# JANE – The Java Ad Hoc Network Development Environment

Daniel Görgen
University of Trier
goergen@uni-trier.de

Hannes Frey
University of Southern Denmark
frey@imada.sdu.dk

Christian Hiedels
Université du Luxembourg
christian.hiedels@uni.lu

## Abstract

*This work describes a Java based development platform which is intended to support ad hoc network researchers in application and protocol design. Software development within this environment is expected to follow a bottom up approach. Basic functionality is implemented in elementary components which can be combined to more complex ones by using well defined interfaces. With dynamically changing network links being rather the common case than a failure situation, asynchronous communication has been selected as the main communication paradigm within this platform. Reusability of components in different execution contexts by providing an appropriate machine abstraction is a further important design decision which drove the platform development. Code written once can be executed in a pure simulation mode, in a hybrid setting with real devices being attached to a running simulation and, finally, in a setting using real devices only. Software development following this three-tier development process paired with the platform's rich visualization features emerged to significantly ease the burden of debugging and parameterizing in such highly dynamic and inherently distributed environments. In conjunction with a core middleware platform a rich set of generic services has been implemented with the most important ones being described in this work. Several application programs have already been implemented on top of these services. These applications which are described in this work as well serve as a proof of concept for both the platform itself and the utilized set of generic services.*

## 1 Introduction

Future paradigms as ubiquitous and pervasive computing rely on countless mobile devices interacting among one another using wireless communication. Groups of these devices may form ad hoc networks at any time. These networks can be divided into three general classes. (1) Pure infrastructureless networks which totally rely on other mobile devices using multi-hop routing for distant interactions, (2) single-hop ad hoc infrastructure using wireless communication as the last hop into a more reliable backbone network and (3) hybrid networks mixing these two types. Hybrid networks integrate infrastructure to improve reliability of mobile multi-hop ad hoc networks. Beside "traditional" infrastructure such as WLAN access points or 3G networks, it is also possible to think of mesh networks or even single installed mobile devices to provide infrastructural duties selforganized.

In all the described scenarios the mobile and wireless network parts are characterized by limited energy resources, a broadcast based communication media, and a dynamically changing network topology due to device mobility or energy conserving sleep cycles. Software development in such an environment requires new tools and programming paradigms which address the special needs under these harsh conditions. In addition, implementing and evaluating applications and algorithms in a mobile ad hoc environment is an extensive task when using real mobile devices in field trials. A large amount of devices are needed and of course enough people to handle them. Thus, it is very hard to obtain reproducible scientific results. Using network simulators for testing and evaluating network protocols for multi-hop ad hoc networks is state of the art. Most of these simulators are not intended to implement applications on top of the simulated network. Moreover, testing applications solely in a simulated and thus idealized environment may produce misleading results or interpretations. A promising approach is the three step development process originally proposed in [25]. Applications are implemented, tested and evaluated in a simulated environment first. Second, the application is tested with real user behavior but still in a simulated environment. Finally, dedicated field trials are used as proof of concept.

The `JANE` development environment described in this work can be used to implement and test protocols and applications by following the described three-tier design paradigm. Component based design and asynchronous communication among components are forming the key ingredients of the `JANE` development environment. The approach is motivated by the observation that realizing a "traditional" view of the network stack and stream oriented synchronous communication does not cope well with the extreme dynamics of an ad hoc environment. For instance, a process blocked at a stream in order to obtain data from another mobile device might frequently be tied up with error handling due to stream disconnection. Error handling, however, should be an exception and not the general case. Event based programming perfectly fits in a frequently changing environment since applications and protocols can adapt immediately when the surrounding of a device changes. Blocking in contrast requires a timeout to occur before the blocked entity can be transferred into the next state (e.g. to realize that the stream sender is no more available).

For ad hoc networks it is a well established fact that ISO/OSI protocol layers require cross layer communication which contrasts the original idea of hiding all protocol peculiarities. For instance, in an ad hoc scenario neighborhood information maintained at the MAC layer should be provided to upper layers as well. This avoids unnecessary message exchange in order to build up the same view at a higher layer. The component based design followed by the `JANE` environment takes the idea of cross layering one step further. Communication among components is possible by using any handler method which has been published at the device. Components themselves are not explicitly layered. This follows implicitly from the components functional dependencies.

The rest of this paper is organized as follows: The first section introduces the `JANE` architecture. Section 3 introduces `JANE`'s network abstraction and the implementations, Section 4 introduces a generic routing framework, and Section 5 a generic neighbor discovery service The three `JANE` execution modes, simulation, hybrid, and platform, are described in Section 6, 7 and 8 respectively. Some application examples which have been realized using `JANE` are shortly sketched in Section 9. Related work is given in Section 10 and the paper is concluded with Section 11.

## 2 The JANE Architecture

In `JANE`, everything is a service. This includes application components, basic middleware services, network layers, routing algorithms, and even hardware such as GPS receivers are represented as a `JANE` service. Services are strict components and cannot be accessed directly. Communication with other services is only allowed using an event based communication mechanism provided by an operating system abstraction. Thus, each device runs a couple of services, locally interacting using asynchronous signals and events. Also the ad hoc network is modeled using event-driven services. `JANE` networks provide asynchronous message oriented communication with other devices. Message sending and receiving is mapped to the local event based interaction primitives. This model perfectly fits to the extreme event-driven environment of mobile ad hoc networks.

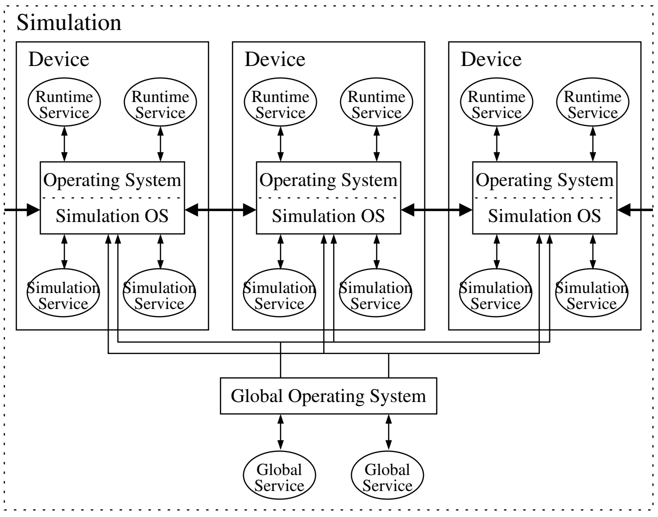


**Figure 1. Service interaction in a `JANE` simulation run.**

### 2.1 JANE Services

A service component consists of at least one Java object which provides methods for event handling. These can be arbitrary methods with no return value. Events can be generated by other services, by the service itself or by an operating system component. By exporting interfaces or additional event handling objects other services are enabled to trigger these methods. At basic, the main service object provides a *start* and a *finish* method triggered by the operating system. A service is able to generate new events using the operating system. For triggering itself, it can start and stop timeouts. For triggering others, it must use the operating systems service interaction component. Each service is uniquely identified, so that it can be addressed by others.

