Simulation of Large Ad Hoc Networks [*]

Valeri Naoumov
Departement Informatik
ETH Zürich
CH 8092 Zürich

Thomas Gross
Departement Informatik
ETH Zürich
CH 8092 Zürich

ABSTRACT

An ad hoc network is formed by wireless mobile nodes (hosts) that operate as terminals as well as routers in the network, without any centralized administration. Research in ad hoc networks often involves simulators since management and operation of a large number of nodes is expensive. However, the widely used simulator NS-2 does not scale; it is very hard to simulate medium scale networks with 100+ nodes. We describe here improvements to NS-2 to meet the needs of large ad hoc network simulations. The modified NS-2 simulator is based on the idea of exploiting the limited interference of wireless communication. The modified simulator has simulated populations of up to 3000 nodes so far and works up to 30 times faster than the original version. We also discuss how the modified simulator is validated.

Categories and Subject Descriptors: I.6.5 [Simulation and Modeling]: Model Development; I.6.4 Model Validation and Analysis — NS-2; E.1 [Data Structures]: Graphs and Networks

General Terms: Algorithms, Experimentation, Performance

Keywords: Network Simulation, Scalability, Ad Hoc Networks

1. INTRODUCTION

Ad hoc (or self-organizing) networks operate without a predefined fixed (managed) infrastructure. In the simplest case, an ad hoc network involves exactly two hosts (e.g., two notebooks communication via the IR port), but interesting scenarios involve a larger number of hosts. The benefits and limits of self-organization can be explored only in large settings – only then can we investigate the scaling of a protocol or management system.

Simulators are attractive vehicles to explore ad hoc networks since it is difficult (and expensive) to manage a large number of real network hosts (nodes). The NS-2 simulator is frequently used for these simulations since it supports the popular WaveLAN cards. Several researchers have proposed algorithms and protocols for ad hoc networks, e.g., protocols to solve the multi-hop routing problem (each node must operate as a router, forwarding packets to their final destinations)[7, 6, 5, 8]. However, exploring these protocols in scenarios with a small number of nodes ($<$ 100 nodes, $100+m^2$) may not expose all subtleties. On the other hand, asymptotic studies may provide upper and lower bounds but do not suggest changes to the protocol or the MAC interface. Since a simulator can record events of interest, it is an indispensable aid in the evaluation and evolution of ad hoc networks.

The simulation of large scenarios is crucial for investigations of ad hoc networks. While there exist many papers that compare the performance of different routing protocols for wireless ad hoc networks, the scalability of routing algorithms often receives little attention. E.g., in [2] the authors use the NS-2 simulator to compare the performance of DSR and AODV. The largest scenario simulated ($2200 \times 600m^2$ area, 100 nodes, 40 CBR sources simulated for 500 seconds) supports the main argument why their comparison is better than many previous ones. The authors complain that they could not simulate larger scenarios because of the slow speed of the NS-2 simulator. Their results show that in many cases DSR and AODV demonstrate similar behavior, But with 100 nodes, DSR delivers 5-50% fewer packets than AODV, depending on the traffic load (number of CBR sources). The lowest percentage of delivered packets shown by DSR is 45%, compared to 75% for AODV. The best performance with 100 nodes is observed simulating 10 CBR sources – DSR and AODV were able to deliver 88% and 92% of packets correspondingly.

As an example of how the investigation of protocols' performance could be continued in [2] we present the results obtained with the help of the modified NS-2 simulator that is described in this paper. Figure 1 depicts the percentage of received packets in the following scenarios:

- $1500 \times 300m^2$ area, 50 nodes;
- $2121 \times 425m^2$ area, 100 nodes;

This research was supported, in part, by the NCCR "Mobile Information and Communication Systems", a research program of the Swiss National Science Foundation, and by a gift from the Microprocessor Research Lab (MRL) of Intel Corp.

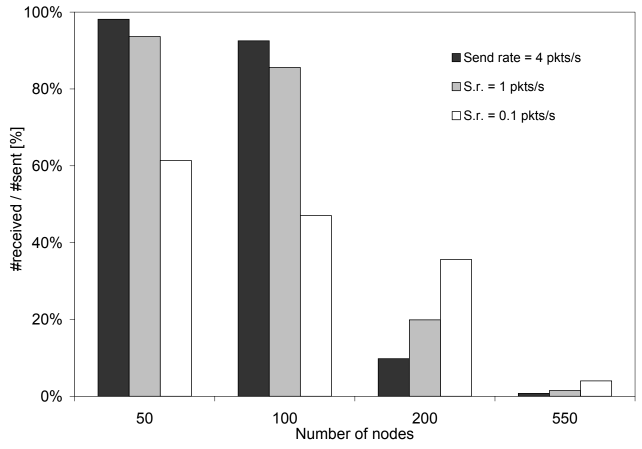


Figure 1: Percentage of packets received (of all packets sent). Pause time 1 s, packet size 64B.

- $3000 \times 600 m^2$ area, 200 nodes;
- $5000 \times 1000 m^2$ area, 550 nodes.

All scenarios were simulated for 600 seconds with the number of active CBR sources equal to 1/3 of the number of nodes involved. Nodes move constantly for the duration of simulation with a speed between 0 and 15 m/s. Three different send rates investigated: 0.1, 1, and 4 packets/s. These experiments show that DSR looses significantly its effectiveness as the network size grows larger and point to the importance of simulating settings with a large number of nodes.

