Designing a Broadband Residential Gateway using Click! Modular Router

Jaime García, Francisco Valera, David Díez, Hugo Gascón, Carmen Guerrero, and Arturo Azcorra
Universidad Carlos III, Madrid, Spain,
jgr@it.uc3m.es
WWW home page: http://www.it.uc3m.es/~jgr

Abstract. Nowadays DSL and cable are the two main access technologies used to bring real broadband Internet to the residential environment but in a near future, with the deployment of FTTH (Fiber To The Home), transfer rates will not be a problem anymore and one of the most important challenges will be located on the provisioning of QoS (Quality of Service) facilities. In MUSE [1] project the overall objective is “the research and development of a future low-cost, full-service access and edge network, which enables the ubiquitous delivery of broadband services to every European citizen”. Within MUSE our main work is related to the design of a RGW (Residential Home Gateway) compliant with this QoS requirements and to the development of a RGW prototype based on the Click modular router [2]. In this paper we propose a new model to design RGW, based on Click, but using user level applications too. This is what we call an Hybrid Model and test validations are presented to support this innovative idea.

1 Introduction

MUSE project looks towards the future European broadband network with the idea of creating a low-cost and a high quality access for users. Although the access and the edge networks are very important research points, the devices located in residential environments are also key components so as to be able to provide good quality services, specially so called Residential Home Gateway (RGW).

The RGW is the first network device accesible by the user as is depicted in figure 1. Every home equipment will be connected (wireless or wired) through the RGW to a broadband but shared environment so, real-time signals such as alarms connected to the fire stations may be sharing the medium together with regular packets such as the messages sent by refrigerators when they detect some

---

1 MUSE is a large integrated R&D project on broadband access. Within the 6th Framework Programme, MUSE contributes to the strategic objective ”Broadband for All” of IST (Information Society Technologies) and it is partially funded by the European Commission.
food have to be bought. As it is seen, some kind of priorization is needed before packets are allowed to flow to the network. Moreover traffic shaping is needed to manage available bandwidth.

Besides all these functionalities, the RGW must perform many other tasks on behalf of the end user: auto-configuration, control, management, service configuration, etc. There are some functionalities not covered in this paper as the QoS service signalling. The IMS (IP Multimedia Subsystem) \cite{ims} will be used in MUSE for the service configuration. Actually, this specification must be adapted to a fix access scenario because nowadays IMS is just specified for the mobile and wireless LAN worlds. MUSE will work together with ETSI-TISPAN \cite{etsi} to make this adaptation. Obviously, the RGW prototype will incorporate this functionality but not at this very beginning stage.

For the implementation stage Linux has been chosen as operating system and it will run over a compact i386 PC compatible device \cite{linux}. Since the RGW have to manage low level packets (link layer) and Linux does not provide this manipulation natively, it was decided to use the Click modular router \cite{click}. This election will be explained in \ref{sec:click} together with the main characteristics of Click. Next section \ref{sec:model} is dedicated to the description of the model designed to implement the RGW. Section \ref{sec:tests} describes the pretended testing scenarios and the tests already performed. Section \ref{sec:conclusion} concludes with the summary of some important abilities acquired in this stage proposing new issues for the future work.

\section{Click Modular Router}

Click is a modular software router developed by MIT LCS’s Parallel and Distributed Operating Systems group, Mazu Networks, the ICSI Center for Internet Research, and now UCLA. A Linux kernel running Click is able to act as a router
in a flexible and configurable way. Routing tasks are made extremely fast, for software routers running on commodity hardware. On a 700 MHz Pentium III, a Click IP router can handle up to 333,000 64-byte packets per second [2].

A Click router configuration is based on interconnected modules called elements which control every aspect of the operation of the router: communicating with devices, packet modification, queueing, dropping policies, packet scheduling, etc. As modularity is the main advantage of a Click router, it is possible to write new modules in C++ with the desired behavior. The router configuration results from gluing elements together in a plain text configuration file using a simple script language.

Click can work either at user level or using a driver program, overriding linux kernel. The second option has been chosen as it lets the router deal with frames faster, avoiding kernel stack. Click router software works, by now, in a Linux kernel 2.2 or 2.4 with a kernel module, but will be soon available for kernel 2.6. As an open source-project many developers are working in order to make Click support kernel 2.6, IPv6 and some other features.

3 Click/Application Hybrid Model

For the RGW prototype we need a software capable of catching all packets at layer two level, modify them, reinject them into the network, sending them up to the upper layers and so on, so we decided to use the Click modular router software (more precisely the 'module Click', as the application Click way to work is not so useful for us). Although we chose Click, it may not be mandatory or desirable to develop new applications at linux kernel level due to two main points:

1. Programming new applications at the kernel level is sometimes very difficult.
2. The creation of new hardware and software network applications is also desirable, and they should be as independent as possible from low level packet facilities (these applications may be programmed in Java, for example).

