DTN Code for ns-2.35

The DTN code has been tested with ns-2.35 [1]. There is no need to re-install ns-2 if ns-2.35 has already been installed.

The DTN code may work with earlier ns-2 versions, too. However, special attention should be paid when modifying packet.h. Moreover, the example script has to be modified: use the default (CMU) IEEE 802.11 model instead of the dei80211mr model [2].

A. Install the ns-2.35 allinone package (if needed)

Installing ns-2 is a simple process:

1. Download the allinone package from the following address:
   http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/
2. In some unix machine, in your home directory:
   ```
   tar -xvf ns-allinone-2.35.tar.gz
   cd ns-allinone-2.35
   ./install
   ```
3. At the end of installation, the installation script prints out the path variables you need to configure

B. Install the DTN code

Adding the DTN code on top of basic ns-2 is quite straightforward:

1. In your home directory:
   ```
   mkdir ns2dtn
   mv ns2dtn.tar.gz ns2dtn
   cd ns2dtn
   tar -xvf ns2dtn.tar.gz
   cd ../ns-allinone-2.35/ns-2.35
   ```
2. Create a directory called dtn and move the files mybundle.h and mybundle.cc there from the ns2dtn directory:
   ```
   mkdir dtn
   mv ../../ns2dtn/mybundle.* dtn
   ```
3. Move the files ll.h, ll.cc and arp.cc to directory mac, drop-tail.h to directory queue and ns-default.tcl to directory tcl/lib (this is done in order to disable ARP in DTN simulations; you may want to rename the original files first):
   ```
   mv ../../ns2dtn/ll.* mac
   mv ../../ns2dtn/arp.cc mac
   mv ../../ns2dtn/drop-tail.h queue
   mv ../../ns2dtn/ns-default.tcl tcl/lib
   ```
4. Move the files tcp.h and tcp.cc to directory tcp; this is done in order to get the hop count from TCP/AODV simulations:
   ```
   mv ../../ns2dtn/tcp.* tcp
   ```
5. Move the file aodv.cc to directory aodv; this is done in order to enable the use of AODV and DTN in the same simulation:
   ```
   mv ../../ns2dtn/aodv.cc aodv
   ```
6. Modify ns-packet.tcl (in ns-allinone-2.35/ns-2.35/tcl/lib):

   From: (search for the below pattern)
   ```
   AODV    # routing protocol for ad-hoc networks
   ```

   To: (one line added)

```
AODV    # routing protocol for ad-hoc networks
Bundle  # bundle protocol for DTNs
```

7. Modify packet.h (in ns-allinone-2.35/ns-2.35/common):
   a. From: (search for the below pattern)
   ```
   static packet_t PT_NTYPE = 73; // This MUST be the LAST one
   ```
   To: (one line added, one line modified)
   ```
   static packet_t PT_DTNBUNDLE = 73;
   static packet_t PT_NTYPE = 74; // This MUST be the LAST one
   ```
   b. From: (search for the below pattern)
   ```
   name_[PT_TFRC]= "tcpFriend";
   ```
   To: (one line added)
   ```
   name_[PT_DTNBUNDLE]="DTNBundle";
   name_[PT_TFRC]= "tcpFriend";
   ```
8. Modify Makefile (in ns-allinone-2.35/ns-2.35):
   a. From: (search for the below pattern)
   ```
   -I./diffserv -I./satellite \
   -I./wpan
   ```
   To: (one line added)
   ```
   -I./diffserv -I./satellite \
   -I./dtn \
   -I./wpan
   ```
   b. From: (search for the below pattern)
   ```
   apps/pbc.o \
   $(OBJ_STL)
   ```
   To: (one line added)
   ```
   apps/pbc.o \
   dtn/mybundle.o \
   $(OBJ_STL)
   ```
9. Run "make clean" and "make" under the ns-allinone-2.35/ns-2.35 folder.