A simulator provides a rich experimentation environment at really low cost. But the problem to obtain a result with reasonable runtime is a serious issue in the simulation of large ad hoc scenarios (100s or 1000s of mobile nodes, areas of approx. $1-10\ km^2$). In such a simulator the most computation-intensive part is the interference computation, since a system with N pairs of transmitters and receivers requires that $O(N^2)$ pairwise interactions are computed. That is why most of the existing simulators show rather poor performance dealing with large ad hoc networks.

There exist several approaches to solve the abovementioned problem of the interference computation. Perrone and Nicol[9] exploit the idea of the well-known N-body algorithm, which is used for calculating the gravitational interactions between objects in simulations of astrophysical models. Since both the gravitational interaction between two bodies and the strength of the radio signal between transmitter and receiver are inversely proportional to the power of the distance, the authors adapt the Barnes-Hut algorithm (one of the algorithms for N-body simulation) to calculate interference in wireless networks. Their results show that the Barnes-Hut algorithm significantly reduces the number of pairwise interactions that must be computed, compared to a brute-force approach.

In this paper we show how the limited interference computation (LIC) algorithm can be applied to improve the performance of the NS-2 simulator. Perrone and Nicol[9] discuss that LIC-based simulations may show better performance and accuracy than the N-body approach, but do not pursue this topic in depth. Furthermore, there exist other significant differences between their work and the material we present here.

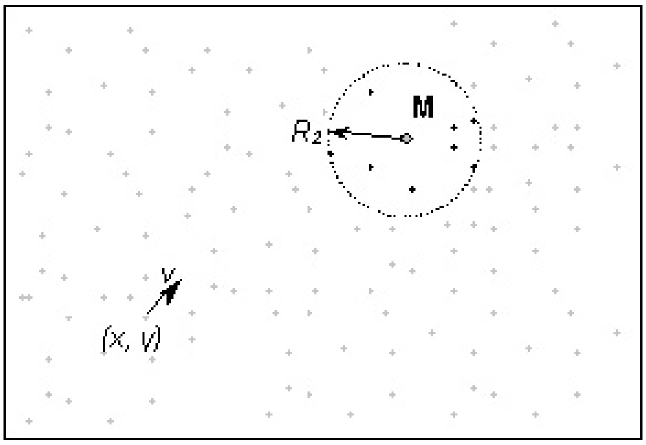


Figure 2: Topography with N mobile nodes.

First, Perrone and Nicol consider only cellular wireless networks with a very low density of nodes (1 node and 1 base station per km^2). Cellular networks have connections only between mobile nodes and the base station(s). Ad hoc networks exploit peer-to-peer communication between any set of nodes that are within transmission range, and there is no distinction made between "nodes" and "base stations". Second, only stationary networks are considered. We operate with ad hoc networks populated by mobile nodes with the variety of densities (from 2 nodes/km^2 to 500 nodes/km^2) and different mobility rates – from stationary to highly dynamic nodes (with speeds up to 20 m/s).

This paper describes the ideas and algorithms we integrated into the NS-2 simulator [3] as well as the validation of these changes. Due to these modifications, we realize a noticeable improvement in CPU time to simulate large ad hoc networks. That improvement allowed us to investigate the efficiency of routing protocols on larger scenarios than have been used in previous studies that were based on NS-2, and we see how the size of an ad hoc network (negatively) impacts various performance metrics.

2. NS-2 SIMULATOR

The NS-2 simulator is a discrete event simulator widely used in the networking research community [3]. It was developed at the University of California at Berkeley and extended at Carnegie Mellon University to simulate wireless networks [4]. These extensions provide a detailed model of the physical and link layer behavior of a wireless network and allow arbitrary movement of nodes within the network. Some of the recently proposed wireless routing protocols (DSDV, TORA, DSR and AODV) [7, 6, 5, 8] are also integrated into NS-2.

Each run of the simulator accepts as input a scenario file that describes the exact motion of each mobile node together with the sequence of packets originated by each node as time progresses. The way NS-2 works can be easily explained by an example. We have the topography - a rectangular simulation area (Figure 2) with N mobile nodes. Each node has a position and a velocity and moves around on the topography. The position of a mobile node can be calculated

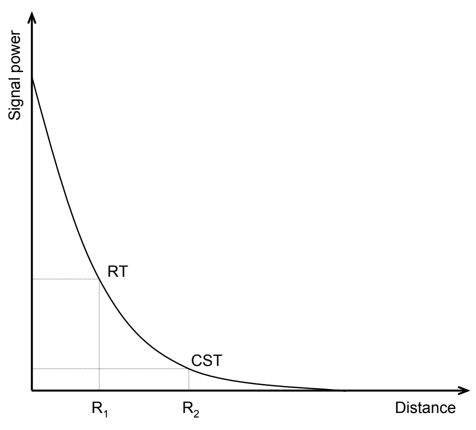


Figure 3: Transmission power as a function of distance from the transmitter.

as a function of time and is used by the radio propagation model to calculate the propagation delay from one node to another as well as the power level of a received signal. We call "receiver" any mobile host receiving a packet.

The power level at which the packet is received is compared to two different values (see Figure 3):

- the receive threshold (RT);
- the carrier sense threshold (CST).

