Multi-homed residential gateway

Authors:
Team 5
Timur Khusainov
Antonio Ramirez
Darwin Valderas
Michel Shahan
Hanbo Zhao

Principal:
Björn Knutsson

Coach:
Raúl Jimenez

October 22, 2007
Abstract

This final report summarizes the development process of the multi-homed residential gateway team 5 made during the CSD course. It is a set of scripts and daemons to configure any Linux platform to work with the needed functionality. Result... The result is the full-featured product attractive to users, as well as a starting point for several other projects and extensions.
Introduction

As the Internet becomes increasingly important for home users, we have seen an explosion in the market for multifunction routers for the home. These boxes often support several of the following functions:

- ADSL/Cable modem
- NAT functionality for sharing a single IP address across multiple machines in the home
- Firewall for protecting the local network
- Usage restrictions, i.e., to filter traffic to/from children computers based on content, destination or day/time
- Wireless base station
- Print server for sharing a printer on the local network
- Storage server to allow local network computers to share data

With the increasing importance of the Internet to the average family, it’s becoming increasingly common that you sign up with multiple service providers, both to provide redundancy when one ISP’s network is failing, and simply to provide more aggregate bandwidth.

This final report summarizes the work team 5 made during the CSD course. Chapter 1 provides a description of the project of multi-homed residential gateway, a study of related work and methods used to achieve the goals. Chapter 2 follows the development process, highlighting the sources found during pre-study, design decisions and implementation details. A review of the testing process is given in chapter 3 and the last chapter shows the possible extensions of the project.
Chapter 1

Project Description

1.1 Objective

The objective of the project is to explore and implement a solution, that extends the basic home router scenario with functions, which would allow a single home router to attach to multiple Internet providers and to manage traffic going in and out of the home network to maximize the utilization of the available bandwidth, achieve load balancing, robustness and flexibility. The project will be known as Multiple Internet Service Providers "MISP", because users can manage traffic between two or more ISPs. The project’s slogan "...you choose" was chosen because of the previously mentioned reasons.

1.2 Measurable goals

The goal of the project is to come up with the set of instructions and scripts, which would enable any x86-based PC running Linux OS to act as a home router capable of managing multiple outgoing Internet connections. It should support the following features.

- Trivial heuristics. The solution must allow alternating between the multiple uplink connections. It should also be possible to block one (or more) of the connections and use the rest. A protocol analyzer, such as Wireshark, will be used to verify the correct operation of this feature. For demonstration and teaching team evaluation purposes, traffic will be generated from the internal LAN, e.g. http traffic, to server X. It should follow ISP A for the first connection attempt. In the next attempt, it should follow ISP B. If the previously explained behavior is achieved, the feature is to be considered as approved.
• Applications filtering. Desired applications can be filtered out (based on the protocol or port number) towards a given ISP. Applications may also have a preferred ISP to transfer data to. For example, SMTP traffic should always go via ISP A, VoIP should always go via ISP B. Successful implementation of this feature will consist in generating e.g. VoIP traffic, as well as SMTP traffic. VoIP traffic should follow ISP A then, while SMTP must follow ISP B. If the protocol analyzer (Wireshark) confirms this behavior, the feature is to be considered as approved.

• Bandwidth utilization. If bandwidth utilization is unbalanced, i.e. there is a lot of traffic on the ISP B link, then the box should favor using the ISP A link. A bandwidth monitor software (e.g. Bandwidth Monitor NG) will be used to examine the traffic passing through several network interfaces. The tool should clearly show that the data starts to be sent over the second link, increasing its bandwidth utilization, at the moment when the first link has bigger utilization ratio.

• Web-interface. Planned functions are network configuration, packet and application filtering configuration. The web interface should simply present the features and they should be configurable/modifiable.

• Reliability. Try to maintain connectivity under any circumstances. When problems occur on one of the uplinks, traffic should be switched to the stable uplinks. Reliability can be tested by simulating one of the worst cases - a link to the Internet is down. The result should be that the traffic is switched to the other operational link.

