# Performance Evaluation of nsclick Simulator for Mobile Ad Hoc Networks

Viet Thi Minh Do*, Lang Xie**, Øivind Kure*
*Q2S, the Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology, Norway
**Department of Telematics, Norwegian University of Science and Technology, Norway
**viet@q2s.ntnu.no, langxie@item.ntnu.no, okure@q2s.ntnu.no**

***Abstract*—In order to evaluate the behavior and performance of protocols for Mobile Ad Hoc Networks (MANETs) both simulation and test bed are often used. The simulation is used in first stages and the test bed is used in final stages of development process when real-world tests are needed. Click Modular Router is known as an efficient software architecture for building flexible and configurable routers for test bed. However, when moving from simulation to test bed, source code usually needs to be reimplemented. As a result, there must be a cost of maintaining two completely different code-bases. The tool *nsclick* was introduced to address this issue. It is constructed by embedding the Click Modular Router inside of the *NS-2* network simulator. The source code with *nsclick* may run both on actual systems as well as under simulator with minor modifications. In this paper, we provided an intensive performance comparison between *nsclick* and *NS-2* by carefully implementing a broadcast algorithm in both simulators. In addition, we designed and implemented an efficient jitter element in *nsclick*. Simulation results show that *nsclick* is a useful and effective tool for evaluating protocols in MANETs. The results also indicate that our jitter element can significantly reduce packet collision and thus improve performance of *nsclick*.**

***Keywords*—nsclick, Mobile Ad hoc Networks, performance evaluation, NS-2.**

## I. Introduction

A mobile ad hoc network is composed of mobile nodes connected without requiring any existing infrastructure or centralized administration. As a promising network type in future mobile application, Mobile Ad Hoc Networks (MANETs) are increasingly attracting researchers. Designing protocols for ad hoc networks is challengeable due to frequent change of network topology, unreliable wireless channel, and channel contention. Simulation has been widely used for protocol design in MANETs. It is more cost-effective to use simulation than real test bed. Simulation also can be used to detect and correct many bugs and issues before testing on a real system. Moreover, simulation is very useful for running large, repeatable scenarios in MANETs. However, it is shown that there are large divergences between simulators [1]. The differences are not only quantitative (not the same absolute value) but also qualitative (not the same general behavior). It is therefore necessary to run real experiments for testing wireless protocols in the final stages of developing process. Source code in simulation is often much different from implementation code in test bed. It is common that much effort need to be taken when moving from simulation to test bed. It is therefore of interest to develop one code base and run it both in simulators and test beds. *nsclick* [2] is one of tools which can perform this functionality. *nsclick* [2] is used as a testing tool for protocols in ad hoc networks in various works [3] [4] [5]. However, to the best of our knowledge, until now there are only two works [6] [7] focusing on evaluating performance of *nsclick* simulator. The work [6] evaluated end-to-end delay and simulation run-time in *nsclick* based on two AODV implementations. [7] compared the performance of *nsclick* with *NS-2* for the ad hoc routing protocol AODV (Ad hoc On Demand Distance Vector). The paper provided a comparison of goodput, latency, and memory used across three different AODV implementations. Our contributions in this paper are a comprehensive performance evaluation of *nsclick* in MANETs and a design of jitter element dealing with packet collision problem. In contrast to the works in [6] [7], this paper used a simple broadcast algorithm to evaluate the performance of *nsclick* and *NS-2*. The broadcast algorithm is likely to help to evaluate the performance more accurately due to the less impact of protocol implementation on performance. Performance of *nsclick* was evaluated based on Packet Delivery Ratio (PDR), packet overhead, simulation run-time, and collision ratio. In addition, we added a new jitter element which is intended to reduce packet collision in *nsclick*. [6] and [7] also implemented a jitter element for *nsclick* but their implementations have the problem of packet reordering. Moreover, these works did not provide the effect of jitter element on packet collision as we did in this paper.

The remainder of the paper is organized as follows. In Section II we overview three network tools: Click Modular Router, network simulator *NS-2*, and *nsclick*. Section III describes evaluation methodology used in this paper. The simulation setup and results are represented in Section IV. The paper is concluded in Section V.

