CLICK

A new programming language for developing software routers

VISHAL PRAJAPATI

April 10, 2009

CLICK

A new programming language for developing

software routers

MTECH SEMINAR REPORT

BY

VISHAL PRAJAPATI
08305030

UNDER THE GUIDANCE OF
PROF. D MANJUNATH
AND
PROF. ANIRUDDHA SAHOO

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY
MAY 2009

ABSTRACT

This seminar report describes Click - a new software architecture for building flexible and configurable routers. In recent years, several proposals emerged, and a very promising architecture is the Click Modular Router, which is not only easily extensible, but also very effective. When Click software is running as a Linux kernel module on conventional PC hardware, the achievable maximum loss-free forwarding rate for IP routing is 357,000 64-byte packets per second, more than commercial routers with far greater cost. The configuration optimization tools can raise this rate to 446,000 64-byte packets per second, enough to handle several t3 lines and 95% of our hardware's apparent limit. The click is made of individual basic blocks called elements, of which each performs simple routing functions like packet classification, queuing, scheduling, and interfacing with network devices. The elements are connected to each other by the links. These links are the route that a packet may follow while it is being analyzed or being modified. A router configuration is a directed graph with elements as its vertices and this flow of the packet as its edges. The language, in which these configurations are written is completely descriptive and support the user-defined abstractions. The simplicity of the language makes it readable by humans and can be understood or manipulated very quickly using tools.

Click Modular Router Webpage: http://read.cs.ucla.edu/click/

A few figures in the paper are taken from the PhD thesis of Eddie Kohler.

CONTENTS

1. Introduction

As now the routers are increasingly becoming more and more powerful, our expectations are also increasing with that. Initially the routers were just the hardware that was simply forwarding the packets that they have got. As routes evolved the responsibilities of them also increased. Boundary routers, which lie on the borders between organizations, must often prioritize traffic, translate network addresses, tunnel and filter packets, and act as firewalls, among other things. Furthermore, fundamental properties like packet dropping policies like RED and Differentiated service also need to be implemented at the router only. As the responsibilities are increased the per packet processing need also increased but as we have faster routers we have survived with that but as now the internet traffic is increasing day by day we have to develop the newer ways to handle that traffic. The most of the router are having very inflexible and static design so that the third party tool or the administrators may not be able to get the full utilization of the capability of the router. Sometimes the router needs to be extended with some functionality but for that we need to access some of the router interfaces that are very crucial, sometime it either they does not exists or they do not exists at right point.

In 1999 Eddie Kohler comes up with the new idea of using the general purpose computer as the router. He and some other colleagues at Laboratory for Computer Science, MIT have developed a new language called Click[1,2]. They kept the language completely declarative and simple so that it can be used easily. This paper describes the working with the click and will give the examples about how to develop the router quickly. The Click is modular language which is made up of fine-grained basic blocks called *elements*. This basic element are doing very small job in terms of processing or modifying the packets like increasing the TTL value, generate the error based on the flags of the packet. The Click is totally inspired by the actual flow of packets so they have make the Click is such a way that it actually simulates the flow of the packets in actual router. If you look at the router configuration of that we are developing via Click is a entirely a directed graph, so each and every path that a packet can take any of the path from input to the first element to the output of the last element. In the directed graph the vertices are the elements and the edges are the routes that a packet can choose any path base on the properties of it.

The paper is formatted in the following way. In section 2, we will be describing the architecture of the click and will discuss about the limitation of the element, the installation procedure in Linux and, how to compile and install the configuration in the usermode and kernel mode. The section 3 will cover the programming concepts of the click and will show how to make the router with simple example and gradually make a complete full-fledged working router. In section 4, we will discuss about the different extensions of the routers that can be created using Click. In section 5, we will discuss some of the experiments that Eddie Kohler and his colleagues have performed and evaluated the Click framework, and then will summarize.

2. Architecture

The basic block *element* is the main and centre of the entire language. It is created to do very simple functions at a time. These fine-grained elements can be connected to each other in any specific order and can be used as a single element and can do complex computations like IP routing. The click's configuration is a directed graph where the vertices are the elements and the edges represent the flow of path that a packet can take while traversing through that element.