To overcome these problems, it was decided to create a new hybrid model where no pure Click nor pure application model will be developed but a combination of these two ones. Figure 2 depicts the hybrid model. The figure shows three main boxes:

Click is the Click software router working at kernel level. It will receive every packet, will wrap them inside a new UDP packet and will forward them up to the Manager or Process application (this will be configured by the Manager).

The Manager will receive fresh packets from the Click module and process them. Depending on the packet characteristics, the Manager could configure the Click module to forward the same kind of packets to a certain process.

Processes n are the user level applications developed to assume certain functions.
This model proposal must be tested to assure that it can possibly be use and that it is not suffering any serious performance problem. When an application is programmed at the Click module level the time a frame expends crossing the Linux kernel is cancelled. This is why the delay imposed by the Manager must be estimated to validate the hybrid model.

![Fig. 2. Hybrid Model](image)

### 3.1 Testing the Hybrid Model Viability

Two tests were defined to validate the hybrid model and new techniques were also proposed to minimize the delay imposed by the packet walk-through between the Click module and the different applications.

**Reception Delay in Click: ToHostSniffers vs. ToHost** The main intention of the hybrid model is to help the programmer to develop RGW applications in an easy and fast way, but there are some issues that must be tested before the model is finally validated. In these tests we try to measure the delay increment introduced by the hybrid model due to the transmission of the packet from the Click layer to the application layer and then back to Click.

In a typical Click application, when a packet has to be relayed to the application layer, the packet has to pass through all the Linux kernel to arrive to the final application. Sometimes this is not the desired performance because of the delay imposed by some kernel operations like the TCP/IP protocols, IPTables, etc. This situation is depicted in figure 3.

In order to prevent packets from passing along the whole kernel, Click provides two different elements to pass the packet up directly to the application layer: ToHostSniffers and ToHost. ToHost element allows Click to send a frame directly to the TCP/IP linux stack responsible then for resending the frame to
the Manager application (that should be listening in a well known port). On the other hand, *ToHostSniffers* element lets Click to send a frame directly to an application level that must be configured to run as if it were a sniffer ².

In order to find out the delay introduced using these two Click elements the following tests were performed:

– The first step is the time synchronization between two hosts A and B.
– Then, from host A, a group of frames are sent towards host B (using a maintained ping), where Click is running. Transmission and reception times are taken down from the Ethereal sniffer output.

Table 1 shows some of the results obtained in this test (time values are expressed in microseconds):

The result of this test reaffirms the previous ideas about the delay imposed by Linux kernel because, as it is shown, the delay obtained using the element *ToHost* is around two times the delay obtained using *ToHostSniffer* (i.e. using *libpcap*[5]).

**Delay introduced by the Manager Application**  This scenario tries to test if the use of a user-level application called Manager does slow down frame management or not (should it really reduce the performance it could always be possible to manage the frames inside the Click module, without passing them to the user level although the flexibility of the development at the application layer would be lost). For this test Click has been installed in a computer with two different configurations:

² a sniffer is a program used to capture data in network, typically in shared medium. Used by network operators and maintenance personnel to troubleshoot network problems
ToHostSniffer & ToHost
\begin{tabular}{|c|c|c|c|c|c|}
\hline
Send at & Received at & Delay & Send at & Received at & Delay \\
792358 & 792370 & 12 & 238694 & 241465 & 2771 \\
242900 & 243063 & 163 & 419857 & 424088 & 4231 \\
374958 & 378089 & 3131 & 560984 & 569566 & 8582 \\
784325 & 785121 & 796 & 674645 & 678154 & 3509 \\
472349 & 476723 & 4374 & 309027 & 310941 & 1914 \\
545403 & 547961 & 2558 & 895921 & 898466 & 2545 \\
545403 & 547961 & 2558 & 44727 & 46137 & 1410 \\
763808 & 765446 & 1658 & 291357 & 293008 & 1651 \\
269527 & 271023 & 1496 & 112631 & 115348 & 2717 \\
819782 & 822051 & 2269 & 884444 & 892718 & 8274 \\
\hline
\end{tabular}

Average delay: 1901, 5\(\mu\)s \\
Synchronization Error: 0.160ms

Average delay: 3760, 4\(\mu\)s \\
Synchronization Error: 7.374ms

Table 1. Delay results

**Direct connection** frames are encapsulated in an UDP packet by Click, and then they are sent again directly to the same interface they came from.

**Manager connection** frames are also encapsulated in an UDP packet, but now they are sent to the Manager. This process can be carried out by a fake interface called fake0 (for example). When the Manager receives a frame, it returns it to the source machine through Click.

In both cases packets had the same size and they were sent by the same source machine. In order to perform the test, a high number of streams of 1000 packets have been sent, with different sizes in each experiment. Information was collected by the source machine with a sniffer application (*Ethereal* [6]). Both configurations are shown in figure 4 and figure 5.