## II. Overview of Tools

### A. Click Modular Router

Click Modular Router [8] is a software architecture for building flexible and configurable routers. Click routers are built from components called elements. To build a router configuration, users choose a collection of elements and connect them into a directed graph. Click uses a textual language

to specify how packets flow through the graph. To extend a configuration, users can write new elements or compose existing elements in new ways. The idea behind Click is to represent the packet flow through a network router as a series of packet manipulations executed by connected elements. Click is widely used to build real systems (e.g., deploying software routers and services) [9] [10] [11].

### B. *Network Simulator NS-2*

*NS-2*, an object-oriented, discrete event driven network simulator, is known as the most popular ad hoc network simulator [12]. It is widely used in academic research and has a lot of packages contributed by different groups. *NS-2* is composed of C++ code and OTcl scripts. C++ is used to implement the detailed protocol and OTcl is used for users to control the simulation scenario and schedule the events. For wireless network simulations, *NS-2* offers a number of features such as energy model, traffic and mobility models, and so forth.

### C. *nsclick*

The simulator *nsclick* [2] was introduced in 2002 as a bridge between simulation and reality. *nsclick* is an integration of network simulator *NS-2* and Click Modular Router. It is constructed by embedding the Click Modular Router inside of the *NS-2* network simulator. In *nsclick*, routing protocols are implemented as Click routing graphs and the routing protocols of *NS-2* is not used. The Click routing graph may run both on an actual system as well as under *NS-2* with minor modifications.

## III. Evaluation Methodology

In order to evaluate performance we implemented a simple broadcast algorithm in both simulators *nsclick* and *NS-2*. If forwarding nodes are not carefully designated, the broadcast operation is prone to broadcast storm problem [13]. An effective approach to this problem is to use jitter. This section describes the broadcast algorithm and our implementation of broadcast and jitter elements in *nsclick* and *NS-2*.

### A. *Broadcast Algorithm*

There are two reasons to implement broadcast algorithm to evaluate performance of *nsclick*. First, broadcast operation is a fundamental service in MANETs. It is a basic building block for various network protocols especially for routing protocols. This operation is frequently used to spread information to the whole network. The second reason is that the evaluation of simulators might be more accurate due to the fact that with simple algorithm the simulation results do not depend much on the implementation of protocols when the implementation is simple.

Our simple broadcast algorithm can be considered as a flood algorithm. Source node floods packets to all neighbor nodes. When a neighbor node receives a packet, if the node have not seen the packet before it will process the packet and reflood the packet to its neighbor nodes; otherwise, the packet is discarded. The process continues until all nodes in the network receive the packet. The flooding algorithm was implemented on top of MAC layer. There is no transport protocol (e.g. UDP, TCP) in our implementation. Application and MAC layer connect directly to the broadcast algorithm. All broadcast packets are sent to the broadcast MAC address. Figure 1 depicts broadcast architecture on *nsclick* and *NS-2*. In *nsclick*, application and MAC physical of core *NS-2* were used while broadcast algorithm was implemented in Click *FloodClassifier* element. In this paper, the term "core *NS-2*" determines the *NS-2* integrated in *nsclick*. In *NS-2* the broadcast algorithm was implemented by a routing agent.

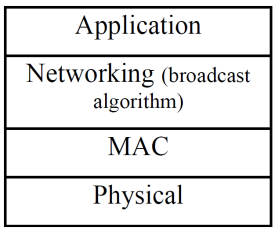


Fig. 1. Protocol stack of broadcast algorithm

### B. *Broadcast and Jitter Implementation*

In our implementation, a node does not immediately re-broadcast the packet it receives for the first time, instead the packet is delayed in its buffer for a random period called jitter.

As mentioned in introduction section, the previous works, which implemented jitter element in *nsclick* (e.g. in [6] [7]), have the problem of packet reordering. In these works, a timer is created to pair to an arriving packet, the firing time of the timer is randomly. As soon as the timer fires, the corresponding packet is sent to all neighbor nodes. Consequently, the order of packet is changed due to the random firing time of the timer. This problem might be more serious in the situation where the order of packet is important (e.g. when the transport protocol TCP is used). We solved this problem in our jitter implementation. In our implementation, when a node receives a packet for the first time, it stores the packet in a table and creates a timer with a random firing time. Unlike other works, the created timer does not pair to the arriving packet. The packet table is an array structure in which packets are stored as the order of their arriving time. As soon as a timer fires, the packet which has the earliest arriving time is sent to all neighbor nodes. The earlier a packet arrives the earlier it is sent, then the problem of packet reordering is avoided.

