Active Probing with ICMP Packets

Internship in the Department of
Electrical and Electronic Engineering,
University of Melbourne

Supervisor : Darryl Veitch

Fabien Viger
fabien.viger@normalesup.org

July 14, 2003

Contents

Chapter 1

Introduction to Active Probing

1.1 The Internet Protocol

1.1.1 Introduction to IP

The Internet is a worldwide network that could roughly be described as a set of hosts, all connected to a core network composed of interconnected routers. When a host wants to send data to another one, it just gives it to the Internet, with the destination address, and the data will automatically be forwarded to the destination. Practically, a router that receives data addressed to X will know which router to transmit the data to, and so on, until X receives the data. This process is called *routing*.

Of course, the data cannot be sent 'as is', some information needs to be appended to make it understandable. That's why data is cut into small pieces (called *packets*), and every packet is encapsulated before its actual sending. This encapsulation has several levels, as shown in Fig 1.1.

1. The destination host must be able to understand what this data is for. Indeed, a host can have many processes receiving data at the same time. Therefore, the raw data must carry some information about its function. This is the purpose of the high-level protocols, such as UDP or TCP, for example.

2. To carry this encapsulated data to the destination, the Internet has to know information such as the destination address, the packet length, the type of service (TOS) the user needs (high bandwidth, low latency, reliability . . .). This is provided by the IP encapsulation. The result is an *IP packet*.

3. To be actually transmitted, the IP packet, considered as a byte stream, has to be formatted, transmitted, and then extracted from the formatted data. An Ethernet link simply adds a 14 bytes header before transmission and removes it just after, but other links like ATM have very different methods.

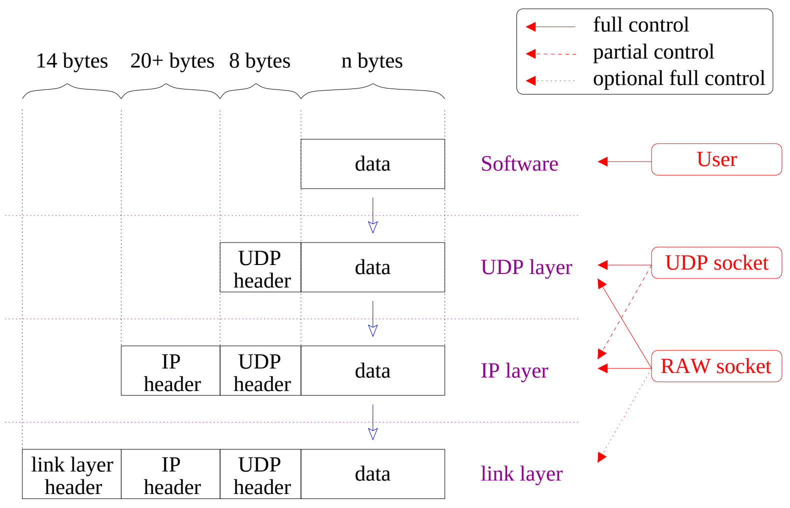

Figure 1.1: The encapsulation sequence for a UDP packet transmitted via an Ethernet link

IP, as described in [9], provides essentially automatic forwarding to the destination and automatic fragmentation and reassembly of oversized packets. However, an IP header actually carries more information than the destination address, TOS, packet length and fragmentation status. This includes the source IP address (the address of the sender), an identification field, a protocol field that specifies the high-level protocol used, and eventual options. Another essential field is the Time-To-Live (TTL), that will be decreased at every IP router. If the TTL of an IP packet reaches zero, the packet will be dropped (A router receiving a packet with TTL=1 will decrease the TTL to zero, and immediately drop it). As the TTL can not exceed 255, this provides a security against infinite loop that could result from bad routing. All this, and more, is described in [1].

The aim of this project is to explore some new possibilities for *probing* the Internet, based on a wider use of the various features that IP offers. Indeed, IP is more complex than a simple destination tag, and we can take advantage of that : if we can make the routers do something else than simple forwarding, we should be able to find some tricky methods to extract information about the router itself. In the remainder of this intoduction we will show some techniques we already use for that purpose, and present some new ideas that should allow us to extend the possibilities of active probing.

1.1.2 ICMP : Internet Control Message Protocol

As far as we have described IP, there is still no way to *interact* with the Internet. That's precisely the purpose of ICMP (Internet Control Message Protocol). ICMP is implemented as a high-level protocol (see Fig 1.2) but is actually part of IP. It was designed to provide various control features : error messages, requests and replies. An example of error message is the ICMP packet a router will send back to an host H if it received a packet from H that couldn't be forwarded.

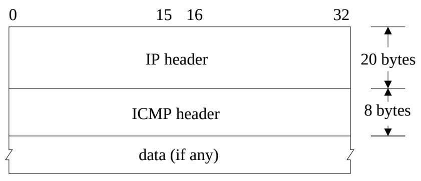

Figure 1.2: Structure of an ICMP packet

This can occur for several different reasons, resulting in different types of ICMP error messages. The one we will focus on later is the one generated by an IP packet dropped because its TTL reached zero. And an example of request is the one used by the well-known *ping* program : any host or router that receives an ICMP *Echo Request* from X is supposed to send back an ICMP *Echo Reply* to X. All the communication between routers (to update routes, or to detect malfunctions, for example) are made with ICMP. See [10] or [1] for further information about ICMP.