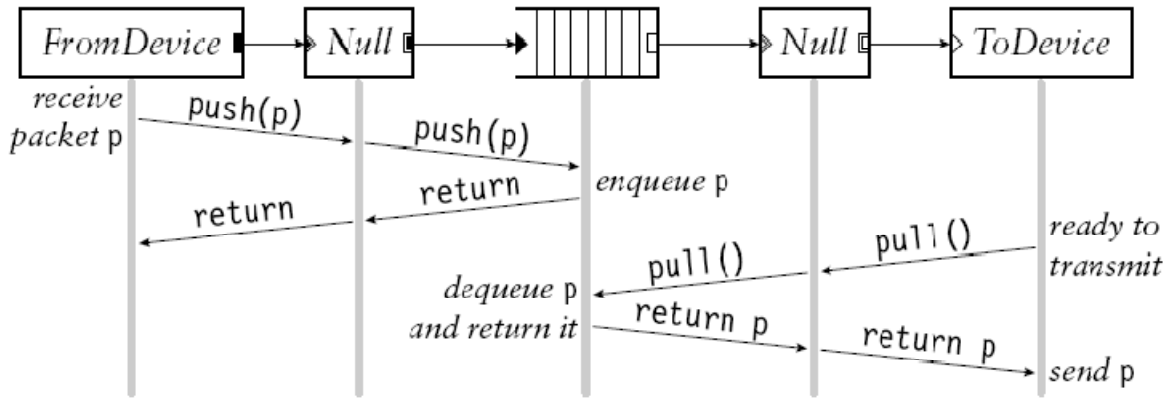

Figure 1: The Push and Pull port operations

2.1 Element

So as we know that the element is the basic block that performs the some operation on the packet and based on the packet property it forwards the packet to the next element via any of the output links that it have. Figure 2 shows the simple router element called Counter which simply counts the packet that passes through it.

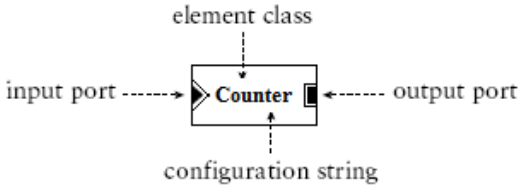

Figure 2: Basic Element

Element Class: It specifies the behaviour of the element like how many ports it has, what is the port type. It's is the basic structure that every element should follow to have unified interface to access it from any other element. Each Click element is class corresponds to a subclass of the C++ class Element, which has 20 virtual functions. But as the all the function are not necessary to implement by all the elements. We need to implement only 3 most important functions to devolve any new element, push, pull and run_scheduler. Following box shows the implementation of a simple element Hub that simply broadcasts the packet to all the outputs that it has got from any of the input with the subnet mask.

Ports: The ports are the connectors to the element these are the entry and exit paths of the packet to and from element. We will describe the ports in details later.

```
class Hub : public Element { public:

  Hub();

  ~Hub();

  const char *class_name() const
  { return "Hub"; }

  const char *port_count() const
  { return "-/="; }

  const char *processing() const
  { return PUSH; }

  const char *flow_code() const
  { return "#/[^#]"; }

  void push(int port, Packet* p);

};
```

Configuration String: This is an optional parameter that is used to initialize the basic element properties to improve the performance.

Method Interface: These are the methods that are provided by the particular element for inter element communication used for runtime arbitrary operation.

2.2 Ports

The click is the completely inspired by the natural flow of the packet, that a packet goes while it is processed by a traditional router. So the designers have developed two kind of the ports *push* and *pull* ports. Figure 1 shows the operations of the push and pull connections.

Push ports: The push ports are designed for simulating the sender initiated operations where there are no time constraints. The element is performing the operation only when the new packet is arrived at the input port and then after processing it, it forwards it to the next element.

Pull Ports: These are the ports that are designed to simulate the receiver initiated operations and where the time constraints are important. For example the Ethernet port is not always ready to send the data, so to send data from Ethernet port the element should always ask for the data from its previous element from its output port. This link is created via pull link.

2.3 CPU scheduling

Click schedules the CPU internally. Click has its own task queue where the one and only thread that runs the click on CPU. Click The task queue is scheduled with the flexible and lightweight stride scheduling algorithm. Here the tasks are elements that want the CPU to process the packets that it has got. This process starts from the first element that gets the packet from outside world. It process the packet and places the packet on next elements input port so the next element is put on the queue so that it gets the CPU time and can now process the packet. This process continues until that packet goes out from the last element. And this process also performed for each packet that comes in at the first element. Click runs on a single thread so any push or pull packet transfer method must return to its caller before another task can begin.

2.4 Configurations and installation

Click can run on two modes User-level and Kernel mode. This user-level driver can be used for the debugging purpose because any bad configuration that loaded by mistake to the user-level driver does not cause any effect on the normal communication because Click runs as a process on the Linux system and each of the Click packet still goes through the Linux networking stack. But the Kernel level driver replaces the complete network stack so that you can now run your own configuration and can make your PC works as a full-fledged router.

The installation of the router configuration can be done with simple steps. The router configuration file is a simple text file that is sent to the driver to parse. The driver then parses the definition, checks it for errors, initializes every element, and puts the router on line. It breaks the initialization process into stages. In the early stages, elements set object variables, add and remove ports, and specify whether those ports are push or pull. In later stages, they query flow-based router context, place themselves on the task queue, and, in the kernel driver, attach to Linux kernel structures.

The user-level configuration can be run by simply calling the click application; it processes the configuration and puts the router online, and more than one configuration can be run simultaneously. While in kernel level installation, as the kernel can run only one configuration at a time, we cannot run more than one configurations at a time. To install a configuration, the user writes a Click-language description to the special file */proc/click/config*. The installation of the new configuration destroys the older router configurations. But Click supports the two techniques for changing the new configuration without losing the older information.

