HWL Documentation

Release 1.0

Robert Sombrutzki

May 01, 2014

CONTENTS

Inhalt:

CHAPTER

ONE

BRN-TOOLS

1.1 Overview

1.1.1 brn-tools

1. **Folgende Verzeichnisse enthaelt brn-tools:**
    (a) brn-ns2-click
    (b) click-brn
    (c) click-brn-scripts
    (d) helper
    (e) jist-brn
    (f) ns2
    (g) ns-3-brn

1.2 click-brn

Enthaelt das Click-Framework inklusive der BRN-Erweiterung.

1.3 helper

Enthaelt die Tools und Werkzeuge zum starten von Simlulationen, Messungen im Testbed und zur Auswertung.

1.3.1 run-sim.sh

Zum Starten einer Simulation: run_sim.sh [ns|ns3|jist] [des-file] [target-dir]

1. **Parameter**
    (a) Verwendeter Simulator (NS2, NS3 oder JiST)
    (b) des-File (Beschreibung der Simulation)
    (c) Ziel-Verzeichnis
2. Umgebungsvariablen

Das Skript kann ueber Umgebungsvariablen zusaetzliche Optionen erhalten. Folgende Mglichkeiten lassen sich so nutzen: * Debugger (gdb): GDB * Leak-Checker (Valgrind): VALGRIND * Profiler (Callgrind): PROFILER * Abschalten der Simulation (benutzt von parasim): PREPARE_ONLY * Abschalten der Evaluation (benutzt von parasim): DELAYEVALUATION

1.4 Wichtig

Die wichtigsten Verzeichnisse sind: #. click-brn #. click-brn-scripts #. helper

Das wichtigste Kommando: #. run_sim.sh

CHAPTER

TWO

CLICK-BASICS

2.1 Installation

2.1.1 Vorbereitungen

1. **Benötigte Software installieren:**
    (a) gcc
    (b) g++
    (c) autoconf
    (d) libx11-dev
    (e) libxt-dev
    (f) libxmu-dev
    (g) flex
    (h) bison
    (i) git
    (j) bc - GNU bc - Rechnersprache mit beliebiger Genauigkeit
2. Account auf gitsar bei Robert beantragen
3. Folgende Einträge in der .ssh/config vornehmen:

```
Host gruenau
    User <username>
    HostName <hostname>.informatik.hu-berlin.de

Host gitsar
    ProxyCommand ssh -q gruenau netcat sar 2222
    User git
```

Der <username> entspricht dem Informatik-Email-Account-Namen <username>@informatik.hu-berlin.de Der <hostname> kann z. B. entweder gruenau oder gruenau2 sein (siehe https://www2.informatik.hu-berlin.de/rbg/Intern_SSL/pools.shtml).

Am komfortabelstenfunktioniert die Verbindung beim Einsatz von SSH-Keys und dem ssh-agent, da man dann nur einmal die Passphrase für den SSH Schlüssel eingeben muss.

2.1.2 Software auschecken

1. brn-tools auschecken:

   ```
   git clone git@gitsar:brn-tools
   ```

2. brn-tools.sh ausführen:

   ```
   cd brn-tools
   ./brn-tools.sh
   ```

3. Umgebungsvariablen setzen. Entweder per `source ./brn-tools.bashrc` oder durch das Kopieren der Einträge in `brn-tools.bashrc` in die eigene `.bashrc`

2.1.3 Testen

Zum Testen der Installation kann die *simple_flow* (oder eine andere) Simulation ausgeführt werden:

```
cd click-brn-scripts/003-simple_flow
run_sim.sh
```

Die Ausgabe der Simulation sollte dabei folgendermaßen aussehen:

```
sim is ns
Send 26 packet, received 26 packets and 26 echo packets are received. OK !
```

Alternativ können über das Shellscript test.sh alle Simulationen angestoßen werden. `test.sh` erzeugt dabei ein pdf namens `testbed.pdf`, welches die Resultate aller Simulationen enthält.

