Enabling cross layer design: adding the MadWifi extensions to Nsclick

Nicolas Letor
PATS research group
University of Antwerp - IBBT
Middelheimlaan 1
B-2020 Antwerpen, Belgium
Nicolas.Letor@ua.ac.be

Peter De Cleyn
PATS research group
University of Antwerp - IBBT
Middelheimlaan 1
B-2020 Antwerpen, Belgium
Peter.DeCleyn@ua.ac.be

Chris Blondia
PATS research group
University of Antwerp - IBBT
Middelheimlaan 1
B-2020 Antwerpen, Belgium
Chris.Blondia@ua.ac.be

ABSTRACT

Simulations have always been a very useful and powerful tool to study and develop networking protocols. *ns-2* is one of the most commonly used open-source network simulators and many protocols have been evaluated with this tool. In order to bridge the gap between simulation and implementation on a real testbed, *Nsclick* was developed. *Nsclick* makes it possible to entirely reuse code between the *ns-2* simulator and a testbed, based on the Click Modular Router. In the quest for high performance wireless networks, researchers have found that the layered architecture networks did not perform well enough. Therefore they focused their research on cross layer design techniques. Although using Click and the MadWifi driver gives us many possibilities to design and evaluate cross layer optimizations in a testbed set up, the *Nsclick* platform does not support these extensions.

In this paper we will introduce an extension to *ns-2* which overcomes these shortcomings of the *Nsclick* platform. This extension makes the wireless features of the Click Modular Router available in Nsclick platform, eliminating the differences between the *ns-2* simulator, within the Nsclick platform, and a real world set up.

Keywords

NS-2, Click Modular Router, Nsclick, Simulation, IEEE 802.11, MadWiFi, Cross layer design

1. INTRODUCTION

The route to develop and evaluate network and routing protocols often has two approaches: via simulation or via implementation on a testbed. However nowadays, as wireless networks and mobility become more and more important, it is becoming more difficult to build mobility aware testbeds. With the increasing popularity of ad hoc networks, we not only have to cope with wireless and mobile nodes issues, but also with larger scaling issues, because wireless ad hoc networks have the tendency to extend beyond a few nodes. Of course, routing protocols can still be tested and evaluated on a wired testbed, but if cross layer designs are desired, then a wired testbed is no longer an option.

NSTools '07, October 22, 2007, Nantes, France

A simulator then seems the only way to go in order to quickly develop and debug cross layer designed ad hoc network protocols. Although offering a lot of functionality, a simulator is always limited in the use of networked applications and cannot be used in a live demonstration, which many research projects nowadays require as a proof of concept. The use of a simulator to design and evaluate a protocol, implies that all the implementation efforts will need to be redone, when building a prototype or testbed set up, as code written for the simulator will typically not be compatible with a real life system.

Nsclick [9] offers an efficient alternative to overcome these incompatibilities. This simulation environment combines the well known and widely used *ns-2*[3] simulator and the Click Modular Router [8]. The use of the Click platform makes it possible to reuse code between the simulator and the testbed hardware.

The Click Modular Router is even capable of using lower layer IEEE 802.11 characteristics using specific drivers. Unfortunately, these functionalities are not available within the *Nsclick* framework which makes it impossible to do simulations using algorithms that heavily depend on these lower layer properties. For this reason, we extended the *Nsclick* architecture to bring cross layer facilities, already available in Click, to the *Nsclick* framework.

This paper is further organized as follows: in the next section, we will introduce in more detail the architecture of the simulation platform. We then continue with an explanation of the changes made to some of its components. In section 4 we present two case studies in which the usability of the extended simulator is shown and we conclude this paper in section 5.

2. DESIGN

In this section we present the different parts that make up the *Nsclick* simulation framework.

2.1 Click Modular Router

The Click Modular Router is a software architecture for constructing and deploying fast, configurable, and flexible packet routers. It consists of two parts: a router configuration and a platform-specific layer that handles the details of running the router configuration on a particular platform.

A router configuration is a graph with packet processing modules called elements at the vertices. The individual elements implement simple router functions like packet classification, queueing, scheduling, and interfacing with network devices. The packets flow along the edges of the graph.

The Click graph of a very simple example configuration of an ethernet bridge[10] is shown in Figure 1. The Click script corresponding to Figure 1 is:

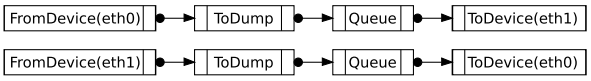


Figure 1: Example configuration

```
FromDevice(eth0, PROMISC true)                  1
       -> ToDump(eth0.dump)                     2
       -> Queue                                 3
       -> ToDevice(eth1);                       4
FromDevice(eth1, PROMISC true)                  5
       -> ToDump(eth1.dump)                     6
       -> Queue                                 7
       -> ToDevice(eth0);                       8
```

In the first line the `FromDevice` element takes packets from the eth0 Ethernet device and pushes them to the `ToDump` element. The `ToDump` element saves the content of the packets in the eth0.dump file, before pushing the packets to the `Queue` element. The `Queue` holds packets until the `ToDevice` element is ready to transmit them on interface eth1. The same happens in lines 5 to 8, but in the opposite direction of this simple Ethernet bridge.

When using a wireless network card, the Click Modular Router can only receive and send Ethernet frames. The driver of the wireless network card presents itself to the kernel (and to Click) as a standard Ethernet device. This driver performs the translation from Ethernet to 802.11 frames and vice versa. As a consequence the kernel, Click and any other part in the networking stack cannot differentiate packets received from a wireless interface from those received from a wired interface.

