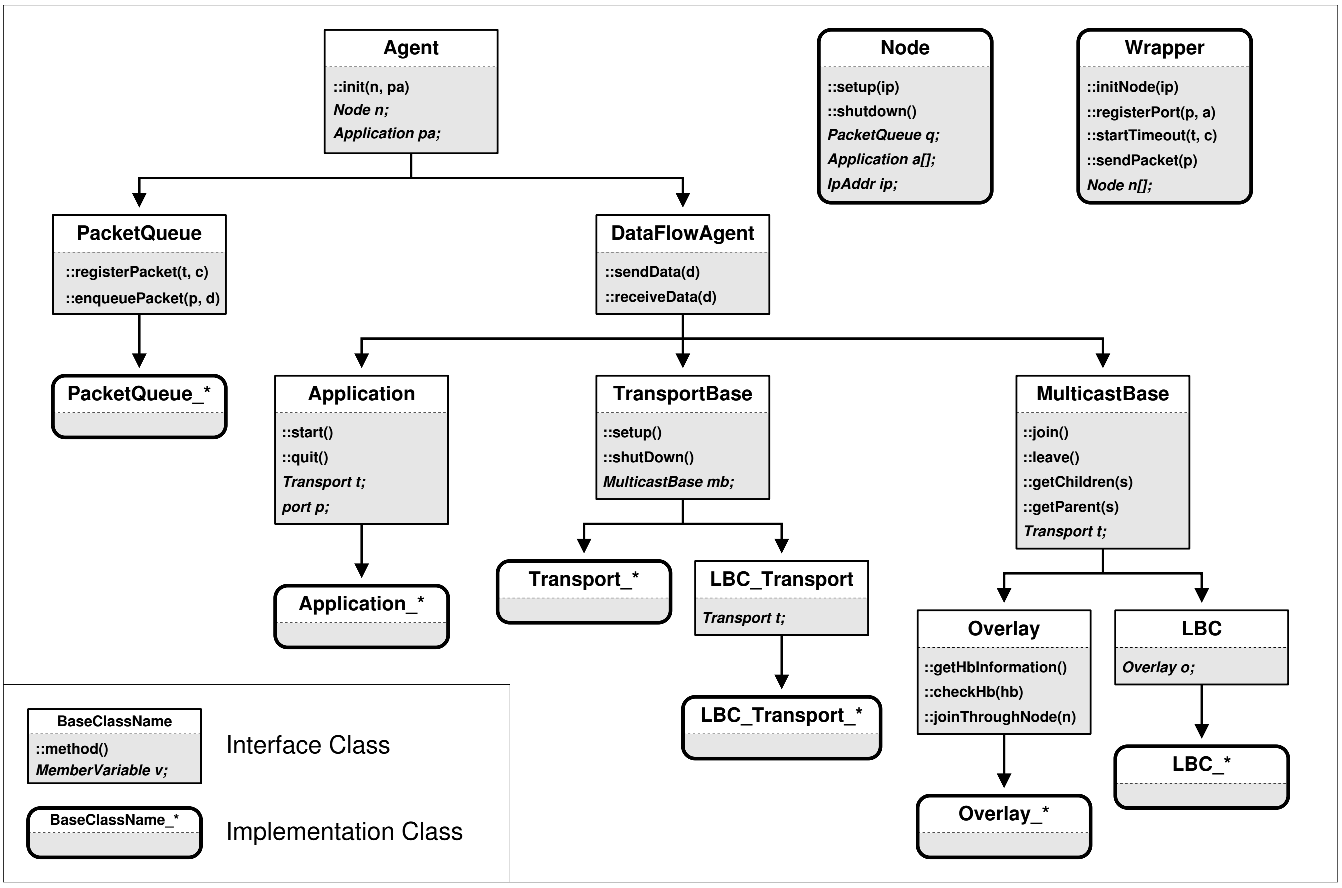

Fig. 2. Simplified inheritance diagram for the Modular Architecture for Application-Layer Multicast.

- standardized interfaces between all modules,
- full interchangeability of modules and thus easy performance comparisons of modules,
- generic scalability enhancements using our approved Local Broadcast Clustering technique,
- flexible composition of overlay-multicast software by reusing previously implemented modules,
- a portable design that can be bound to arbitrary base systems using respective wrappers,
- a system-independent operation and thus simple migration of services e.g. between event based simulation environments and real-world devices.