As depicted in Fig. 1, `JANE` distinguishes three service types: *runtime service*, *simulation service* and *global service*. A runtime service can be executed in all `JANE` environments while a simulation service has access to simulation and thus global knowledge. This includes exact device positions of all devices, exact simulation time, simulation control and visualization settings. Moreover, this service type is able to generate events for services on other devices using the same asynchronous service interaction primitives as used for local interaction. Thus, it is possible to implement network services based on generating events on neighboring devices. Simulation services are also used to implement services that avoid computational expensive network load using global knowledge and direct device interaction. This is also needed, when services are evaluated and some necessary services should not influence simulation results. In contrast to the other services which are always assigned to a mobile device, global services are simulation services which are instantiated only once within the

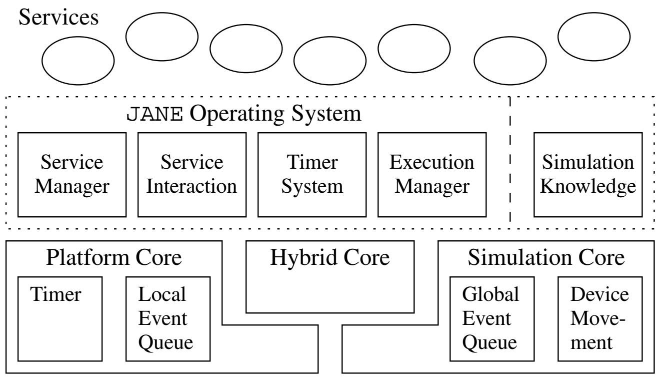


**Figure 2. The Architecture of the JANE operating system.**

simulation. These services are mapped to each device. As simulation services, these are also used to realize functionality based on simulation and global knowledge but using a central approach not a distributed. Other purposes are realizing global visualization for painting background graphics or implementing global statistics. The services, in particular runtime services, do not see any difference between these service types when interacting. For any service type it is always possible to find out the service identification of the service causing the execution of an event handler. Simulation and global services can also retrieve the hosting device of the initiating service. This enables them to reply directly to the device they received an event from.

## 2.2 Operating System Abstraction

The event driven core of a JANE device is hidden in an operating system abstraction (see Fig. 2). In contrast to other simulation environments, it is not possible to access the event queue or other simulation internals directly. Services are implemented on top of this operating system and thus can easily be ported to other environments. The basic duties of the JANE operating system are service management, service interaction, event execution management and timeout management. The service interaction directives are described in section 2.3 in detail.

Service management handles service starting and shutdown at runtime and at device startup. Each service is able to start and shutdown other services. To lookup other services for interacting, a simple service discovery mechanism is provided. This is based on the Java class hierarchy and lists all identifications of services implementing a given class or interface.

Instead of adding events to the simulation event queue directly, a service is able to set and remove timeouts. This enables a service to generate events by itself, e.g. to realize periodic tasks or to timeout communication tasks. Timeouts are set using time deltas relative to the device's local clock.

The execution manager schedules the events on a device. Here, it is possible to reorder contemporary events, e.g. by using service priorities. The current implementation uses a FIFO ordering. The execution manager also ensures that every event is executed atomic and that each service handles only one event at a time.

## 2.3 Service Interaction

JANE provides two types of asynchronous service interaction: (1)Signaling and (2) event firing and catching.

A *service signal* addresses an event handler object using a known receiver identification and calls asynchronously a public method of the receiving object. By default a service can be addressed by signals using its service ID. Additionally, every service is able to register objects for signal receiving. The signal receiving event is executed in the context of the receiving service (also in the case of registered handler objects). To keep things simple, it is possible to export interfaces for the service and its event handler objects. Other services can request signal proxies automatically generated by the operating system using the exported interfaces. Thus, a service can send signals to another service and asynchronously execute a signal handler method by calling exactly the same method on the generated signal proxy. To reply to an asynchronous call, the sender can pass callback objects. The receiver gets a signal proxy as a method argument, which also generates asynchronous signals that are send back again. Thus, also more complex interactions can easily be implemented. An example is a message status handler for network communication where the current status of a message propagation task is signaled asynchronously.

A service can also interact with others by firing a *service event*. These events are not addressed to a specific receiver as a signal. Instead, other services can register event handlers which are called when such an event occurs. To specify the events which are of interest event templates are used. A fired event is matched to all registered templates implementing the same class or one of it's superclasses. Registering a superclass template can be used to receive all derived events. An event object is matched using its attributes. A template attribute with a null value matches all possible values. Initialized attributes must be equal to the corresponding event attribute. In case of a Java *Class* attribute, the *instanceof* relationship is used instead. By default, an event contains the unique identification of the sender and its Java class. Thus, a receiver is able to specify the sender by providing one of the interfaces the sender implements. It is possible to enable template matching also for every event attribute given as Java object. The template matching is restricted by a given reflection depth, which is one (only service event attributes are matched) by default. Using a higher reflection depth causes additional computational load. But a fired event is only reflected to the maximum depth of all registered templates of the same event type. The registered event handler object can implement a default handler method which is able to handle every event type. Or it can implement an event handler interface specified by the event as it is known from signals. Thus, the event can execute the correct method and pass all necessary event attributes. The event handler does not need to cast the event and it is possible to implement event handler methods for different events within one object.

## 3 JANE Networks

In JANE, also a network is implemented as a set of services and event handling objects. The main purpose of JANE is to simulate mobile ad hoc networks but it is also possible to simulate hybrid settings by starting multiple network services. In a hybrid setting, a wireless ad hoc network is combined with a wireless infrastructure network. Also multiple wireless ad hoc networks are possible and devices can be connected with fixed wired network links. Due to the open service architecture it is possible and also simple to realize additional networks beside the ones presented in this paper.

A JANE network is represented at the link layer. Basically, it provides asynchronous message oriented unicast and broadcast communication which addresses direct neighbors in a wireless ad hoc setting. Other services can communicate over the network by signaling a message communication task to one of the running network services. To retrieve message status events, the client is able to provide a callback handler which can be signaled by the network service. At minimum, a network signals success, failure and timeout for reliable communication and at least it signals when a message has been completely processed locally, e.g. completely put on the media.

Networks can also provide an extended network interface which is mainly intended for wireless networks. It provides additional communication paradigms addressing the broadcast property of the wireless media. In most wireless networks, unicast communication is much more reliable than broadcasts since the link layer uses acknowledgments to increase reliability. Thus, there is a tradeoff between using the media's broadcast property and message reliability. To cope with this problem, the extended link layer provides reliable communication paradigms which also use the broadcast property: (1) An *addressed multicast* sends a message to a known receiver set in the direct neighborhood and waits for acknowledge messages from all receivers. In the best case, the message is transmitted only once. (2) An *addressed broadcast* is an addressed multicast and addresses also a known receiver set but the message is also received and processed by all other devices within the senders broadcast region. When a subset of the neighboring devices is known, a broadcast to all neighbors is therefore much more reliable. A special case addresses only one device. This is a unicast while all receiving neighbors are implicitly set into promiscuous mode, e.g. using an additional header flag.

JANE networks provide configuration at message level. Other services are able to change the network's behavior for each message task. Usage scenarios are the reduction of signal strength while broadcasting and the adaptation of timeouts and retries when using reliable communication paradigms. Thus, it is possible to adapt the network very dynamically and not only globally for all services.

For simulating network communication, a message must specify its size. This reduces computational load and also enables the developer to use simple message implementations which are not necessarily optimized for network communication. The developer can also piggyback additional e.g. statistical data which should not increase the message size. Additionally, messages are able to define arbitrary shapes for network visualization. Received messages are delivered as events, so that a service must explicitly register itself or an event handling object for message reception. The event receiver has also access to an enriched message header which also provides signal strength respectively distance information of the message sender.