In *nsclick*, in order to implement broadcast algorithm and jitter we added two new elements into Click, *FloodClassifier* and *JitterFlood*. *FloodClassifier* element has one input port and two output ports. This element uses a table to store recent packets to detect the duplication of packets. When a packet arrives the element, if the element has not seen the packet before, the packet is stored in the table as well as pushed to the first output port; otherwise, the packet is pushed to the second output port. Each packet is removed from the

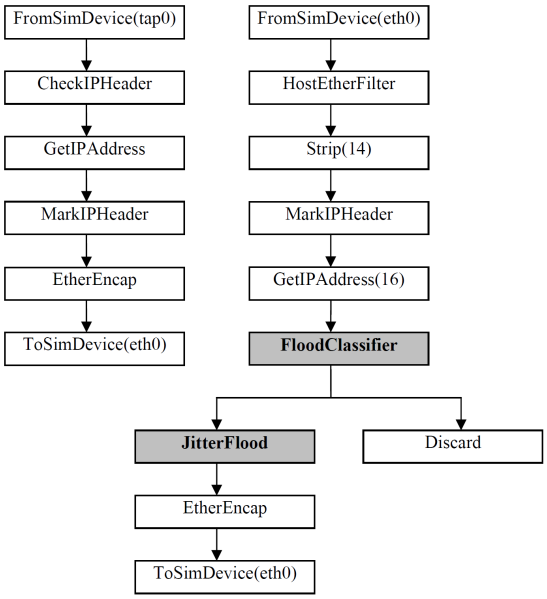


Fig. 2. a)(left) b)(right) Click routing graph for broadcast router configuration

table after five seconds by a timer. *JitterFlood* element has one input port and one output port. When receives a packet, this element stores the packet and its arriving time in a table and creates a timer. The firing time of the timer is randomly chosen from (0, JITTER_MAX] in uniformly way. In this paper, JITTER_MAX is set to 10 mseconds. As soon as a timer fires, the earliest arriving packet in the table will be pushed to the output port.

*nsclick* routers use elements *FromSimDevice* and *ToSimDevice* to receive and send packets. The simulated device transmits "raw packets" which will be transmitted using the core *NS-2* physical layer models. In this paper, *nsclick* use 802.11 MAC layer and power model provided by core *NS-2*. Figure 2 presents the Click routing graph for broadcast router configuration in our *nsclick* implementation. The new elements are presented as brown blocks in Figure 2. Packets can reach Click routing from simulator network interface (eth0) or from application (tap0). Figure 2a) and Figure 2b) depict the routing graph for packets from application and network interface, respectively. After being read by *FromSimDevice(tap0)* element, packets are pushed to *CheckIPHeader* element to check their validity. Then *GetIPAddress* element sets destination IP address annotation from packet data. IP header annotation is set by *MarkIPHeader* element. After that *EtherEncap* element encapsulates packets in Ethernet header. Finally, packets are sent to simulated network device by *ToSimDevice(eth0)* element. Packets through *FromSimDevice(eth0)* are pushed to *HostEtherFilter* element. Ethernet packets sent to other machines are dropped by this element. This element expects Ethernet packets as input and acts basically like Ethernet input hardware for a device with address as argument of this element. *Strip* element is used to get rid of the Ethernet header. IP header annotation and destination IP address annotation are set by *MarkIPHeader* and *GetIPAddress* elements. Then the packets are classified by *FloodClassifier* element. If the packet have not been seen before, *FloodClassifier* sends the packet to *JitterFlood* element; otherwise the packet is pushed to and then discarded by Discard element. *JitterFlood* element delays the packet for a randomly time jitter before sending it to *EtherEncap* element. The packet is encapsulated in Ethernet header before being pushed to simulated network device by *ToSimeDevice* element. Implementing jitter in *NS-2* is quite simple. It was implemented in operations of the broadcast routing agent.

## IV. Performance Evaluation

We run simulations and implemented the same simulation set-up in both simulators *nslick* and *NS-2*. All simulations run on an Intel Pentium D workstation with 2GB of RAM, CPU 3.00GHz, Ubuntu Linux SMP 9.04. Our measurements were taken using NS-2.34 and *nsclick* with Click-1.8.0. All results provided were averages over five executions of each simulation scenario. We used the confidence level of 0.95 and dropped data from the initial of the simulation.