2.1.4 Troubleshooting

- **Wenn die Simulation mit dem Fehler `*** buffer overflow detected ***: ./ns terminated`** abbricht liegt das vermutlich daran, dass die falsche TCL Version verwendet wird. Abhilfe schafft entweder die Deinstallation von TCL auf dem System oder aber das Sicherstellen, dass ns2/bin/tcl vor den Systembinaries im Pfad liegt.
- **Falls das Ausführen von test.sh Warnungen und/oder Fehler erzeugt liegt dies unter Ubuntu** Systemen evtl. daran, dass die dash als System Shell verwendet wird. Eine mögliche Lösung besteht darin, eine andere System Shell mittels `sudo dpkg-reconfigure dash` festzulegen.

2.2 Simulationen

Im Ordner `simulation/click-brn-scripts/` liegen Simulationsscripte für verschiedene Experimente. Beim Ausführen der Scripte (siehe *run_sim.sh*) werden verschiedene die Ausgaben der verschiedenen Knoten gesammelt. Anschließend werden diese Ausgaben analysiert und ausgewertet und die Resultate im Ordner `simulation/click-brn-scripts/<SCRIPT>/<NUMBER_OF_EXPERIMENT>` gespeichert.

Eine Simulation wird durch die folgenden beiden Dateien definiert:

- Mes-files beinhalten alle Geräte, die für das Experiment vorbereitet werden.
- Des-Files beinhalten eine grobe Beschreibung vom Experiment. Z.b.: Die Dauer des Experiments; das Verzeichnis für die log-files; Netzwerk-Topologie; etc. ...

2.2.1 run_sim.sh

Um eine Simulation auszuführen wird das *run_sim.sh* Script verwendet, welches sich im Verzeichnis */helper/simulation/bin/* befindet. Das Script nimmt als Parameter den zu verwendenden Simulator (*ns* oder *jist*) und den Pfad zur *des* Datei der Simulation entgegen:

```
run_sim.sh ns <des-File>
run_sim.sh jist <des-File>
```

2.3 Experimente im Testbed

2.3.1 run_measurement.sh

Ähnlich wie bei der Simulation verwenden wir das selbst geschriebene Skript *run_measurement.sh*. Dieses führt grob folgende Arbeitsschritte durch:

1. für jeden Knoten (siehe *.mes-Datei) wird eine Screen-Session hergestellt
2. über diese Screen-Session werden per ssh Befehle abgesetzt
3. außerdem werden per NFS Informationen über die Knoten eingeholt (z. B. Architektur-Info)
4. Treiber laden
5. Treiber konfigurieren
6. Click starten
7. Zusätzliche Pre- und Post-Skripts ausführen

2.4 Weitere Dokumentation

- Search click documentation: http://read.cs.ucla.edu/click/docs
- Publications about click and stuff that uses click: http://read.cs.ucla.edu/click/publications
- Manual how to program click elements: http://read.cs.ucla.edu/click/doxygen/class_element.html
- Information about click elements: http://www.read.cs.ucla.edu/click/elements
- Network Simulator 2 (NS2) Docu: http://isi.edu/nsnam/ns/

CHAPTER

THREE

CLICK-ELEMENTE BAUEN

(Hier fehlt noch Doku)

3.1 Handler

3.1.1 Handler bauen

Handler sind Zugangspunkte, mittels derer die Nutzer mit einzelnen Elementen zur Laufzeit des Click-Routers interagieren können. Man unterscheidet dabei zwischen read und write handlern, die sich jeweils wie Dateien in einem Dateisystem verhalten. Im folgenden wird dies anhand eines Beispiels erläutert.

1. Falls nicht vorhanden, füge eine Klassen-Methode mit dem Namen add_handlers() zum Element hinzu. Diese Funktion dient als Sammelstelle für die Registrierung aller Element-Handler:

   ```
   void
   MyElement::add_handlers()
   {
           //todo
   }
   ```

2. Als nächstes folgt die read/write Handler im einzelnen. Dazu werden die Methoden *add_read_handler* und *add_write_handler* verwendet. Das Argument *name* definiert den Namen des Handlers, *func* steht für die aufzurufende Funktion und *thunk* für zusätzliche Parameter. Vollständige Signatur und Beispiel:

   ```
   void add_read_handler (const String &name, ReadHandler func, void *thunk)
   void add_write_handler (const String &name, WriteHandler func, void *thunk)

   // Beispiel
   add_read_handler("info", read_my_param, (void *) H_READ);
   add_write_handler("debug", write_my_param, (void *) H_DEBUG);
   ```

3. Nun müssen die Funktionen der neu eingeführten Handler definiert werden. (Die Besonderheit ist hier das Schlüsselwort static. Dies hat tiefergehende Gründe, die hier nicht weiter diskutiert werden.)