For simulating wireless ad hoc networks JANE currently provides three network implementations with increasing simulation detail: a collision free and a shared network model and an implementation of the IEEE802.11 MAC protocol. For hybrid settings it provides a simple wireless single-hop infrastructure and a simple wired network.

### 3.1 Linkcalculator

The JANE simulation core provides a link precalculation where network implementations can be based on. Depending on a maximum communication range, this component generates events when a device comes into communication range of another device. These events are calculated from mobility data of the mobility sources. Since devices are moved on straight lines, exact link establishment events can be calculated easily. Link information and link events are provided to simulation services. Network services can thus depend their decisions on a reduced device set or use these links directly without any adaptation. Precalculation improves the simulation performance in large network scenarios where the average links per device is lower than the total amount of devices or when communication link changes are less frequent than communication events on each device in average.

### 3.2 Collisionfree Network

The simplest network implementation models the wireless ad hoc network as spontaneous but reliable unidirectional links between neighboring devices with arbitrary but circular communication ranges. Thus, a message is only lost when a link breaks down due to device mobility. Moreover, this network does not simulate a shared media. Unicasts are also delivered over unidirectional links but are only signaled as successful, if the receiver has a link to the sender for “virtual" acknowledgement. The network is realized as global service and is based on the exact linkcalculator links. Devices that like to take part within the network must start a local service that registers themselves within the global service. To simulate packet losses when devices are mutually within their communication ranges, the network additionally asks an exchangeable component which can base the packet loss decision on the positions and communication ranges of the two devices, as well as on arbitrary random distributions.

This network is mainly intended for simulations where device mobility patterns or spontaneous network structures are the main focus. Moreover it provides fast simulation runs since the overhead for network simulation is very low. For instance, a usage scenario for this type of network is measuring the hop count of a message needed from sender to receiver using a geographic routing protocol [17]. No detailed network simulation is needed in this case.

### 3.3 Shared Network

This network models the wireless ad hoc network as a shared medium with arbitrary circular communication ranges. No radio propagation model is used. Message reception only depends on the sender’s communication range. To avoid collisions, an idealized carrier sense and RTS/CTS mechanism is used. The coordination of the concurrent media access is achieved using global knowledge, which avoids concurrent access using a fair scheduling. The idealization leads to reliable unicast communication in fully connected and stationary networks. Broadcast messages can be lost when the hidden terminal problem occurs, i.e. no carrier sense is possible. Unicasts can be lost due to mobility which causes link breaks or when a receiving device comes into communication range of another currently sending device. As the collision free network, this network is also realized in a global service on top of the linkcalculator. It can also use a randomized message reject component to simulate message losses.

Due to the simple coordination function and thus low simulation overhead, this network is also used to achieve fast simulation runs. This implementation is used when the behavior of a shared network is needed, but the main focus is still on device mobility and network structure. Regions with a higher device density are simulated more realistically in particular when addressed broadcast is compared with unicast communication. An example scenario is the *en-passant communication* (see section 9.1) where devices efficiently exchange data objects while passing each other. The device mobility is the main focus but also the usage of the media’s broadcast property is important.

### 3.4 802.11 MAC

JANE also contains an implementation of the IEEE802.11 standard for ad hoc networks. The implementation is divided into three services: the medium, the physical layer and the MAC layer. A global service simulates the radio propagation model of the shared medium, e.g. a two-ray-ground model. It uses discrete slots for calculating the receive respective the interference signal strength observed on other devices which is signaled to the device’s local physical layer services. Due to all observed signal strength, the physical layer decides whether a signal is received or not. Like the ns2 [14] physical layer implementation, it also regards capture effects, so that a signal is received, when the signal to noise ratio is greater than a specific threshold. The MAC layer implements only the ad hoc part of the IEEE802.11 standard. The sending of MAC frames is mapped to turning the radio on for a time delta calculated from the frame length and the used data rate. As mentioned above, messages in JANE always provide their size for network simulation.

Additionally to the standard, the implementation provides also the extended link layer directives. This is achieved by an implicit promiscuous mode. The sender sets an additional flag within the MAC header, so that a unicast is received by all devices in communication range. In a piggybacked header, all other receivers are listed and a flag indicating addressed broadcasts or addressed multicasts. The additional receivers acknowledge the received message with a link layer unicast. The possibility of extending the 802.11 MAC protocol for supporting addressed multicasts and broadcasts directly has not yet been investigated. This network can also be configured on a per message basis. Currently, signal strength variation, frame retry counters and timeout deltas can be adapted for each message that is communicated. Simulating the network with such a high detail causes very expensive simulation runs. But for investigating routing protocols or also regarding link layer effects, such a detailed simulation is necessary.

### 3.5 Wireless Infrastructure

For allowing simulations of hybrid network scenarios, a wireless single-hop infrastructure network is contained in JANE as well. It consists of two runtime services and two global services. To declare a device as a part of the infrastructure, a base station runtime service is started. They are positioned at fixed places and are connected to each other, independent of their distances. A governing global service, the base station network service, manages these devices and forwards messages between them.

Other devices, running a client runtime service, can move freely around and are connected to a base station, when they are located in sending range (see Fig. 3). Another global service, the client network, organizes the connections between clients and base stations and manages the

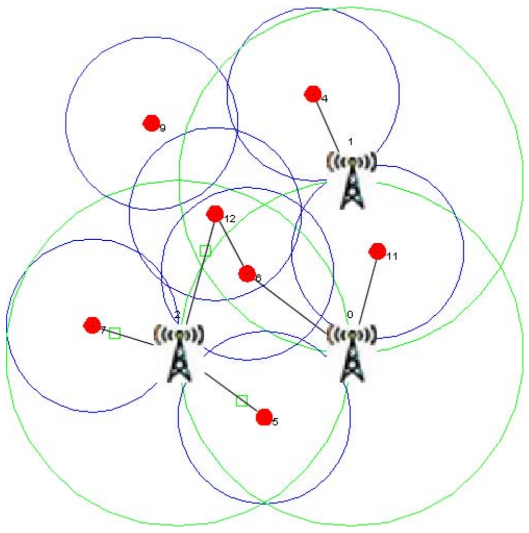


**Figure 3. A hybrid scenario showing three base stations and seven mobile devices. Circles represent sending ranges and lines represent connections.**

links provided by JANE's linkcalculator in a network graph. Furthermore it allows queuing of messages, which can be used to simulate erroneous behavior like lost or delayed messages and congestion. Packet collision and changing of transmission ranges are laid aside purposely, thus every device has an unchanging sending capacity. In other words, the medium is not shared. JANE's ability to combine different network protocols in the same simulation allows adding of other protocols as required. When the focus lies on the mobility of devices, this abstraction is sufficient.

To transmit a message, a device sends a unicast to the base station, to which it is currently linked. The message is forwarded internally in the base station network service to the base station to which the receiver is linked to and obtained finally via unicast. If the receiver is currently not available, types of messages can be specified, which are stored on the base station until the receiver finally arrives. Broadcasted messages reach all other devices linked to the same base station, but are not forwarded by the base station, thus a cell broadcast. If a device moves, it will leave the sending range of its base station eventually, but will be logged into another one immediately, if there is one available. This handover between base stations happens without effect on message transfers. This network can be used as a simple implementation of a UMTS network.

### 3.6 Fixed Infrastructure

Based on the collision free network JANE provides also a wired network model. Fixed links between devices can be added in advance to the linkcalculator and the links are simulated as permanent links within the collision free network. Thus, messages can be send completely reliable over these links. But as the wireless implementation, this implementation can only address direct neighbors at the link layer and a routing protocol is needed for addressing an arbitrary device within this network.