In all simulations 50 nodes are placed randomly in the area of 1000m x 1000m. The transmission range of each node is 200m. Network is 802.11 with the rate of 2 Mbps. Mobility of nodes follows the Random Waypoint model with pause time 100 seconds. Each node randomly selects moving direction and move there at a random speed uniformly chosen from $(0, V_{max}]$, where $V_{max}$ is the maximum speed of the node. Upon reaching the destination the node stays there for some pause time. Upon expiration of the pause time, the next destination and speed are again chosen in the same way and the process repeats until the simulation ends. Each sender broadcasts total 100 packets (512 bytes/packet) with different rates during the simulation. To avoid synchronized transmitting, each sender starts to transmit packets at different times. They broadcast packets continuously until 100 packets are sent. We used constant traffic rate CBR as traffic type in all simulations. The value of jitter is randomly generated from (0, 10 mseconds] in uniformly way. Simulation time was set to 300 seconds except where the simulated time was described explicitly.

We considered four performance metrics to evaluate performance of *nsclick* and *NS-2* as follows.

- *Packet delivery ratio (PDR)* is computed as the ratio of the total number of packets received by receivers to the product of the total number of packets transmitted from sources and the number of receivers.
- *Packet overhead* is measured as the total number of duplicated packets received by all nodes in network.
- *Simulation run-time* is defined as the computation time for a simulation.
- *Collision ratio* is computed as the ratio of the number of collision packets to the total number of packets sent by all senders multiply the number of network nodes. In other words, *collision ratio = # collision packets/(# sent packets * 50)*