• User and development documentation. Should cover all the functions and provide a thorough description of implementation, so that it could be developed further by other people.

• Advanced heuristics. The box should keep track of and remember bandwidth utilization patterns, and try to assign applications/connections accordingly. I.e. it could remember that connections to download sites will normally consume a lot of bandwidth. It was proposed as an extra feature by the principal and was treated as such. . . .

1.3 Related work and market research

The team conducted a preliminary market survey for the multihoming routers and gateways on the market and found some proprietary hardware and open-
source software solutions.

1.3.1 Proprietary hardware solutions

The following products were found: RADware LinkProof [12], PePlink’s Balance series [11]. These are proprietary solutions, closed-platform hardware boxes. Product design (e.g. LinkProof’s 19”-rack chassis), marketing (advertisement brochures contain added value information only for e-commerce, ISPs and enterprises) and sales models (limited retail distribution) are aimed at business users.

1.3.2 Open-source software solutions

The following products were found: Lokiwall [4], OpenWrt [6]

Lokiwall  Lokiwall is in many aspects similar to MISP. It is a set of scripts to configure firewall on Linux platform. It supports dual-routing and traffic shaping. The project is still in a very early stage though. There is no web-interface, and the scripts are hard-coded to work with only two uplinks. It is not a direct competitor to MISP, since it is focused firewall functionality, has a much more limited set of features and hard to configure.

OpenWrt  OpenWRT is a small linux distribution designed to work on certain models of residential gateways. It has many features similar to MISP (multi-homing, web-interface), but it is limited to a certain hardware implementations with closed architechture. So it is limited in flexibility.

1.4 Unique Contribution

This solution would intensify the competition between ISPs and empower end-users. Since they can easily compare and combine services of different ISPs, having a flexible and reliable Internet connection and select the ISPs whose policies best suit users.

Our solution is designed to work on Linux platform and to be distributed under GPL license, so it would be possible for anyone to use and develop it further. It could be installed on any x86 PC running Linux, and work as a residential gateway, providing many additional features, such as load-balancing, increased bandwidth and reliability, e.g. inbuilt bittorent client, web-server, ftp-server etc (these features are not in the scope of this project, more in the chapter 4). It makes our software implementation of multi-homed
gateway a foundation for a variety of more specialized solutions, useful for different categories of users.

1.5 Methods and Equipment

The target environment for this project will be a small PC (from now on we will refer to it as the box) running some Debian based Linux distribution, that will act as a gateway, it will be configured with three or more Ethernet network interfaces, two of these interfaces are going to be connected to different ISPs, and the remaining interfaces will be connected to an internal LAN with a private network range. The internal networks traffic to the internet is going to be NATed, and routed through the different ISPs, depending on the type of traffic, link features or other configurable options that will be present in the box, as mentioned in the project goals. Network configuration scheme is shown in figure 1.1.

![Network configuration scheme](image)

Figure 1.1: Network configuration scheme
Chapter 2

Project Development

Extensive/detailed description of the project (phases, milestones, presentations, etc) The MISP project is the result of an extensive search/research process. Information gathering consisted of technical HOW-TOs analysis, discussion forums postings, application and command manuals analysis and extensive search-engine searches.

2.1 Project Development Structure

The MISP project was divided into modules according to required functionality:

- Trivial heuristics;
- Applications filtering;
- Bandwidth utilization;
- Web-interface;
- Reliability;
- Advanced heuristics.

Each module was sub-divided into phases which are explained next.

- The Pre-study Phase
  During the pre-study phase, the team concentrated on gathering useful information. This phase also included group brainstorming and discussions, as well as small simple tests. The goal of this phase was to understand what the module was about. Furthermore, it was expected to find different approaches/tools to solve the same problem.
• The Design Phase
The design phase consisted of testing the gathered tools and approaches. The idea was to find out how they work in practice and how could they be used in hour implementation. The result of these tests gave us a better idea of how the given module could have been implemented. In some modules, different tools were combined while in other a simple tool or approach was enough.