1.2 Active Probing : an Overview

Even ICMP offers to the user very limited possibilities to interact with Internet routers. That's why the only tools that non-privileged users may use to probe the details of the Internet structure are based on end-to-end measurements. This measurements are made at the end points of a connection : assistance from the network core is not required. Active probing methods are particular cases of end-to-end measurement, consisting of sending an artificial flow of traffic to a controlled host, which can allow us to get some clues about what is inside the blackbox.

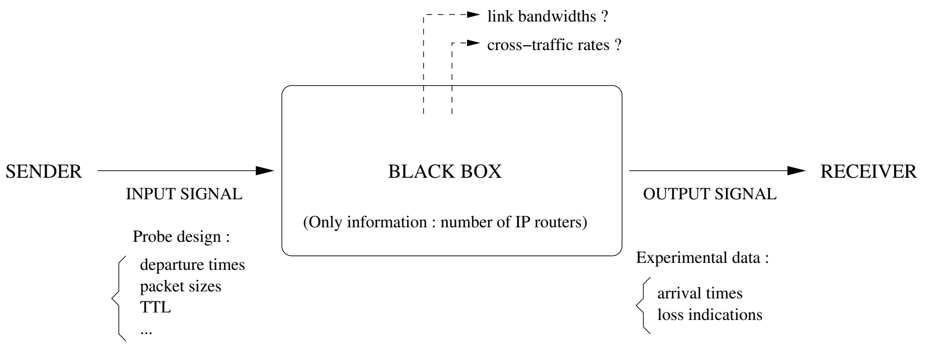

Figure 1.3: A *Black Box* problem

The main desired characteristics of active probing methods are :

Non invasiveness : they should not add a significant stress on the network. This is a necessary condition for a wide deployment of active probing tools.

Efficiency : their duration should scale according to both the users needs and the network volatility.

Various information may be extracted from the network, using active probing. This includes for example hop bandwidth, link rates, or network load. The knowledge of those could allow a more efficient and reliable use of the Internet :

For users :

- Flexible Internet applications and protocols could adapt themselves to available bandwidth variations. This could become crucial for the transfer of data with real-time constraints and/or significant volume.
- The cost of Internet access is generally connected to its bandwidth. Users need to check whether the service offered by their providers matches its price.

For providers :

- Dynamic bandwidth and load estimation could be used to design more efficient and robust routing protocols.
- Using traffic statistics, providers could locate sources of inefficiency and plan capacity upgrades.

The complexity of the Internet makes it very hard for active probers to extract useful information. Indeed, we want to extract information from the routers we don't control directly (otherwise, it would be easier to call the manufacturer). And we can be sure that, because of the extent of the Internet, there will always be some *cross-traffic* (packets coming from other users) on a route we don't control entirely. The cross-traffic is of course unpredictable, and the network load varies a lot, so that a packet sent one second after an identical packet, on the same route, may take significantly more (or less) time to arrive than its predecessor. The difference between routers, the diversity of the cross-traffic (video, http, ftp), the complexity of the Internet connectivity graph, the different types of links (optical, multiplexed, ATM, ...), all these factors are part of the challenge that any active prober faces.

1.3 An Active Probing Project

Our active probing project belongs to CUBIN (Centre for Ultra Broadband Information Networks), which is part of the Department of Electrical and Electronic Engineering of Melbourne University. In this section we give an example of actual active probing. This will introduce the active probing background and the concepts used in a convenient way.

1.3.1 Testbed

The following components are always present in active probing experiments :

- The *Sender* emits a scheduled and pre-defined probe stream. It must be accurate enough to send probes at the desired times, with negligible error.

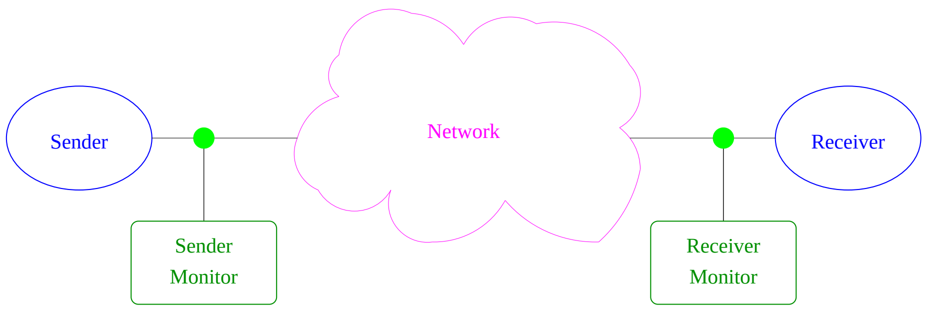

Figure 1.4: Active probing components

- The *Sender Monitor* timestamps every probe departure. If this differs from the scheduled sending time, the monitor is the one we trust.
- The *Network* forwards the probes to the destination. This can be any IP network : LAN, Internet . . . .
- The *Receiver Monitor* timestamps every probe arrival.

To run an experiment, we first have to design a given probe stream, then to give it to the Sender. When it's done, the experimental data are collected in the *Monitors* and the data can be post-processed for analysis.