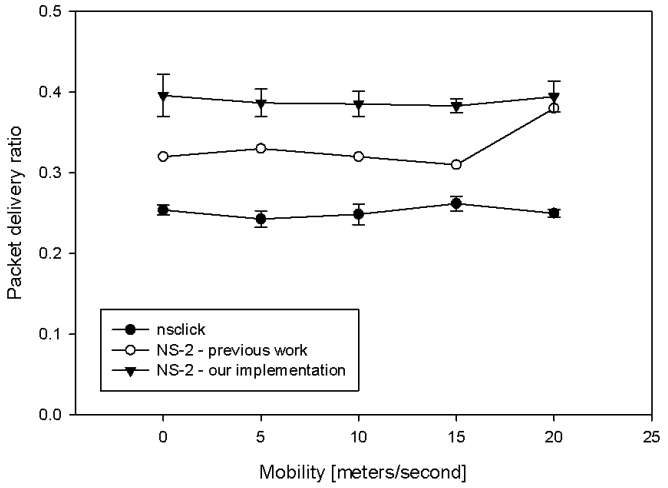


Fig. 3. Packet delivery ratio

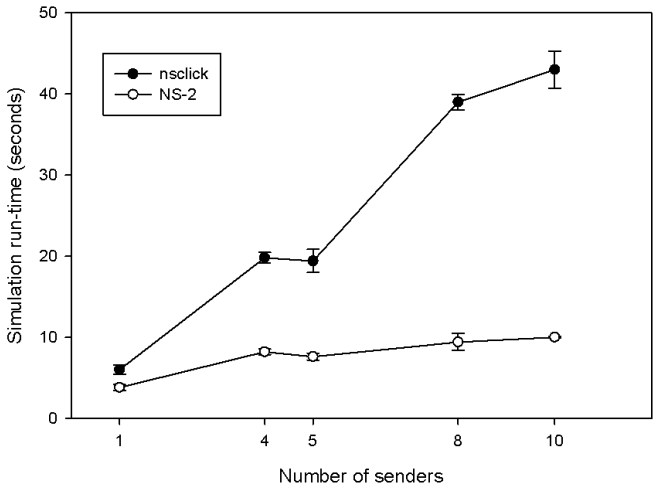


Fig. 5. Simulation run-time vs. number of senders

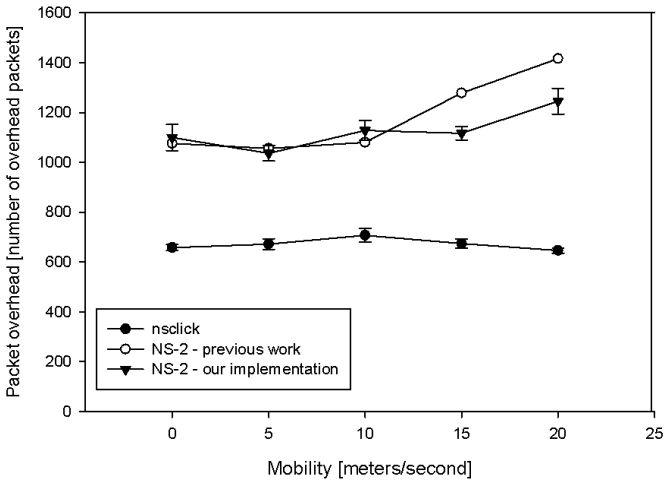


Fig. 4. Number of overhead packets

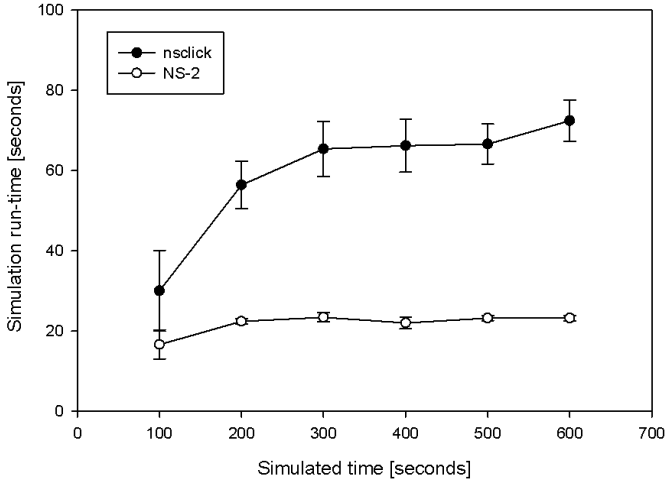


Fig. 6. Simulation run-time vs. simulated time

### A. *Packet Delivery Ratio and Packet Overhead*

We conducted experiments to compare PDR and packet overhead between *nsclick* and *NS-2*. Other purpose of these experiments is to confirm the correctness of our implementation by comparing our results with results of previous work in [1]. We set the explicit parameters in *NS-2* and *nsclick* in this run series as the same in the previous work [1]. In other words, 10 senders broadcast packets with the rate of 4 packets/second (the packet size is 512 bytes so the source traffic rate is 32 Kbits/second). We extracted PDR and packet overhead of *NS-2* from the previous work [1] to make the comparison more clearly. Figure 3 and Figure 4 depict PDR and packet overhead of broadcast algorithm in both *nsclick* and *NS-2*, respectively. The first thing to notice is that there is just small difference between PDR as well as packet overhead of our implementation and the previous work. This is likely due to the difference of some implicit simulation parameters. However, the trend and the value of PDR and packet overhead of *NS-2* in our implementation are somehow similar to the previous work. In addition to carefully debugging and implementing, this similarity is contributed to validate our implementation. The second thing to notice is that PDR of *nsclick* is a bit lower than that of *NS-2*. The reason is that the size of packets in *nsclick* is larger than packet size in *NS-2* (*nsclick* has to append Ethernet header to all data packets). Packets in *nsclick* consume more bandwidth and packet loss ratio increases in *nsclick*. As a result, PDR of nsclik is lower than that of *NS-2*. The overhead of *NS-2* is slightly higher than that of *nsclick*. This difference is due to the difference of time management

in *NS-2* and *nsclick*. However, the results indicate that *nsclick* performs quite reasonably when compared to *NS-2*.

### *B. Simulation Run-time*

In this section, we run three experiment series to evaluate simulation run-time. This section aims to investigate the effect of parameters (number of senders, simulated time, network traffic rate) on simulation time in *nsclick* and *NS-2*. Network traffic rate is defined as accumulation of source traffic rates of all senders in the network.