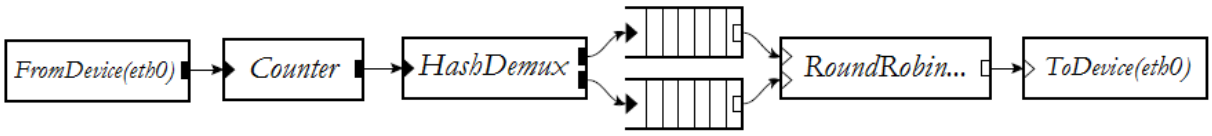

Figure 3: Simple Router Example - 2

Handlers: The Click elements can have any number of the handlers. And use can see this as the files in the Linux's */proc* folder as the files and can access any of the handlers. So we can use this files for the configuring the elements dynamically. For example, the *Meter* element has the functionality to send the limited number of the packets/second through it. So we can very well change the parameter dynamically so that it can send more or less number of packets.

Hot Swapping: The handlers can support the much of the finicality but adding a new element or removing an element from the existing configuration is very much complex to implement so the other option is the hot swapping. This finality is provided by Click but we need to be very cautious while developing the new configuration for hot swapping because the incorrect installation of the hot swapping configuration costs the loss of the data at the time in process at different elements. In this process the new configuration takes places the states of the older click elements to the newer one but for that your elements name should not different and there parent class should not differ.

3. Click programming language

This Click language is wholly declarative. It has features for declaring and connecting elements and for designing abstractions called *compound elements*, and that is all. Element classes are written in C++ using an extensive support library. The Click router configurations are also simple enough so that they can be optimized by the tools that can process the router configuration files.

The box is showing the syntax of the configuration that is used to create routers.

The connection statement "T [0] -> eth(0);" creates a connection from T's output port 0 to eth(0)'s input port. Elements must be declared before they are used in connections.

```
T :: Tee(2)              // declaration
T [0] -> eth(0);         // connection
T [1] -> Discard;        // connection
```

3.1 The simple example of router

Now let's look at some simple configurations of the router that can be created by the Click environment.

Example 1: This example cannot be said as a router but as a start let's just have an example where we are just counting the number of packets coming in and then discarding the packets. Figure 3 shows the actual router configuration. And the following block shows the configuration file of the same example.

Figure 4: Simple router - Example – 1

```
FromDevice(eth0) -> C :: Counter;
C -> Discard;
```

By looking at the configuration file and the graphical representation it can be seen that there is much same. So now onwards we will be giving only the graphical representation.

Example 2: Let's extend the same example with adding the virtual queue that implements a

version of Stochastic Fairness Queuing. Here the HashDemux element divides the flow of the packets in two flows based on the parameters that are passed to the element on the time of initialization and after that the RoundRobin element simply collects the packets from the both the queue one after another. As you can see here the connectors in the early part up to the queue are shown as the filled triangle inside the element box because these are the push links and the connectors after the queue are hollow because the links on that path are pull type. Because the ToDevice element sent the packet to the Ethernet port so we don't have any control over the Ethernet port behaviour. So whenever the device becomes ready to send the packet the ToDevice element pulls the packet from the previous device. And this sequence initiates the other flow of the task queue in the CPU scheduling algorithm.

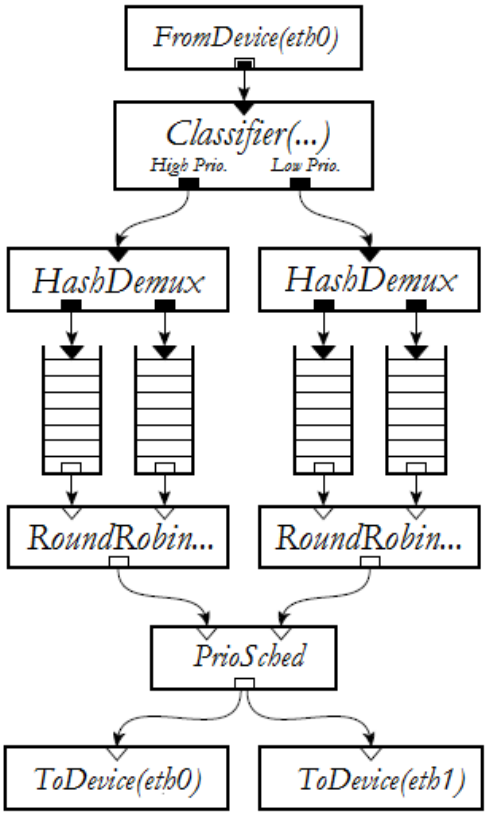

Figure 5: Simple Router Example – 3

Example 3: Now let's add the Classifier[3] element, this element simply processes the packet and sends the packet on the one of the output port based on the contents packet contains and the initialization parameters that given by the user. Also lets add the priority scheduler that always seeks the packet from the its first input port and if there is no packet on that input port then only it searches the packet on its second input port.

Example 4: Now let's add the packet size constraints and let's limit the network traffic from the ports.