It is still possible that devices with fixed links are mobile but the normal setting is to use fixed positions for these devices. To include a device into the network it has to register itself at this global service.

## 4 Generic Routing Framework

Reliable end-to-end communication between a source and one or a set of destination devices forms a basic communication primitive which is prevalent in many ad hoc network applications. The choice of the appropriate routing protocol to enable message transport from source to destination depends on the network characteristics. A non dynamic network might be addressed by proactive routing protocols building up a routing infrastructure which, once installed, may change very infrequently and thus produce a minimum amount of control overhead. However, as dynamism increases proactive routing should be exchanged by reactive routing which is setting up a routing path only if communication between two end devices is required [9]. Finally, when location context is available due to GPS, for instance, the application of highly scalable geographic routing protocols [16] might be a good choice. Summarizing, as ad hoc network properties like dynamism or location context may vary both spatially and temporally, an ad hoc networking environment should provide a tool box of different routing protocols which might transparently or even explicitly be selected in order to fit the needs of the current ad hoc networking scenario.

The JANE environment follows a routing approach which enables the implementation of new and the combination of existing routing algorithms in a uniform way. This is achieved by providing a generic routing service which implements the common routing functionality, including message queuing and interfacing with a one hop communication service (e.g. the link layer service). The characteristic of the desired routing protocol will be determined by routing algorithms being separate services running on top of the generic routing service. These algorithms will decide the next forwarding device (or forwarding devices), and will tell the routing service by using the appropriate signal, where to forward the message next.

In order to start message forwarding any service using the generic routing service has to obtain a routing header from the desired routing algorithm. Routing header and payload are provided to the routing service which is responsible to invoke the routing start method of the routing algorithm matching the header. For this purpose, a routing header has to store the service ID of the routing algorithm which is responsible to handle it. This mechanism is used for message reception during routing as well. Whenever a message arrives at the device it is first passed to the generic routing service, which in turn stores the payload in a message queue and passes the routing header to the appropriate routing algorithm by using the routing algorithm ID stored in the routing header.

The routing service provides the routing algorithm with a set of commands which can be used by a routing algorithm in order to determine the next routing action. The routing service commands drop, ignore, and deliver can be used in order to terminate routing of the currently handled message. At this, drop and ignore will simply remove header and payload from the routing service's message queue. Deliver does the same, but causes the routing service as well to deliver the message to its intended destination service, which has to reside on the current device. Distinguishing between drop and ignore might be necessary for protocols which are allowed to receive a message but need not to take any further forwarding action. For instance, a device using a routing protocol flooding a message towards a final destination might receive a message more than once and has, thus, to ignore all message copies received more than once. In contrast, drop will be issued by a routing protocol whenever the message can't be forwarded further on due to a routing failure.

When deciding to forward a message to the next hop the routing algorithm might use three possible primitives. Unicast can be used to send the message to exactly one reachable neighbor device, multicast will address a subset of all, and finally broadcast can be used to send the message to all one hop neighbors. At this, the routing algorithm gets informed by the generic routing service when the message was completely passed onto the wireless channel. In addition, for reliable forwarding the generic routing service might detect a forwarding error when a receiver is no more available. In this case the routing algorithm is informed by issuing its forwarding error handler routine. Finally, if the underlying one hop communication service supports promiscuous mode, a routing algorithm will be informed by all received routing messages sent by any neighboring device.

As an additional feature the generic routing service provides a multimodal protocol design as well. For instance, combining protocols is reasonable for localized geographic routing which often employs a combination of a greedy routing and a planar graph recovery strategy [16]. Recovery is invoked whenever greedy routing fails. The generic routing service implements this principle in a general way by providing a delegate command. In case of a routing failure a routing protocol can use this command in order to instruct another existent routing algorithm to continue handling the message. However, selecting the right protocol is not part of the generic routing service but has to be decided by the routing algorithm itself. This is accomplished by providing the header of the desired algorithm to the routing service. The generic routing service in turn only serves as the mediator between the delegation source and destination algorithm. Note that the delegation mechanism is not only useful to recover from routing failures, but might also be used to switch between protocols explicitly. For instance, in a Geocast scenario (i.e. all devices within a certain area are the message's receivers) a message might first be sent towards the destination area by using a single path geographic or even topology based protocol. On arrival at the target border the routing service might be instructed to switch to a restricted flooding of the devices located within the target area.

## 5 Neighbor Discovery

The majority of ad hoc routing protocols, for instance, require a permanent view on the one hop neighbor devices which are available at the moment. This is accomplished by periodically exchanging short beacon messages which keep the current entries in a neighbor list alive. When a beacon message from a specific neighbor is no longer received, the neighbor entry is removed from the neighbor list again. Some routing protocols require more information than the current availability of neighbors. For instance, devices running geographic routing protocols need to exchange information about the current physical location among each other. Information about neighbor devices is not only of interest from a routing point of view but may be required in certain application scenarios as well. For instance, an application might automatically exchange an electronic business card with other users which are immediately reachable in the one hop neighborhood and which matches a certain user profile.

In order to save energy resources and communication bandwidth it is reasonable to avoid each protocol and application to implement its own beaconing mechanism. In addition, factorizing out a common functionality which is implemented in one generic service is likewise a good software design. The JANE platform provides two main abstractions in order to provide a permanent view on all neighbor devices in vicinity. Determining the beacon intervals and interfacing with the network in order to send beacons to all immediate one hop neighbor devices is accomplished by the beaconing service. The service keeps a simple data structure, which stores all neighbor devices it reached a beacon message from, and removes neighbors after a certain timeout interval. At this, beaconing period and the timeout interval depend on the implementation of this service. For instance, these times might be fixed, randomized, or depending on the current network dynamic.

Any service implementing the beaconing service interface has to provide a method which enables other services to append data to each beacon message. With this method several protocols can use a running beaconing service in order to exchange their data among the neighboring devices.

Since the beaconing service just provides information about current available one hop neighbor devices, providing additional information like location information and storing locally available neighbors, which are reachable in more than one hop has to be accomplished by additional services running on top of the beaconing service. Implementations providing this service can be found in the JANE neighbor discovery package. The common interface of a neighbor discovery service provides methods in order to request addresses and data items known about neighbor devices in vicinity. At this, data can be requested by providing the neighbor discovery service a data filter. The method will return only these stored data items which are matching the provided filter.

In order to keep the exchanged messages small, a neighbor discovery service is not responsible to provide a complete local view on the network graph. Instead, a service may only request the set of neighbors which are reachable in a certain amount of hops. For one hop neighbors, however, a neighbor discovery service provides the devices which can be reached using this device as a gateway. In a reverse way, for a given neighbor device the set of gateway devices can be requested as well. More precisely, for a destination device which is an $i$-hop neighbor, this method returns those 1-hop neighbors which can reach the destination in $i - 1$ hops.

The current JANE neighbor discovery package has two neighbor discovery implementations. One provides information about all neighbors which can reach each other, and one which provides both information about the one and the two hop neighbors. A more sophisticated implementation keeping information about a parameterized number of $n$ hop neighbors is not implemented so far, but is in principle covered by the neighbor discovery interface as well.

## 6 Simulation Core

JANE's primary duty is to simulate a mobile ad hoc environment. As most ad hoc network simulators, also JANE's simulation core contains a discrete simulation kernel with a central event queue. On top of this kernel all other simulation components have been build. For each simulated device, the simulation internals are hidden by a simulated operating system. It maps the operating system components to the simulation specifics. A device manager is responsible for the device mobility provided by a mobility source which will be discussed in section 6.2. It might additionally own a linkcalculator component as described in section 3.1 in order to speed up the simulation of wireless networks. Moreover, the device manager is responsible for the visualization of a device into a decoupled visualization component.