If the power level falls below the CST, the packet is discarded as noise. If the received power level is between CST and RT, the packet is marked as a packet in error before being captured. Otherwise, if the power level is above RT, the packet is received without errors.

Once the receiver starts receiving a new packet, it checks that its receive state is presently "idle", meaning it does not currently process any packet. If the receiver is not idle, one of two things can happen:

- If the power level of the packet already being received is at least 10 decibels greater than the power level of the newly received packet, we assume capture of the current packet, discard the new packet, and allow the receiving interface to continue with its current receive operation.
- Otherwise, a collision occurs, and both packets are dropped.

Consider some mobile node M willing to transmit (Figure 2). Beyond the distance R_2, the transmission power level of M's signal is below the CST. The simulator must compute the mobile nodes that are within this range (so they are affected by M's signal). Therefore NS-2 simulator checks for every node in the topography that is currently participating in a data exchange whether the power level of M's signal is above CST or not. This check causes a large number of unnecessary steps that increase the cost of the computation exponentially in N, since a system with N pairs of transmitters and receivers requires that $O(N^2)$ pairwise interactions are computed.

3. IMPROVEMENTS TO NS-2

The problems one experiences using NS-2 to simulate large ad hoc networks can be divided into two main types: fundamental problems and system idiosyncrasies. The first type of problems restricts the use of NS-2 due to the very high runtime of the simulator, the second seriously limits the number of scenarios that can be simulated.

There are three (C++) classes of the NS-2 simulator implementation that are essential for understanding our modifications:

- The *Channel* class simulates the actual transmission of the packet at the physical layer. Method *send()* of this class allows the MAC object to transmit a packet on the channel for a specified duration of time.
- The *MobileNode* class has *double X, Y* as the node's coordinates, and *update_position()* is the method responsible for the node's movement.
- NS-2 is an event-driven simulator. The class *Scheduler* runs by selecting the next earliest event, executing it to completion, and returning to execute the next event. Method *schedule()* of this class puts an event in the calendar queue.

3.1 Solving fundamental problems

NS-2 is slow when used with large ad hoc network scenarios. E.g., to simulate 300 mobile nodes randomly moving over the area of $3 \times 3km^2$ with intensive random traffic connections, 10 seconds of simulated time require around 1000 seconds on a PC with a 1GHz Pentium-III and 512MB. So if we want to make 10 runs with the same scenario type and increase the simulation time to 600 seconds (at least that much time is required to explore the behavior of the network), the whole simulation will take more than a week. Such a performance is not very useful, especially when investigating protocols.

Since radio signals decay exponentially with increasing distance between transmitters and receivers, after some distance from the sender, no receiver is further affected by the transmitter's signal. This distance of course depends on the carrier sense threshold. Exploiting this fact we report on two ways to enhance NS-2 performance, both based on additional data structures to keep track of the nodes' position: the *Grid* and the *List*.

3.1.1 Grid-based node organization

We divide the simulation area into cells[1] to form a grid (Figure 4). The grid itself is a 3D array of pointers able to store, if necessary, *all* mobile nodes in the topography if they are all located in the same cell. This organization of the array allows each cell to know which mobile nodes are currently located in it and which not. Every time the position of a node is required to be known, the *update_position()* method of the class *MobileNode* is triggered[2]. In this method

[1]These "cells" have no relationship to the cells of the mobile phone net or of [9].

[2]Since this method is invoked every time the position needs to be known, even if the node has not moved at all or is still in the same cell, node mobility does not influence the cost to maintain this data structure. Further optimization is possible but would complicate a fair performance evaluation.

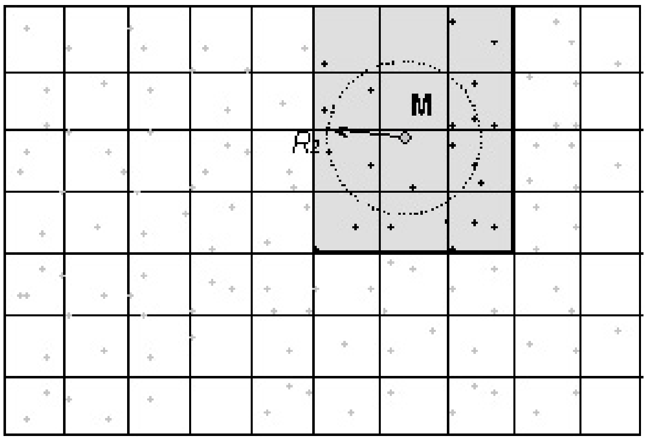


Figure 4: Simulation area divided into cells, with cells affected by M's transmissions in grey.

we integrate functions that calculate the position of a node in the grid to determine the cell a node belongs to.

We include in the method *Channel::sendUp()* a function responsible for getting the list of nodes affected by the current transmission. First the function determines the radius determined by CST and then selects those nodes that belong to the grid's cells covered by a circle with this radius. Subsequently, only nodes in these cells are involved in a computation, while the original NS-2 considers all nodes on a channel. See Figure 4 for an example, where node M is ready for transmission.