1.3.2 Timestamping

Timestamping is very important as it is the only measurement we have, apart from loss. Moreover, it requires a very good accuracy, as the Internet goes faster and faster : the transmission of a 100 byte packet on a Gigabit link (that are present on most of the international routes) is made in less than one microsecond. Three levels of accuracy are available : the standard Linux timestamping, the TSC-based timestamping, and the DAG card. The TSC-based timestamping, based on the TSC register in modern CPUs, which is incremented at every CPU tick (less than 1 *nanosecond* in any CPU faster than 1GHz), requires a TSC-modified environment, that is the combination of slightly modified Linux kernel and NIC (Network Interface Card) driver. It gives very good accuracy, by considerably reducing the noise coming from the operating system and the clock skew. This is fully described in [2]. The DAG card, developed by the Computer Science Department of the University of Waikato, New Zealand, is a full-hardware solution, that acts as a passive tap recording any network traffic without disturbing it. For more information, see [5]. Coupled with a GPS timer, it provides “perfect” timestamping that allowed us to check the accuracy of the previous solutions. Most experiments are made with the TSC-based timestamping.

The timestamps we collect are the arrival times τ^* and the departure times τ (see Fig 1.3). The Departure (resp. Arrival) Time of a packet is the time when the packet has fully left (resp. arrived in) the *Sender* (resp. *Receiver*) As the clocks on the *Sender Monitor* and the *Receiver Monitor* may not be synchronized, we use the inter-departure times (IDT) $t_i = \tau_{i+1} - \tau_i$ and the

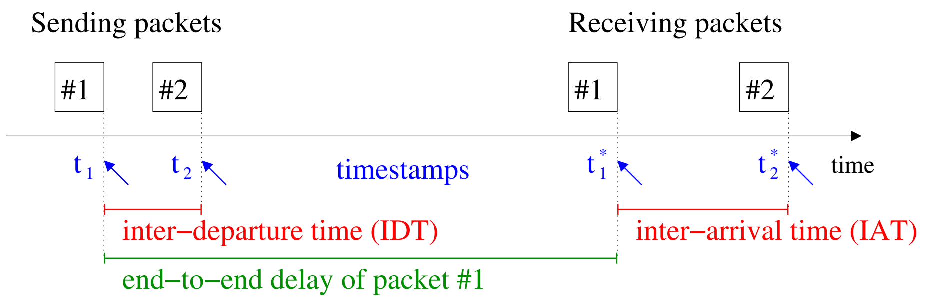

Figure 1.5: Timestamps and key values for active probing

Inter-arrival times (IAT) $t_i^* = \tau_{i+1}^* - \tau_i^*$ of the packets, rather than the end-to-end delays $d_i = \tau_i^* - \tau_i$ (the delays are the difference between the departure and the arrival times of the packets). The delay variation δ of two packets is the difference between the delays of both packets, which is exactly the difference between their IAT and their IDT.

$$\delta_i = d_{i+1} - d_i = ((\tau_{i+1}^* - \tau_{i+1}) - (\tau_i^* - \tau_i) = (\tau_{i+1}^* - \tau_i^*) - (\tau_{i+1} - \tau_i) = t_i^* - t_i \quad (1.1)$$

1.3.3 Probe parameters

The probe stream is totally defined, probe by probe, by three parameters. For probe i, we have :

- The inter-departure time t_i of probe i
- The *Size* of probe i. Here, we call size the total length (in bytes) of the probe, considered as IP packet (eg. link layer header excluded).
- The *TTL* of probe i.

Those three parameters allow us to design many experiments, and most of the active probing techniques can be implemented using UDP probes with the above parameters.

The inter-departure time t is more important than one would think at first look. Typically, it is either big (i.e. around 1 second) or zero. A big IDT means that the probes are going to be independant : the network conditions seen by the probes won't be correlated, thanks to their volatility. We can also send the probes back-to-back. In this case, they will probably encounter similar network conditions, especially if they stay back-to-back. To send back-to-back probes, we set the IDT to zero. The actual IDT is not zero, but equal to the transmission time (also called *service* time) s_2 of the second packet. Setting the IDT to zero is just more convenient. We can for example send a stream of independant pairs of probes, where probes are back-to-back within pairs, as shown in Fig 1.6.

The back-to-back probes are not easy to send *really* back-to-back, since the software needs to send two probes in a very short time ($5\mu s$ for 56-bytes probes). The software we use obtain various performance in sending back-to-back probes, depending on the hardware. On Pentium-III 500MHz with Linux, TSC-modified kernel 2.4.14 and 100Mbits LAN, we couldn't send probes closer

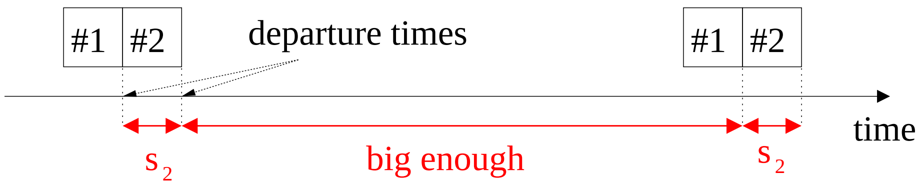

Figure 1.6: Independant Pairs of Back-to-Back Probes. The sizes of probes represent their service time.

than $25\mu s$. With an Athlon 1.7Ghz, Linux kernel 2.4.20-8 and 100Mbits LAN, that was $7.5\mu s$. But we can still use a trick : just before sending back-to-back probes, we send a 1500 bytes packet to a LAN neighbour (so that it doesn't interfere with our probes on the Internet route). This will cause our two probes to be queued in the NIC sending queue, because a 1500 bytes packet takes more than $100\mu s$ to be sent. And as soon as the big packet is finally sent, the two probes are sent just after, hopefully back-to-back. We ran some tests with the DAG card to validate this method, and the results were satisfying (at least 99% of the probes were really back-to-back)

1.3.4 Experimental data analysis

When an experiment is done, the experimental raw data (probes departure and arrival times, τ_i and τ_i^*) is first filtered to eliminate some outliers results, such as lost probes (i.e probes that the *Receiver Monitor* didn't see), or probes that have not been sent at the correct time, for example. Then, we can analyse the filtered data the way we want to. To give a practical example, a typical experiment is to send many probe pairs with the same given inter-departure time, and to plot the inter-arrival time histogram. The two following graphs are from two such experiments, both made with 114 bytes (912 bits) probes, sent in pairs, with an IDT of $50ms$ for the first one and of $0ms$ for the second one. The route was from the University of Waikato, New Zealand, to CUBIN.