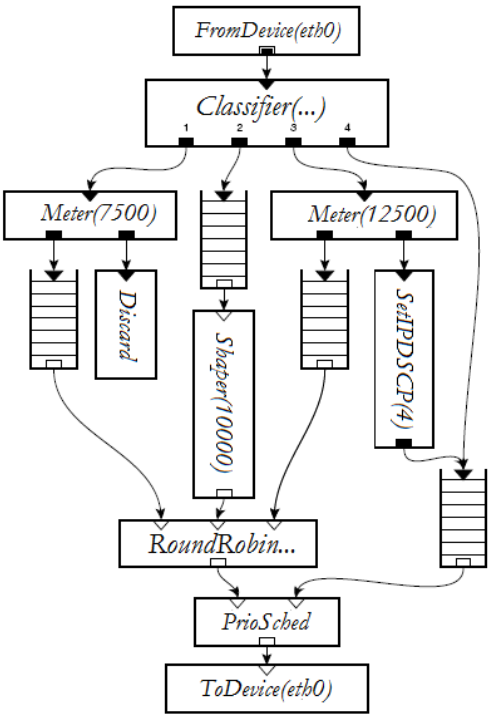

Figure 6: Simple Router Example – 4

So in figure 6 we are showing that the classifier distributes the packets in 4 different flows inside the configuration. The first flow coming out from the out port 1 goes through Meter element that has the property of sending maximum fix number of packets from it. So the element has the argument as 7500 says that it will allow only 7500 packets/second to pass from it. Same is the 12500 packets/second. The 2nd output of classifier is directly connected to a queue which is connected to Shaper case with the output 3 but here the limit element which also posses the same functionality of passing only the limited number of packets/second to its output port but here the packets are not dropped instead they are queued and will be dropped only when the queue is full. The queue is also having the verities of different dropping policies so that you can drop the packets from the front or from the end of the queue. And from the 3rd input

port the flow is first passed to the direct output competing with other similar flows but if the number of packets/second increases from 12500 then the overflowed packets are sent to a lower priority queue which is works as the best effort delivery.

Example 5: full-fledged IP router. Figure 7 shows the full-fledged bridge IP router connecting two networks. Almost all the network standards are taken care of in this router configuration. So let's describe some of the important elements.

FromDevice(…): This element gets the packet from the device, set as the parameter of the element. The element gets the interrupt from the device and stores the packet in a temporary queue and when the element gets scheduled by the Click then it sends all the stored packets to the next elements as per the packet content.

Classifier: This element checks the content of the packet and based on its contents it forwards the packet on one of its output. Here we are classifying the packet based on its type, and passes it accordingly to the next element.

ARPResponder: This element generates the replies of the ARP requests sent to the router. The table of contents of IP/Network to the Ethernet address has to passed as the argument as the part of initialization parameter.

Paint(…): This is the element just used as a flag for the internal use of the Click to mark the packet and you can identify the packet afterwards based on the colour that packet is painted of. Here we are checking that the packet coming from the specific Ethernet port should leave only from that Ethernet port.

Stripe(…): On the basis of what information is passed to the element at the time of its initialization, the Stripe element removes first few bytes of information. Here it removes first 14 bytes of information which is essentially removes the packet type the Ethernet address of the sender and receiver.

LookupIPRouter(…): This is the element forwards the packet comes to its input port by looking in to its IP layer and based in its destination IP and the configuration parameter set at the time of its initialization, it forwards the packet to one of the appropriate output port.

ICMPError: The element sends the reply based on the error that is set at initialization time. So whenever a packet comes in the reply for that particular packet is generated. Here while the conditions of PaintTee, IPGWOption, DecIPTTL and, IPFragmenter are not satisfies the error message should be generated so the second output port which is usually used as the outputting the error messages sent to the ICMPError elements.

ARPQuerier(…): As we have seen the element ARPResponder generates the replies of the ARP requests that are coming from the input port. This query messages are sent by the ARPQuerier element for gating the Ethernet address of the particular machine based on its IP address.

4. Extensions of element

As we know Click's design is completely modular, so that we can merge the different combinations and have a various combinations of functionality. We can have verity of the elements that we can merge such that we are able to built very complex systems by simply joining the different elements in specific order. Let's see some of the combinations that we can build by the Click.

4.1 Differentiated services

Example 4 in the above section is actual implementation if the differentiated service where we have made the 4 different flows treated differently based on the where they are coming from. So the most important element in this configuration is the Classifier that specifies that which packet should follow which path. And we can have various traffic limiters on the different paths. Here the first 3 flows are the having guaranteed service commitments and they each are having the limit of number of packet sent per second guarantee, which is managed by the Meter element, while the last flow coming out from the 4th output is the