Depending on the size of the simulation area, the number of mobile nodes, and the maximum transmission power, the algorithm chooses the appropriate division of the topography into cells (the number of cells per X and Y side). The average chosen cell size is about $R_2/2$. With this cell size we can achieve the optimal performance of our simulator. If we have cells noticeably smaller than $R_2/2$ (see Figure 5) then a smaller number of mobile nodes resides in the affected cells but outside the area that is covered by the circle with radius R_2. On the over hand, the more cells we have, the more computation is involved to determine the set of nodes affected by the current transmission session, since for each cell and every mobile node on a network the algorithm must check whether a node belongs to a given cell or not.

If a "transmitting node" has originated a packet to transmit, our algorithm gets the position of this node and determines the CST radius depending on the transmission power. Then the algorithm calculates which cells are covered (entirely or partially) by the CST circle and gets the list of mobile nodes that reside in those cells. Next it traverses the list and calculates the signal propagation time between the current node (receiving node) from the list and the transmitting node. Then the time of the next event (start time of packet receiving process) and the type of this event (receive) are passed to the *Scheduler*, where the rest of event-processing takes place.

3.1.2 List-based node organization

Another data structure that can be used to improve NS-2 performance is a double-linked list of mobile nodes, ordered by their X-coordinates. We add new members to the *MobileNode* class, *MobileNode *prevX, *nextX* and a new static member *xListHead*, to keep track of nodes. When a new node is created its constructor simply appends the node to the list[3]. Once the coordinates are assigned, the list is sorted in ascending order based on the X-coordinate.

Every time a node moves its position, the list is updated. Once the *Channel::sendUp()* method is called, our algorithm determines the set of nodes that are affected by the current transmission. This set is formed by the nodes in the list with X-coordinates from $X - R_2$ to $X + R_2$ that have their Y-coordinates in the range $Y - R_2$ to $Y + R_2$, where $(X,\ Y)$ are the coordinates of the current (transmitting) node.

To get the most out of this improvement, a user must always set the X-side of a simulation area to be larger than the Y-side. This choice limits the number of nodes in the X-list that belong to $[X - R_2, X + R_2]$. This restriction can be trivially met by renaming the sides, if necessary.

3.1.3 Computational cost

These changes have the potential to save a significant amount of time if the simulation area is noticeably larger than R_2 and the number of randomly distributed mobile nodes is large enough. If the simulation are is less than $2 \cdot R_2$ then no benefit is gained.

In case of large areas, we use the following rule to divide the simulation area when the grid data structure is used. A is the size of the simulation area, R_2 is the radius corresponding to CST, l is the side of a cell we are trying to determine, (X, Y) are the coordinates of the transmitting node, T_0 is the CPU time to check if a node hears the transmission or not[4]. T_1 is the CPU time spent to check whether a node belongs to a certain cell and N is the total number of nodes involved in the simulation. From here we have:

- CPU time to check all nodes in the network whether they hear the current transmission: $T_{CPU_0} = T_0 \cdot N$;
- CPU time to determine the set of nodes located in the cells affected by the current transmission: $T_{CPU_1} = T_1 \cdot N \cdot (\frac{2 \cdot R_2}{l})^2$,
  where $2 \cdot R_2/l$ is the number of cells per side of the affected area;
- CPU time to check all nodes from the above set whether they hear the current transmission: $T_{CPU_2} = T_0 \cdot N \cdot \frac{(2 \cdot R_2)^2}{A}$,
  where $(2 \cdot R_2)^2$ is the size of the affected quadratic area.

The modifications to NS-2 are beneficial if $T_{CPU_0} > T_{CPU_1} + T_{CPU_2}$, or in total

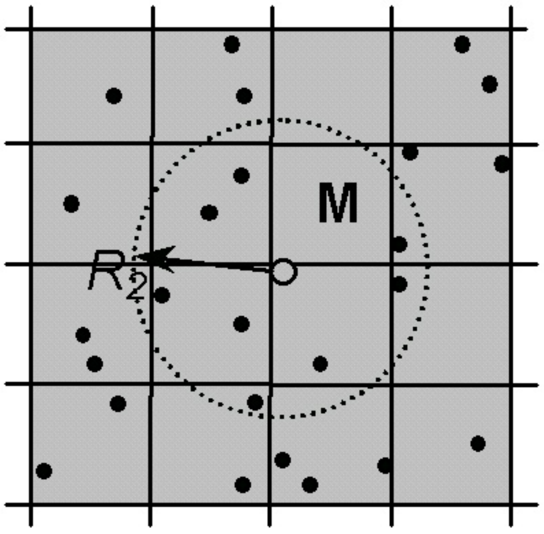
$$T_0 \cdot N > T_1 \cdot N \cdot (\frac{2 \cdot R_2}{l})^2 + T_0 \cdot N \cdot \frac{(2 \cdot R_2)^2}{A}$$

From here we find that the size of a cell's side must be:

$$l > \sqrt{k_1 \cdot \frac{4 \cdot R_2^2}{1 - \frac{4 \cdot R_2^2}{A}}}, \qquad (1)$$

[3]In the NS-2 simulator, the coordinates of the node in the simulation area are not yet known at this time.

[4]T_0 includes the time needed to find the distance between two nodes, to calculate the signal propagation delay, to put a receive-event in the calendar queue, to get the event out of the queue later, and to execute it to completion (i.e., one of receive packet, discard packet, or receive in error).



