– SimANet – A Large Scalable, Distributed Simulation Framework for Ambient Networks

Matthias Vodel
Chemnitz University of Technology / Dept. Computer Science, Chemnitz, Germany
Email: vodel@cs.tu-chemnitz.de

Matthias Sauppe and Mirko Caspar and Wolfram Hardt
Chemnitz University of Technology / Dept. Computer Science, Chemnitz, Germany
Email: {saum | mica | hardt }@cs.tu-chemnitz.de

***Abstract*— In this paper, we present a new simulation platform for complex, radio standard spanning mobile Ad Hoc networks. *SimANet - Simulation Platform for Ambient Networks* - allows the coexistence of multiple radio modules with different communication technologies and protocol stacks within one node, which can be used concurrently. By the usage of efficient data structures like *Randomised Skip Quadtrees*, SimANet allows the analysis and evaluation of large scale, heterogeneous network topologies in both static and dynamic simulation scenarios based on different movement models. The software design enables a modular extension with additional models for power consumption, communication complexity or barrier simulation. Furthermore, an integrated MPI library provides the possibility to run distributed test cycles on parallel computing systems. Thereby, special sliding time window algorithms avoid the typical disadvantage of a slow network interconnection structure and allow a dynamic load balancing on the available hardware resources during the runtime. With the main focus on the evaluation of abstract multi-interface, multi-standard communication concepts, we compare the functionality and the complexity of SimANet with well-known simulation tools like ns2-MIRACLE, ns2-NW-Node, OMNet++ and TeNS. Simulation results for different application scenarios estimate features like versatility, practicality or usability in large scale network topologies with up to 10^5 nodes.**

***Index Terms* — simulation framework, parallelisation, wireless communication standards, mobile Ad Hoc networks, multi-standard, multi-interface, ambient networking**

I. INTRODUCTION

The number of wireless communication technologies is steadily rising. Due to different application areas of wireless communication networks within the private and industrial sector, a multiplicity of radio standards has been developed [1]. Thereby, each communication standard has specific, application optimised characteristics concerning power consumption, data transfer rate, frequency band or transmission range. Due to physical and functional differences within the protocol stacks of the several communication standards, an interoperability between these technologies is not possible.

Looking forward to the next generation of mobile technologies, one essential ability will be the integration of different wireless communication standards. To avoid the incompatibilities between available technologies, different research approaches, like Software Defined Radio (SDR) [2], Cognitive Radio [3] or Ambient Network [4] (see Figure 1), provide interesting solutions. In [5], another concept for the radio standard spanning communication in mobile Ad Hoc Networks is introduced, which offers the possibility to connect standardised radio modules on a hardware near layer. Regarding to these current research projects, mobile Ad Hoc networks enable large-scale network topologies in a completely new dimension.

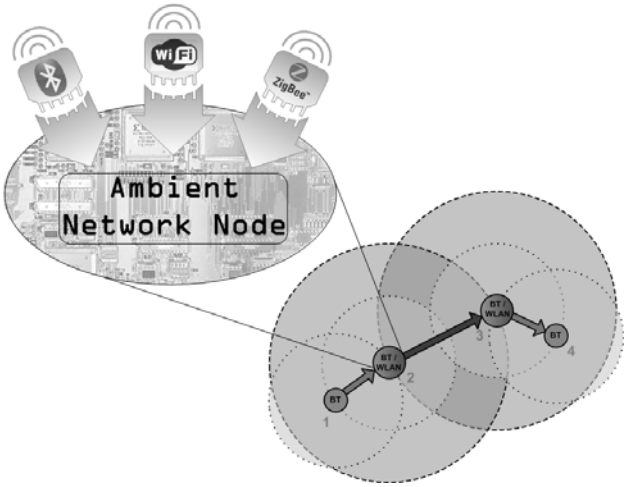

Figure 1. Ambient networking: Integration of several wireless IEEE 802.x communication standards - Next generation wireless networks use a heterogeneous, multi-standard infrastructure for an optimised communication

To handle the complex development of future communication technologies, powerful simulation tools are used to analyse the network behaviour in realistic

application scenarios. Thereby, the most currently available simulation frameworks for mobile Ad Hoc and sensor networks deal with one single radio standard and focus small- or medium-scaled networks [6][7].

To evaluate the performance and the advantages of multi-interface, multi-standard concepts, capable simulation platforms are necessary.

II. Related work

During the last decade, a lot of powerful simulation tools for wired and wireless network topologies have been developed. With the focus on the requirements of modern devices like mobile phones, laptops or PDAs, simulation scenarios with a heterogeneous, dynamic node infrastructure are essential. An efficient analysis of the simulation results reduces the development costs and accordingly the time-to-market of new technologies.

Classical simulation tools like *GloMoSim - (Global Mobile Information Systems Simulation Library)* [8] or *SSFNet - (Scalable Simulation Framework)* [9] are command-line-based. Accordingly, the realisation of specific application scenarios and the user interaction is difficult. For GloMoSim multiple, platform independent extensions for the visualisation of the simulation results exist. Furthermore, the simulator includes different movement models to support dynamic network topologies. In contrast to GloMoSim, SSFNet focuses on static application scenarios. An important feature of SSFNet is the possibility to parallelise the simulation. This speedup enables the analysis of a large scale network behaviour. Both toolkits are limited to a single communication interface per node.