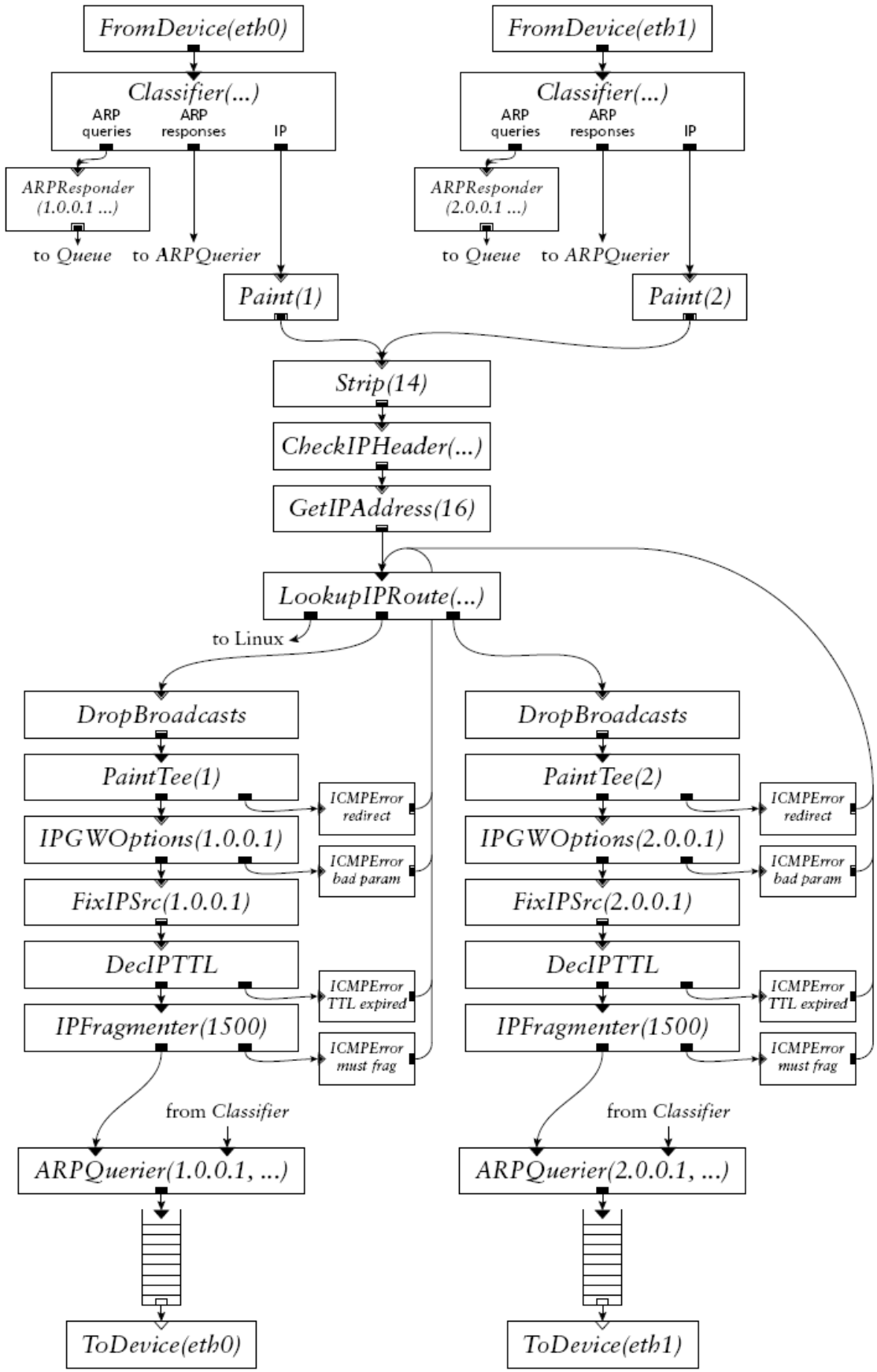

Figure 7: full-fledged IP router Example - 5

best effort flow which will be only served if there is no packet to send from the RoundRobin element, which is collecting the packets from the guaranteed customers.

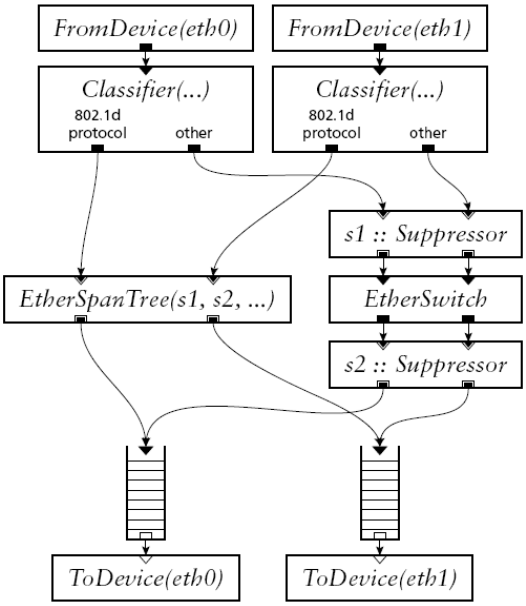

Figure 8: Ethernet Switch

4.2 Ethernet switch

Figure 8 shows the implementation of the Ethernet switch by the Click language. This configuration works as a learning bridge between two networks and forwards only those particular packets that are destined to that network. For example if some packet comes from the input port eth0 and sometime after if the same another packet comes from say eth1 having the same source IP address then the router made the entry to its routing table that the particular host is reachable from both of the links. This configuration is compliant to the original 802.1d standard, so that you can very well have you other ordinary router working with this Click configured PC router.