Firstly, we performed experiments to test the effect of the number of senders on simulation run-time. The number of senders is set to 1, 4, 5, 8, 10 senders. Results are illustrated in Figure 5. We can see that the simulation run-time of *nsclick* is much higher than that of *NS-2*, especially when the number of senders is large. The reason is that in *nsclick* there is a lot of system calls inside Click to *NS-2* for the current time and this slows down the entire system (Click normally uses system calls to determine the current time and transfer packets to and from Ethernet devices and the kernel). As can be seen from the results, the simulation run-time increases when the number of senders rises up in both *nsclick* and *NS-2*. The increase speed of *NS-2* by mobility is lower than that of *nsclick*. For example, the simulation run-time in both *NS-2* and *nsclick* with one sender is about 6 seconds; with 10 senders, simulation run-time in *NS-2* is 10 seconds, in *nsclick* is 43 seconds. Nevertheless, the simulation run-time in *nsclick* increases in a linear way with the number of senders.

Secondly, we performed experiments in *nsclick* and *NS-2* with different simulated times from 100 seconds to 600 seconds. The number of senders is fixed at 10 senders, maximum speed of nodes is set to 20 m/s and source traffic rate is fixed at 32 Kbits/second. Figure 6 represents simulation run-time in *nsclick* and *NS-2* with various simulated times. We can see that the simulation run-time of *NS-2* remains stable while the simulation run-time of *nsclick* increases when the simulated time grows up. Also, the simulation run-time of *nsclick* increases in a linear way with the simulated time.

TABLE I
SIMULATION RUN-TIME IN NSCLICK FOR DIFFERENT NUMBER OF SENDERS AND DIFFERENT SOURCE TRAFFIC RATES

| # senders | 16 Kbps | 8 Kbps | 4 Kbps |
|---|---|---|---|
| 2 | 11 | 15 | 31 |
| 4 | 19 | 16 | 6 |
| 5 | 47 | 21 | 15 |
| 10 | 34 | 24 | 17 |

Finally, we run experiments only in *nsclick* with different number of senders and different source traffic rates. Table I presents the simulation run-time of *nsclick* with different number of senders and different source traffic rates. As seen from Table I the simulation run-time of *nsclick* is different in scenarios which have the same network traffic rate. For example, the simulation run-time of *nsclick* in scenario with 5 senders and 16 Kbps source traffic rate is 47 seconds while the simulation run-time in scenario with 10 senders and 8 Kbps is 24 seconds (the network traffic rate in this situation is 80 Kbps). Another thing to notice is that if the network traffic rate increases, the simulation run-time can increases or decreases. The scenario with 2 senders and 4 Kbit/s source traffic rate, the scenario with 10 senders and 4 Kbit/s source traffic rate, the scenario with 5 senders and 16 Kbit/s source traffic rate are examples. Therefore, it could be included that the network traffic rate does not directly affect on the simulation run-time of *nsclick*.

Briefly, the number of senders and simulated time impact on simulation run-time of *nsclick* while the network traffic rate does not. The simulation run-time of *nsclick* increases in a linear way with the number of senders as well as the simulated time. In other words, it is possible to run *nsclick* with the large scale simulation.