• The Implementation Phase
This phase was the actual application of the gathered tools by following the design guidelines. Also, the implementation phase included minor testing to make sure that the expected behavior was achieved.

2.2 Trivial heuristics module

2.2.1 Pre-study
We started with searching the Web for multi-homing and dual-routing solutions on Linux. Several approaches were found:

1. A solution based on iproute2 was found in section 4.2 of Linux Advanced Routing and Traffic Control HOWTO [8];
2. A solution based on iptables was found in [13];
3. A solution, which uses iptables for marking connections and iproute2 for forwarding decisions [10];
4. OpenWRT;
5. Lokiwall.

The team then spent some time studying the usage and specifics of iptables and iproute2 utilities. It was also found out that all of the found solutions require some kernel modification, so we studied how to do that in the most effective way. Linux man pages and [9] were useful in that.

2.2.2 Design
We started with the first approach. The linux kernel had to be compiled with ROUTE target. We made a shell script, based on [8], and started testing. It soon became clear that it does not suite us, because it only gave statistic
load balancing, while we need sequential one.

For the second approach we needed Linux kernel patched with support for the ROUTE target and the "nth" match module according to [13]. These patches are available in NetFilter’s "patch-o-matic-ng" subversion module. We had several problems with it, since it turned out some kernel versions are incompatable with certain versions of ROUTE module, and nth module was recently superceded by statistics module included in iptables versions above 1.3.7. So it took some time to get the development system properly patched. When the testing started we have found, that we achieved the goal of trivial heuristics from the MISP box itself. But when we were testing from the computer in LAN using NAT we faced a problem. We couldn’t get responses from destination through of the uplink interfaces. It was dropped by MISP box.

Then we switched to the third approach and it worked the way we wanted.

2.2.3 Implementation

This module was implemented by using iptables version 1.3.8, since the Linux distributions we had on our PCs had iptables 1.3.6 we had to compile a new version, which required kernel compilation as well. Iptables makes use of the MARK and CONNMARK modules to mark packets and connections respectively. IPROUTE2 holds the routing information for every mark. In our implementation, the connections follow one ISP or the other depending on the mark they have. In order to achieve the alternating behavior between the two ISPs, one of every two connections will receive a mark of "1". In the same way, the second of every two connections will receive a mark of 2. Once the connection is marked, it needs to know which of the two exit paths it will follow. As a result, two separate routing tables were created; one table for each mark value. These tables contain the LAN’s network associated with the corresponding interface and a default gateway associated with the exit interface of the MISP box. In this way, the first table will contain one exit interface and the second table the other exit interface. Once the tables and the marks are set, iptables will match marked connections to the previously created tables. Since each table has a different default gateway, it will mean that connections marked with "1" will have a different gateway than those marked with "2". The process is illustrated in figure 2.1. For detailed command list see the development documentation in appendix A.
2.3 Application filtering module

2.3.1 Pre-study

The application filtering module’s pre-study phase involved research on various traffic/packet/connection filtering methods. The idea of this phase was to come up with applications, approaches or tools that would allow implementing the feature. The most significant results of this phase were the following:

- Layer 7 filtering approach using L7-filter, a classifier for Linux’s Netfilter [1]
  L7-filter presented a promising approach to deal with application filtering because of its pattern-filter design. L7-filter is an optional module of iptables that contains several build-in patterns for various applications. The matching is performed based on these patterns regardless of the protocol or port used. Furthermore, users can add or modify patterns according their needs making L7-filter flexible. L7-filter contains ready-made patterns for most popular applications such as Skype,
BitTorrent and MSN messenger among others. Finally, L7-filter is reasonably straightforward to use; it only needs information concerning source, destination and the pattern to be matched.

- Layer 4 port filtering approach. Traditional Layer 4 filtering was another possible approach to achieve filtering of applications. Most protocols can be filtered by knowing the corresponding port. Iptables is ideal to perform this type of filtering because of its firewall capabilities.