B. Modules and Interfaces

This section presents the MAAM's most important modules and discusses their particular responsibilities and interfaces. The object oriented design, guaranteeing the arbitrary interchangeability of modules, is shown in Figure 2. The diagram differentiates between *Interface Classes* and *Implementation Classes*. The former are required for interface declaration and contain methods and member variables that we identified as elementary for the interaction of the MAAM's sub-layers (cf. Figure 1). Implementation classes usually extend an interface class, in order to integrate true functionality such as overlay-multicast algorithms or acknowledgment strategies. The following sections thus mainly focus on interface classes, as they reflect the architecture's properties.

1) Agents: form the super class of every module that integrates functionality. Agents know on which node n they run (e.g. for accessing packet queue sub-layer), and of which parent application pa they are part of. Both of these parameters are set when initializing the agent using its init constructor.

2) Data Flow Agents: are derived from (standard) agents and extend their interfaces for the handling (sending and receiving) of data packets[2]. They thus form the super class of modules that actively process multicast packets or their headers.

3) Applications: While true multicast applications will most likely handle user input via complex user interaction, we for the development of the MAAM restrain the interface of application modules to basic functionality such as startup and finalization. As application modules will process multicast packets, they extend data flow agents and refine the latters' interface by the elementary methods mentioned above. Note that during its initialization, it is the application's task to configure the multicast service that will be used for data dissemination: Indeed, when created, an application instantiates a specific transport as well as a multicast base module. For passing multicast data between modules, the latter are then linked by setting the modules' respective member variables

[2]When speaking of data packets, we refer to packets containing true multicast data and not e.g. control packets or acknowledgments.

(transport modules e.g. disseminate data using a multicast base mb, while the latter hand received data to a transport t). After its initialization, an application however is only required to know what transport module t it uses for data dissemination and on which port p it runs.

4) Transport Bases: implement reliability mechanisms as required by a specific application. Their main task will thus be to integrate the handling of packet losses, by acknowledging data and triggering/performing packet retransmissions. To achieve optimal performance, these procedures are required to be tuned depending on an application's characteristics. Indeed, the use of positive or negative acknowledgments (or a combination of both) is known to have different effects on latencies as well as throughput. In order to offer the highest degree of flexibility, neither the strategy employed for acknowledging data nor the (packet) format used for acknowledgments is specified by the MAAM: It is entirely left to transport module implementations. The latter can thus be highly optimized, in order to accurately meet an application's reliability requirements. Although they usually will be optimized for one application, transport modules may however be used by other applications because of their exchangeability. They can thus easily be evaluated and compared in the context of different traffic scenarios. As transport modules directly affect data dissemination, they refine the interface of data flow agents by including two generic methods for setting up and shutting down the transport[3]. For data forwarding, transport modules additionally rely on a so-called multicast base mb which is presented in the following section.

5) Multicast Bases: are responsible for ensuring a member's connectivity within the multicast group. Their interface thus offers two abstract methods, join and leave, via which the joining to and leaving from a multicast group can be triggered[4]. As they provide group connectivity, multicast base modules on the other hand also manage neighborhood information, which is required for data forwarding and acknowledging. Multicast base modules thus offer two additional methods, getChildren and getParent. Note that both methods take an address s as argument, as returned values will most likely depend on the address of a specific multicast source: While the first method is used for retrieving the next-hop nodes to which multicast data is forwarded, the second method returns a list of parent nodes from which e.g. packet retransmissions may be requested. Although multicast base modules manage group connectivity, they do not directly implement overlay-multicast algorithms. Instead, the latter will be integrated by so-called Overlay modules, which are explained in the following section.

6) Overlays: integrate entire overlay-multicast algorithms. They will thus manage overlay connections to other group members and be responsible for the routing of multicast data. Since overlay modules (just as all modules) have a generic interface, they may be used by arbitrary transport modules. The latter are thus unaware of the overlay topology they actually use for data dissemination. Note that overlay modules extend communication bases, which provide the interface used by transport modules. Transport modules will thus only access the multicast base part of an overlay's interface. This differentiation between multicast bases and overlay modules has been made in order to provide transparent support for *Local Broadcast Clusters*. These (and the overlay module's additional interface methods) are explained in the following section.