   ```
   #include <click/straccum.hh>
   ...
   static String
   read_my_param(Element *e, void *thunk) {
           StringAccum sa;
           // do foo hier
           return String();
   }
   ```

```
static int
write_my_param(const String &string, Element *e, void *vparam, ErrorHandler *errh) {

        MyElement *e = (MyElement *)e; //cast
        e->cmd = string;
}
```

Quelle: http://www.read.cs.ucla.edu/click/element?s[]=handler

3.1.2 Handler verwenden

Handler können auf verschiedenste Weisen verwendet werden.

- In einem Click-Script: Dazu wird meist ein *Script*-Bereich am Ende eines Click-Scripts eingeführt.

```
...

Script(
read MyElement.info,
write MyElement.debug 3
);
```

Quelle: http://www.read.cs.ucla.edu/click/elements/script?s[]=handler

- ControlSocket

Quelle: http://www.read.cs.ucla.edu/click/elements/controlsocket?s[]=handler

3.1.3 Weiterführende Links

-

3.2 OpenSSL

Bei der Verwendung der Funktionen aus der SSL-Library ist es notwendig, dem Compiler mitzuteilen, dass er die ssl-lib mitlinken soll. Dies geschieht durch das Hinzufügen der folgenden Zeile an das Ende der Quelldatei eines Elementes, in dem die SSL-Includes zur Anwendung kommen:

```
ELEMENT_LIBS(-lssl)
```

CHAPTER

FOUR

CLICK-SKRIPTE BAUEN FÜR NETWORKING

Vorwissen: Click-Paper (http://read.cs.ucla.edu/click/publications)

Zuvor eine historische Notiz, die dem Leser den Ursprung einiger hier verwendeter Elemente erklären soll.

Historisch ist das HWL-Netzwerk aus zwei Projekten hervorgegangen: Zum einen aus dem MIT-Projekt "Click Modular Router", welches das Framework liefert, und zum anderen aus der Kooperation zwischen dem Berliner Freifunk-Projekt "Berlin-Roof-Net" (BRN) und der HU-Berlin (SAR-Group). Diese Kooperation hatte die Erforschung und Konstruktion eines drahtlosen Ad-Hoc-Netzwerkes zum Zweck und ergänzte das Framework um weitere wichtige Netzwerkelemente, die den Betrieb des BRN-Netzwerkes ermöglichten. Die weitere Entwicklungs- und Forschungsarbeit wurde am Institut für Informatik in Form des BRN2-Netzwerkes fortgesetzt, baut jedoch nach wie vor auf den Arbeiten von Click und BRN auf. Die Kombination aus Click-System (Framework), BRN (speziell entwickeltes Netzwerk) und BRN2 (zusätzliche Komponenten) ist Teil des HWL-Netzwerkes (Humboldt Wireless Lab).

Diese historischen Vorgänge spiegeln sich im ganzen Dateisystem und der Namensgebung wider.

Bevor man beginnt ein Click-Skript zu entwickeln, sollte man erst einmal einen Entwurf machen, in dem man festhält, welches Ziel man in der Simulation verfolgt. Das Skript, welches man am Ende entwickelt, liefert den Netzwerkknoten, auf dem es läuft, einen vollständigen Netzwerkstack. Dementsprechend müssen Netzwerkgeräte, Routing-Protokolle, u. Ä. in den Entwurf mit einbezogen werden. Die Vorüberlegungen beinhalten:

- Welche Elemente oder Elementklassen dafür benötigt werden und
- wie diese Elemente miteinander verbunden sind.

Höchst wahrscheinlich benötigt man ein Wifi-Device, welches stellvertretend für die physikalische Schicht steht. Sodann wird ein Routing-Protokoll benötigt, stellvertretend für den Vermittlungsschicht.

Achtung: Unsere Click-Architektur verwendet Ethernet-Routing und unterscheidet sich daher von üblichen Schichtenmodellen. Das bedeutet, dass bei der Verknüpfung eines Wifi-Devices mit einem Routing-Modul die Elemente BRN2EtherEncap(), BRN2EtherDecap(), BRN2Encap(), BRNDecap() verwendet werden müssen. Näheres dazu im nächsten Abschnitt.