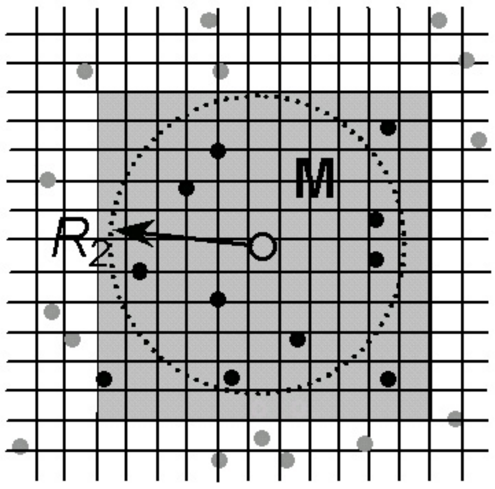


Figure 5: Tradeoffs for the size/number of cells: smaller cells contain on average fewer nodes beyond the transmission range but incur higher bookkeeping overhead since more cells must be checked.

where $k_1 < 1$ is the constant that represents the ratio T_1/T_0.

From this we conclude that when the simulation area A is less than $(2 \cdot R_2)^2$ this model cannot be used directly. In this case our algorithm sets the number of cells for the X and Y side equal to 1, and the overall performance does not change. The cost of maintaining the grid is minor and amounts to only around 2% to the total runtime of the original NS-2 simulator.

The list-based data structure can also successfully be used in almost all scenarios. More over, simulations that are based on this model run 5-60% faster than those based on the grid, as we discuss in Section 4. The cost of picking up the involved nodes from the list is lower than the cost of finding those nodes in the grid, and this difference accounts for the performance improvement. The cost of updating a node's position in the List is also low and is compatible with the same cost in the grid.

Let T_3 be the time needed to check if a node in the list belongs to the quadratic region affected by a transmission. Then the time to check all the nodes in the X-list that belong to the interval from $X - R_2$ to $X + R_2$ whether they also fall inside the region $Y - R_2$ to $Y + R_2$ is

$$T_{CPU_3} = T_3 \cdot \frac{2 \cdot R_2}{\sqrt{A}}$$

The time (T_{CPU_2}) to check the set of nodes that belong to the quadratic region whether they fall inside the circle with radios R_2 stays the same as in the grid.

Then the inequality looks like follows: $T_{CPU_0} > T_{CPU_3} + T_{CPU_2}$, i.e.,

$$T_0 \cdot N > T_3 \cdot \frac{2 \cdot R_2}{\sqrt{A}} + T_0 \cdot N \cdot \frac{(2 \cdot R_2)^2}{A}$$

From here we have the quadratic inequality over R_2:

$$\frac{4R_2^2}{A} + k_2 \cdot \frac{2}{\sqrt{A}} \cdot R_2 - 1 < 0$$

where $k_2 < 1$ is the constant that represents the ratio T_3/T_0.

This inequality can be resolved if

$$R_2 < \frac{\sqrt{\frac{k_2^2+4}{A}} - \frac{k_2}{\sqrt{A}}}{\frac{4}{A}}$$

or simply

$$R_2 < \frac{\sqrt{A}}{2} \qquad (2)$$

since $k_2 << 2$. So the only restriction for the use of the list data structure is that the side of a simulation area must be larger than $2 \cdot R_2$

There exist other cases when these changes do not improve performance, e.g., when all nodes are located in some small region of the simulation area, or when the density of nodes is very low so there is no interaction between the nodes. However, we speculate that the first situation does not occur frequently in a simulation, and really sparse ad hoc networks exhibit other problems.

3.2 Solving system problems

System idiosyncrasies cause another class of problems for the simulation of large ad hoc networks with long simulated times. Although these problems are "conceptually" not very interesting, they can impose hard limits on what the simulator can be used for. The bottleneck of the NS-2 simulator is a high dependence on the current Tcl/Tk release.

Tcl is used in NS-2 to describe scenarios, i.e., connection and movement patterns. The Tcl part of the simulator is also responsible for writing the data into the output trace file.

In the trace file, each entry describes one of the events types. Either a send, receive, drop, or forward operation is applied to a packet. Recording one entry may require up to 300B. With a large number of nodes and intensive traffic the output file can grow correspondingly. But the maximum file size that Tcl can handle is 2.1GB. This size can be used, e.g., to simulate for 50 seconds a highly dynamic ad hoc network with 800 nodes distributed over an area of $5km^2$. With 2000 nodes this time is even less – 10..20 seconds are sufficient to reach the file size limit. But certainly a longer time is required to obtain a realistic picture of a network's behavior.

This problem can be overcome if we remove file I/O from the Tcl/Tk part. We realized that by pipelining the simulator's output directly into a text (or archive) file. Then the only file size limitation that exists is determined by the operating and/or file system on the host machine – it's hard to get around this limit. Fortunately, modern OSs support big files and we have produced files with sizes $>$ 20+ GB.

Another Tcl-dependent problem has the same source - the absence of support for long integers. Each event (described above) in the simulator has its unique ID, represented by an unsigned integer, and this event ID is incremented for each event. 32 bits are not enough to simulate large populations of nodes with intensive traffic. In our experience, in the above mentioned scenarios, we run out of unique IDs within 70-150 seconds of simulated time.