### 6.1 Simulation Operating System

Each simulated device has its own operating system component. Thus, it is possible to realize different operating system behaviors on each device. One aspect is the device's local clock. It is possible to use the time given by the event queue, so that all devices have one global time base. Another implementation uses randomized drifting clocks, so that also timeouts with the same delta can have different simulated durations. The execution manager uses the event queues zero execution time per event by default, but it is possible to simulate a fixed or a randomized execution time for each service event on this device. The timeout manager uses the local clock for timeout delta calculation and operates directly on the event queue. Timeout handlers are executed as all other service event handlers by the device's execution manager.

The simulation operating system must support all three service types. Thus it must also provide access to simulation knowledge. For event generation on other devices it has direct access to all other devices and thus, for example, can send signals to services on another device. These events are handled by the execution manager on the receiving device.

### 6.2 Device Mobility

Since JANE is intended for a mobile environment, the simulation core must realize device mobility. At startup, it is possible to add *mobility sources* to the simulation. A mobility source provides enter and move events for a set of devices. A move event specifies the position of a device at a given time. The device manager moves the device on a straight line with a constant velocity from one position to another within the time delta given by two consecutive move events. Thus, arbitrary device movement can be approximated using a sequence of lines.

Beside simple stationary scenarios where device placement follows given rules and simple random waypoint and random walk variants [10] also more complex mobility sources have been implemented. The *pathnet* mobility source uses a graph given as XML file for device movement. Vertices are crossings or endpoints and contain routing probabilities to endpoints for outgoing edges. Endpoints (e.g. rooms) can be arbitrary randomized position generators and can be entered over at least one graph edge. Edges can have an arbitrary width so that devices are moved on a lane and not only on a strict line. Devices are moved on the pathnet by providing the next endpoint and the moving speed. The device's route within the pathnet is chosen at random due to the given routing probabilities. The endpoint provides the final position. Pathnet move events can be generated at random or by a given timetable. The timetable realization groups the device in classes and schedules events for a set of classes at a set of pathnet endpoints. The device enters and leaves these events randomized and also one of the possible endpoints are chosen at random. By combining different mobility sources, complex mobile settings can be generated easily.

Some mobility models also provide an interactive setting. There, the devices can also be moved by user interaction with the simulation GUI at runtime. The device movement is restricted by the underlying mobility models. Using the pathnet model, the user can only move devices between pathnet endpoints. This setting is used for interactively “playing” with the mobility scenario during testing or debugging and also for application demonstration.

At startup, devices can be grouped together. Each device group is assigned to a mobility source, so that it is possible to model different device behavior. Each group can be started with a different set of services. This feature is used to set up simulation with heterogeneous functionality and behavior very simple. Device can be stuck to fixed positions providing selforganized ad hoc infrastructure features while the rest randomly moves on the plane. Other scenarios are used to analyze applications only on a small subset of devices while the rest only act as transparent multi-hop communication routers. Different mobility patterns can also be used in combination with different simulated user behavior patterns implemented as services on a device. Besides grouping, it is of course possible to assign each device its own service set.

### 6.3 Visualization

For testing, debugging and demonstration JANE provides extensive visualization possibilities. Every service is able to visualize its state by providing arbitrary shapes like lines, rectangles, ellipses or collection of shapes. Shapes can be defined using simulation positions or by using device addresses as points. Thus, the network for example is able to simply paint network links or a message progress of a transmission between two devices. The shape rendering is decoupled from the simulation so that it does not influence the simulation behavior. The rendering unit can use a GUI canvas (e.g. Java2D and OpenGL [28]) for drawing and, if desired, the unit can also render to file e.g. as Postscript, PNG or XML.

Using the visualization is computational expensive and slows the simulation down. But it gives sometimes better insights as it can be achieved by evaluating trace files. For long running statistic evaluation, the visualization can completely be deactivated. This causes no additional computation load if the service shape generation is implemented appropriately by providing the shapes only on demand.

The visualization GUI can be extended to pass user interaction to a running simulation. This causes, of course, non determinism and thus non reproducible simulation runs. But for testing and also for presentation this is a very helpful feature. Such user interaction can use the same operating service abstractions like a global service. A user interaction is therefore able to generate arbitrary events for each running service on each simulated device. In combination with a mobility source which also supports user interaction, it is also possible to influence the device mobility using the visualization GUI.

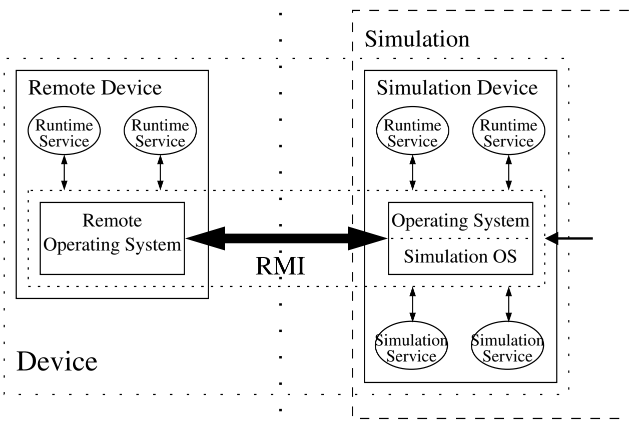


**Figure 4. JANE merges services on extern device with the services on a simulated device on a virtual device with a virtual OS. This is achieved using Java RMI.**

## 7 Hybrid Setting

In particular for applications in mobile ad hoc networks, it is necessary to evaluate and test the application components not only with simulated, but also with real user interaction. But only a simulation enables the application developer to test applications using a very large amount of (simulated) devices. Thus, it is helpful if real devices can be connected to the simulation and users are able to interact with others using a real device on a simulated network. This hybrid setting makes it also possible to integrate simulated-only devices, e.g. as transparent message routers in a multi-hop scenario or to represent passive users. Moreover, also simulated user behavior can be used to create large scenarios for tests with real user interaction.

Real devices are connected to an existing device within the simulation. The client is able to choose the device or is assigned to an arbitrary unconnected one. The extern device can start arbitrary runtime services. Services on the simulated and on the connected real devices are merged transparently as if they were on the same platform (see figure 4). A service does not see any difference when interacting with another service. It is possible to start only the non runtime service within the simulation and the rest on the real device or letting everything within the simulation and just start an extern application GUI instead of a service simulating user behavior. This transparency can easily be achieved due to the event based interaction between services. Synchronous operating system calls causes the simulation to stop between two event executions until it is completely processed, so that an event stays atomic and no inconsistencies can occur. Services and registered event handlers are stored within the extern operating system and the simulated operating system, respectively. Only event handler call descriptions are transmitted. These calls are appended to the simulation event queue or to a thread driven event queue on the extern device to decouple the operation of simulation and extern devices. This causes, of course, non reproducible simulation behavior. However, the hybrid scenario is used for real user behavior which is indeed non deterministic.

The connection is achieved using Java RMI. This enables the connection of extern devices also over a network. Using WLAN for connection allows to integrate also small wireless devices as PDAs which gives the user a good feeling of the real live behavior and operation of the application.

## 8 Platform Core

All implemented runtime services can be tested also in an execution environment using real devices and a wireless network interface. This is possible without any modification of the services. Thus, the application code can be completely implemented and tested within a simulated environment and can afterwards be used on a real device without any additional effort.