To overcome this limitation a special Click Wifi Driver[1], known as the Madwifi Stripped Driver, was initially developed. The Click Wifi Driver was a stripped down version of the Multiband Atheros Driver[2] which contains only the very basics of the 802.11 MAC protocol and the RTS/CTS mechanism. This Stripped Madwifi Driver does not perform any encapsulation or de-capsulation, instead it only accepts 802.11 frames to transmit and passes on the received 802.11 frames to the Click Modular Router. Currently, this functionality is incorporated in and directly available from the MadWifi driver, when using the wireless interface in monitor mode.

```
struct click_wifi_extra {
  u_int32_t magic;
  u_int32_t flags;

  u_int8_t rssi;
  u_int8_t silence;
  u_int8_t power;
  u_int8_t pad;

  u_int8_t rate;
  u_int8_t rate1;
  u_int8_t rate2;
  u_int8_t rate3;

  u_int8_t max_retries;
  u_int8_t max_retries1;
  u_int8_t max_retries2;
  u_int8_t max_retries3;

  u_int8_t virt_col;
  u_int8_t retries;
  u_int16_t len;
};
```

Figure 2: Click Wifi annotations

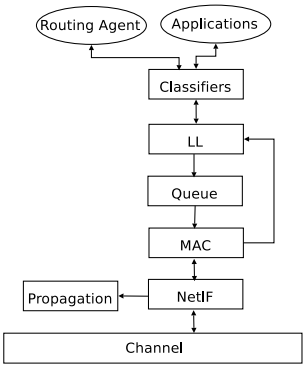


Figure 3: Mobile Node Interface

Besides the basic functionality of transmitting and receiving frames, the MadWifi Driver, in combination with the Click framework, provides some extra functionality. The Click router possesses an interface along which the transmission parameters can be communicated to the driver on a per packet basis. The router does so by extending every packet with a Click Wifi Extra header(figure 2). This header communicates the following information to the driver on a per packet based way:

- the transmission rate,
- the maximum number of retries,
- and if the RTS/CTS mechanism should to be used.

Once a frame is transmitted, the driver will report to Click about the actual transmission. The driver indicates in the same Click Wifi Extra header if the transmission was successful and how many retries were needed, before returning the transmitted packet to the Click router for inspection.

At the receiver side, an incoming frame will be marked with the received signal strength indicator (RSSI) to inform higher layers on the reception quality, again on a per packet basis.

2.2 ns-2

The *ns-2* simulator is a discrete event simulation framework, designed to evaluate transport protocols, networking routing protocols and multicast protocols over wired and wireless networks. *ns-2* simulations consist of nodes, links and events. These nodes represent communication stations and routers in the network, which are connected by links.

In figure 3, the architecture of a (mobile) node is shown. At the bottom, the Channel object represents the physical broadcast medium to which the nodes are connected. Every node that is connected to the same Channel object, is connected to the same LAN. This medium can be either a wireless or a wired medium.

The next item in the communication stack is the network interface (NetIF), representing the physical network interface connected to a Channel object. In a wireless network, the physical network interface is responsible for determining if a transmitted frame can be heard. Every interface connected to the same channel object will receive the frame. The associated propagation model computes the received signal strength, based on the position of the sender and the receiver. If the received signal strength is greater than a certain threshold, the receiving interface will accept the frame and pass it on up the stack, shown in Figure 3.

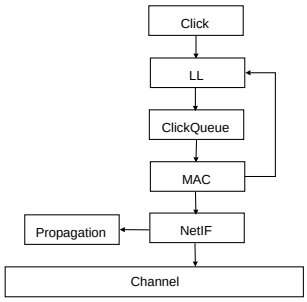


Figure 4: Nsclick Ethernet Interface

A MAC layer is connected on top of the network interface. *ns-2* implements, amongst others, an IEEE 802.3 and 802.11 MAC layer. The implementation of the *ns-2* 802.11 MAC is a 802.11 DCF with RTS/CTS mechanism and its functionality is comparable to the Madwifi driver support of the Click Modular Router, providing only basic transmission functions without, e.g. the management functions for infrastructure based networks.

To finish up, the Link Layer (LL) provides the fragmentation and reassembly of frames while Classifiers are responsible for delivering the frames to the appropriate applications or the routing agents.

2.3 Nsclick

Nsclick is an integration of the Click Modular Router with the *ns-2* network simulator. The purpose of *Nsclick* is to run the same Click router configuration[1] both on an real system, as well as in combination with *ns-2* to facilitate testing, debugging and evaluation of network code. The simulator is very useful for running large and repeatable scenarios. Building a large testbed is expensive and repeatability is hard to achieve in a testbed. Another major advantage is that only one code-base has to be maintained.

The link between the *ns-2* simulator and the Click Modular Router is made in the Classifier. Click presents itself to *ns-2* as a special Classifier, as shown in Figure 4. This Classifier intercepts all packets and translates them from a *ns-2* packet format to Click raw data format before sending them through the Click router configuration. When the packets are processed, the Classifier translates them back from Click raw data format to the *ns-2* packet format and sends them down the stack to the *ns-2* MAC layer, except packets destined for the local node. These packets are delivered to a `ToHost` element, which delivers them to the corresponding applications. Packets originating from the local applications are handed over to a `FromHost`, which inserts them in to the Click Modular Router.