This problem can be solved by changing the event-counter type to float. In Tcl, the float type uses twice the number of bits as integers. With this modification, we can simulate 2000 mobile nodes with intensive traffic moving around the area of $20km^2$ for 2400 seconds. Then the simulation exhausts the space on a state-of-the art disk drive. Although the changes are easy to describe, the actual problem is serious.

4. EXPERIMENTS

The goal of these experiments is to compare the performance of the original NS-2 with the performance of the modified simulator for large ad hoc wireless networks. All experiments are carried out on a commodity PC with a 1 GHz Pentium-III and 512MB RAM.

4.1 Grid-based setup

We use 100 different scenario files with varying movement patterns and traffic loads to run the DSR routing protocol against each of these scenario files on both versions of the simulator. We concentrate on the DSR wireless routing protocols since previous studies rated it to attractive ([1] compares the performance of the following routing protocol: DSDV, TORA, DSR and AODV).

Different movement patterns were created varying the simulation time, the number of nodes, the size of simulation area, the pause time, and the range of a node's speed. The *pause time* is the time that a node rests between two moves. Each node begins the simulation by waiting for *pause time* seconds. Then it selects a random destination within the simulation area and moves there with a speed randomly chosen from the range of permitted node speeds. When a node reaches its destination, the node pauses again for pause time seconds and then proceeds in the same way for the duration of the simulation.

We experimented with various numbers of TCP and CBR sources and two packet sizes (512 and 1024 bytes). All peer-to-peer connections are started at times uniformly distributed between 0 and the corresponding simulation time for a given movement pattern.

The results presented here were measured with the help of the DSR protocol. In principle, the efficiency of the modified NS-2 simulator must not depend on the routing protocol. To confirm this claim, we also performed several experiments with TORA, DSDV, and AODV. The results obtained in

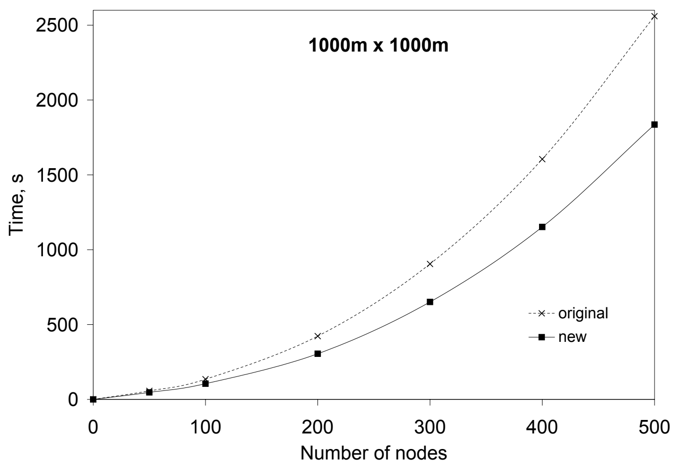


Figure 6: Runtime of original and modified (grid-based) simulator, area: $1km \times 1km$.

those experiments confirm that our extensions work independently of the routing protocol.

Figures 6, 7, and 8 show the performance of the original and the grid-based modified NS-2 simulator on different scenarios, and Figure 9 highlights the speedup of the modified simulator in comparison with the original one.

For each experiment we apply the same scenario (chosen from the set of pre-generated scenarios) for both simulator versions and then compare runtime and output trace files. In all cases we obtained absolutely identical network behavior (we observe byte-to-byte correspondence between any two output trace files generated from the same scenario setup). This result means the extensions do not introduce any errors into the final simulation results.

We can see from the figures that as the execution time for the original version of the NS-2 simulator grows exponentially, whereas the time of the modified simulator grows linearly for the range that is depicted here. The exception is the scenario with a simulation area of $1000 \times 1000m^2$. In this case, the CST induced radius is about $550m$ (at transmission power of 0.28 W). This radius is about half the side of the simulated area. Therefore it is difficult to realize a noticeable improvement when using the modified NS-2 simulator (since only nodes placed at opposite sides of the simulation area do not interfere with each other). As a result the original and modified simulators exhibit similar runtime behavior. The situation is different with areas of size $3 \times 3km^2$, $5 \times 5km^2$ and $10 \times 10km^2$. Here the modified NS-2 simulator works 4 to 20 times faster than the original. We notice a more pronounced improvement in execution time with bigger numbers of mobile nodes in larger simulation areas.

4.2 List-based setup

Figure 10 depicts the relative advantage of the list-based representation over the grid-based simulator. The list-based simulator realizes an additional performance benefit due to lower cost of identifying the set of nodes affected a transmission, as discussed in Section 3.1.3. We see that at low densities of nodes in each scenario, the difference between the grid-based setup and the list-based setup is not high,

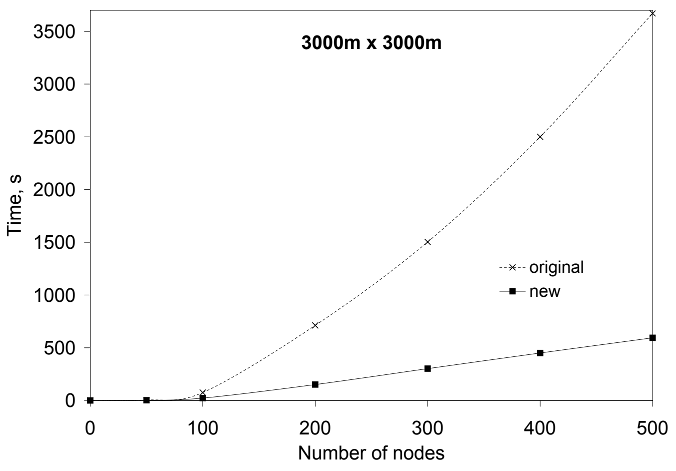


Figure 7: Runtime of original and modified (grid-based) simulator, area: $3km \times 3km$.