C. Try the example script

Run the example script in ns2dtn directory by launching the shell script:

1. `./simulate_dtn.sh &`
2. Wait patiently, the simulation takes a few hours.
3. Study the scripts and the C++ files and figure out what they do.

NOTE: The setdest program (in /ns-allinone-2.35/ns-2.35/indep-utils/cmu-scen-gen/setdest) can be used for creating traces for random waypoint type of node mobility.

D. Try the simulation campaign

Download the other gzipped archive "Scripts for simulation campaign" and extract it in some directory (after you have installed the DTN code):

1. Extract the files:
   ```
   tar -xvf ns2dtn_campaign.tar.gz
   ```

2. Modify the path to *libdei80211mr.so* library as needed in all four simulation scripts and then launch the simulations in all four subdirectories:
```
cd AODV_C
emacs bundle-test-large-scen-aodv.tcl
nohup ./simulate_aodv.sh &

cd ../DTN_C
emacs bundle-test-large-scen.tcl
nohup ./simulate_dtn.sh &

cd ../DTN_SW_C
emacs bundle-test-large-scen.tcl
nohup ./simulate_dtn.sh &

cd ../DTN_AODV_TRACE_C
emacs bundle-test-large-scen.tcl
nohup ./simulate_dtn.sh &
```
3. When the simulations are over (in a few days, depending on your hardware), start Matlab and plot the results with the script that is included in the package:
```
matlab -nodesktop
>> dtn_vs_aodv_vs_adaptive
```

This simulation campaign is a comparison between AODV, pure DTN (epidemic routing or spray-and-wait routing), and the adaptive routing scheme described in our paper [3]. See the section 4.3 of the paper for details.

E. How to use/modify the DTN parameters

Default values of the DTN bound variables

Hello message interval [ms]:
```
Agent/Bundle set helloInterval_ 100
```

Bundle retransmission timeout [s]:
```
Agent/Bundle set retxTimeout_ 1000.0
```

Delete forwarded bundles or not (zero means no deletion):
```
Agent/Bundle set deleteForwarded_ 0
```

Use antipackets or not (zero means no antipackets):
```
Agent/Bundle set antiPacket_ 1
```

Bundle routing protocol (antipackets are always sent using epidemic); 0 = broadcast, 1 = epidemic, 2 = binary spray and wait, 3 = PRoPHET:
```
Agent/Bundle set routingProtocol_ 1
```

Maximum number of bundle copies in binary spray and wait:
```
Agent/Bundle set initialSpray_ 16
```

Bundle drop strategy; 0 = drop tail, 1 = drop oldest, 2 = drop most spread, 3 = drop least spread, 4 = drop random:
```
Agent/Bundle set dropStrategy_ 0
```

Bundle buffer size [bytes]:
```
Agent/Bundle set bundleStorageSize_ 100000000
```

Congestion control on or off (zero means off):
```
Agent/Bundle set congestionControl_ 0
```

Congestion control threshold:
```
Agent/Bundle set bundleStorageThreshold_ 0.8
```

Counters that can be used in the Tcl script

Antipacket queue size [bytes]:
```
Agent/Bundle set cqsize_ 0
```

Retransmission bundle queue size [bytes]:
```
Agent/Bundle set retxqsize_ 0
```

Bundle queue size [bytes]:
```
Agent/Bundle set qsize_ 0
```

How long the interface queue has been non-empty [s]:
```
Agent/Bundle set ifqCongestionDuration_ 0
```

Bundles sent:
```
Agent/Bundle set sentBundles_ 0
```

Bundles received:
```
Agent/Bundle set receivedBundles_ 0
```

Duplicate bundles received (destined to this node):
```
Agent/Bundle set duplicateReceivedBundles_ 0
```

Duplicate bundles received (destined to other nodes):
```
Agent/Bundle set duplicateBundles_ 0
```

Bundles dropped (due to insufficient buffer resources):
```
Agent/Bundle set deletedBundles_ 0
```