Because *Nsclick* uses the *ns-2* simulator as a platform for the Click Modular Router, it is not possible to use the available Click WiFi elements which interface with the lower MAC functions by using the earlier described headers. As such it is not possible to control the lower MAC functions or receive feedback information as would be possible in a testbed set up with the MadWifi drivers. It is however this type of information we would like to use in cross layer optimization schemes and which we have integrated in the *Nsclick* framework.

[1]Some very minor modifications have to be made to the Click router configuration.

3. MADWIFI STRIPPED IN NSCLICK

The objective of this *Nsclick* extension is to support the Click Wifi elements in *Nsclick*, which control the Madwifi driver. These elements communicate with the driver through the Click Wifi annotations of a packet, shown in Figure 2. Before sending the packet, these Wifi annotations are converted in a click_wifi_extra header and placed in front of the packet. The driver decodes this header and configures itself with the indicated options.

The *ns-2* 802.11 MAC was adapted to behave like the Madwifi driver. The modified MAC interprets the click_wifi_extra header in front of the packet, which is used to set the transmission rate, retry count and usage of the RTS/CTS mechanism.

Besides controlling the behavior of the driver, the wifi elements also provide a feedback mechanism to monitor the state of transmitted packets. Packets delivered to the driver are handed over again to Click after successful transmission or after failure of such event. Routing algorithms in Click can thus benefit from link layer information indicating success or failure of transmission and the retransmission count to get a packet delivered.

Furthermore, we modified the Network Interface (netIF) to account for the different rates at which packets can be sent. The higher the transmission rate is, the stronger the received signal has to be for the receiving hardware to be able to decode it correctly.

For each rate a signal strength threshold is defined at which the packet can successfully be decoded and received. If the packet can be received, a received signal strength indication (RSSI) is saved in the header of the packet, before the packet is sent up to Click (Figure 4). An example configuration with four rates and the corresponding thresholds is given :

```
# set multi-rate PHY parameters
Phy/WirelessPhy set RateCount_ 4
Phy/WirelessPhy set Rate0 11e6
Phy/WirelessPhy set Rate1 5.5e6
Phy/WirelessPhy set Rate2 2e6
Phy/WirelessPhy set Rate3 1e6

# 100m , 150m , 200m and 250m
Phy/WirelessPhy set RXThresh0 1.427e-08
Phy/WirelessPhy set RXThresh1 2.818e-09
Phy/WirelessPhy set RXThresh2 8.916e-10
Phy/WirelessPhy set RXThresh3 3.652e-10
```

An additional function is the ability to switch between channels from within Click. The `ClickWirelessInfo` element contains all wireless configuration parameters like bssid, ssid and channel id. Any change of channel id in this `WirelessInfo` element will trigger a switch between channels in the *ns-2* simulator. This switch is executed by the following `SwitchChannel` function in the *ns-2* scenario:

```
#setting network channels [1..13]
set netchan Channel/WirelessChannel
for {set i 1} { $i < 14} {incr i} {
        set chan_($i) [new $netchan]
}
#switch channel function
proc SwitchChannel { i whichif whichnewchannel } {
        global ns_ chan_
        [$ns_ set Node_($i)]  changechannel \
                $whichif $chan_($whichnewchannel)
}
```

In short, the nsmadwifi (*Nsclick* + MadWifi) extension presented here, adds the following features of the MadWifi driver to the *Nsclick* simulator platform:

- transmission of 802.11 raw frames, with the possibility to set transmission parameters like transmission rate and the en-

abling of the RTS/CTS protection mechanism and with the possibility to retrieve the received signal strength indication (RSSI);

- availability of feedback information of the transmitted frames, indicating failure or success and the retry count;
- and the possibility to change the wireless channel from within the Click Modular Router.

The nsmadwifi extension has been made publicly available and can be downloaded from [4], together with a quick start guide. This quick start guide describes three scenarios covering each one a basic feature of nsmadwifi.

4. CASE STUDIES

In this section, we present two case studies, briefly showing research performed using the nsmadwifi extension, presented in this paper. The first study will take advantage of the fact that using the madwifi extensions, an IEEE 802.11 infrastructure mode access network can be created with Click. The basic idea is, that while *ns-2* only takes care of the lower level 802.11 MAC protocol, the higher level management functions of 802.11 are performed by the Click Modular Router, creating plenty of opportunities to develop cross layer handover schemes.

The second case study will show how to take advantage of the extension to develop a new cross layer neighbor sensing protocol for wireless mobile ad hoc networks. By making an ad hoc routing protocol aware of 802.11 messages, a more mobility sensitive neighbor sensing protocol can be designed.

4.1 A Feedback based smooth Mobile IP Handover

4.1.1 Introduction

In [7] we described a buffering mechanism that uses IEEE 802.11 feedback information in order to realize a smooth Mobile IP handover. The results published in this work were obtained using a real-life testbed, but in order to have some more dynamic and scalable results, we easily ported the used Click scripts to the *ns-2* simulator using the MadWifi extensions we here introduced.

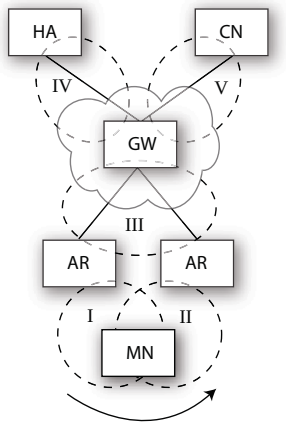


Figure 5: Reference network

In the testbed setup, all wireless communication was performed using Atheros based wireless interface cards and the MadWifi driver in monitor mode. This made it possible to implement the higher level functions of an IEEE 802.11 infrastructure stack in Click, which would handle all management tasks of the MAC layer (scanning, authentication, association) while the basic lower level MAC tasks (RTS/CTS, retransmissions,...) were performed by the firmware and/or nic. As stated before, the combination of Click and the MadWifi driver enables us to create raw 802.11 frames and inject them on the wireless medium.