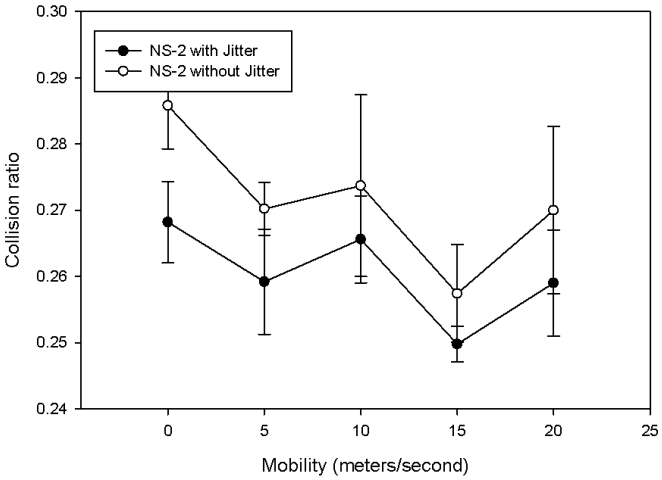


Fig. 7. Collision ratio with and without jitter in NS2

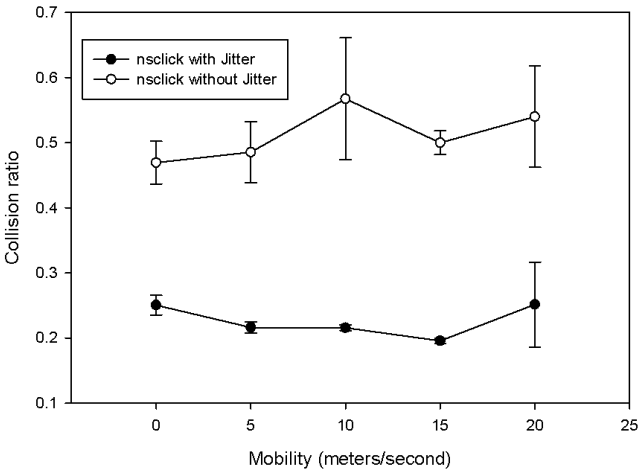


Fig. 8. Collision ratio with and without jitter in nsclick

### C. *Collision ratio*

In this section, we run two experiment series to evaluate the effect of our jitter element, broadcast jitter was ignored in the first series while it was included in the second series. Simulations were performed in both *nsclick* and *NS-2*. Collision ratio in implementation with and without jitter are illustrated in Figure 7 and Figure 8. As can be seen from Figure 7, collision happens more frequently in *NS-2* when broadcast jitter is not used. In Figure 8 the collision is much serious in *nsclick* when the broadcast jitter is ignored. The results show that jitter implementation is more important in *nsclick* than in *NS-2*. In *NS-2* jitter helps to reduce collision caused by the broadcast storm problem [13]. In *nsclick* jitter not only deals with the broadcast storm problem, but also helps to reduce collision caused by integration of simulator *NS-2* and Click of *nsclick*. In Click the time management is performed using the *gettimeofday* system call. Routers in Click are usually distributed, so the returned time is unique to each computer and the time for different computers may not the same. In other hand, in *NS-2* the time between different nodes is identical because it is a discrete event simulator. *nsclick* is integration of *NS-2* and Click, it uses simulation time as in *NS-2*. In *nsclick*, the system calls is performed inside *NS-2* and *NS-2* feed simulation time to *nsclick*. The time between different nodes in *nsclick* therefore is identical. The identical time leads to collision in transmitting data of neighbor nodes. Jitter in our broadcast implementation in *nsclick* helps to deal with this collision type.

## V. Conclusion

This paper presents an intensive performance evaluation of *nsclick* in mobile ad hoc networks and a design for an effective jitter element in *nsclick*. The simulation results show that it is possible to use *nsclick* with large scale simulations and that it is important to implement a jitter in *nsclick* to reduce collision. *nsclick* is an effective tool to evaluate and test new protocols in MANETs. It is a good tool for researchers who want to test new protocols in both simulation and test bed. In near future, we intend to move the code from *nsclick* to a real test bed system in order to compare the performance of *nsclick* and Click.

## References


[1] 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*. New York, NY, USA: ACM, 2002, pp. 38–43.

[2] M. Neufeld, A. Jain, and D. Grunwald, "Nsclick:: bridging network simulation and deployment," in *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*. ACM Press, 2002, pp. 74–81.

[3] M. Voorhaen, E. Van de Velde, and C. Blondia, "Morhe: A transparent multi-level routing scheme for ad hoc networks," vol. 197, pp. 139–148, 2006.

[4] M. Voorhaen and C. Blondia, "Analyzing the impact of neighbor sensing on the performance of the olsr protocol," in *Proceedings of 4th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt06)*, APRIL 2006.

[5] A. Jain, M. Gruteser, M. Neufeld, and D. Grunwald, "Benefits of packet aggregation in ad-hoc wireless network," Tech. Rep., 2003.

[6] B. Braem, "Implementation and evaluation ad hoc on-demand distance vector routing," 2005.

[7] M. Neufeld, A. Jain, and D. Grunwald, "Network protocol development with nsclick," *Wirel. Netw.*, vol. 10, no. 5, pp. 569–581, 2004.

[8] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, 1999.

[9] S. Doshi, S. Bhandare, and T. X. Brown, "An on-demand minimum energy routing protocol for a wireless ad hoc network," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 3, pp. 50–66, 2002.

[10] C. Kim, "Floodless in seattle: A scalable ethernet architecture for large enterprises," in *SIGCOMM*, 2008.

[11] R. Chertov, S. Fahmy, and N. B. Shroff, "A device-independent router model," in *INFOCOM 2008*, 2008.

[12] S. Kurkowski, T. Camp, and M. Colagrosso, "Manet simulation studies: The incredibles," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, pp. 50–61, 2005.

[13] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999, pp. 151–162.