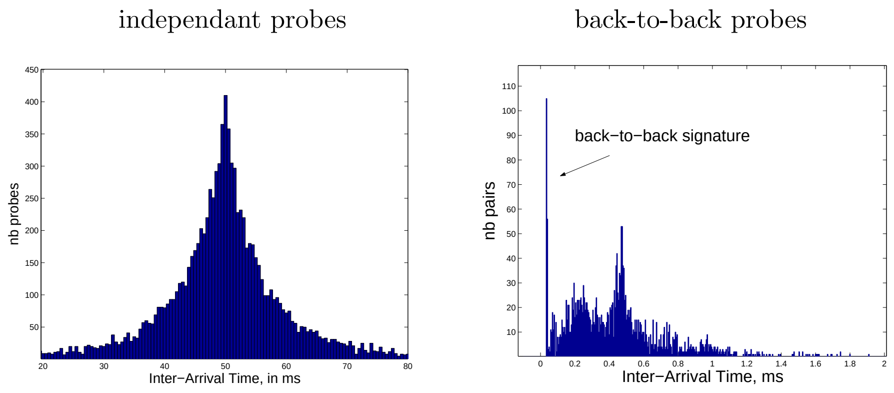

Figure 1.7: Experimental traces : histogram plot of the inter-arrival times t_i^* of pairs that were sent with fixed t_i (left : $t_i = 50ms$, right : $t_i = 0ms$)

The first graph shows a nice symmetric distribution centered at $50ms$. This is typical of the *independance signature*, as the delay variation $\delta_i = t_i^* - t_i = d_{i+1} - d_i$ of the probes is the difference between two quantities (the delays d_i and d_{i+1}) that both follow the same distribution. This is true thanks to the independance of d_i and d_{i+1}.

The second graph exhibits an interesting peak at $9\mu s$. We also see that this peak is at the smallest inter-arrival time observed. The immediate conclusion is that it represents the probes that arrived as close as possible : back-to-back. Therefore, the peak's location is the inter-arrival time of back-to-back pairs, which corresponds to the transmission time of the second probe in the pair. We can also guess the last link bandwidth, in this case we obtain, with the peak at $9\mu s$ and a size of 912 bytes, a bandwith of $\sim$ 100Mbits. This is what we expected.

$$bandwith = \frac{2^{nd}\ probe\ size}{Peak\ location} \tag{1.2}$$

1.4 Aim of the Project : A Wider Range of Probing Tools

Many techniques can be implemented on this basis, like the packet quartets method [3], that gives an estimation of router bandwidth. However, it makes sense to enlargen the playground even if it hasn't been fully explored yet. In this project, the aim was to gather as many ideas as possible to expand the set of active probing techniques. This includes creating new methods, of course, but first of all we had to upgrade our active probing basic tools in order to give us more degrees of freedom in the design of experiments. This is what is described in this section.

1.4.1 Choosing the protocol

All the probes we had been using so far were UDP packets. This protocol was chosen because of :

1. its simplicity
2. its flexibility : the smallest valid UDP packet (eg. with no data) is only 28 bytes long (link layer header excluded), when the smallest TCP is 40 bytes long. Since the smallest valid IP packet is only 20 bytes long (a simple header), it makes sense to be as close from this limit as possible.
3. its wide implementation : UDP belongs to the three most used Internet Protocols (TCP, ICMP, UDP)

However, it can possibly make sense to use other protocols, and there's no reason to limit ourselves to UDP.

1.4.2 Spoofing

Probes are either *direct*, i.e. they reach the receiver, or *hop-limited*, i.e. they are dropped on their way to the destination because of their too small TTL. Actually, they are not simply dropped : the router that drops them also sends an error message – an ICMP *Time-Exceeded* packet – back to the sender, as shown in Fig 1.6.

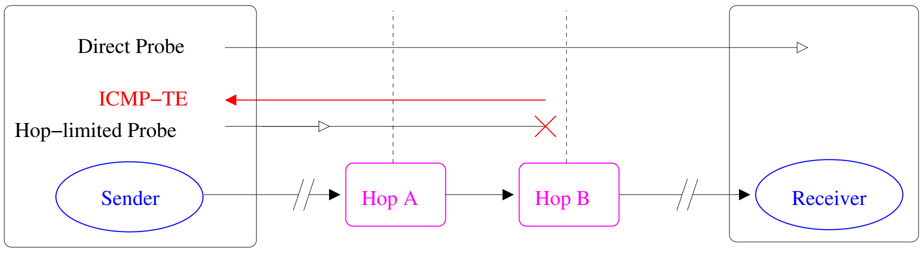

Figure 1.8: Direct and hop-limited probes

Now, if we change the source address of the hop-limited probes, and replace it by the destination address, the ICMP error message will be sent to the destination, following the *same path* as direct probes. Thus, *spoofed* hop-limited probes can be considered as direct probes that are just transformed in an ICMP *Time-Exceeded* on their way, as shown in Fig 1.7. This is another interesting feature that should be implemented.