4.3 Mobility Extension

Click is also capable of supporting the IP-in-IP protocol so now we can also implement the Mobility extension that is required when a mobile user roam in some foreign network with the temporary address and still able to get all the packets that are sent to its original address. As Figure 9 shows the scenario where the mobile host 1.0.0.11 is currently located on network 43, where it has the temporary address 43.0.0.6. Its home node, 1.0.0.10, should encapsulate packets destined for 1.0.0.11 and forward them to 43.0.0.6.

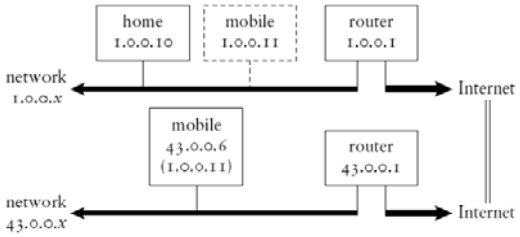

Figure 9: Network configuration for Mobility Extension

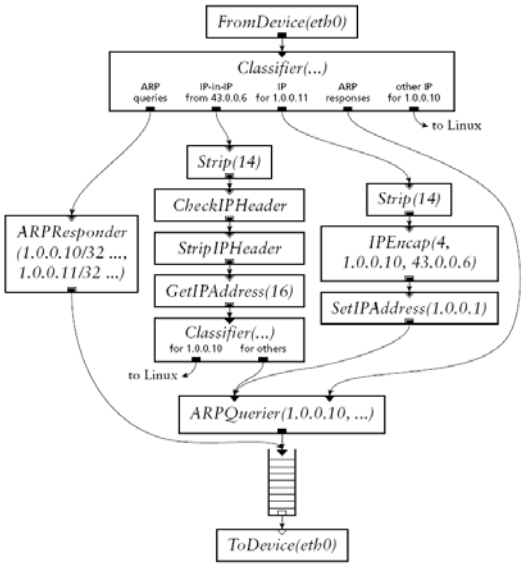

Figure 10: Click Configuration for implementation Mobility with IP-In-IP protocol

Assume that the configuration shown in Figure 10 is installed at the home node, 1.0.0.10. Here the main element that performs the major part is the Classifier by which the entire routing of the packet is taken care of. As we know that Click is

completely defined by the flow of the packet so if we can identify the packet where it is destined to then we can very well process packets by the use of suitable elements. So here we have identified 5 different flows that can be arrived at any time to the router. First flow is about the ARP query which is directly responded by the ARPResponder element. Second flow is showing the packets that are coming from the mobile node and are the IP-in-IP packets so the home node gets the original packet after stripping out the outer layer of the packet and then it treats the packet as normal packet received from the local user and do the according job. The third case is the type of packets that are received from other nodes and are destined to the mobile node. So here the home node itself creates the IP-in-IP packet and forwards it to the appropriate destination. The fourth case is for the ARP responses that need to be generated for each of the ARP query. And the fifth case is for any other packets type that arrives at the home node.

5. Evaluation

The graphs and statistical-figures of this section are referenced from the Eddie Kohler's Thesis and Paper.
This section describes the experiments that are performed by Eddie Kohler and his colleagues. We will discuss about Click's performance with the *Linux 2.2.16* kernel network stack. More details about the experimental setup are described in the Appendix. By performing the experiments the author states that the bottleneck is not the finicality or the CPU constraints that are not capable of supporting this much processing but the real bottleneck is the call of the virtual function, the IO latency. The statistics also suggests that the limitation of the forwarding rate is because of the PCI bus performance or the memory.

5.1 Forwarding rates

To check the maximum forwarding rate author had performed an experiment in which they wants to check that if we sent the packets from a saturated host how many packets can be forwarded without any loss. Here author sends the 64 bytes of packets because the intention is the forwarding rate is not depend on the bandwidth but here the main bottleneck is the per element processing of the packet so they kept the packet size minimum and performed the experiment. Here the Click configuration used is showed in Figure 7, but here the input and output interfaces are extended to 8 instead of 2. So there are 161 elements, 19 elements per interface and 9 elements are shared.

Figure 11 shows the graph that author got from the experiment here they have compared the 3 ways of implementations. The Linux line shows that it can forward very less amount of small size packets and as the input rate increases the forwarding rate decreases because of the use of the calling procedures of the Linux networking stack and slow table lookup algorithms. The calling and returning of the function call is the bottle neck of the forwarding rate. While in the Linux with the polling mechanism of the Click works far more batter then that of Linux basic driver.