- Lokiwall firewall script [4] Lokiwall is a set of scripts providing firewall capabilities. These scripts can be modified according to the user’s needs and they are not more than a set of iptables rules.

2.3.2 Design

The design phase for the application filtering module consisted on discovering how the tools found worked in practice for our needs. This allowed the team to decide which approach was the most suitable. The combination of approaches was also a possibility. Layer 4 filtering was considered as the simplest, but limited solution to filter application. As it is problematic to filter applications using dynamic range of ports and peer-to-peer connections (eg. BitTorrent, Skype, SIP) using this approach. During the design phase this approach was tested with successful results for http, ssh and ftp protocols. Hence, we decide to spend more time exploring L7-filter. The first step was to enable the Layer 7 filtering support in the Linux Kernel as well as applying the necessary patch. Next, iptables had also to be patched with the needed files for L7-filter. For a detailed command list see the development documentation. Various tests were performed with L7-filter. The module seemed to perform as expected as long as one uplink was used. In multi-homing scenario we faced a problem though.

Problems with using L7-filter in multi-homing scenario We have realized there is a big problem with using statefull inspection (L7-filter) for application filtering. The reason is that Layer 7 patterns don’t match the handshake tcp packets, so there is no way of controlling through which interface the connection will be established. The following packets of the connection are matched, but if they are sent through another interface the destination rejects them. When we have made a rule to match SYN, RST, ACK, ACK ACK packets and send them through the same interface, as the application’s traffic matched by layer7 iptables module, it works fine. But then there is no way to distinguish between handshake packets from different
application, other than port filtering. This problem is illustrated at picture 2.2 with the example of http traffic.

Figure 2.2: Problem with Layer 7 filtering in multi-homing scenario

The team discussed this with the principal. The following two possible solutions were offered.

**Router approach** Host opens a connection, a router forwards it randomly to one of the uplinks. After some time there is a data packet coming in this connection, so the router can match it and see if the random forward decision was right or not. If it was wrong then a router sends RST to destination and makes a synthetic SYN packet (looses tcp sequence, must use tcp splicing) over the same socket. Problems: RTT, TCP options, loose TCP sequence need TCP slicing, will have problems if the destination consists of many hosts on one IP-address (server farm).

**Proxy approach** Router buffers all the packet in the connection until it can make a layer7 match and then it makes a forwarding decision and sends all the queued packets. Problems: splicing, TCP options, RTT
2.3.3 Implementation

What we have made so far is a set of rules to filter by port number. And then user can configure which interface will be a default one, so that all the traffic not matched by port filtering will go through that one. Port filtering can be combined with trivial heuristics, of course.

2.4 Bandwidth utilization module

2.4.1 Pre-study

The team came up with the following approaches:

1. Combined with trivial heuristics. User defines a bandwidth utilization ratio threshold (75 percent by default). MISP box checks the bandwidth utilization ration on all interfaces before making forwarding decision. If none of the interfaces’s utilization ration is beyond threshold forward by trivial heuristic rules. If some of the interfaces has crossed the threshold exclude them from trivial heuristic algorithm.

2. MISP box checks the bandwidth utilization ration on all interfaces before making forwarding decision. The it forwards each connection to the uplink with the least utilization ratio.

3. Forward connections to the certain interface until it reaches utilization ratio threshold, then move to next one, continue like this in round robin fashion.

Also we realised that we can use the queuing theory algorithms to optimize bandwidth utilization. Having these in mind we started a research for the useful information in the Internet.

2.4.2 Design
Chapter 3

Testing

Testing consisted on experimenting with different situations, which could cause a fault in the given module. It was scheduled to perform several trials with different protocols and simulate various scenarios in which the module was expected to follow a certain behavior.