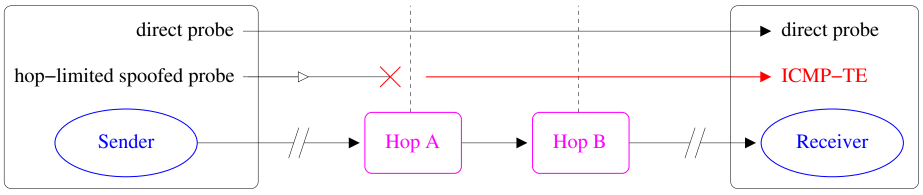

Figure 1.9: Direct and *spoofed* hop-limited probes

1.4.3 Expanding *Ping*

Another possibility is to use the classical *ping*. To *ping* a router, we simply send an ICMP *Echo Request* packet addressed to it, and it will send an ICMP *Echo Reply* packet back to the sender. Using *spoofing*, the *Echo Reply* answer can also be sent to the receiver. This is very similar to the *spoofed Time Exceeded* idea, but the router is no longer targeted with TTL, but directly with its address. And it's important to notice that the *Echo Reply* size is always the same as the *Echo Request* size, while the size of an ICMP *Time Exceeded* is (in most cases) fixed to 56 bytes.

1.4.4 Using Reordering

After enlarging the diversity of the probe parameters, we can also extract more information from the probe arrivals than only inter-arrival times. Reordering, which is often considered as a pathological behaviour, is also extra information. Moreover, since it seldom happens, two reordered packet should have very few chances to reorder again. Such an event is interesting, because, contrary to of the timing analysis techniques, it doesn't suffer from any distorsion caused by the cross-traffic noise on the rest of the route. Therefore, during the post-processing of the results, we shouldn't always drop the reordered probes : they carry an information that is much more meaningfull than a single timestamp.

Chapter 2

Active Probing's Software

2.1 The software 'as it was'

All our active probing tools have been grouped in a single package, the *Linux Sender Package*. It contains all the software for the standard Linux solution as well as for the TSC-enhanced solution. We can basically split the package programs in three categories : sending, monitoring, and converting monitor outputs (*post-processing*). Here, we will give a brief description, for more complete documentation, look at the active probing online documentation ([4]), or see the Linux sender package `README` file, also available on the webpage.

2.1.1 Sending

In the standard Linux solution, two different programs that communicate via a memory shared segment achieve the sender's function.

linuxSend will read a probe stream description file and write this description in the shared memory segment. The format of the input file is a sequence of probe descriptions, and each probe description is a vector of three 64-bit doubles.

$$Data = \begin{bmatrix} t_1 & Size_1 & TTL_1 \\ t_2 & Size_2 & TTL_2 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

linuxps must run first, as it waits in an infinite loop to receive some probe stream data from *linuxSend*. When it does, it starts sending, waiting between each probe for the given inter-departure time.

Probes are all UDP packets, with the same source and destination port, used as a tag : any UDP packet with the same port will be considered as a probe by the monitors. They are sent using a standard UDP socket, that requires no root privilege, and the TTL is set using the `setsockopt()` function. The length of the packet is adjusted by choosing the size of the UDP data, the data itself is filled with zeros. Each probe must of course get a unique identifier if we want to be able to detect loss, duplication, or reordering. This probe identification, called *serial number*, is stored as an integer in the 32 first bits of data of the UDP packets.

The TSC enhanced solution works in exactly the same way, the only difference is that the timing is done using the CPU's TSC register instead of the Linux function `gettimeofday()`. The programs names are also different : `tsclinuxps` and `tsclinuxSend`.

2.1.2 Monitoring

In the standard Linux solution, the user can use `tcpdump` as sender and receiver monitor (for more information about `tcpdump`, see [6]). Since it produces quite good timestamping, and provides filtering that allows us to consider only the probe packets, no other monitor has been developed. For example, the following command line produces a record of every UDP packet with port 8888 detected by the computer.

```
tcpdump udp port 8888
```

But the standard `tcpdump` cannot use the TSC timestamping, and that's why two monitors have been developed for the TSC solution :

tscreceiver uses a UDP socket set with the appropriate port to receive probes, that are timestamped with the `ioctl()` function, which allow us to access the timestamp created by the TSC-modified NIC driver. Its output is a `.tscdt` file that contains a raw array of probe receipts, in each receipt we have the timestamp and the probe's serial number. It can be used as a receiver monitor only.

TSC-tcpdump can act as a sender or receiver monitor. It is a modified tcpdump that takes advantage of the TSC timestamping available with the TSC environment.

The DAG monitoring is not included in the package since the network monitoring, which is the primary function of the DAG card, is always implemented in the DAG packages.

2.1.3 Post-processing

The output files from the monitors have three possible formats : the `.tscdt` files that come from `tscreceiver`, the `tcpdump` standard output, or the DAG card output. They all need to be converted in the general `.dt` format that is simply a sequence of arrival times stored in 64 bits double-precision floats, starting with the first probe (eg. the probe with serial number 1), and ending with the last probe monitored. Any lost probe has an arrival time of -1, and any duplicate or late probe (i.e. that arrived after a probe with a bigger serial number) is considered as lost. No filtering need to be done as the monitors already achieve that function : all the packets they monitored were probes.