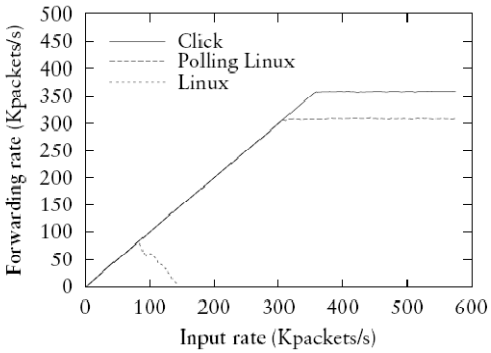

Figure 11: Experiment – 1 maximum forwarding rate

Let's look at the statistics of the experiment. Click's maximum loss-free forwarding rate is 357,000 packets per second. By having the poling algorithm of the Click on the Linux driver (Poling Linux) author had got the maximum loss free sending rate of 308,000 packets per second. Here author has stated that the no matter the forwarding rate of Click and the Poling Linux is more exciting but they are not scalable enough as the Linux IP routing table lookup algorithm. So the scalability is still the milestone to achieve in Click framework.

5.2 Comparison between IP router configuration and non-IP router configuration

Figure 12 shows the comparison between the IP routers and non IP routers configurations. Here we can see that the simple line is the Click router's simple configuration where there is no processing done at the element level. By that configuration the maximum loss free rate goes to 452,000 packets per second. Here the rate is limited by the PCI bus not by the CPU. Author had done the same experiment with different configurations as showed in the graph. By analyzing the graph we can say that the even the more complex flow of that a packet goes in or adding more number of elements in the path of the packet the performance is decreasing gradually. And even after reaching the maximum limit the performance does not goes down instead remains at the same state giving maximum possible forwarding rate.

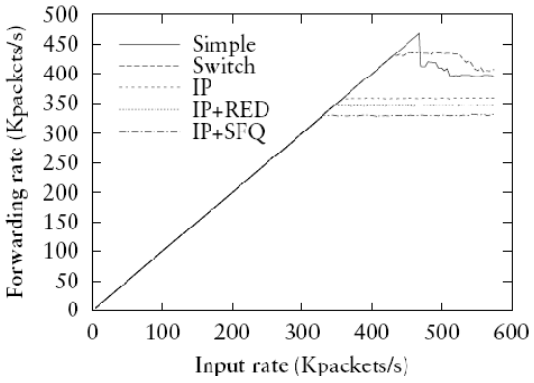

Figure 12: Experiment – 2 Comparison between the IP router and Non IP router

5.3 Forwarding cost per element

Author had take the experiment of they have already performed with the 8 interfaced IP routers and then checked the CPU time taken by each any every event. Figure 13 shows the results that they have got. Here the time is in nanoseconds and cost was measured by Pentium III cycle counters. Each measurement is the accumulated cost for all packets in a 10-second run divided by the number of packets forwarded. And as the counter is also costing a large CPU usage to process this information we can say that these figures are also more than the actual statistics.

There are various operations that cost the CPU timing they are as follows:

Packet Polling: This is the time, PollDevice takes to read the packets from the Tulip's receive DMA ring.

Task	**Time (ns/packet)**
Polling packet	562
Refill receive DMA ring	139
Click forwarding path: push	1572
Click forwarding path: pull	85
Enqueue packet for transmit	135
Clean transmit DMA ring	412
Total	2905

Figure 13: CPU time cost breakdown for the Click IP router

Element	**Time (ns/packet)**
Classifier	70
Paint	77
Stripe	67
CheckIPHeader	457
GetIPAddress	120
LookupIPRoute	140
DropBroadcasts	77
PaintTee	67
IPGWOptions	63
FixIPSrc	63
DecIPTTL	119
IPFragmenter	29
ARPQuerier	206
Total	**1455**

Figure 14: Execution times of some of the individual elements involved in IP forwarding

Refill receive DMA ring: After taking the packet from the Tulip's DMA ring PollDevice adds a new descriptors to the receive DMA ring so that the Tulip may receive more packets. The time taken to add new descriptor cost is marked as Refill receive DMA ring.

Click forwarding path: This is the time taken by the packet to traverse from the IP routers configuration and reached at the end. The time consumed by Linux's equivalent of the push path is 1.65 μs, slightly more than Click's 1.57 μs.

Enqueue packet for transmit: This is the time ToDevice spends placing a packet on the Tulip's transmit DMA ring.

Clean transmit DMA ring: ToDevice must also remove transmitted packets from the DMA ring; this cost is measured by "Clean transmit DMA ring".

The Click push forwarding path is by far the most expensive task. Overall, Click code takes 60% of the time required to process a packet; device code takes the other 40%.

Figure 14 shows the per element processing time. Where we can see that the processing time is different for each of the element. The most expensive element CheckIPHeader because of it first has to locate the packet start byte and then has to compute the offset to check the IP. So it requires more number of IO commands. While the IPFragmenter takes the smallest time because as we have used only 64 bytes fixed size packets none of the packet needs to be fragmented so the IPFragmenter simply forwards all the packets without any processing. As we required at least one virtual function call that only costs 29 μs so that performing any operation on packet will take at least the amount that every element is already taking so that we have very less possibility of optimization but still after applying the different optimization algorithms to the Click configurations author have achieved 30% of the improvement in the time requirement. The most optimization is done at the Classifier element that is 24% of improvement can be seen.