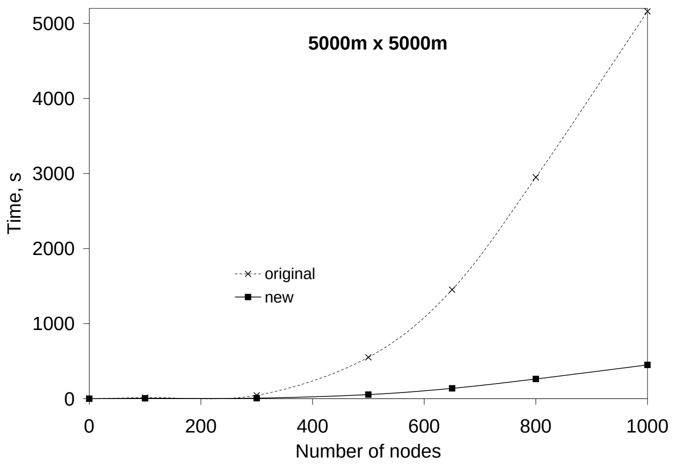


Figure 8: Runtime of original and modified (grid-based) simulator, area: $5km \times 5km$.

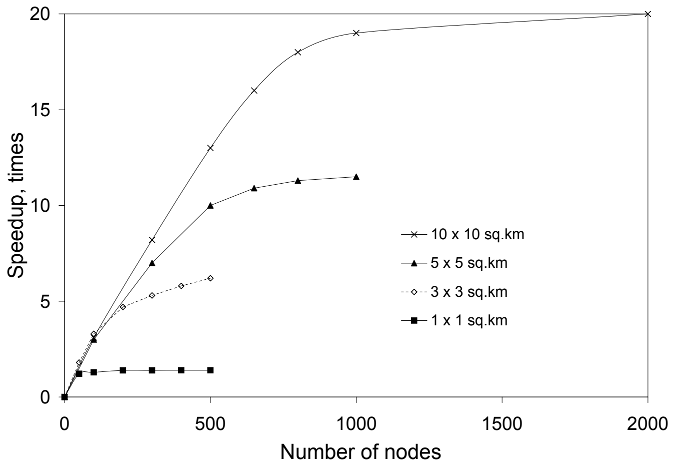


Figure 9: Speedup of the grid-based simulator relative to the original one.

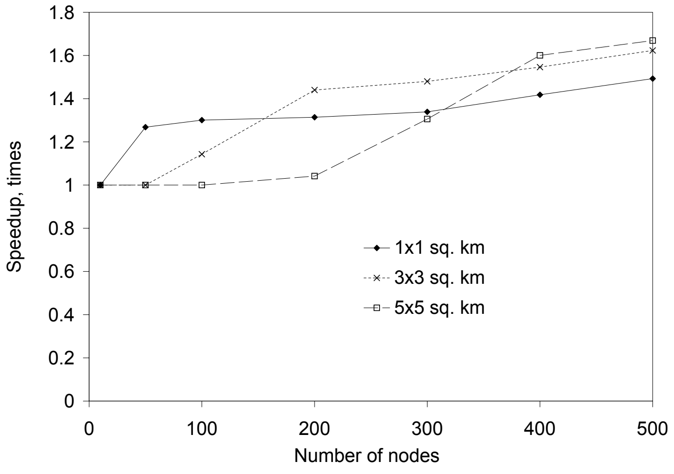


Figure 10: Speedup of the list-based setup compared to the grid-based one.

only 5-15% in favor of the list. However, as the number of nodes grows, the difference becomes more significant, reaching about 60% in the scenario with 500 nodes and an area of $5 \times 5km^2$.

5. VALIDATION

To validate the modified NS-2 simulator we repeated the experiments reported in [1]. We use the same setup as described in [1] and present results for the simulation of 50 wireless mobile nodes communicating with each other and moving around in a rectangular $1500m \times 300m$ flat space for 900 seconds of simulated time.

To create movement and communication scenarios we use the scripts mentioned above. We run our simulation with movement patterns generated for 7 pause times: 0, 30, 60, 120, 300, 600, and 900 s and generate scenario files with 70 different movement patterns, 10 for each value of pause time. The node speed ranges from 0 to 20 m/s.

As traffic sources we choose constant bit rate (CBR) sources with the sending rate set to 4 packets/s. We use three different communication patterns corresponding to 10, 20, and 30 sources. The packet size is 64 B. All communication patterns are peer-to-peer, and connections are started at random times chosen between 0 and 180 seconds. The 70 movement patterns taken in conjunction with the 3 communication patterns lead to a total of 210 different scenario files.

Figure 11 shows the percentage of data packets received (of all packets sent). The results obtained by the modified simulator ("new") are identical to the number reported in [1]. I.e, the output trace files have byte-to-byte correspondence. Since the list-based and the grid-based modifications of NS-2 always produce absolutely identical results, we include only a single figure.