Table 2 shows the results obtained in these tests.

<table>
<thead>
<tr>
<th>Packet size</th>
<th>Direct connection</th>
<th>Manager Connection</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 bytes</td>
<td>120(\mu)s</td>
<td>250(\mu)s</td>
<td>130(\mu)s</td>
</tr>
<tr>
<td>540 bytes</td>
<td>200(\mu)s</td>
<td>330(\mu)s</td>
<td>130(\mu)s</td>
</tr>
<tr>
<td>1060 bytes</td>
<td>290(\mu)s</td>
<td>430(\mu)s</td>
<td>140(\mu)s</td>
</tr>
<tr>
<td>1440 bytes</td>
<td>365(\mu)s</td>
<td>500(\mu)s</td>
<td>135(\mu)s</td>
</tr>
</tbody>
</table>

Table 2. Delay introduced due to the Hybrid model

Taking into account this result it can be concluded that the usage of the Manager increases the time around 130 – 140\(\mu\)s, (this is packet size independent result). Nevertheless, the Manager will not always directly resend packets, because sometimes it has to send packets to another user-level applications or Click.
modules (through a fake interface for example). Then, time used for managing frames could be similar in both cases (Click handling or Manager handling). It must also be noticed that we are working to achieve that packets arrive to Manager without passing by Linux kernel stack (using ToHostSniffer), and this will decrease reception times, but in this scenario just the Manager presence is tested.

4 Test Scenarios

Probed the hybrid model viability, the next step is to test the Click module stand alone. Figure 6 shows the test scenario used to test the Click modular router as a basic NAPT (Network Address Port Translation) device. The scenario is the following: a video server listens for HTTP connections on its 8282 TCP port. A
user in the laptop introduces in its browser the corresponding URL of the server. The laptop and the video server belong to two different networks so the laptop sends the HTTP request to its default gateway. The gateway device runs this simple Click program:

```
t1::Tee;
t2::Tee;
q1::Queue;
q2::Queue;
fhw::FromHost(aml0f,192.168.0.2/24);
fh1::FromHost(eth1f,163.117.140.38/24);
fdw::FromDevice(aml0);
fd1::FromDevice(eth1);
c1::Classifier(12/0800,-);
c2::Classifier(12/0800,-);
elementclass FixChecksums {
  // fix the IP checksum, and any embedded checksums that include data // from the IP header (TCP and UDP in particular)
  input -> SetIPChecksum
  -> ipc :: IPClassifier(tcp, udp, -)
  -> SetTCPChecksum
  -> output;
  ipc[1] -> SetUDPChecksum -> output;
  ipc[2] -> output
}
fdw->c1;
c1[0]->Strip(14)->MarkIPHeader(0)->StoreIPAddress(163.117.140.32,12)->
FixChecksums->EtherEncap(0x0800,00:40:F4:8A:54:F7,00:40:F4:67:3D:34)->q1;
fd1->c2;
c2[0]->Strip(14)->MarkIPHeader(0)->StoreIPAddress(192.168.0.1,16)->
FixChecksums->EtherEncap(0x0800,00:09:92:00:A9:A0,00:04:e2:2a:32:8b)->q2;
c1[1]->ToHost;
c2[1]->ToHost;
fhw->t1;
t1[0]->q1;
t1[1]->q2;
fh1->t2;
t2[0]->q1;
t2[1]->q2;
q1->ToDevice(eth1);
q2->ToDevice(aml0);
```

Basically, this program changes the IP source address for the upstream packets (from the private network to the public one) and the IP destination address for the downstream direction. The instantaneous traffic rate at the laptop input is depicted in figure 7 (high quality video) and 8 (medium quality). Neither delay or packet losses were noticed in these tests.
Another important tested value is the maximum number of packets that have ever been in a queue at once. It is a value stored in the Click environment and it is easy to read. Click creates a virtual file directory where elements can write and read different values depending on the element itself. For example, a queue element (q1 and q2 in our program) creates a read-only variable called highwater_length where it stores the maximum number of packets in a queue. At the end of these experiments just four packets were waiting at the same time to be transmitted. In other words, the Click module is able to process all frames at these rates.
5 Conclusions and Future Work

Click is a modular router designed for an easy configuration and high performance when it is used in the Linux kernel area. Nevertheless, working at the kernel space has some problems: it is difficult to program and debug at this level and the final application is too operating system dependent.

In this paper we proposed and validated the benefits of working with a hybrid model, using Click to capture low level frames (link layer frames) and process them at higher layers.

Two different tests were presented where the low delay imposed by the transfers between the Click level and the Manager application is probed to be acceptable.

In the future the Click core module must be well designed to allow an easy configuration and integration with the Manager and the different processes. An important study must be done in this field, because of the notorious impact this decision may have in the future work. It is also important to define an API to allow other developers to write Click-independent RGW processes.

Acknowledgments

This article has been partially granted by the European Commission through the MUSE project.

References