The conversion from `.tscdt` files, done by *tscdttodt* is immediate, since it contains a set of probe arrival times and serial numbers. This is not true with the `tcpdump` outputs, that only contains a sequence of records for each probe received. To convert them, *tcpdumptodt* has to extract the timestamp and the serial number from each record, which, provided that the serial number is in the first 32 bits of UDP data, is quite easy thanks to the `-x` option that forces `tcpdump` to include a snapshot of each packet captured, byte per byte. Here is an example of tcpdump output :

```
16:34:44.706726 134.78.129.1.845 > 134.78.129.87.649: udp 64 (DF)
      4500 005c fbb5 4000 ff11 7a3d 81c7 8101
      81c7 810d 034d 0289 0048 6f9d 3f19 cce0
      0000 0001 0000 0000 0000 0000 0000 0000
      0000 0000 0000 0001
```

The conversion from the DAG output varies with the DAG implementation. On most of them, we have the possibility to pipe the DAG output to `tcpdump` input, thanks to the DAG team that provides this tool. This allow us to let `tcpdump` do the hard work of filtering probes and producing an output we are able to convert. Even when this feature is not available, the DAG package includes at least a filter tool that can make the filtering by itself. However, in this case, we still have to convert the DAG output to the friendly `.dt` format. This is done by *dagtodt*, that extracts the probes serial number in a similar way to *tcpdumptodt*.

Any further treatment of the `.dt` files is let to the user. We actually use *Matlab*, that can at the same time interpret these file and analyse the data.

2.2 Upgrades

To achieve the aim of the project, that is to enlargen the playground of active probing, we need a global upgrade of the whole package : it was based on UDP probes only that were easy to send, to receive, and to filter. If we want to deal with *spoofed* ICMP probes, we need to design a new sender, but also new monitors, and therefore new converters.

2.2.1 'Make your own' IP packet

To implement every new feature, especially *Spoofing*, we had to get rid of the restrictive predefined socket types, that don't give us access to all the IP options. Instead, we now use a socket of type `RAW`, that gives us also full access to the high-level protocol header, and even to the IP header (see Fig 2.1). Actually, it forces us to build both of them entirely. The only field that will (or may)

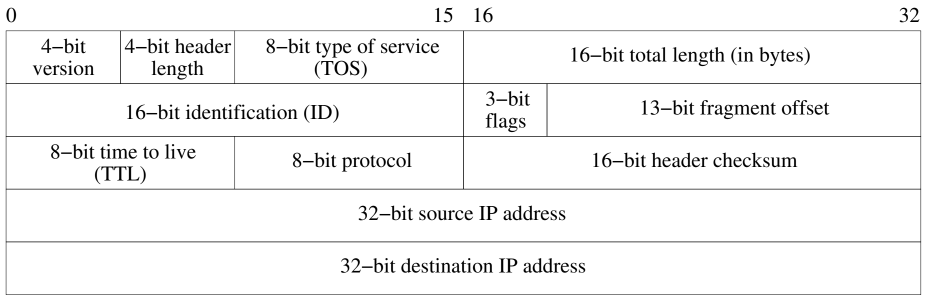

Figure 2.1: The IP header structure, without options

automatically be computed by the kernel, if they are left to zero, are the IP checksum, the ID field and the source IP address. Doing this, we are now able to specify any source IP address instead of the actual one. Setting the probe's

TTL is also very easy, and choosing the protocol as well. It also improve our sending accuracy, as we do part of the usual kernel work (i.e. build the UDP & IP header, compute checksums…), which duration could vary with different kernel versions.

Now, let's see what we can do about the probe *serial number*. We can get rid of the 32 bits of data in the payload that were necessary to encode it, by storing this serial number in the ID field of the IP header instead. The only problem is that the 16 bits of the ID field could be too narrow for this purpose : on very big experiments, it could be possible to use more than $2^{16} = 65536$ probes. However, we can just make the ID field loop back to 1 when it reaches 65536, because the probe stream stays more or less well ordered : it is hard to imagine that a probe could be reordered more than 65536 probes after or before its original place.

The other important information that the probes have to carry is the *authentification key*. This is the value that all of our probes will carry, and that will allow us to recognize them among all the packets that our monitor could receive. With UDP probes, the old package used the port number to carry the authentification key. We can still do that. Anyway, we have to write the whole UDP header, that contains the port number, so it is not a problem to keep the old authentification field. It would have been better to write it in the IP header (to be independant from the UDP protocol), but there's no room for it.

In the end, we have a C function `write_ip_header()` that only takes source and destination IP addresses, the TTL, the probe size (IP packet size), the ID and the protocol as arguments and that writes a complete IP header, setting the unspecified fields to default values.

2.2.2 Expanding the probe family

Because of the full control of the packet encoding, we can now build probes based on any protocol. But what for ? Actually, at least three types of probes could be interesting :

- ICMP *Echo Request* will allow us to *ping* a router
- ICMP *Echo Reply*, that doesn't interact with routers more than UDP, is an alternative to UDP probes
- ICMP *Time Exceeded* is also such an alternative, slightly different because when the TTL of an ICMP-TE packet reaches zero, the packet is silently dropped, i.e. no ICMP-TE is generated. This was designed to avoid infinite loops.

The ICMP *Echo* packets are convenient because their structure is, as shown in Fig 2.2, very similar to UDP packets. Thus, their ICMP header contains two fields, the sequence number and the ICMP Identification, both 16 bits, that can replace the UDP source and destination port for the probe authentification.