The execution manager of the platform uses a simple thread driven FIFO queue for contemporary events. Timed events of the timeout manager are driven by the standard Java timer system and are queued to the execution manager. A multithreaded core is also possible but was not needed yet since only single prototype applications have been tested so far.

Since the platform does not provide network communication within the operating system, the network communication is implemented within a runtime service which maps the communication events to network communication and vice versa. The implementation uses UDP unicasts and broadcasts and maps the extended link layer features to these primitives. For demonstration purposes, the network connectivity can be reduced by simply discarding messages due to virtual device positions and sending ranges.

Messages are serialized using the standart Java object serialization. The network implementation also allows to hook in specialized serializer for a more efficient bit packing. Some of the standard services in particular the beaconing service provides more efficient message packing.

For applications and algorithms using device positions a service has been implemented which provides the current device position from GPS or by user interaction. Additionally, a GUI service can visualize the local running services and if beaconing is used, also the current neighborhood (see fig. 5).

## 9 JANE in Action

We have implemented `JANE` to investigate applications in mobile multi-hop ad hoc networks in order to derive new

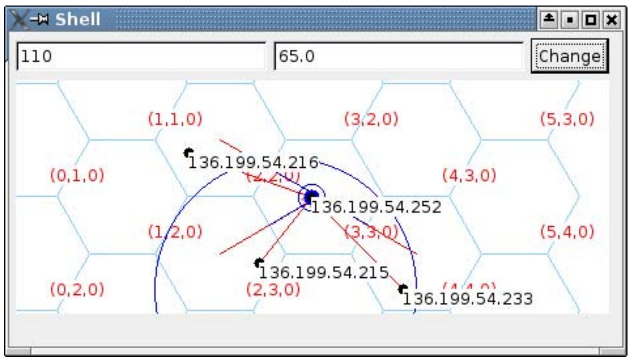


**Figure 5. Visualization of geographical cluster based routing and neighbor discovery seen from the perspective of node 136.199.54.252.**

selforganizing and new communication patterns in this type of network. Also several applications were developed using hybrid networks, combinations of ad hoc with wireless infrastructured networks. In the past years a couple of applications have been realized. The experiences and also the application needs led to an iteration and extension process that resulted in the current version of `JANE`. Exemplarily, some of these `JANE` applications and some work-in-progress applications are sketched in the following sections.

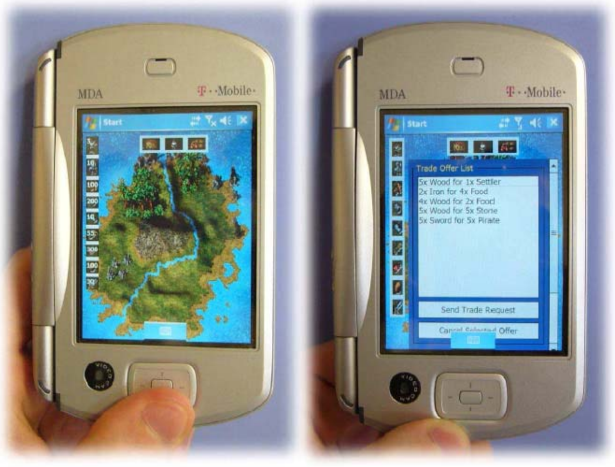
### 9.1 En-passant Communication

The en-passant communication pattern [19] is used to efficiently synchronize data objects when devices passes one another. It is possible that not only two devices are involved in this process, e.g. when two groups pass one another. The data exchange protocol handles this by using the extended link layer features of `JANE`. Data objects and protocol messages are propagated as addressed broadcast. Thus, all neighboring devices receive and store data objects. When an object is not needed locally it is stored and provided altruistically to others. If possible, it won't be transmitted any more when it is needed locally.

The en-passant communication has also been evaluated in field trials using `JANE`'s platform mode (fig.8). Field trials are very extensive and thus only smaller scenarios with up to six devices were realized. Although the addressed broadcast has been realized using only UDP unicasts and broadcasts, the advantage of this communication paradigm has been observed.

### 9.2 UbiSettlers

UbiSettlers [22] is a real-time strategy multi-player game, running in an ad hoc network in combination with an infrastructured network. It is roughly inspired by a popular German board game called "The Settlers. Every player controls an island and tries to establish an infrastructure, con-



5x Wood for 1x Settler
2x Iron for 4x Food
4x Wood for 2x Food
5x Wood for 5x Stone
5x Sword for 5x Pirate

**Figure 6. Screenshot of the UbiSettlers game.**

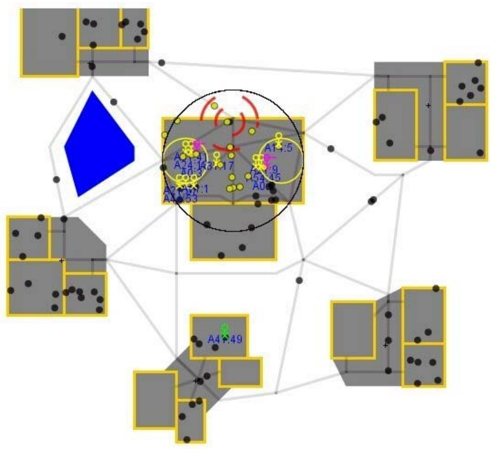


**Figure 7. The UbiBay application on top of a pathnet mobility model. The auction marketplace is located in the middle.**

sisting of buildings, which are constructed using resources like Stone, Grain, Iron Ore, Wood and other. The buildings again provide bonus on the gathering of resources, thus enabling players to get more powerful structures. UbiSettlers is working just fine in single-play, but teamwork with other players is better and in fact emphasized by things like allowing only collaborating players to construct high-level buildings like a cathedral. Another way to get additional resources is trading. Each player is able to send out trade offers stating a proposal to give some own, not needed resources in return for other resources. Other players receiving such an offer can respond and establish trading with a simple handshake model. Devices not running UbiSettlers, forward messages altruistically, by using an implementation of the Lightweight Mobile Routing protocol (LMR). The graphical user interface of UbiSettlers is designed for using on PDAs in an intuitive way. Every action in the game is visualized, enabling players to react immediately.

### 9.3 UbiBay

The UbiBay application [18] realizes auctions solely on top of a mobile multi-hop ad hoc network. Auctions take place at so-called marketplaces, geographic locations with a higher device density. Devices does not need to be at a marketplace to start or to join an auction. It is possible to send agents to the location using position based routing strategies which are hosted by the devices currently located there. These agents act on behalf of the users and return to the users device when the auction is finished. Figure 7 depicts a screenshot of the ubibay application simulated using a pathnet mobility model.

### 9.4 NetNibbles

Basically, NetNibbles is a multi-player variant of the well-known game Nibbles (also known as Snake) designed for ad hoc environments. Players control a never stopping Snake and collect edible objects which lengthen the body of the Snake permanently. At the same time, they try to avoid obstacles, such as walls and the bodies of other Snakes from other players. Multiple players thereby establish a logical group. Like in most games the players can achieve some score and they usually want to propagate their high score. For that purpose, a top ten high score list is introduced and maintained by a game server residing in a special device. Every time a player achieves a new high score, which is determined based upon the top ten list stored locally on each device, that high score list is updated and propagated within the logical group using ad hoc communication. After terminating a full game session, the central game server will be updated only once. Due to the fact that different players - maybe in different groups - can achieve new high scores at the same time, the high score list might be updated at different places concurrently. The central server finally needs to conciliate the potentially conflicting different high score lists, integrating them appropriately.