Other simulation tools, for example *NCTUns - (National Chiao Tung University Network Simulator)* [10] or the *OMNeT++* framework [6], provide a well arranged graphical user interface. NCTUns is an integrated network and traffic simulation platform with a flexible simulation engine and very specific purposes for the inter-vehicle communication. The popular OMNeT++ framework provides detailed models for IEEE 802.11, Ethernet, Fiber Distributed Data Interface (FDDI) and Token Ring on the MAC protocol layer. Additional extensions like *Castalia* [11] and other dedicated mobility framework enhance the versatility. But neither NCTUns nor OMNeT++ provide the functionality for multiple radio interfaces or protocol stacks.

In the context of parallelisation, OMNeT++ provides basic features on the basis of MPI. The conversion of a given project to a distributed hardware environment requires a lot of modifications. Different research approaches, i.e. in [12], try to provide comfortable solutions. However, the features for a parallel simulation in OMNeT++ are limited to a conservative, blocking event synchronisation. This clashes with the communication dominant aspects of network simulation tools.

The most famous simulation framework is the *Network Simulator (ns) - version 2* [13], which includes multiple realistic simulation models to analyse the behaviour of a given network topology. The original version focuses on classical, static application scenarios within small and medium scale networks [14].

To avoid these limitations, several extensions of ns2 improve the functionality. In the research field for mobile Ad Hoc and sensor networks, we consider three capable projects. *TeNS - The Enhanced Network Simulator* [15] is an extended version of ns2, which enables a multiple interface support for mobile nodes. TeNS only supports the IEEE 802.11 MAC layer protocol for all interfaces. Simulations with TeNS are limited by the usage of a single radio technology. Application scenarios of multiple, concurrently working communication standards are not possible.

An addition to the support of multiple wireless interfaces, the *Module-based Wireless Node* project (*MW-Node*) [16] also allows coexistence of different communication technologies and routing protocols within one node. MW-Node maps one radio interface to one routing protocol and accordingly it handles each interface as a dedicated communication unit. There is no possibility to use an multi-standard, multi-interface routing protocol (e.g. EBCR [17]), which takes intelligent decisions about the choice of the optimal radio interface.

ns2-MIRACLE [14] allows the coexistence of multiple communication modules within each layer of the protocol stack. The primary disadvantages of MIRACLE correlate with the fundamental characteristics of ns2. The usage is very complex and time-consuming. Furthermore, ns2 is not able to simulate large network topologies with more than 100 nodes efficiently [14].

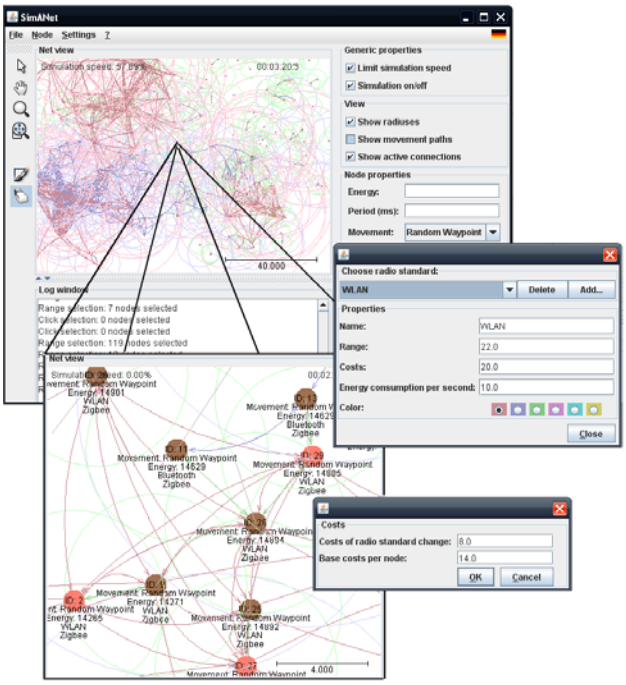

Figure 2. Screenshot of *SimANet - Simulation Platform for Ambient Networks*

PDNS - Parallel/Distributed NS [18] provides an extension for ns2 to enable simulations in a parallel computing environment. Similar to OMNet++, PDNS uses a conservative (blocking based) approach for the synchronisation and accordingly the possible speedup is suboptimal. Furthermore, the integration in such a complex framework like ns2 constrains PDNS to several conceptional limitations.

To provide reliable simulation results for a first order validation of new multi-standard communication concepts before moving to implementation on a specific hardware platform, an easy to use, modular simulation platform is necessary. Consequently, the primary objectives are versatility and practicality.