Our testbed is unfortunately limited in space and mobility, thus so far, we were only able to perform tests on a basic scenario, using two static access points and a static mobile node, which was forced to perform a handover by filtering out beacons and thus triggering a new scanning phase. The reference network is shown in figure 5. Using *ns-2*, it is now easy to scale up the number of access points and study the protocol behavior when multiple handovers are performed.

The protocol itself also uses cross layer information from the 802.11 MAC to optimize the handover. The old Foreign Agent (oFA) will need to buffer packets while the previously connected Mobile Node (MN) will perform a hard handoff and scan for a new Access Point (AP). Standard buffering is however likely to introduce duplicate packets when buffers are over-dimensioned while small buffers will introduce packet loss. By accounting which frames are successfully delivered by an AP to the connected Station (STA), the oFA can decide to buffer only those frames on which transmission failed, minimizing the total number of packets that need to be buffered and avoiding duplicating packets. This transmission feedback information was available on our testbed through the MadWifi drivers. Frames which were delivered to the MadWifi driver, are again delivered to Click when they are processed by the driver. This results in a packet which now holds the transmission state, delivered or dropped, of this packet and also the number of retries which were needed to transmit the packet. Using the presented extensions to *Nsclick*, this information is now also transparently available in the simulator environment.

4.1.2 Implementation

A two stage buffering mechanism is used to implement the feedback based queuing. A FA will need to set up such a buffer mechanism for each of its registrated MNs. The general idea is to store a newly arrived packet in a first stage regular queue at the IP layer and deliver a copy of it to the link layer for transmission. When a successful transmission is reported by the link layer, this packet is removed from the queue and dropped at the FA. In case of a transmission failure, the packet moves from the general queue to a second stage circular queue, holding packets which recently failed transmission. This is likely to occur at the handover of a MN as the oFA unaware of the handover will continue to try delivering packets which will not be received by the MN as it has broken its connection. When the MN is later on connected with a new FA, this nFA can request the oFA to forward all packets it failed to transmit and thus are stored in its circular buffer along with those which are scheduled for transmission and of which a copy is hold at the first stage buffer.

The Click implementation of this buffering mechanism is illustrated in figure 6. The key elements are marked in grey and will be further discussed here, a complete discussion would take us to far and can be found in [7].

The PaintSwitch element at the top will distinguish between newly arriving packets and feedback packets arriving at the buffer compound. Firstly, newly arriving packets will be copied at the

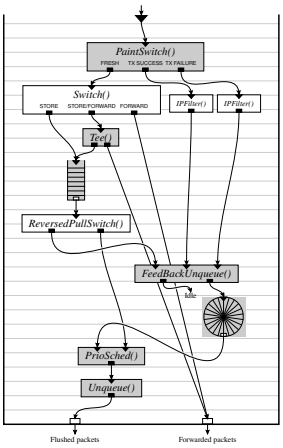


Figure 6: Click implementation of an IEEE 802.11 Feedback based buffer

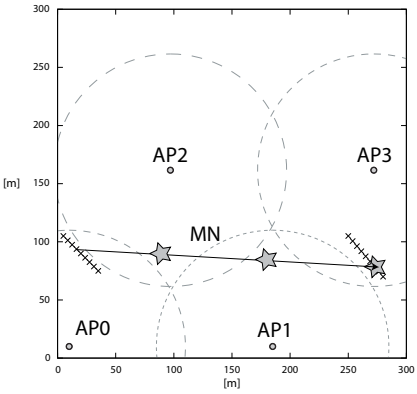


Figure 7: Map of the simulated environment

Tee element and while a copy is forwarded further on the Click graph to get transmitted, a copy is stored at the SimpleQueue at the left hand side, which implements the first stage buffer. Secondly, feedback packets holding the transmission information are fed into the FeedBackUnqueue element which reacts by pulling a packet from the first stage queue. In case of a successful feedback packet, this packet will be dropped, otherwise it will be sent to the FrontDropQueue at the right side via the second output port of the FeedBackUnqueue element.

In case a flush of this buffer compound is requested, the Unqueue element at the bottom will be activated and both the circular and the regular queue will be emptied. The priority scheduler in front of the unqueue element will ensure that the older packets which already failed transmission will be pulled before the more recent packets stored in the first stage queue.

4.1.3 Results

Using the nsmadwifi extension, we simulated an extended scenario as an example of the use of this extension. The Click configurations which were used on the testbed, were reused with only those modifications already necessary to port Click to *Nsclick*. Nothing was changed in the code of Click nor the Click configuration scripts concerning the raw packet mode or the feedback mechanism.

In figure 7, the simulated environment is shown. In an area of 300 by 300 meters, four access points were deployed. Each AP operates in a different channel than its neighbours. A MN will start from one of the 10 start points, marked with a cross at the left side, and will move at a speed of $20m/s$ towards one of the 10 endpoints at the right side. Start and end points are chosen randomly at the beginning of the simulation. Figure 7 also indicates the locations at which a handover is expected to occur by a star.

As in the reference network shown in figure 5, all access points are connected to a gateway which will relay to the Corresponding Node (CN) and the Home Agent (HA). An RTP CBR stream of 30 packets per second was sent from the CN towards the MN. A STA will detect a link failure when it misses 3 consecutive beacons and will respond to this by scanning for new APs.