4.1 Elemente zum Senden und Empfangen im BRN2-Netzwerk

Jedes zu übertragene Paket benötigt einen Ethernet-Header, um einfache Ad-Hoc-Kommunikation, Infrastruktur-Kommunikation oder Ethernet-Routing (unter Zuhilfenahme eines Routing-Protokolls) durchzuführen. Der Ethernet-Header enthält unter anderem die Quell- und Zieladresse. Für das Hinzufügen und Entfernen der Ethernet-Header benötigt man die folgenden Click-Elemente:

- BRN2EtherEncap()

- BRN2EtherDecap()

Zusätzliche Informationen, die für die BRN2-Netzwerkkommunikation wichtig sind, werden in einem sogenannten BRN2-Header verpackt. Für das Hinzufügen und Entfernen der BRN-Header, benötigt man folgende Click-Elemente (die Makros in BRN2Encap dienen der Default-Einstellung):

- BRN2Encap(BRN_PORT_FLOW, BRN_PORT_FLOW, BRN_DEFAULT_TTL, BRN_DEFAULT_TOS)
- BRN2Decap()

Beispiel:

```
tls
    -> BRN2Encap(BRN_PORT_FLOW, BRN_PORT_FLOW, BRN_DEFAULT_TTL, BRN_DEFAULT_TOS)
    -> BRN2EtherEncap(USEANNO true)
    -> [1]device_wifi;
```

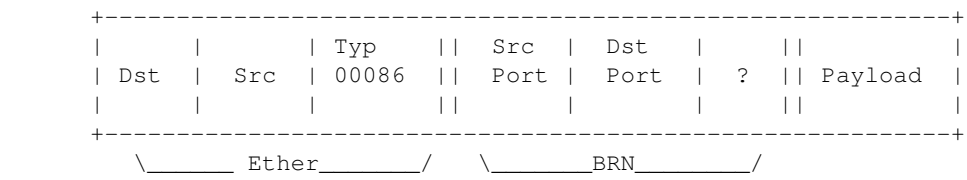
Neben der Verwendung von BRN2Encap im Click-Script, lässt sich dieser Header auch direkt bei der Paketverarbeitung innerhalb der Click-Elemente (C++) einsetzen. Dies ermöglicht eine größere Kontrolle über die Paketinformationen. Die entsprechende Funktion nennt sich “add_brn_header()”. Hier ein Beispiel:

```
tls
    // -> BRN2Encap() /* Diese Funktion wird vom "tls"-Element implizit übernommen*/
    -> BRN2EtherEncap(USEANNO true)
    -> [1]device_wifi;
```

In diesem Beispiel enthält das tls nicht nur die Packet-Generierungsfunktion *Packet::make()* sondern auch *BRNProtocol::add_brn_header()* und *BRNPacketAnno::set_ether_anno()*.

4.2 Aufbau eines BRN-Basispakets

Aufbau:



Beim Einsatz von DSR wird aus dem Ether-Header ein DSR-Header gemacht.

4.3 Besonderheiten in der BRN-Architektur

Das Click-Skript wird beim Aufruf mit run_sim.sh vorbereitet. Dabei werden einige Variablen durch spezifische Informationen über das Netzwerkgerät, den Knoten, etc. ersetzt. Die wichtigsten Informationen stehen in der mes-Datei (oder auch Nodetable genannt).

4.4 Hilfsscripte

Mittlerweile existieren eine Reihe von Hilfsscripten welche häufig benötigte Funktionalität implementieren. Diese script finden sich im *helper* repository unter *helper/measurement/etc/click* und können mit *#include* Anweisungen eingebunden werden. Wichtige Script sind zum Beispiel:

- brn/brn.click Definition von Konstanten, insbesondere die Konstanten welche im BRN Header gesetzt werden um die unterschiedlichen Protokolle auseinanderzuhalten (z.B. BRN_PORT_DSR). Diese Konstanten können von *Classifier* Elementen verwendet werden, um einzelne Pakete unterschiedlicher Protokolle getrennt zu behandeln.
- device/wifidev_linkstat Definition von WIFIDEV einer Abstraktion der WLAN Karte, welche sich unter anderem um das Link Probing kümmert. Die Zuordnung der Ein- und Ausgänge ist:

  output:

  - 0: To me and BRN
  - 1: Broadcast and BRN
  - 2: Foreign and BRN
  - 3: To me and NO BRN
  - 4: BROADCAST and NO BRN
  - 5: Foreign and NO BRN
  - 6: Feedback BRN
  - 7: Feedback Other

  input:

  - 0: brn
  - 1: client
  - 2: high priority stuff (higher than linkprobes)
- brn/helper.inc Definition von Macros FROMDEVICE und TODEVICE für unterschiedliche Szenarien (z.B. Simulation)
- routing/routing.click Abstraktion der unterschiedlichen Routingprotokolle. Stellt ein einheitliches Interface für alle Routingprotokolle zur Verfügung.

4.5 Simulation & Debugging

Wer eine genauere Analyse des Netzwerkverkehrs machen möchte, der sollte sich die Dumps anschauen. Das Dumping muss jedoch zuvor aktiviert werden im Click-Skript. Dazu fügt man diese beiden Zeilen an vorderster Stelle im Click-Skript ein:

```
#define RAWDUMP
#define RAWDEV_DEBUG
```

4.6 Problembehandlung

In den äußersten Fällen, da plötzlich Fehler unangemeldet auftreten und bei allen Debugging-Anstrengungen hartnäckig bestehen bleiben, hilft ein kompletter Neubau:

```
make clean
make elemlist all
```

Dies ist typischerweise der Fall, wenn die Initialisierungsliste des Konstruktors verändert wird. Zum Beispiel so:

```
BRN2DSREncap::BRN2DSREncap()
  : _link_table(),
        _me(),
        _neuer_Eintrag()
{
  BRNElement::init();
}
```

CHAPTER

FIVE

INTEGRATION VON NS2 UND CLICK

5.1 PollingDevice

5.1.1 Original

`FromDevice` scheduled sich solange selbst bis von oben kein Packet mehr kommt oder das device belegt ist (not ready). Im Simulator wird anhand der `ClickQueue` entschieden, ob das Gerät frei ist oder nicht. Ist kein Paket in der queue (max. 1 Paket (`QueueSize == 1`) ist das Gerät frei:

```
int LLExt::ready() {
    ClickQueue* pcq = (ClickQueue*) ifq_;
    if (pcq) {
        return (!packetpending_ && pcq->ready());
    }

    // No ClickQueue? Then we're always ready.
    return 1;
}
```

dequeue (ClickQueue) nimmt ein Paket aus der Queue, wenn die Uebertragung beendet wurde) und gibt es noch oben (click):

```
Packet* ClickQueue::deque()
{
    Scheduler& s = Scheduler::instance();
    double dcurtime = s.clock();
    fprintf(stderr,"Time: %f", dcurtime);
    Packet* retval = pq_->deque();
    return retval;
}
```

Das Gerät ist frei, sobald `ClickQueue::deque()` aufgerufen wurde.

5.1.2 Verbesserung

`TXFeedback` signalisiert, dass das Geraet wieder bereit ist. Maximal ist 1 Paket im Simulator

5.1.3 Problem

Wenn die Uebertargung abgeschlossen wurde und das Paket aus der ClickQueue, wird es nach oben gegeben. Das jedoch nicht (unbedingt) sofort, sondern erst nach einer Verzögerung. Im tcl file lässt sich diese Verzögerung einstellen:

set netifq	Queue/ClickQueue
set netll	LL/Ext
#LL set `delay_`	1ms
LL set `delay_`	0ms

Durch die 1 ms Verzoegerung gibt es einen unterschied zwischen Pollen und Nicht-pollen: Nicht pollen (txfeedback basiert) erkennt das frei-werden des Geraetes erst 1ms spaeter (nach dem `delay_`) Die alte Version hatte diese Verzoegerung nicht. Dadurch gibt es unterschiede in der Geschwindigkeit.

loesung (mehr Workaround): Verzoegerung auf 0ms setzten.

CHAPTER

SIX

FAQ

6.1 Simulation

6.1.1 Simulation starten nicht bzw. es werden nur leere Ordner erzeugt

Ursachen: Die brn-tools.bashrc wurde nicht in der .bashrc eingebunden.