### 9.5 HyMN

HyMN (Hybrid Multimedia Network system [5]) is designed for users interested in live multimedia news from certain events like e.g. Football Championships, Olympic Games etc. For instance mobile devices from football fans create an interest group in a local ad hoc network partition. Multitudes of such groups co-exist e.g. football fans in pubs, those watching another match, traveling ones, and more. In each of these cases, a considerable number of devices have shared interests and might join forces in a local setting. The devices in the ad hoc network running HyMN organize themselves in clusters, where the clusterheads maintain uplinks to a backbone network in order to receive multimedia news related to the interests of the ad hoc members. Thus, the football fans will receive information such as small videos, pictures or text messages each time something interesting is happening during the match. The received multimedia files remain stored on the mobile devices and will be provided to devices joining the group at a later time. For optimization, the HyMN backbone splits the files into chunks, which are sent concurrently to different clusterheads within a single ad hoc network partition. Until everyone received the complete file, the chunks are exchanged via Wi-Fi among interested devices.

### 9.6 Distributed Script

Distributed Script [20] is intended to enable students to create a script during a university lecture in a distributed manner. The lecture itself is modeled as a geographic context given by locations and times. Inside a lecture context, devices are able to communicate over a few hops using a geographic bounded version of DSR (Dynamic Source Routing [24]) and geographic bounded broadcasting strategies. Due to the well known geographic locations, devices located there can be addressed using position-based routing strategies as GCR(Geographic Cluster-based Routing [17]). Thus, it is also possible for absent students to create consistent parts of a script by communicating with the lecture context. Newly created material is initially propagated using SPBM (Scalable Position-Based Multicast [29]) to address as much interested students as possible. Missing lecture material is exchanged between neighboring devices directly, also outside a lecture context when devices passes by (so-called En-Passant communication [19]).

As all `JANE` applications which have been developed including a graphically user interface also this application can be used in all three `JANE` modes. A hybrid mode example of `JANE` is depicted in figure 8.

## 10 Related Work

This section gives an overview over other popular simulating environments. While comparing them with `JANE`, especially the power to build arbitrary working applications using the vast set of available services emphasizes `JANE`'s ability to serve as a middleware.

GloMoSim [8] is a network simulator which focuses on scalability. It uses the capabilities of the parallel discrete-event simulation language Parsec [7]. Implemented protocols are build to use a layered approach and standard APIs are used between different layers, which enables users to integrate new models easily. Simulation scenarios are described via text files. It has various applications, transport and routing protocols, as well as miscellaneous mobility schemes. Because of the parallel approach, GloMoSim allows to run simulations with thousands of devices, which can be visualized either during runtime or later.

QualNet [4] was developed by Scalable Network Technologies based on GloMoSim. It has a lot more network models and protocols, as well as more tools to ease creating simulation scenarios and is sold as a pure sequential and a parallel version.

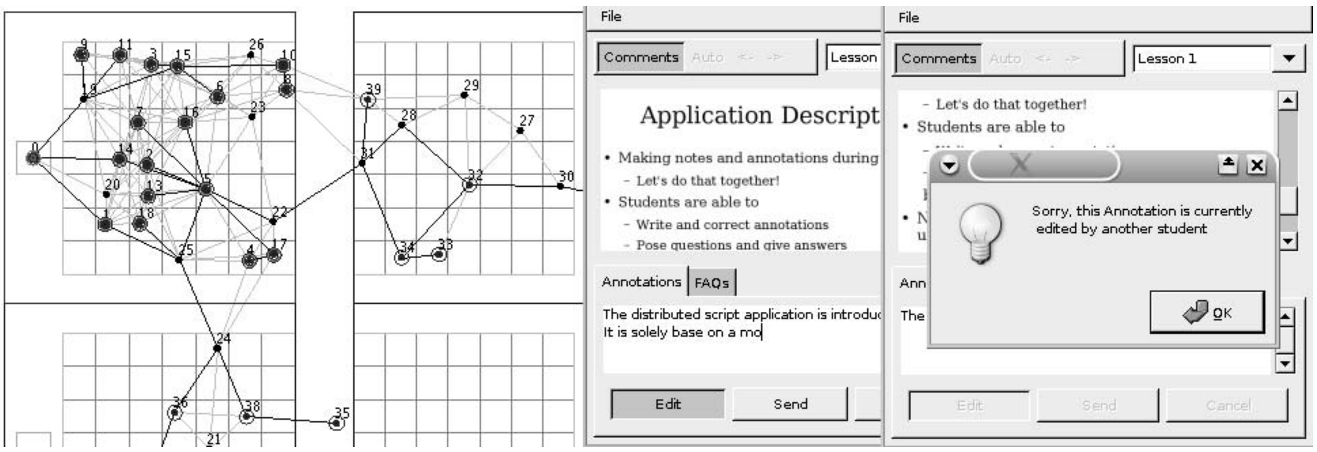
Ns/2 [14] is a discrete event based network simulator written in C++ and probably the best known and widely accepted simulator for computer networks. The first version ns/1 was already developed back in 1995, and meanwhile, it is the best supported simulator available. In the beginning, it did only support stationary networks, but was enhanced with mobile ad hoc network abilities in 1998 by the Monarch Project of the Carnegie Mellon University [23]. Now a huge number of different network protocols, as well as all popular routing algorithms are disposable, e.g. a complete implementation of IEEE 802.11 [1], as well as different projects to integrate UMTS networks [6, 15]. Simulations are defined through scripts written in OTcl [3]. Ns/2 performs the simulation and stores results in a trace file, which can be analyzed and visualized with external tools, like the Network Animator "nam" [13]. New protocols are directly integrated in the source code of ns/2 and are available in simulation scripts after compiling. However, it is generally known, that it takes a long time for getting used to ns/2 [11]. Also it is worth mentioning, that ns/2 scales quite bad as soon as more than a few hundred devices are analyzed and its memory requirements are huge.

With ANSim (Ad Hoc Network Simulation) [21], a network simulator was developed at the University of Bruchsal, that purposely skips a detailed simulation of the MAC-Layer and transmitting protocols and concentrates on mobility of devices. The gained amount of processing power is used to simulate large scenarios as quick as possible. Results are visualized directly during running time and scenario parameters can be specified in a GUI, which can be used as a scenario generator for ns/2 and GloMoSim, too.

OPNET Modeler [2] is a commercial simulation environment that allows detailed simulations of vast networks. It provides hundreds of vendor specific and generic device models. Mobile devices can be placed and moved anywhere in a 3-dimensional area. Its modeling paradigm allows defining the behavior of individual objects at a "Process Level", which are interconnected to form devices at a "Node Level". Devices finally are linked at a "Network Level". Simulations can be parallelized to use multiple processors.

A parallel network simulator for simulating very large stationary multi-protocol networks is Dartmouth Scalable Simulation Framework (DaSSF) [26], which is a C++ implementation of the Scalable Simulation Framework (SSF) API [12]. Parallel simulations with tens of thousands devices are possible using shared and distributed memory configurations on a variety of different architectures.