6. MEMORY CONSUMPTION

The additional memory consumption of the modified NS-2 simulator is small in comparison with the size of memory occupied by the original system. The additional amount of memory needed for the modified algorithm to store the 3D

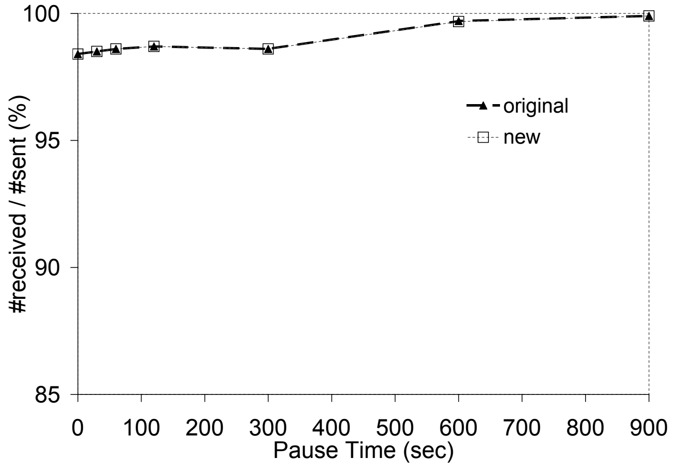


Figure 11: Percentage of data packets received (of packets sent) for DSR, original and modified simulator. 10 CBR sources.

array of pointers (the grid) is expressed by the formula:

$$PtrSize \cdot CellsPerX \cdot CellsPerY \cdot NumberOfNodes,$$

where $PtrSize$ is the size of the pointer type in bytes (4 for the current implementation), $CellsPerX$ and $CellsPerY$ are the number of cells for the X and Y side of the simulation area, and $NumberOfNodes$ is the total number of nodes.

Table 1 shows the amount of memory occupied by the original NS-2 simulator when 1/3 of the nodes act as traffic sources constantly sending data to some destinations[5].

If we set the number of cells for the X and Y sides to 20 and set the number of nodes to 500, the additional memory consumption is 800KB. It is less than 1% of the memory occupied by the original NS-2 with this number of nodes.

The additional memory consumption for the list-based setup is even smaller:

$$2 \cdot PtrSize \cdot NumberOfNodes,$$

since each node has only two additional fields, *MobileNode *prevX, *nextX.*

7. CONCLUDING REMARKS

The original NS-2 simulator shows poor performance for large ad hoc wireless networks. To alleviate this scaling problem, we base the computation of the interactions on the truncation algorithm, which exploits the real-life properties of signal propagation. Consequently, NS-2 performs much more effectively (up to 30 times faster) when simulating larger areas and/or larger numbers of mobile nodes than the original, unmodified simulator.

The validation shows that our modifications of the NS-2 simulator produce the same results as the original system and do not introduce any new errors. Of course, every simulator abstracts from the real system, and NS-2 delivers only an approximation of a real wireless network.

The price for this performance improvement is an increase in the amount of memory that is required. However, the additional memory consumption is negligible in comparison with the size of memory that is needed in any case. Given the memory size of today's workstations, this tradeoff in favor of a reduced execution time seems to be worthwhile.

The simulation of large scenarios is crucial for ad hoc networks. The benefits of self-organization appear attractive, but they must be demonstrated in large-scale settings. This modified simulator offers one path to investigate such scenarios [6]. Given the interest in ad hoc networks and the well-established role of NS-2 for network simulations, we hope the modified simulator enables interesting research in ad hoc networks.

[5]These measurements are carried out using the ns2.1b6 version of NS-2. The more recent version ns2.1b9a-gcc32 has noticeably higher memory consumption.

Number of nodes	Memory(MB)
10	8
50	13
100	23
300	73.1
500	138.7

Table 1: Memory allocated by original NS-2 simulator.

8. ACKNOWLEDGEMENT

We thank Cary Kornfeld for his suggestion to keep mobile nodes in a sorted list and for insightful comments on earlier drafts. Comments by the reviewers are appreciated.

9. REFERENCES

[1] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MOBICOM'98*, pages 85–97, 1998.

[2] S. Das, C. Perkins, and E. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *INFOCOM'2000 (1)*, pages 3–12, 2000.

[3] K. Fall. ns notes and documentation. *The VINT Project*, 2000.

[4] D. Johnson. Validation of wireless and mobile network models and simulation. In *DARPA/NIST Network Simulation Validation Workshop, Fairfax, Virginia, USA*, May 1999.

[5] D. Johnson, D. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, C. Perkins (ed), Chapter 5, pages 139–172, 2001.

[6] V. Park and S. Corson. The Temporally-Ordered Routing Protocol (TORA) Specification. draft-ietf-manet-tora-spec-00.txt, Internet Draft, IETF, Octorber 1999. (Work in progress).

[7] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[8] C. Perkins, E. Belding-Royer, and S. Das. Ad Hoc On Demand Distance Vector (AODV) Routing. draft-ietf-manet-aodv-10.txt, Internet Draft, IETF, March 2002. (Work in progress).

[9] L. Perrone and D. Nicol. Using n-body algorithms for interference computation in wireless cellular simulations. In *MASCOTS 2000 Intl. Workshop Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 49–56, 2000.

[6]Available for download from `www.lst.inf.ethz.ch`