Figure 8 shows us a sequence graph of a single run. The three handovers are distinctly present in the graph. The detail on the graph, shows the first handover more clearly. It is clear that no

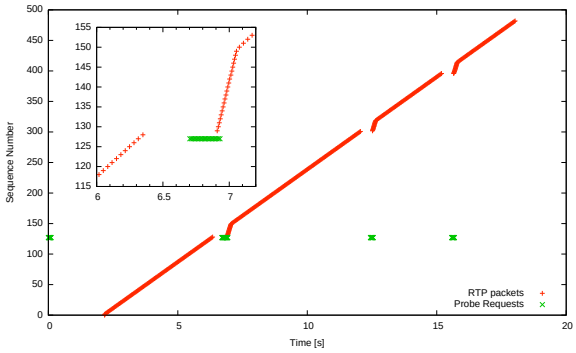


Figure 8: Sequence graph of three handovers

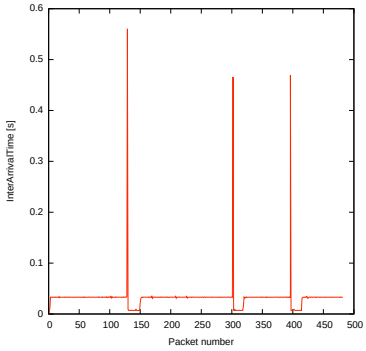


Figure 9: Packet interarrival time at the Mobile Node

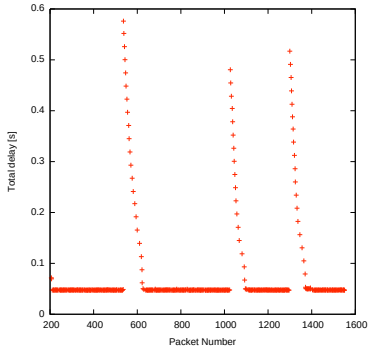


Figure 10: Total transmission delay from HA to MN

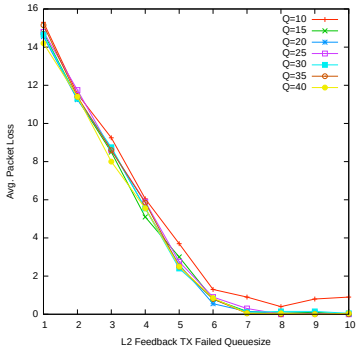


Figure 11: Average number of packets lost in relation of both buffer sizes

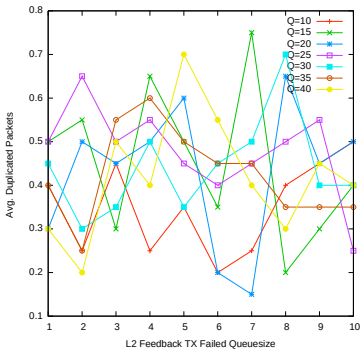


Figure 12: Average number of packets duplicated in relation of both buffer sizes

packets were lost nor packets were duplicated when the buffered packets at the old FA were forwarded towards the new FA. A successful smooth handover took place. Figures 9 and 10 show some more results from the same test-run. As the MN will need to scan for a new AP, a connection gap will exist. This can be noted from figure 9 which shows the interarrival time of packets arriving at the MN. Again the three handovers are visible and the first one introduces a lager delay due to the lesser channel conditions at that point. Once the first packet has been flushed from its queue, all packets follow back-to-back, which results in a temporary drop of the interarrival time. Off course, the total delay, from CN to MN, a packet experiences, depends on the moment at which the packet arrives at the AP during the handover. Figure 10 shows us this total delay. The first packet which could not be delivered to the MN while it was scanning will experience the largest delay, which gradually decreases based on the constant interarrival time of the CBR traffic.

In figures 11 and 12 we finally present a comparison of various queue-sizes and the resulting packet loss and packet duplication. Figure 11 shows us the average packet loss related to the combination of the size of the feedback queue (x-axis) and first stage buffer (the various indexes). From the results it shows that a second stage queue of minimum 7 packets will suffice to realise a lossless handover when combined with a first stage buffer of at least 15 packets.

In [7] we also showed packet duplication was an issue that needed attention. Figure 12 shows the average number of packets duplicated during an handover. From the scattered results for the various combinations of queuesizes in this figure, it is clear that increasing the buffersize, both in total as for the separate queues, does not increase the number of duplicated packets as was the case with a plain circular buffer. Buffersizes can be easily extended to cope with various packet rates and handover delays without introducing packet duplication.

4.2 A cross layer neighbor sensing mechanism for wireless ad hoc networks

4.2.1 Introduction

Common ad hoc routing protocols like the Ad hoc On demand Distance Vector (AODV) [11] protocol and the Optimized Link State Routing (OLSR) [6] protocol employ a hello advertisement protocol as a neighbor sensing mechanism: each mobile station advertises itself by broadcasting periodically a "Hello" message to all neighbors. Such a routing protocol only detects a link break when the link times out, as there are no new Hello messages that arrive. Inherently there is delay between the actual link break, i.e. the node is out of range, and the link break detection, i.e. when the routing protocol reacts upon this link break event by recomputing the routing table .