7) Generic Local Broadcast Clustering: Although we included generic support for LBCs in our architecture, their use is entirely optional and fully transparent for standard transport and overlay modules. This is achieved by making the LBC functionality available through LBC and LBC transport modules that extend multicast base and transport base modules respectively. Applications will thus be unaware of using an LBC transport module instead of a (standard) transport module, as both provide the same interface from the application's point of view. When employing LBCs, group members are connected either within the overlay, or inside an LBC. Transitions between both states however are possible at any time. The design goal of generic LBC support was to keep additional functionality in overlay modules as low as possible. To do so, an LBC module maintains a node's connection state and has full control over the overlay module o chosen by the application. It can thus trigger processes for joining or leaving the overlay, depending on the node abandoning or entering the broadcast range of nearby overlay nodes. LBC modules however have no knowledge about the actually used overlay: they might thus be combined with *arbitrary* overlay topologies. LBC modules e.g. are responsible for broadcasting the LBC heartbeats, in case the node has presently joined the overlay. Information about the node's "strength" inside the overlay is retrieved by calling the overlay module's method getHbInformation and included inside the heartbeats. When receiving a heartbeat hb, an overlay node can compare its own strength with the heartbeat sender's strength, by passing the heartbeat to the method checkHb implemented by the overlay module. As the weaker node will retire from the overlay (in order to become an LBC node), we can by appropriately defining heartbeat information e.g. minimize topology reconfigurations inside the overlay. When leaving an overlay node's broadcast range, LBC nodes will have to join the overlay. To enable such nodes to quickly join an overlay, overlay modules additionally offer a method joinThroughNode. The latter takes the address n of a node as parameter, which is known to currently have joined the overlay - this might e.g. the overlay node to which the former LBC node was connected. In analogy to LBC modules, an LBC transport module has control over a (standard) transport module t. LBC transport modules e.g. implement acknowledgment strategies optimized for the (1-hop) broadcast delivery of data.

8) Packet Queues: are responsible for providing packet management as well as different queuing techniques. Agents register for specific packet types, by passing a type identifier t and a callback c to the method registerPacket. When receiving packets, a queue can thus directly pass the received data to the registered callback. While standard packet queues will directly hand packets p enqueued for a destination d to the network, more complex queues can buffer and aggregate packets with

[3]These will be called by the application when started or finalized.

[4]This triggering will usually be done by a transport module, when the latter is set up or shut down using its respective methods.

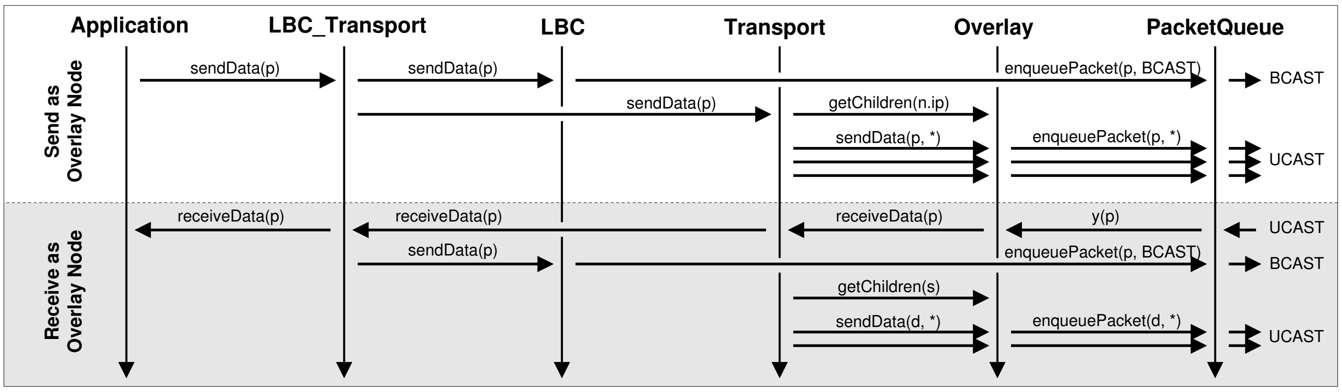

Fig. 3. Simplified flow control chart exemplarily showing the MAAM's data sending and forwarding processes.

the same destination, in order to save medium accesses. Because of a packet queue's exchangeability, the effects of e.g. packet aggregation can easily be studied and compared.

9) Nodes and Wrappers: Nodes encapsulate information which is maintained for single devices, such as e.g. a packet queue q and a list of its running applications a[]. Node structures are only of interest when operating the MAAM within simulation environments, as the latter will usually handle several nodes in parallel. A wrapper, as mentioned in Section II-A, acts as a special layer, linking the MAAM to one specific base system. It encapsulates system specific issues that allow the sending and the reception of packets on different ports. Additionally, a wrapper must handle the scheduling of timeouts using functionality provided by an operating system. This is done, by respectively implementing the method startTimeout which accepts a delay d and a callback c as arguments. All methods provided by a wrapper are globally known to all MAAM modules, so that the latter do not explicitely require a reference to the actually used wrapper.