III. SIMANET

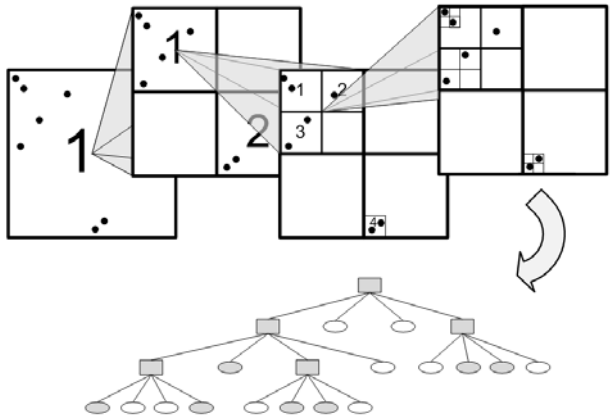

Figure 3. Runtime-efficient object localisation in a two-dimensional area with an complexity of *O(log n)* (in expectation and w.h.p.): exemplary the representation of an optimised *Compressed Quadtree* data structure with seven objects: inner tree nodes (grey rectangles), localised objects (grey circles) and empty squares (white circles)

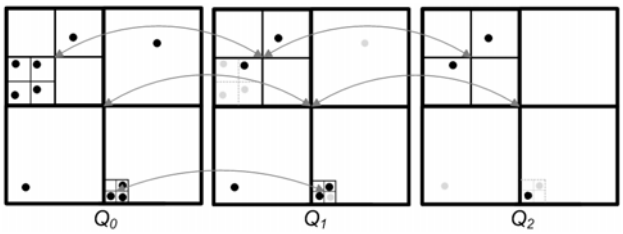

Figure 4. Randomised Skip Quadtree data structure including three double linked, compressed quadtrees. The link pointers of the several squares are represented by the dark grey arrows. The nodes will be copied from Q_0 to Q_1 and from Q_1 to Q_2 with a probability of 0.5. Accordingly, in each quadtree representation, the number of nodes and the quadtree complexity is decreasing (represented by the light grey quadtree structure and the deleted, light grey nodes).

With the focus on a scalable framework for heterogeneous, mobile Ad Hoc networks, we designed *SimANet - Simulation Platform for Ambient Networking* (Figure 2). Similar to related simulation tools like ns2-MIRACLE or ns2-MW-Node, SimANet is able to handle multiple communication interfaces and different radio standards within one network node concurrently. Based on these capabilities, we are able to run behaviour simulations of radio standard spanning routing concepts and complex topology optimisation algorithms for large scale network topologies.

A. Features

SimANet was designed to be as modular as possible to enable a simple and fast extension with additional models for power consumption, movement, communication complexity or barrier simulation. Due to the strict separation of simulator frontend and backend, different operational modes are possible. A default GUI mode provides multiple features for the interaction with the node topology and supports miscellaneous options for the import and export of data. For systems without a dedicated graphic mode, SimANet provides a powerful command line mode including a multifunctional scripting language, which enables an automation of miscellaneous simulation workflows. Additionally, a special environment enables application scenarios on parallel computing systems.

```
Find(p):
  // Find the smallest square that covers
  // the area of the point p by walking
  // through the trees from right to left
  InterestingSqare tempIS := rightmostTree

  while tempIS != null
       tempIS :=
           FindInterestingSquare(tempIS, p)

       // save current pointer and jump to
       // the next left hand tree(Qk -> Qk-1)
       lastIS := tempIS
       tempIS := tempIS.left

  // check whether point is really linked
  // in the found interesting square
  if pointDirectlyLinked(lastIS, p)
       Quadrant q := getQuadrant(lastIS, p)
       return lastIS.quadrants[q]
  else
       return <p not found>

// returns the smallest square
// that covers the area of p
FindInterestingSquare(IS, p):
  Quadrant q := getQuadrant(IS, p)

  // calculated quadrant empty?
  if IS.quadrants[q] is empty
       return IS

  // no further recursion possible?
  if IS.quadrants[q] is single point
       return IS

  // recursion
  recSquare := FindInterestingSquare
                        (IS.quadrants[q], p)
  if recSquare is not empty or single point
       return recSquare
  else
       return IS
```

Source Code Listing 1: Localisation process based on the Skip Quadtree data structure. (Pseudocode)

B. Design

The basic simulation engine has to manage all networking nodes within the working area, including neighbourhood localisation, communication and movement. Especially the localisation of nodes during a range search has to be done thousands of times in each simulation step. Therefore, an optimised runtime for an average and worst case localisation process is a primary objective of the central simulation engine. The used data structure has a significant influence on the performance regarding memory and operating speed.

For SimANet, the *Skip Quadtree* [19] was chosen. It is an improvement of the common compressed geometric quadtree (Figure 3), whose worst case depth is $O(n)$ [19], whereas n represents the number of network nodes. Thus, node localisations and range searches, which are needed to find neighbour nodes, would be inefficient.