One possible solution [5] to decrease this gap is to increase the rate at which the Hello messages are sent, in order to react to a link break more quickly. However the authors of [12] showed that this approach is not feasible as it increases the signaling overhead on the network exponentially and thereby it reduces the available capacity for normal communication. As an alternative solution, they proposed a link layer feedback mechanism (similar to this extensions feedback mechanism) to notify the routing protocol when a failed transmission occurred, triggering the removal of the link. While this scheme works well in lightly loaded networks, it has an adverse effect when the network is more loaded, because in a heavily loaded network a packet transmission failure could also be the result of contention of the medium.

In this case study we will show a cross layer neighbor sensing mechanism for OLSR, which combines the ideas of both previous solutions and which takes advantage of the features provided by this extension.

4.2.2 Protocols

The first feature of the nsmadwifi extension we use is the possibility to receive and transmit raw 802.11 frames, including 802.11

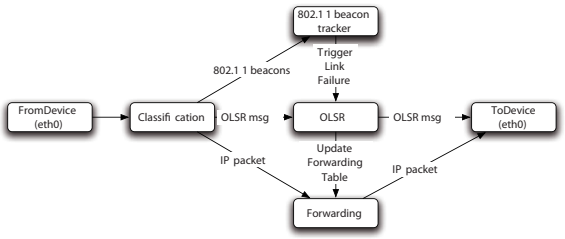


Figure 13: Cross layer neighbor sensing scheme

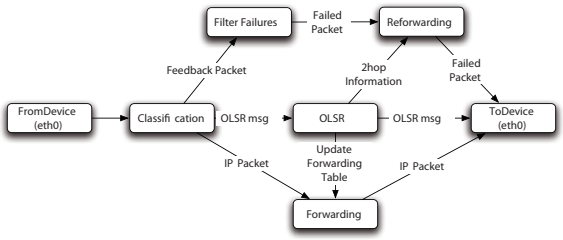


Figure 14: Fallback scheme

beacons. The 802.11 standard mandates the use of IBSS beacons for a wireless network interface in infrastructure-less mode. These IBSS beacons are similar to the beacons of a wireless access point and their purpose is to advertise the presence of a mobile station to other ad hoc stations in the network (on the MAC layer). These IBSS beacons are typically sent at a 100 ms interval, which is 10 to 20 times faster than the typical OLSR Hello interval rate.

We propose a cross layer neighbor sensing for the OLSR protocol (figure 13), which combines the fast presence advertisements contained in 802.11 beacons (layer 2), with the link symmetry information contained in the slower hello messages (layer 3) to increase the sensitivity of the OLSR to a link break. This cross layer neighbor sensing comprises two key components:

- the hello message signaling (layer 3): link sensing is performed using regular HELLO messages as described in the OLSR RFC, but instead of updating the validity time of links to three times the HELLO emission interval, it is only updated to one time the HELLO interval. Additionally the ethernet address of the neighbors interface is registered in the link information entry.
- the 802.11 beacon tracker (layer 2): the beacon tracker updates the validity time of each link for which it receives an 802.11 IBSS beacon from the corresponding neighbor. It performs its task by analyzing incoming beacons and looking up the correct OLSR link information entry using the ethernet source address as a key. Finally the beacon tracker extends the validity time of the link with 1 second.

The synergy between these two components allows OLSR to discover more quickly a link break, while avoiding the increase in OLSR signaling overhead.

To complement this cross layer neighbor sensing scheme we propose a "fallback" scheme to close the intermediate gap between the link break event and the link break trigger. The "fallback" scheme augments the classic OLSR packet forwarding with a forwarding path for failed packets, as illustrated in figure 14. This path consists of a FilterFailures element and Reforwarding element. The FilterFailures element analyzes the layer 2 transmission feedback packets and sends the failed packets to the Reforwarding element, which in turn sends these failed packets to a different neighbor than the original next hop neighbor. A failed packet is forwarded as follows: first the original next hop IP address is determined by looking up the destination ethernet address in the link information entries; secondly based on this next hop IP address, a suitable neighbor is determined using the two hop information table of OLSR. In case of multiple possible neighbors, the neighbor, which has most recently seen the required next hop, will be selected. And at last the packet will be forwarded to this neighbor, marked with a flag in the IP header that is has been forwarded by the fallback scheme to avoid loops. Figure 15(b) illustrates the "fallback" scheme for a strip topology scenario, which we will discuss in the next section.

Table 1: OLSR parameters

Parameter	Value
HELLO_INTERVAL	1s
TC_INTERVAL	5s
NEIGHBOR_HOLD_TIME	3s
TOP_HOLD_TIME	15s

The main disadvantage of the "fallback" scheme is that it wastes bandwidth. As long as the routing protocol does not discover the link break it will continue to retransmit packets to a disappeared neighbor, before it forwards these packets using an alternative route. Only when the routing protocol detects the link break and recomputes the routing table, the transmission failures of these packets can be avoided. By combining the cross layer neighbor sensing with the "fallback" scheme into a hybrid scheme, the disadvantages of both can be mitigated. The cross layer neighbor sensing increases the reactiveness of the routing protocol to link breaks, while the "fallback" scheme covers the gap between the link break and the detection of it.

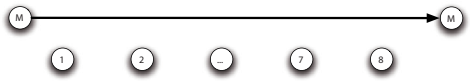
4.2.3 Results

Basic Scenario.