ICMP *Time exceeded* packets are normally only generated by routers when a packet must be dropped because its TTL has reached zero. That's why they have been designed to carry at least the IP header of the original packet, plus the 8 first bytes of the packet datagram, as shown in Fig 2.3. Since these 8 bytes are usually the first 8 bytes of the higher-level protocol header, they should,

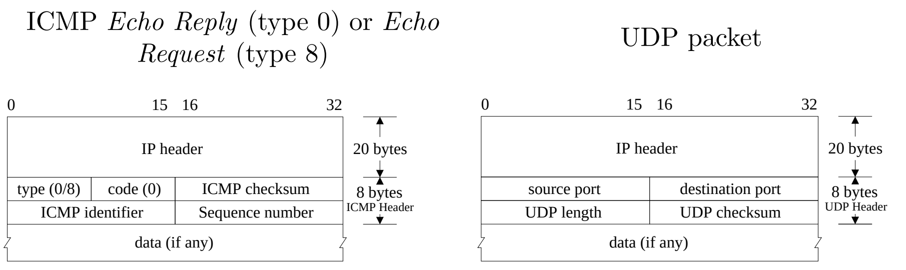

Figure 2.2: Comparison between ICMP *Echo* packets and UDP packets

combined with the IP header, help the user that receives an ICMP-TE to know which packet was dropped.

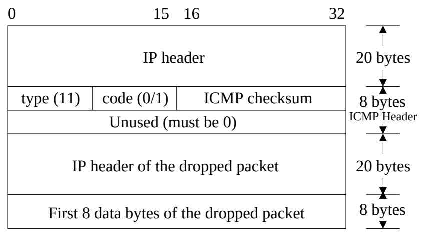

Figure 2.3: A typical 56-bytes ICMP *Time Exceeded* packet

Note that our probes (UDP or ICMP *Echo*) carry all their information in the 28 first bytes : the *serial number* is in the IP header, and the *authentification key* is in the 8-bytes UDP or ICMP header. Therefore, receiving an ICMP-TE coming from a dropped packet, we are able to recognize it as a dropped probe, and get the dropped probe serial number.

We can also use that to send ICMP-TE probes, what was not so easy because, unlike ICMP *Echo* probes, there's no room for the authentification key in the ICMP-TE header. But we can just send the ICMP-TE probe as if it had been generated by one of our probe, dropped by a router. Which means, we include a faked probe in the 28 bytes following the ICMP-TE header. The monitors will be able to recognize it as a probe in the same way as they would do it for ICMP-TE really coming from a dropped probe.

Another limitation of the old package is that a single experiment could only send probes to a single destination. What if we want to send UDP probes to our receiver, and, in the same experiment, *ping* a router of the corresponding route ? That's why it should be useful to send some probes to an alternative destination IP address. This alternative IP address will be given as argument to the *[tsc]linuxps*.

Another option could be not to compute the UDP or ICMP checksum, or to give a false checksum. That could provoke some reactions if the routers check it on the way.

To make all this options available without compatibility loss, we have to keep the old probe stream description file format, so that old input files are still recognized. But with this format, a probe is only described with its inter-departure time, its size, and its TTL. The trick is to use the unused bits in the TTL encoding : a TTL is always coded on one byte, and the input file uses 8 bytes to encode it. Practically, we use following bits (bits 0 to 7 are already used for the TTL) :

- bit 8 : *spoofing* flag. If set, the probe's source address will be spoofed.
- bit 9 : *Echo Reply* flag. If set, the probe will be an ICMP *Echo Reply*.
- bit 10 : *Echo Request* flag. If set, the probe will be an ICMP *Echo Request*.
- bit 11 : *No Checksum* flag. If set, the UDP/ICMP checksum won't be computed.
- bit 12 : *Alternate Destination* flag. If set, the probe will be sent to the alternative destination IP address. We could have coded the alternate destination address itself instead of only a flag. This will be part of future enhancements.

If all the flags are clear, the probe will be a UDP packet, exactly as the old version (except that the serial number will be in the ID field instead of being stored as data). In addition, if the TTL is zero, then instead of sending the probe we send the ICMP *Time Exceeded* that would have been generated if the sender was a router and if the probe had reached it with a TTL of zero.

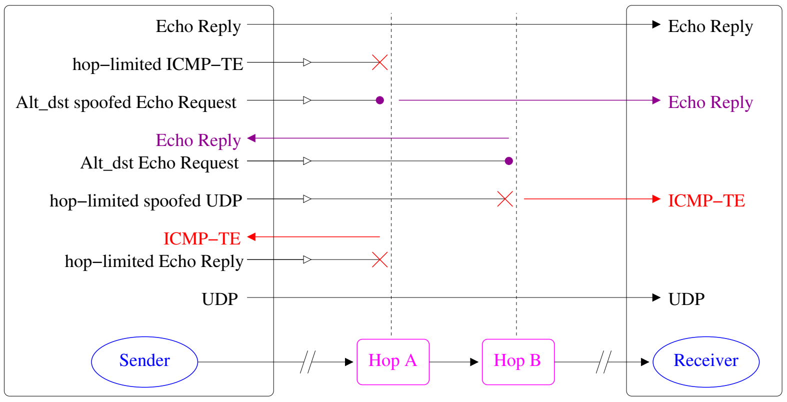

Figure 2.4: Some of the different kinds of probes we can send. Crosses are probes dropped by a router because of their expired TTL. Dots are *Echo Requests* that arrive at their destination.