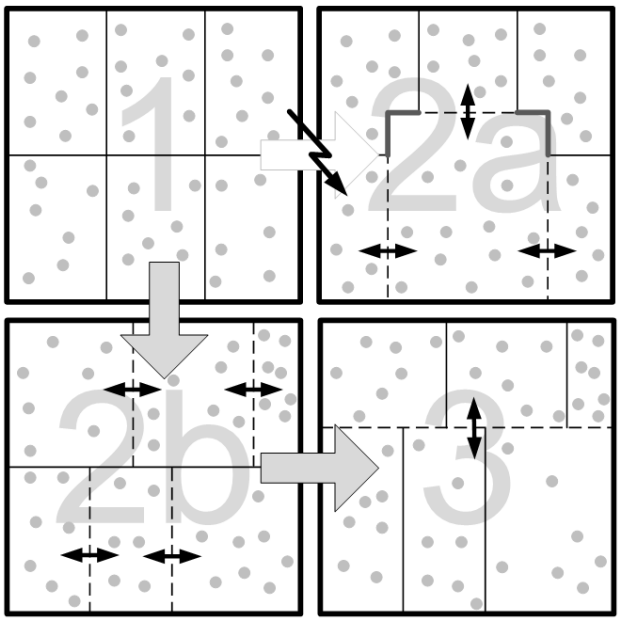

Figure 5. Distributed simulation scenario with six computing nodes: dependent on the current node distribution, the simulation engine is able to reallocate the borders of the several sectors during the runtime. This allows an optimised usage of the available hardware resources and reduces the network load within the cluster. Thereby, the reallocation process is critical to avoid polygons like in the case 2a. A structured algorithm realises the cases 2b and 3.

The Skip Quadtree solves this problem by managing several compressed quadtrees in a double linked list, represented by Q_0 ... Q_{k-1}. Each tree has a corresponding point set, S_0 ... S_{k-1}. S_0 is equal to the initial input set and includes all point in the simulation area. The additional quadtrees represent subsets of the input set S_0. S_{i+1} is derived from S_i. Thereby, each point from S_i will be inserted into the set S_{i+1} with a predefined probability (default value 0.5). The value k represents the number of created quadtrees, normally $\log n$. Congruent inner nodes in adjacent trees are double linked. Figure 4 shows an exemplary skip quadtree.

The localisation process of a chosen object starts with a search within the rightmost tree. If the object isn't found, the algorithm steps back to the smallest square that covers the area of the object (i.e. the last visited inner node) and uses its link to the equal-sized and -positioned square in the left neighbour tree. After that the algorithm repeats this procedure starting from there. The processes for inserting and deleting an object are similar. Source code listing 1 represents the localisation process in pseudocode.

Interestingly, the number of searching steps per tree is constant w.h.p., as proven in [19]. Accordingly, the total number of steps per query is $O(\log n)$.

Another feature of the Skip Quadtree, which is very important in relation to the SimANet platform, is the ability to answer queries for a range search within a defined radius around a chosen node in $O(\log n)$. Further detailed information regarding these algorithms are presented in [19].

C. Parallelisation

As already mentioned, one goal of SimANet is the ability to handle huge simulation scenarios. To improve the simulation speed and the maximum number of simultaneous nodes, the simulator backend was designed to support a parallel computing environment using *MPI (Message Passing Interface)*. Thereby, an efficient parallelisation seemed to be difficult, because the simulation of networks is communication dominant. In SimANet, we use a *geometric partitioning algorithm*, where each processor is simulating its own area in shape of a rectangle. Simulated nodes near the edges of these sectors are critical, because the transmission ranges of the radio modules cross the borders. Accordingly, the simulation speed slows down significantly. Another problem concerns the time synchronisation of the distributed simulator instances. The possibility to ensure isochronous simulation times would produce a heavy network and CPU overhead. Consequently, we wouldn't be able to create optimised simulation scenarios with an adequate speedup in a parallel computing environment.

The idea of simulating into the future and going back in time if necessary (as proposed in [20]) turned out to be inadequate due to the amount of simulated network traffic at the sector borders. The states of the simulator instances are changing too fast to achieve a sufficient speedup.

In order to minimise network communication and synchronisation overhead, several approaches were implemented. The most important one is a model of *tolerated time inaccuracy*, using a *sliding time window* per node. Typically, sliding window algorithms are used for the flow control during a data transmission within a distributed communication network. Thereby, the algorithms allow a continuous data flow between an asynchronous transmitter and receiver. We adopt this concept for the simulation of multiple network nodes in a distributed computing environment.

The key idea is to allow a specified maximum drift between the times of neighbour computing nodes, defined as t_d. To exploit the physical network topology, each CPU has to communicate with its directly attainable

neighbourhood. Thereby, the several computing units exchange their current time stamps to calculate an upper limit of the simulation time, forming a *sliding time window*. If this limit is reached, the simulation has to be suspended. In an optimal case, this will never happen, because new requests are sent to the neighbour CPUs after simulating the half of the current time window, which will enhance the time window.

```
Init:  long timeLimit            := t_d
       list listResponseTimes    := <empty>
       int  countResponsesNeeded := 0

       TIME_HALF := t_d / 2
       TIME_UP   := t_d

If TIME_HALF is reached:
  countResponsesNeeded := #neighbour CPUs
  listResponseTimes.clear()
  for each neighbour CPU p do
     Network.Send(p, TIME_REQUEST,
                  currentTime() + t_d)

If TIME_UP is reached:
  // if not all necessary time responses
  // have been received, pause simulation
  if countResponsesNeeded != 0
     pauseSimulation()

// neighbour CPU asks for current time
If incoming "TIME_REQUEST" Message:
  if timeLimit >= msg.requestedTime
    // send time response immediately
    Network.Send(msg.source, TIME_RESPONSE,
                  currentTime() + t_d)
  else
    // delay time response
    Create delayed Transmit Event when the
    Condition is reached

// time response received from a neighbour
If incoming "TIME_RESPONSE" Message:
  countResponsesNeeded :=
                    countResponsesNeeded -1
  listResponseTimes.add(msg.time)

  if countResponsesNeeded = 0
    // all queried neighbour processor
    // have responded -->
    // update the own time limit
    timeLimit =
              listResponseTimes.getMinimum()

    // update triggers for time events
    TIME_HALF := currentTime() + (timeLimit
                 - currentTime()) / 2
    TIME_UP   := timeLimit

    listResponseTimes.clear()
    if simulation is paused
       continueSimulation()
```