C. Event Handling and Module Interaction

The MAAM is an entirely callback-driven architecture. CPU time is thus spent inside the MAAM only right after the occurrence of events (timeouts or packet reception). The main execution loop is encapsulated by the wrapper. In case of simulation environments, the wrapper itself will e.g. be called by the environments core, as events occur for the simulated nodes. In real-world environments, the wrapper implements a main loop that listens on sockets and waits for the occurrence of timeouts. As MAAM modules usually won't provide reentrant-safe code, wrappers must also make sure, that only one event is processed at a time.

To clarify the interaction of modules, we in the following briefly discuss the sending as well as the forwarding of data, as it occurs for overlay nodes in case the MAAM's LBC extension is used. This process is shown in Figure 3. Note that we omit the wrapper layer (which would be located on the diagram's right side) as it only translates the queue's activities into system calls. As can be seen, the application sends a data packet (top part of Figure 3) by handing a packet object p to its transport, which in this case is an LBC transport module. As the node has joined the overlay, the LBC transport is required to broadcast the data packet. The module does so by passing a copy of the packet to its multicast base, which in this case is the LBC module. After adding an individual packet header, the latter enqueues the packet at the packet queue for a broadcast destination. In addition to broadcasting, the packet is then required to be forwarded along the overlay. This is done by letting the LBC transport hand the packet to the standard transport module. As added packet headers will usually depend on the (overlay) next-hop, packet duplication for data forwarding is done in the transport and *not* in the overlay module. The former thus retrieves a list of next-hop nodes[5] from the overlay, to which packet copies are then forwarded one by one. For the same reasons as for the LBC transport (addition of packet headers), data is sent via the transport's multicast base and not directly via the packet queue.

Nodes having joined the overlay usually receive data through unicast messages of overlay neighbors (shown by the arriving UCAST message on the bottom part of Figure 3). The queue extracts the packet's type and invokes the respective callback y with the received message as argument. For unicast data packets, the callback will be implemented within the Overlay module. The latter thus receives the packet, processes its headers and passes the packet to its transport, which in this case is the standard transport module. After processing and removing its own packet headers, the transport module hands the received packet upwards to the LBC transport. In analogy to data sending (see above), the packet is then forwarded to LBC nodes (via the LBC transport's broadcast) and to overlay neighbors (via the standard transport module).

The procedure of data sending and reception for LBC nodes are similar. Difference are, that data packets are not forwarded along the overlay, as the node is only locally connected. Additionally, the packet queue will hand data, which is received through broadcast messages, directly to the LBC and not to the overlay module.

III. Summary and Future Work

In this contribution we presented our Modular Architecture for Application-Layer Multicast. By decomposing application-layer multicast services into interchangeable modules, we on

[5]Note that the retrieved list of next-hop nodes depends on the multicast packet's source, which in this case is the node itself.

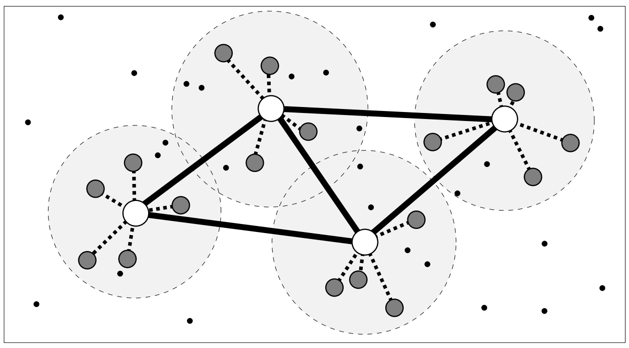

Fig. 4. Local Broadcast Clustering applied to overlay topology.

the one hand enable the service's adaptation in the context of different application and network scenarios. On the other hand, we simplify its development on the whole, by making protocol components reusable: The development of application-layer multicast services thereby can become as easy as plugging different, previously implemented modules into a new executable application. Scalability issues, arising from the service's application-layer nature, are improved by including our technique of Local Broadcast Clustering, which generically extends any overlay topology implemented within our architecture. Since we additionally abstract from a specific network access, developed services can for evaluation purposes easily migrate between various network environments: Overlay topologies and reliability mechanisms may, in a first step, be tested and optimized within event-based network simulation software. In a second step, they may then be used by true applications on real devices, without any changes made to their source code. As the testing and the tuning of application-layer multicast services is thus greatly simplified, we look forward to providing extensive evaluations and functional MANET multicast software in the near future. Our ongoing activities in this area will be made available under [13].