Bundles forwarded:
```
Agent/Bundle set forwardedBundles_ 0
```

Return receipts sent:
```
Agent/Bundle set sentReceipts_ 0
```

Return receipts received (destined to this node):
```
Agent/Bundle set receivedReceipts_ 0
```

Duplicate return receipts received (destined to this node):
```
Agent/Bundle set duplicateReceivedReceipts_ 0
```

Duplicate return receipts received (destined to other nodes):
```
Agent/Bundle set duplicateReceipts_ 0
```

Return receipts forwarded:
```
Agent/Bundle set forwardedReceipts_ 0
```

Average end-to-end bundle delay:
```
Agent/Bundle set avBundleDelay_ 0
```

Average end-to-end receipt delay:
```
Agent/Bundle set avReceiptDelay_ 0
```

Average bundle hop count:
```
Agent/Bundle set avBundleHopCount_ 0
```

Average receipt hop count:
```
Agent/Bundle set avReceiptHopCount_ 0
```

Average link lifetime (exponential averaging weight: 0.02):
```
Agent/Bundle set avLinkLifetime_ 0
```

Average number of neighbor nodes (exponential averaging weight: 0.02):
```
Agent/Bundle set avNumNeighbors_ 0
```

Send a bundle: parameters

```
$bundleagent send $dst_node_id $bundle_size [bytes] $lifetime [s] \
                  $custody_transfer [0/1] $return_receipt [0/1] \
                  $priority [no effect yet]
```

Mobility and traffic files

```
source scen_n40_pt2_ms20_t5000_x2000_y2000
source trafficgen.tcl
```

F. Main C++ Functions

- `sendHello`: Periodically broadcast a Hello message that contains the identifiers of those bundles that are stored in this node, bundle storage size and current bundle storage occupancy. Moreover, other (e.g., PRoPHET specific) information is broadcast as well. The size of the Hello message is 20 bytes, it is currently hard coded.
- `checkStorage`: Periodically check what bundles and receipts in the storage can be forwarded.
    - Update the bound variables for queue size and non-empty interface queue duration.
    - Update the number of neighbor nodes and the average link lifetime (i.e. average contact duration) and remove all expired bundles from the storage.
    - Check if the interface queue is empty (or if there is only one packet). If this is the case, we can proceed and select the next packet to be transmitted. Antipackets and Hello messages (in the management buffer) have the first priority, retransmissions have the second priority and only if these two buffers are empty, we can send an ordinary packet.
    - Request for missing bundle fragments.
    - Take bundles from the bundle storage (in a specific order), fragment them and put the fragments to the bundle buffer waiting for transmission. Congestion control can be applied here: if the bundle storage occupancy in the neighbor node is above the congestion control threshold, we do not forward bundles to this node.
- `recv`: Receive packets and process them accordingly.
    - Hello message: update neighbor information, e.g., bundle identifiers.
    - Nack (negative acknowledgement): retransmit the requested packet.
    - Ordinary packet: first check if all packets of a bundle have been received; if yes, then forward/receive/delete the bundle. Different bundle dropping strategies can be applied if the bundle storage is full.
    - Custody ack: delete the bundle from storage.
    - Return receipt / antipacket: delete the corresponding bundle from storage and add its identifier to the database (so that possible copies can be deleted when they arrive).

G. References

[1] UCB/LBNL/VINT, “Network Simulator – ns (version 2),” Jul. 2012.

[2] University of Padova, Department of Information Engineering, “dei80211mr: a new 802.11 implementation for NS-2,” Jul. 2009. [Online] http://www.dei.unipd.it/wdyn/?IDsezione=5090.

[3] J. Lakkakorpi, M. Pitkänen, and J. Ott, “Adaptive Routing in Mobile Opportunistic Networks,” in Proc. ACM MSWiM 2010, Bodrum, Turkey, Oct. 2010, pp. 101-109.