Source Code Listing 2: Adapted sliding time window algorithm, which enables a continuous simulation process in a distributed computing environment. (Pseudocode)

The higher the value of t_d, the higher the inaccuracy, but also the higher the parallelisation efficiency. By now, an optimal td has to be determined manually for each simulation scenario. Further work has to be done on dynamic, optimal control of t_d. The pseudocode representation of the dynamic sliding time window process is shown in source code listing 2.

```
while(true)
  // equalisation within one row
  if right neighbor rn exists
    l_1 = getLoad()
    l_2 = rn.getLoad()
    if abs(1 - l_1/l_2) > THRESHOLD_X
       x_1 = getMiddlePosition().x
       x_2 = rn.getMiddlePosition().x
       d_x = x_2 - x_1
       moveRightBorder(x_1 + d_x * l_2 /
                       (l_1 + l_2))
       commitBordersGlobally()

  // equalisation among rows - only row
  // masters (one per row) may execute this
  if rowmaster
    if bottom neighbor bn exists
       l_1 = row.getLoad()
       l_2 = bn.row.getLoad()
       if abs(1 - l_1/l_2)> THRESHOLD_Y
       y_1 = getMiddlePosition().y
       y_2 = rn.getMiddlePosition().y
       d_y = y_2 - y_1
       moveBottomBorder(y_1 + d_y * l_2  /
                        (l_1 + l_2))
       commitBordersGlobally()

  // execute the whole loop regularly
  sleep for t_sleep milliseconds
```

Source Code Listing 3: Dynamic reallocation algorithm for distributed simulations in a parallel computing environment. (Pseudocode)

Another improvement regarding parallelisation is a dynamical resize and reallocation of the simulation areas to the available hardware resources depending on the computing load. This has to be done in a structured way in order to keep the topology simple and the topology management overhead low. Resizing a single rectangle in both dimensions in a grid is complex and results in arbitrary shapes of the former neighbour rectangles (see Figure 5 – case 2a).

Figure 5 illustrates an exemplary scenario. The resizing processes in the vertical and horizontal dimensions are done independently. In a first step, the grid of nodes can be treated as rows, stacked onto each other (Figure 5 – case 2b). Each row contains several rectangles with computing units, located next to each other. The load balancing algorithm works as follows: If the load difference between two neighbour areas exceeds a certain percentage, the border between these sectors is moved accordingly. This process is repeated until the load is balanced among all sectors in the row. Note that the rectangles are resized only in the horizontal direction;

the top and bottom edges are not modified. In the second step, load balancing is done among the several rows, again moving the borders between neighbours, if the load difference between the rows is too high (Figure 5 – case 3). But this time, the borders between the rows are moved vertically. The left and right edges of rectangles are not changed. Using this approach, the rectangle shape of each simulation area is ensured. The source code listing 3 illustrates the dynamic reallocation process.

IV. Simulation Results

For testing the performance of SimANet, all simulation scenarios were launched on the *Chemnitz High Performance Linux Cluster (CHiC)*, which provides a highly parallel computing infrastructure with more than 500 computing nodes and an *InfiniBand* network interconnection. Furthermore, for the comparison to related network simulation tools, several test cycles are performed on a conventional desktop PC with 2 GB of RAM and a single core Pentium 4 CPU.

A. Performance I – Simulation engine

First simulations measure the performance of three known data structures for the object localisation in an *n*-dimensional area. As already mentioned, this data structures represent the core of each simulation engine. We compare the compressed quadtree, the proposed skip quadtree and a trivial list implementation with object identifier and X/Y coordinates. The test cases include three different application scenarios. In each scenario, the runtime for 3 million random node localisations was measured. Due to the different parameters of the several test cases, the simulation results in figure 6 illustrate relative values. Thereby, the runtime of the proposed skip quadtree represents the 100% reference value.

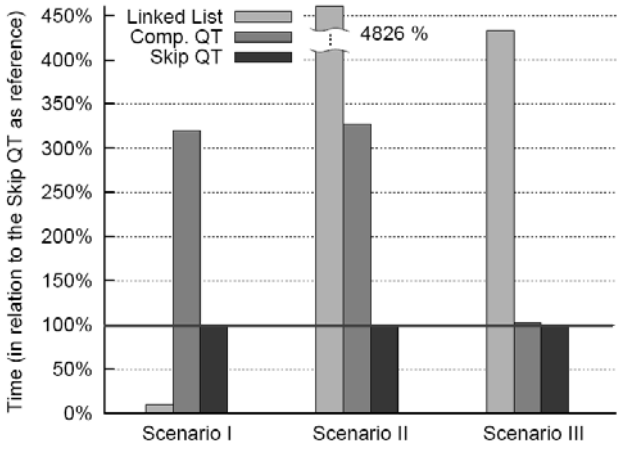

Figure 6. Performance analysis I: three different data structures for the object localisation within the simulation engine - Trivial object list with identifier and X/Y coordinates; Compressed Quadtree; Skip Quadtree. The measurements of the Skip Quadtree represent the reference value of the simulation time.