APPENDIX
LOCAL BROADCAST CLUSTERING

When operated in wireless environments, application-layer multicast services suffer from severe scalability issues. Indeed, because application-layer multicast services forward data using ordinary unicast transport links, the wireless medium will be increasingly stressed as soon as many group members are located (physically) close to one another. As the low bandwidth available in wireless environments will have to be shared by many devices, multicast throughput is likely to drop. Additionally, areas of increased group member density will require many devices to synchronize for medium access. Until synchronization is reached, collisions and growing back-offs will occur, resulting in raising latencies.

Local Broadcast Clusters (LBCs, as seen in figure 4) enhance the scalability of application-layer multicast services by making use of the wireless medium's broadcast capability: Group members having joined the overlay broadcast any received multicast data. As data is forwarded to any group members within transmission range, the latter are not required to join the overlay and thus become *locally joined nodes*. Overlay nodes additionally periodically broadcast dedicated heartbeat messages. By doing so, they form their LBC and signal their presence to locally joined nodes. As soon as a locally joined group member moves out of an overlay node's broadcast range, it will cease to receive multicast data as well as heartbeat messages. It is thus required to join the overlay, in order to be included in the latter's data dissemination process. In analogy, an overlay node receiving heartbeat messages may retire from the overlay, since it finds itself within broadcast range of a nearby overlay node and thus receives the latter's broadcasted data. Using this dynamic overlay adjustment, the area covered by the overlay's nodes' broadcast ranges is continuously adapted to a multicast group's circumference.

LBCs successfully reduce the number of medium accesses required for data forwarding. Especially in areas of increased group member density, node synchronization will be simplified, as only very few nodes are involved in data forwarding. As additionally less nodes will join the overlay, overall control flow required for the overlay's maintenance is accordingly decreased [11], [12].

REFERENCES

[1] Sung-Ju Lee, Mario Gerla, and Chin-Chuan Chiang, "On-demand multicast routing protocol," in *Proceedings of IEEE WCNC 1999*, New Orleans, USA, Sept. 1999, pp. 1298–1304.

[2] Elizabeth M. Royer and Charles E. Perkins, "Multicast operation of the ad-hoc on-demand distance vector routing protocol," in *Proceedings of MobiCom '99, Seattle, WA, USA*, Aug. 1999, pp. 207–218.

[3] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS 2000*, Santa Clara, California, USA, June 2000, pp. 1–12.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM 2002*, Pittsburgh, PA, USA, June 2002.

[5] Min Ge, Srikanth Krishnamurthy, and Michalis Faloutsos, "Overlay multicasting for ad hoc networks," in *Third Meditteranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, Bordum, Turkey, Jun 2004.

[6] Ahmed Sobeih, William Yurcik, and Jennifer C. Hou, "Vring: A case for building application-layer multicast rings (rather than trees)," in *12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Volendam, The Netherlands, Oct 2004.

[7] Chao Gui and Prasant Mohapatra, "Efficient overlay multicast for mobile ad hoc networks," in *The Wireless Communications and Networking Conference (WCNC)*, New Orleans, Louisiana, USA, Mar. 2003.

[8] Gavin Holland and Nitin H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proceedings of IEEE/ACM MOBICOM '99*, Seattle, WA, USA, August 1999, pp. 219–230.

[9] Ahmad Al Hanbali, Eitan Altman, and Philippe Nain, "A survey of TCP over Ad Hoc Networks," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 3, pp. 22–36, May 2005.

[10] Martina Zitterbart, Burkhard Stiller, and Ahmed N. Tantawy, "A model for flexible high-performance communication subsystems," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 507–518, 1993.

[11] Peter Baumung, Martina Zitterbart, and Kendy Kutzner, "Improving delivery ratios for application layer multicast in mobile ad-hoc networks," *Elsevier Special Issue on Computer Communications*, vol. 28, no. 14, pp. 1669–1679, 2005.

[12] Peter Baumung, "Stable, congestion-controlled application-layer multicasting in pedestrian ad-hoc networks," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Taormina, Italy, June 2005.

[13] Peter Baumung et al., "The Modular Architecture for Application-Layer Multicast," http://maam.pcb-net.org, 2005.