3.1 Test of the trivial heuristics module

3.1.1 Test-bed

Picture 3.1 illustrates the infrastructure of the test environment. The workstation named Alice, which is located on the left hand side of the MISP box, represents the internal network (i.e. LAN). On the other hand, the PC at the very right end called Bob stands for a host on the external network. All the traffic of the conversation between Alice and Bob goes through the MISP box which functions as the gateway of internal network. The MISP box has two links to the school network simulating two links to different ISPs.
3.1.2 Testing tools

**D-ITG**  D-ITG[2] (Distributed Internet Traffic Generator) is a platform capable of producing traffic at packet level accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) with random variables. D-ITG supports both IPv4 and IPv6 traffic generation and it is capable to generate traffic at network, transport, and application layer. D-ITG is conceived to be used as a distributed active measurement tool, able to perform measurement of one-way-delay (OWD), round-trip-time (RTT), packet loss rate, jitter and throughput using the various platform components: sender, receiver, decoder and log server.

**Features:**

- Transport layer protocols supported: TCP, UDP, ICMP, SCTP and DCCP.
- Application layer protocols statistically replicated at packet level: Telnet, VoIP (G.711, G.723, G.729, Voice Activity Detection, and Compressed RTP), DNS, network games, etc.
- TOS (DS) and TTL IP header fields can be set.
- Multiple traffic flows can be generated simultaneously thanks to its multi-thread implementation

**Network Traffic Generator**  Network traffic generator [5] follows a server/client model for generating high volumes of traffic on a network. This could be used to test the ability of our gateway to handle continuous high traffic loads.

3.1.3 Test Procedures

First of all, it is essential to get the script running on the MISP box. Meanwhile, two Wireshark [7] (a network analyser program) instances are opened on both interface one and interface two of the MISP box to capture the traffic for further observation and analysis. Then different types of traffic are generated from Alice to either Bob or public hosts.

**Transport layer test**  As iptables mainly operates on transport layer, this is the layer the major focus of our test will be put in.
**ICMP** Ping google.com from Alice. From the captures on wiresharks, it can be seen that every other ping went through interface one while the rest went through interface two.

**TCP** A script file is prepared to simultaneously generate more than one flow, like those shown in the following examples. Six TCP flows with different constant bit rate are generated. Thousands of TCP packets are sent per second through each traffic flow with constant inter-departure time between packets. The size of each packet is equal to 512 bytes.

According to the log file on the receiver, three packet flows went through the interface one and the other three went through interface two. The delay and jitter are acceptable.

**UDP** Six UDP flows with different constant bit rate are generated. Thousands of UDP packets are sent per second through each traffic flow with constant inter-departure time between packets. The size of each packet is equal to 512 bytes.

According to the log file on the receiver, two packet flows went through the interface one and the other four went through interface two. The delay and jitter are acceptable.

**Application layer test** Hybrid traffic flows are generated. Each of them has flows with more than one application layer protocols. Different combinations have been created and tested.

According to the log file on the receiver, VoIP and DNS packet flows went through the interface one and Telnet went through interface two. The delay and jitter are acceptable.

**Stress test** A Perl script is used to download multiple files from a set of servers using CURL. Before running the script, one single CURL command was executed several times to make sure that the traffic of one downloading session pass through only one interface. Then several Perl scripts are executed in order to fetch files from multiple sets of servers simultaneously.

The downloading tasks were separated to go through two interfaces which met the expectation. However, the MISP box crashed twice while the traffic was heavy.
Chapter 4

Future work

In this chapter we define some possible extensions to our project and proposals for future work.

1. Implement a layer 4 NAT-box to solve problem with application filtering on layer 7, as described in section 2.3.2;

2. Bittorrent client which makes use of multiple uplink connections;

3. Sharing of uplinks between neighbours via LAN/WLAN;

4. Integration with TSlab’s ISP in-a-box project;

5. Implement network storage solution using a hard drive of the PC used as a gateway;

6. Load balancing for web-servers inside corporate networks, such as Linux Virtual Server (LVS) [3], but keeping the multiple out-inbound link;

7. Implement multi-homing between Ethernet vlans.
Chapter 5

Conclusion
Bibliography