Scenario I measures a worst case scenario for common compressed quadtree data structures. Thereby, a small scale network topology with only 62 nodes is arranged in a line, where the distance from one node to the next one doubles each time. Accordingly, the compressed quadtree reaches its worst case depth of *n*. As expected in this scenario, the trivial list implementation is significant faster than both quadtree data structure, which results from the small number of network nodes. The *second scenario* combines the topology of scenario I with 5000 additional, random distributed nodes. Due to the increased number of available nodes, the performance of the list implementation is absolutely weak. The compressed quadtree needs more than 300% of time in comparison to the skip quadtree approach. The *scenario III* represents a random distribution of 500 nodes with a defined hotspot, where the node density is very high. In this case the difference between compressed quadtree and skip quadtree is minimal.

In conclusion, the skip quadtree approach is better within node topologies where the depth of a representing compressed quadtree is $>O(\log n)$. The trivial list implementation is not feasible and inefficient for a large number of network nodes.

B. Performance II – Parallelisation

In figure 7 we present some performance measurements using parallelisation. Simulation scenarios with 1000 to 100000 randomly distributed network nodes were run on 1, 2, 4, 8 and 16 processors using the CHiC. As expected, only very large scenarios could be parallelised efficiently, which results from the communication bottleneck between neighbour sectors/processors. The results of small simulation scenarios, for example 1000 network nodes on two processors, offer a moderate speedup of 1.18. Further tests on four or more processors produce a speedup even below 1.

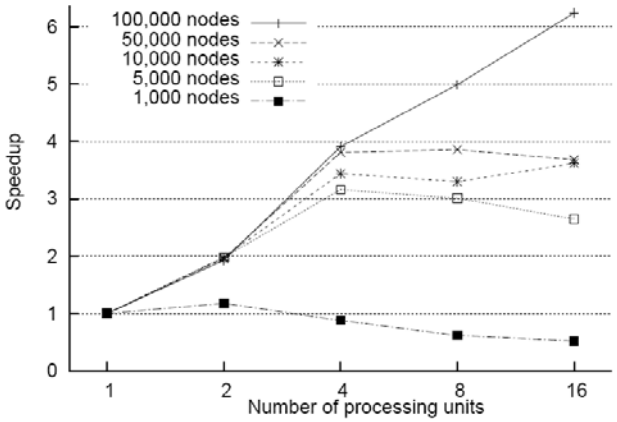

Figure 7. Performance analysis II: several, distributed simulation scenarios on the CHiC. The simulation of 50000 nodes on four processors almost reaches the theoretical speedup limit. Even larger network sizes provide additional speedup. Without using the sliding time window algorithm, the parallelisation didn't result in measurable speedup at all.

Parallelisation of larger networks performed much better. A scenario of 5000 nodes gained a speedup of 3.16 on four computing nodes; a size of 50000 reached a speedup of 3.81 on four computers. Due to the necessary

communication effort, the usage of more processors does not improve the speedup significantly.

Considering the communication aspect, we expect further improvements by optimising the communication library. By now, we use *jMPI* [21], a native Java MPI implementation using TCP. An implementation like *mpiJava*, which uses Java Native Interfaces and UDP, promises a significant performance boost. Unfortunately, this requires some special adjustments on the computing cluster.

Another bottleneck in a parallel computing environment concerns the dumping of simulation snapshots into an output file. The current implementation uses a master-slave algorithm, where a master computer collects serialised data from the slave computing nodes and writes it into a file. To avoid this problem, SimANet can use the centralised and diskless storage architecture of the CHiC. A possible solution manages one single dump file, which is located in this storage network. Thereby, each computing node is able to append its own data into this file. To ensure the simulation continuation during the dumping process, the caching of temporary data is necessary.

Neither OMNet++ nor ns2 achieve such a performance regarding parallel simulation scenarios. Both frameworks only provide conservative, blocking approaches to synchronise the several computing nodes. Accordingly, this prevents a suitable speedup.

C. Memory Usage

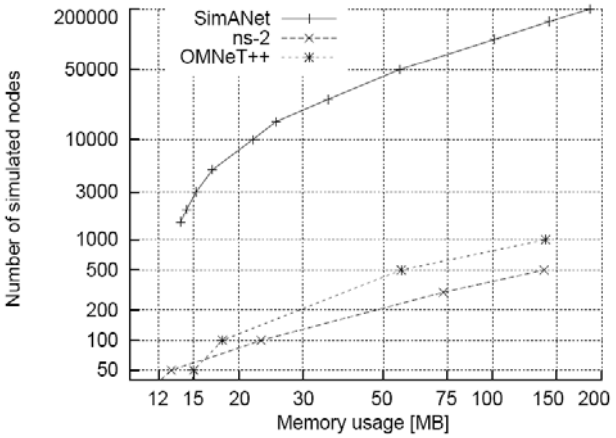

Figure 8. Memory usage of ns2 [7], OMNet++ and SimANet, dependent on different simulation scenarios (GUI enabled). With SimANet less than one kB is needed per simulated node. The linear memory increase per node in large scale networks results from the efficient neighbourhood management.