The Staged Network Simulator (SNS) [30] uses a performance technique, to improve the simulation scale. It is based upon ns/2, but is able to simulate around 50 times



Application Descript
- Making notes and annotations during
  - Let's do that together!
- Students are able to
  - Write and correct annotations
  - Pose questions and give answers

The distributed script application is introduc
It is solely base on a mo

Sorry, this Annotation is currently edited by another student

**Figure 8. Screenshot of JANE in hybrid mode running the Distributed Script application. The two Application GUIs on the right are connected to devices visualized within JANE's simulation GUI on the left.**

faster on certain scenarios. Thus, it can be used to perform simulations of very large networks of up to 10000 devices. The concept is to eliminate redundant computations through function caching and reuse. SNS supports all protocols that are implemented in ns/2.

The Georgia Tech Network Simulator (GTNetS) [27] is a distributed simulator. GTNetS does not use distributed shared memory as DaSSF. Instead, it allows the creation of *remote links* between two devices in different simulators. Connectivity of such a link is determined after a packet has been transfered to the receiver's simulator. GTNetS has initially been developed as teaching tool since the common simulators lacks in simple extensibility and ease of use.

## 11 Conclusion and Future Work

We have presented JANE, an application development, evaluation and testing environment for mobile ad hoc networks. It was pointed out that JANE is more than just a simulation environment but provides the user a middleware platform for applications in an event-triggered ad hoc networking environment.

JANE is not only intended for pure multi-hop ad hoc networks. It is possible to use different network types at the same time to realize applications for hybrid networks combining pure mobile ad hoc networks with wireless infrastructure, like is shown in the UbiSettlers, NetNibbles and HyMN applications.

The generic routing framework provides standard position-based and topology-based routing protocols which can be chosen and adapted by the application for each communicated message. This framework can also be used to combine standard protocols to new, multimodal protocols as the Distributed Script application does by combining the position-based GCR with the topology-based DSR.

While the simulation and the hybrid mode of JANE has been evolved to a productive environment, the platform mode of JANE has still a proof-of-concept character rather than a ready to use application platform. The mapping of the JANE communication primitives to UDP unicasts and broadcasts does not provide the possibilities of the environment as it should be. A deeper integration of JANE's network capabilities within a Linux kernel module is still work in progress. Although the single threaded platform event-queue is good enough for application prototype tests and demonstrations, a multithreaded core is much more suitable for realizing large environments or multi-application tests. The environment is extended continuously. Beside standard position-based and topology-based unicast routing protocols currently available within the routing framework, also other protocols, for example multicast protocols, should be realized. Finally, the proposed local communication paradigms have not yet been realized within a standard DCF like 802.11. Since addressed mulitcast/broadcast is well suited for wireless ad hoc networks it is planned to adapt the 802.11 MAC or to propose a new DCF for wireless ad hoc communication.

## Acknowledgements

This work is funded in part by DFG, priority programme SPP1140 "Selforganized infrastructures for networked mobile systems".

## References

[1] Ieee 802.11 support for ns/2. http://yans.inria.fr/ns-2-80211/.

[2] Opnet technologies inc.: Opnet modeler. http://www.opnet.com/products/modeler/home.html.

[3] Otcl. http://otcl-tclcl.sourceforge.net/otcl/.

[4] Scalable network technologies: Qualnet family of products. http://www.qualnet.com/products/qualnet.php.

[5] A. Andronache, M. R. Brust, and S. Rothkugel. Multimedia content distribution in hybrid wireless networks using weighted clustering. In *2nd ACM Workshop on Wireless Multimedia Networking and Performance Modeling, WMuNeP 2006, Torremolinos, Malaga, Spain, October 2006*, 2006.

[6] J. Antoniou, V. Vassiliou, A. Pitsillides, G. Hadjipollas, and N. Jacovides. A discrete event based simulation environment for enhanced umts 3rd generation networks. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 369–370, New York, NY, USA, 2004. ACM Press.

[7] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, 1998.

[8] L. Bajaj, M. Takai, R. Ahuja, R. Bagrodia, and M. Gerla. Glomosim: A scalable network simulation environment. Technical Report 990027, 13, 1999.

[9] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM) '98*, pages 85–97, Dallas, TX, USA, 1998.

[10] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483–502, sep 2002.

[11] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[12] J. Cowie, A. Ogielski, and D. Nicol. Modeling the global internet. *Computing in Science and Engineering*, 1:42–50, January/February 1999. http://www.ssfnet.org/homePage.html.

[13] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu. Network visualization with nam, the vint network animator. *IEEE Computer*, 33(11):63–68, Nov. 2000.

[14] K. Fall and K. Varadhan. The *ns* Manual. The VINT Project – A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. http://www.isi.edu/nsnam/, 1989-2006.

[15] T. I. for Wireless and M. Communications. Eurane enhanced umts radio access network for ns/2. http://www.ti-wmc.nl/eurane/.

[16] H. Frey. Scalable geographic routing algorithms for wireless ad-hoc networks. *IEEE Network: Special issue on Ad Hoc Networking: Data Communications & Topology Control*, July 2004.

[17] H. Frey and D. Görgen. Planar graph routing on geographical clusters. *Ad Hoc Networks, Special issue on Data Communication and Topology Control in Ad Hoc Networks*, 3(5):560–574, Sept. 2005.

[18] H. Frey, D. Görgen, J. K. Lehnert, and P. Sturm. Auctions in mobile multihop ad-hoc networks following the marketplace communication pattern. In *6th International Conference on Enterprise Information Systems ICEIS'04*, Porto, Portugal, 2004.

[19] D. Görgen, H. Frey, and C. Hutter. Information dissemination based on the en-passant communication pattern. In *Fachtagung "Kommunikation in Verteilten Systemen" (KiVS)*, Kaiserslautern, Germany, 2005.

[20] D. Görgen, M. Transier, P. Sturm, and W. Effelsberg. Distributed script – prototyping a mobile application for multihop ad-hoc networks. Technical Report TR-06-02, Universität Trier, Trier, Germany, Mar. 2006.

[21] H. Hellbrück and S. Fischer. Basic analysis and simulation of ad-hoc networks. Technical report, 2001.

[22] C. Hiedels, C. Hoff, S. Rothkugel, and U. Wehling. Ubisettlers-a dynamically adapting mobile p2p multiplayer game for hybrid networks. Submitted to 4th IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P'07), White Plains, NY, USA, March, 2007.

[23] D. B. Johnson, J. Broch, Y.-C. Hu, J. Jetcheva, and D. A. Maltz. The cmu monarch projects wireless and mobility extensions to ns. In *E. Tang (Publisher): Proceedings of the forty-second internet engineering task force, Chicago, IL, USA*, 1998.

[24] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[25] J. K. Lehnert, D. Görgen, H. Frey, and P. Sturm. A Scalable Workbench for Implementing and Evaluating Distributed Applications in Mobile Ad Hoc Networks. In *Western Simulation MultiConference WMC'04*, San Diego, California, USA, 2004.

[26] J. Liu and D. M. Nicol. Dartmouth scalable simulation framework (dassf), version 3.1. user's manual. http://www.crhc.uiuc.edu/~jasonliu/projects/ssf/.

[27] G. F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the workshop on Models, methods and tools for reproducible network research MoMeTools03*, pages 5–12, 2003.

[28] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 2.0). http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf, 2004.

[29] M. Transier, H. Füssler, J. Widmer, M. Mauve, and W. Effelsberg. A Hierarchical Approach to Position-Based Multicast for Mobile Ad-hoc Networks. *Wireless Networks - The Journal of Mobile Communication, Computation and Information*, 2006.

[30] K. Walsh and E. G. Sirer. Staged simulation: A general technique for improving simulation scale and performance. volume 14, pages 170–195, New York, NY, USA, 2004. ACM Press.