6. Limitation and future work

We can say that the Click has successfully achieved the goal of the keeping the language simple and readable. The language is also very well processed by the custom built tool. And can very well be parsed. Any tool never be complete there is always something to add into it, that can be a new feature, some plug-in, some bug, or some latest technologies implementation in the related field. So is the case with Click, Click has most of the elements that are required to implement most of the configurations available today. But some implementations required coarse grained implementation, like BGP routers concepts where there is need of dynamic implementation of the policies is not easily implementable with Click.

The compound elements can very well created with the use of abstractions that is provided by the Click but the compilation of the router still open ups the abstraction, so weather it's an abstraction at the user level at the internal level it is still the same full structure. So we are not getting any advantage of creating abstraction at the performance level instead this abstraction is creating the overhead making the more complex and hits the performance level, because it takes more time to process than that of native elements.

Click is having the internal CPU scheduling that makes it simpler to implement the routers in the learning mode. But as we move to the implementation at the real-time level we need to optimize the router for the batter performance and we do not have any control over the CPU scheduling

7. Related work

There has been several proposal suggested in this area. And many of them have got some advantages and some disadvantages. Most of the implementations are having not having enough communication between the network and user level. So they failed to get enough advantage of the network properties. For example x-kernel has the much of the same structure as Click, it has nodes instead of the elements of Click, it also generates the graph and the packets are passed through the graph but the graphs in x-kernel cannot be cyclic by the configuration and it has the layered structure that prevents some of the native functionality that we need to have while developing network router, like the IP

router needs to have recursive call to itself but it is not possible in this configuration.

One more implementation called Scout was also developed by Abhiram Khune, that is specifically designed to support the real-time and Quality of service applications. The system was powerful enough to have a firm end-to-end latency. So this was better suited then that of x-routers. It was also having support of the cyclic paths that was the main advantage over x-kernel mechanism.

8. Conclusion

Click is one of the most successful software router developed that is so close to the actual networking stack implementation. The flexibility, modularity and the easiness of developing new router configurations makes it more successful. Due to the modularity there is no limit of the extensibility of the router. It has the compatibility with the traditional routers that are available today in the market. And the easy to implement language of Click makes it even more attractive. The declarative style of the router configuration makes the configuration even readable by the tools so that it can be automatically parsed by the tools and also can be optimized by the language processing tools. The experiments that author had performed shows that the Click is having enough performance that can be even enhanced by the optimized use of the elements or by performing the optimization on the elements themselves to have a fine-tuned router for a particular scenario. By the experiments performed by the author we analyzed that by applying the optimization we can reduce the CPU usage by 34%. And the still we can achieve the maximum forwarding rate of 400,000 minimum-size packets per second.

A. Bibliography

[1] **Kohler, Eddie**. *The Click Modular Router*.
17th Symposium on Operating Systems Principles„ December 1999.

[2] **Kohler, Eddie, Robert Morris, Massimiliano Poletto**. *Modular Components for Network Address Translation*. Rome Laboratory under agreement number F30602-97-2-0288.

[3] **Kohler, Eddie**. *Click for Measurement*.
UCLA Computer Science Department Technical Report TR060010, February 2006

B. Appendix

1. Experimental Setup(*Taken from Eddie Kohler's Thesis*)

The experimental setup consists of a total of nine Intel PCs running a modified version of Linux 2.2.16. The PCs are patched by the Click modules with the kernel mode. Out of the 9 machines 4 works as the source hosts, 4 machines works as the destination nodes and one node works as the router. The router host has eight 100 Mbit/s Ethernet Controllers connected, by point-to-point links, to the source and destination hosts. During a test, each source generates an even flow of UDP packets addressed to a corresponding destination; the router is expected to get them there.

The router host has a 700 MHz Intel Pentium III CPU and an Intel L440GX+ motherboard. Its eight DEC 21140 Tulip 100 Mbit/s PCI Ethernet controllers [13] are on multi-port cards split across the motherboard's two independent PCI buses. The Pentium III has a 16 KB level-1 instruction cache, a 16 KB level-1 data cache, and a 256 KB level-2 unified cache. The source and destination hosts have 733 MHz Pentium III CPUs and 200 MHz Pentium Pro CPUs, respectively. Each host has one DEC 21140 Ethernet controller. The source-to-router and router-to-destination links are point-to-point full-duplex 100 Mbit/s Ethernet.

The source hosts generate UDP packets at specified rates, and can generate up to 147,900 64-byte packets per second; the destination hosts count and discard the forwarded UDP packets. Each 64-byte UDP packet includes Ethernet, IP, and UDP headers as well as 14 bytes of data and the 4-byte Ethernet CRC. When the 64-bit preamble and 96-bit inter-frame gap are added, a 100 Mbit/s Ethernet link can carry up to 148,800 such packets per second.