SimANet was designed to be as lightweight as possible. Each simulated network node uses only about 0.85 kB of memory in average. This value highly depends on the neighbourhood management of each node. In the current implementation, each simulated node maintains a list of neighbour nodes, which are used for communication. Thus, the memory usage per node may vary around ±200 bytes, depending on the network density. As figure 8 shows, the additional memory used by the skip quadtree data structure is minimal, because the node payload is stored only once, of course. The GUI, implemented in Java Swing, uses 12 to 16 MB of memory; the simulator backend (without GUI) needs less than 1 MB. This memory-saving architecture enables us to simulate large scale Ad Hoc networks also on moderate computer hardware.

In comparison to simulation tools like ns2 or OMNet++, which need >>100 MB of memory to simulate 1000 nodes, SimANet is substantially more efficient. Furthermore, the initialisation time for creating a 1000 node simulation scenario with OMnet++ needs more than 30 minutes on a Pentium 4 workstation.

V. Conclusion

The simulation of realistic network scenarios like in [22] verifies the restricted capabilities of OMNet++ regarding large scale network topologies. In our opinion, the exorbitant memory usage, the huge initialisation time and the missing features for a parallelisation make the simulation tool not feasible for large scale simulation scenarios. As already mentioned in [7] the limitation of the ns2 framework is similar to OMNet++. In the original version, simulation scenarios with 100+ nodes are not feasible. Using the proposed improvements and modifications, the limit will be enhanced to 3000 nodes. Anyway, this value represents the upper bound of the ns2 software. At present, the number of available extensions and simulation models for ns2 or OMNet++ are definitely higher and consequently, these frameworks support more specific application scenarios. The conceptional disadvantages still remain.

In contrast to that, the presented SimANet platform enables the simulation and visualisation of 10000+ nodes on a standardised single CPU computer system. Due to the usage of Java, the development of additional, application specific extensions is simple and very fast. Features for a parallelised operation mode allow an additional improvement of the simulation performance. Thereby, first simulation results demonstrate an excellent speedup. Further comparisons with related simulation frameworks like ns2 or OMnet++ verify the advantages of SimANet regarding first order validations of new communication concepts.

References

[1] T. Cooklev. Wireless Communication Standards - *A Study of IEEE 802.11., 802.15., and 802.16*. IEEE Press, New York, USA, 2004.

[2] Adam S. Harrington, Chin-Gi Hong, and Anthony L. Piazza. *Software Defined Radio - The Revolution of Wireless Communication*. International Engineering Consortium, 2004.

[3] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks Journal (Elsevier)*, 50(13):2127–2159, September 2006.

[4] Bengt Ahlgren, Lars Eggert, Brje Ohlman, and Andreas Schieder. Ambient networks: Bridging heterogeneous network domains. *In Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile*

Radio Communications (PIMRC), Berlin, Germany, September 2005. IEEE Computer Society.

[5] Matthias Vodel, Mirko Caspar, and Wolfram Hardt. Performance analysis of radio standard spanning communication in mobile ad hoc networks. *In Proceedings of the 7th IEEE International Symposium on Communications and Information Technologies (ISCIT)*, pages 848–853, Sydney, Australia, October 2007. IEEE Computer Society.

[6] A. Varga. *The OMNet++ distrete event simulation system*. http://www.omnetpp.org, 1999. [Online].

[7] Valeri Naoumov and Thomas Gross. Simulation of large ad hoc networks. *In Proceedings of the 6th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWIM '03)*, pages 50–57, San Diego, USA, 2003. ACM Press.

[8] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. Glomosim: *A scalable network simulation environment*.Technical report, 1999.

[9] J. Cowie, A. Ogielski, and D. Nicol. *The SSFNet network simulator*, http://www.ssfnet.org/homePage.html, 2002. [Online].

[10] S.Y. Wang and Y.B. Lin. NCTUns network simulation and emulation for wireless resource management. *Wiley Wireless Communications and Mobile Computing*, 5(8):899–916, December 2005.

[11] Australia's ICT Research Centre of Excellence. *Castalia - a simulator for WSNS*. http://castalia.npc.nicta.com.au/, 2008. [Online].

[12] Y. Ahmet Sekercioglu, Andras Varga, and Gregory K. Egan. Parallel simulation made easy with OMNet++. *In Proceedings of the 15th European Simulation Symposium (ESS2003)*, Delft, The Netherlands, October 2003.

[13] Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. *In WNS2 '06: Proceeding from the 2006 Workshop on NS-2: the IP Network Simulator*, page 13, Pisa, Italy, 2006. ACM Press.

[14] Nicola Baldo, Federico Maguolo, Marco Miozzo, Michele Rossi, and Michele Zorzi. ns2-MIRACLE: a modular framework for multi-technology and cross-layer support in network simulator 2. *In Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools (ValueTools '07)*, pages 1–8, Nantes, France, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[15] Siddharth Saha. *TeNS - the enhanced network simulator*. http://www.cse.iitk.ac.in/users/braman/tens/. [Online].

[16] Laurent Paquereau and Bjarne E. Helvik. A module-based wireless node for ns-2. *In WNS2 '06: Proceeding from the 2006 Workshop on NS-2: the IP Network Simulator*, page 4, Pisa, Italy, 2006. ACM Press.