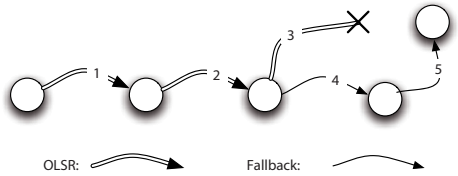
The first scenario we have analyzed is the strip scenario (figure 15(a)), consisting of eight stationary nodes in line and a nineth mobile node moving in parallel away from the first node. The distance between the stationary nodes is 175m and the transmission range was reduced to 200m at 2Mbps. The velocity of the mobile node varied from from 2m/s , 5m/s , 10m/s ,15m/s to 20 m/s. A 128Kbps CBR stream flows from the first node to the mobile node and the packet size was 1000 bytes. The OLSR parameters are shown in table 1.

The simplicity of this scenario allows us to trace the reactions of the routing protocol and the network as a whole back to one single link break event. In a more complex scenario it would have been more difficult to correlate the multiple events to different link breaks.

The sequence graph in figure 16(a) shows seven distinct link breaks for a mobile speed of 5m/s. As can be seen, the last two link breaks have a much greater impact. This is caused by the fact that OLSR only disseminates partial topology information, instead of full topology information. When the mobile node reaches the end of the line topology, it will have a direct link with station seven



(a) Strip topology



(b) Fallback scheme

Figure 15: Strip scenario

and eight, and therefore this mobile only select one single multipoint relay (MPR). As MPRs only advertise links to its selectors to the ad hoc network, only a partial topology is disseminated. When the mobile station moves out of range of its MPR, the relay will inform the network about this link break, causing the whole network to lose track of the mobile node's whereabouts. This continues until a new MPR has been selected and this new relay starts advertising the link with the mobile node. To tackle this issue OLSR can disseminate the full topology information. The schemes, where full topology dissemination is active, are denoted with FT (Full Topology), however we omitted the sequence graphs as they add no value to understanding the cross layer schemes.

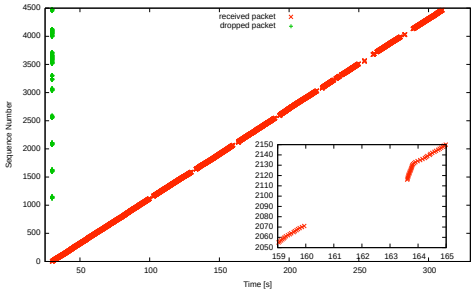
When we compare the sequence graph of the basic OLSR scheme (figure 16(a)) with the sequence graph of the cross layer neighbor sensing mechanism (figure 16(b)), it becomes clear that the cross layer neighbor sensing mechanism decreases the gap between a link break and the detection of this event, although not completely.

The sequence graph of the fallback scheme (figure 16(c)) shows that packet loss can be avoided, but at the expense of increasing the end-to-end delay and packet reordering (figure 17(b)). When a link break occurs, the packets will still be forwarded by the routing protocol using the broken link until it notices this link break. All these packets, which are retried by the 802.11 MAC until the retry count has exceeded, add up significantly to the total delay as multiple retransmission not only take up a lot of time itself, but also delay other frames waiting for a transmission attempt in the interface queue. However figure 17(a) shows that the cross layer neighbor sensing mechanism does not suffer from this problem. Therefore the combination of both schemes keeps their respective advantages while mitigating their disadvantages (figures 16(d) and 17(c)), i.e. increasing packet delivery while avoiding the increased delay.

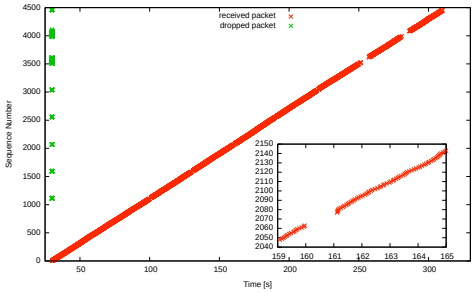
Figure 18 shows for different velocities of the mobile node that both the cross layer neighbor sensing and the fallback scheme decrease the packet drop ratio significantly. The combination of both schemes, the hybrid scheme, eliminates almost all packet loss for this scenario, when the routing protocol disseminates the full topology information.

Random waypoint scenario.

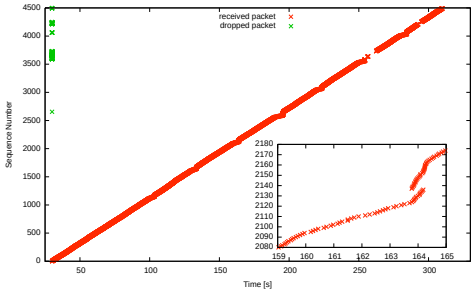
The second scenario is a random way point scenario, consisting of 32 mobile stations placed in a 1000m by 1000m square topology. The mobile stations have a 250m transmission range at



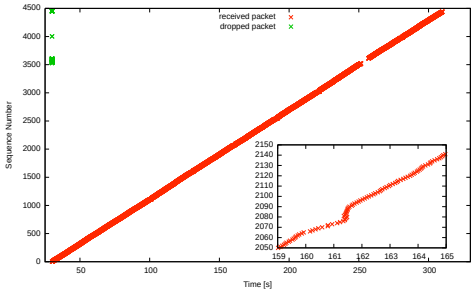
(a) OLSR



(b) Cross layer neighbor sensing scheme

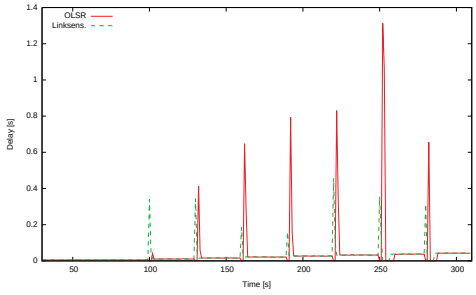


(c) Fallback scheme

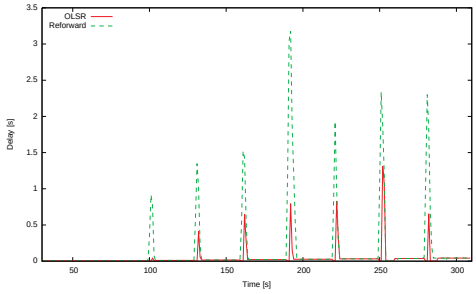


(d) Hybrid scheme

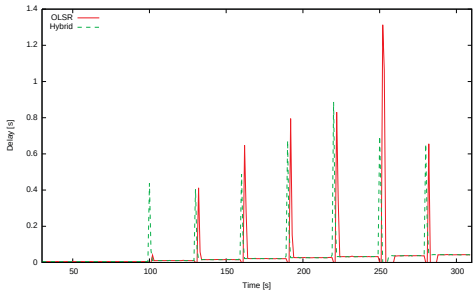
Figure 16: Strip scenario 5m/s



(a) Delay cross layer neighbor scheme



(b) Delay fallback scheme



(c) Delay hybrid scheme

Figure 17: Delay Strip scenario 5m/s

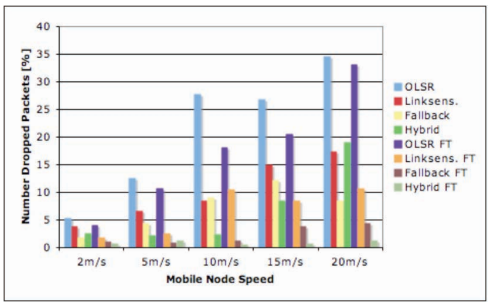


Figure 18: Strip scenario packet drop

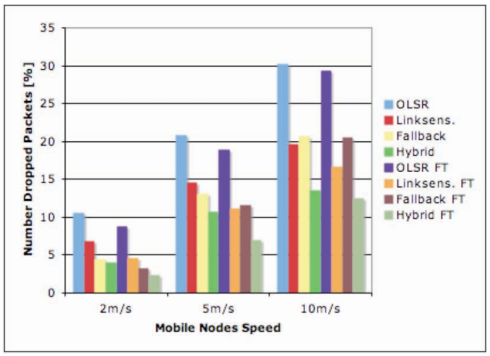


Figure 19: Random scenario packet drop

2Mbps and depending on the performed scenario, a fixed velocity of 2m/s, 5m/s or 10 m/s. From the 32 nodes, eight pairs are chosen randomly and each pair of nodes will build up a one way 16Kbps CBR stream with packet sizes of 250 bytes. This low stream rate was selected in order to avoid link saturation. The results were averaged over 10 runs. Figure 19 shows that both schemes (cross layer scheme and the fallback scheme) manage to decrease packet loss with 5 to 10%, both combined they provide a much stronger reduction, by halving the packet loss ratio.

5. CONCLUSION

In this paper we first described how we adapted the *Nsclick* simulator in order to support the Wifi extensions already available in Click. Furthermore we illustrated its use in the development, debugging, and validation of cross layer design techniques in wireless networks. We presented two case studies, showing its use in the development of cross layer algorithms for wireless infrastructure networks and mobile ad hoc networks. Both cases are heavily dependent on link layer information otherwise not easily available in a *Nsclick* environment. This MadWifi extension introduces this functionality in a uniform and reusable way in *Nsclick*, making the development of other cross layer protocols in Click straight forward.

The power of the combination of the MadWifi driver and the Click Modular Router and hence this extension lies in its simplicity: it provides access to the core of the 802.11 standard: the MAC protocol. This simplicity offers the researcher the necessary freedom to quickly and easily develop and analyze its own protocols or cross layer algorithms using an 802.11 MAC. It does so by providing three core functions: raw frame transmission, feedback of transmission and reception information and controlling the transmission parameters on a per packet basis.

Acknowledgment

This research was partially funded by the Institute for the promotion of Innovation by Science and Technology in Flanders (IWT, Flanders) through the GBOU Contract 20152 ”End-to-End QoS in an IP Based Mobile Network”.

REFERENCES

[1] Madwifi Stripped Driver. http://pdos.csail.mit.edu/ jbicket/madwifi.stripped/.

[2] Multiband Atheros Driver for WiFi. http://sourceforge.net/projects/madwifi/.

[3] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[4] Nsmadwifi extension. http://www.pats.ua.ac.be/software/nsmadwifi.

[5] M. Benzaid, P. Minet, and K. Agha. Integrating fast mobility in the olsr routing protocol. In *Proceedings of the Fourth IEEE Conference in Mobile and Wireless Communications Networks*, September 2002.

[6] T. Clausen and P. Jacquet. RFC 3626: Optimized Link State Routing Protocol (OLSR), October 2003.

[7] Peter De Cleyn and Chris Blondia. An L2 Transmission Feedback based buffering scheme for Mobile IP handovers in an IEEE 802.11 wireless network. *WSEAS Transactions on communications*, 5(6):1055–1060, June 2006.

[8] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, pages 263–294, August 2000.

[9] Michael Neufeld, Ashish Jain, and Dirk 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*, pages 74–81. ACM Press, 2002.

[10] Michael Neufeld, Graham Schelle, and Dirk Grunwald. *Nsclick User Manual*. University of Colorado, Boulder, CO 80309, August 2003.

[11] C. Perkins, E. Belding-Royer, and S. Das. RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing, July 2003.

[12] 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.