[17] Matthias Vodel, Mirko Caspar, and Wolfram Hardt. Energy-balanced cooperative routing approach for radio standard spanning mobile ad hoc networks. *In Proceedings of the 6th International Information and Telecommunication Technologies Symposium (I2TS)*, pages 42–47, Brasilia, Brazil, December 2007. IEEE Region 9.

[18] Dr. George Riley and Alfred Park. *PDNS -parallel/ distributed ns*. http://www.cc.gatech.edu/computing/ compass/pdns/index.html, 2004. [Online].

[19] David Eppstein, Michael T. Goodrich, and Jonathan Z. Sun. The skip quadtree: A simple dynamic data structure for multidimensional data. *In SCG '05: Proceedings of the 21st Annual Symposium on Computational Geometry*, pages 296–305, Pisa, Italy, 2005. ACM Press.

[20] G.F. Riley, R.M. Fujimoto, and M.H. Ammar. A generic framework for parallelization of network simulations. *In Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 128–135, 1999.

[21] Steven Raymond Morin, Ian Harris Member, Seshu B. Desu, and Department Head. jMPI: Implementing the message passing interface standard in java. *In IPDPS Workshop on Java for Parallel and Distributed Computing*, 2002.

[22] Roland Bless. Using realistic internet topology data for large scale network simulations in OMNet++. *In 2nd International OMNeT++ Workshop*, Berlin, Germany, January 2002. TU Berlin.

Matthias Vodel was born in Germany in 1982. He received the German Diploma degree (equal to M.Sc.) in Computer Science with the focus on computer networks and distributed systems from the Chemnitz University of Technology / Germany in 2006. The major field of study during his master thesis deals with new routing strategies and topology optimisation algorithms in mobile Ad Hoc networks.

In 2005, he studies one term at the University of Sussex / United Kingdom at the Dept. of Informatics. Currently, he works as a research assistant and Ph.D. student at the Department of Computer Science, Chair of Computer Engineering at Chemnitz University of Technology, Germany.

Actual research projects focus self-organisation in mobile Ad Hoc networks and radio standard spanning wireless communication technologies like Ambient Networking, Cognitive Radio or Software Defined Radio. Additional fields of interest concern network security, protocol engineering and embedded systems.

During a research exchange at the King Mongkut's University of Technology Northern Bangkok / Thailand in May 2008, Mr. Vodel received the best paper award for the conference paper "EBCR - A Routing Approach for Radio Standard Spanning Mobile Ad Hoc Networks".

Matthias Sauppe was born in 1985 in Germany. Currently he is a German Diploma student (equal to M.Sc.) for computer science at the University of Technology in Chemnitz / Germany and is going to finish his study in March 2009. His major field of study focuses electronics and embedded design.

Besides the academic studies he works as student research assistant at the computer engineering Dept. under guidance of Mr. Vodel.

In 2003, Mr. Sauppe successfully participated several computer science competitions and reaches the final round of the German National Computer Science Olympics, where he got delegated into the German National Team for the International Baltic Olympiad in Informatics (BOI). With the German National Team he received the 3rd in 2004.

His research interests include microprogramming, Linux Kernel / driver engineering and wireless communication technologies.

Mirko Caspar was born in the former German Democratic Republic in 1980 and received the German Diploma degree (equal to M.Sc.) in Computer Science, focuses electrical engineering and embedded systems, from Chemnitz University of Technology, Germany, in 2006.

He is currently research assistant at the Department of Computer Science, Chair of Computer Engineering at Chemnitz University of Technology, Germany where he also is pursuing his Ph.D. degree under guidance of Prof. Dr. W. Hardt. His special field of interests are Test-Automation for Embedded Systems, Organic/Pervasive Computing and Radio-Standard-Spanning Communications.

For his Diploma/Master thesis, Mr. Caspar received a special award within the "SAX-IT Nikolaus-Joachim-Lehmann-Preis"

Prof. Dr. Wolfram Hardt is professor for computer science and head of the Computer Engineering Group at the Chemnitz University of Technology. He was born Germany 1965 and received the German Diploma degree (equal to M.Sc.) in Computer Science in 1991 from the University of Paderborn. Accordingly, Prof. Hardt received the Ph.D. degree in Computer Science from the University of Paderborn in 1996.

From 2000 to 2002 he was chair of the Computer Science and Process Laboratory at the University of Paderborn / Germany. After that he worked as chair of the operating systems Dept. at the University of Kassel / Germany. Since 2003 Prof. Hardt became chair of the computer engineering Dept. at the Chemnitz University of Technology / Germany. Furthermore, since 2006 he is dean of the Faculty for Computer Science and the scientific director of the university computing centre at the Chemnitz University of Technology / Germany. He is editor of a scientific book series about self-organising embedded systems and has published more than 50 papers.

Prof. Hardt is member of the Association for Electrical, Electronic and Information Technologies (VDI/VDE), the Association for Computer Science (GI) and the Association for Computing Machinery. Since 2006 he is committee member of the DATE conference – "Design Automation and Test in Europe"; Topic: System Synthesis and Optimization. His research interests include Hardware/Software Codesign, Organic Computing and Reconfigurable Hardware.