All those features have been grouped in a single file, `spoofing.c`, the key function is `send_probe()` that takes following arguments : the destination IP address, the alternate destination IP address, the source IP address to use for spoofed probes, the source IP address to use for normal probes, the authentication key (UDP port or ICMP identifier), the serial number, the TTL (including the flags), the size of the probe, and the optional data to include.

We've also added options to the `linuxps` and `tsclinuxps`. For instance, the following command line starts the sender with the destination address `receiver.cs.mit.edu`, an authentification key of 7777 and defines the alternate destination address to be `63.45.123.78`, the source address for spoofed probes to be `spoof.cs.mit.edu`, and the source IP address for normal probes to be `127.0.0.1`.

```
tsclinuxps -a 63.45.123.78 -s spoof.cs.mit.edu
  -S 127.0.0.1 receiver.cs.mit.edu 7777
```

2.2.3 Monitoring and Post-Processing

Now that we can send these different types of probes, filtering the packets to recognize the ones that are probes becomes more difficult. The main difficulty is that there's no standard way to extract the authentification key from all kinds of packets : UDP and ICMP *Echo* carry it in the same place, but it's not the same for ICMP *Time Exceeded.* Neither `tcpdump`'s options nor socket types (for `tscreceiver`) allow such a powerful filtering. That's why we included in the new package the `packet_is_probe()` function that takes a raw packet as argument, and tests if it is a probe or not, checking the authentification key at the right place, depending on the packet type. Moreover, if the packet is indeed a probe, `packet_is_probe()` extracts the probe's serial number from the raw packet. The Monitors now filter the probes in a different way :

- `tscreceiver` uses a RAW socket (instead of an UDP socket) that receives any kind of packet, but it will record a probe arrival only when the packet is acknowledged by `packet_is_probe()`
- `tcpdump` cannot filter probes properly, so it just captures any packet that could be a probe (like any UDP packet with the right port number and any ICMP packet)

The post-processing of the `tcpdump` output requires some filtering. `tcpdumptodt` now uses `packet_is_probe()` for that purpose. Otherwise, the only change in the post-processing of the data is that we don't want to drop the reordered packets any more. Besides, it can still be done by the user afterwards. This implied a few modifications in both `tscdttodt` and `tcpdumptodt`.

2.2.4 Compatibility

Most of the new features are optional :

- The IP addresses are given to `linuxps` and `tsclinuxps` as options
- To send the new kinds of probes, we have to set some flag bits in the input file that were always zero before.

But still, even if we don't use the optional features, there are still a few modifications :

- The serial number is in the ID field now
- The type of socket used by `linuxps`, `tsclinuxps` and `tscreceiver` now require more user privileges.

- The data will not be filled with zeros, but with random bytes. This will avoid compression. This could eventually be an option in futures versions.

For a user with superuser privileges, any script that worked with the old package will work with the new one. More precisely, if the user gives an old probe description input file that he used with the old package, the new one will send the same probe stream, and the output `.dt` files will have the same format. The only constraint is to use the same package for an experiment : it is impossible to use the new senders and the old monitors, for example. The compatibility works also backwards : any input file designed for the new package that doesn't take advantage of the new features can be given to the old one.

2.2.5 Future Enhancements

We left undone some possibly interesting enhancements :

- Use TCP packets. This could be very useful, since TCP, as the most used protocol, could have a higher priority in some future TCP-optimized routers.
- Deal with IP fragmentation. This is a way to indirectly interact with routers, like TTL. In a ideal case, this could even allow us to transform a single probe to a pair of back-to-back probe in the middle of the route (if the packet is fragmented on the way).
- Optimize the sending software to be able to send back-to-back probes without queuing them behind a big packet. We could also include this queueing trick as an easy-to-use option.
- Use the IP options to enlarge the IP header. This will causes the ICMP-TE created by dropped probes to be bigger (they have to contain the full original IP header).
- Encode the alternate destination address in the input file instead of giving it to the sender program as an argument. This will allow us to send probes to much more than two destination in the same experiment.

Chapter 3

Reordering : A New Direction for Active Probing

With the new active probing package, promising techniques are finally available. We focus here on methods that use reordering. Indeed, if we can make some packet reordering happen at a specific location in the route, that could provide some very efficient measurement techniques on the router concerned.

3.1 ICMP Time Exceeded

The method described in this section was the original idea of this project. This idea first led us to develop the software in order to implement it, and this development then become the principal aim of the project. In this section, we present some of the possible applications of the enlargment of active probing tools based around this theme.

3.1.1 Methodology

We send identical pairs of probes. Here we describe what happens for one pair (see Fig 3.1). The two probes are sent back-to-back. The second probe is direct. The first probe is hop-limited, and *spoofed* : it will be dropped at a given router h (that we can choose with the probe's original TTL), and an ICMP *Time Exceeded* will be generated by the router h and sent to the destination, thanks to the *spoofing.*

ICMP *Time Exceeded* takes time to be generated, so that the second probe has a chance to arrive before the ICMP is ready, and to pass it. This chance essentially depends on the time that the second probe will take to arrive at the router. This time will of course depend on the *service time* s_2 of the second probe, that is the time needed by the previous router $h-1$ to send it entirely. The service time s_2 is proportionnal to the second probe size p_2.

$$s_2 = \frac{p_2}{\mu^{h-1}} \tag{3.1}$$

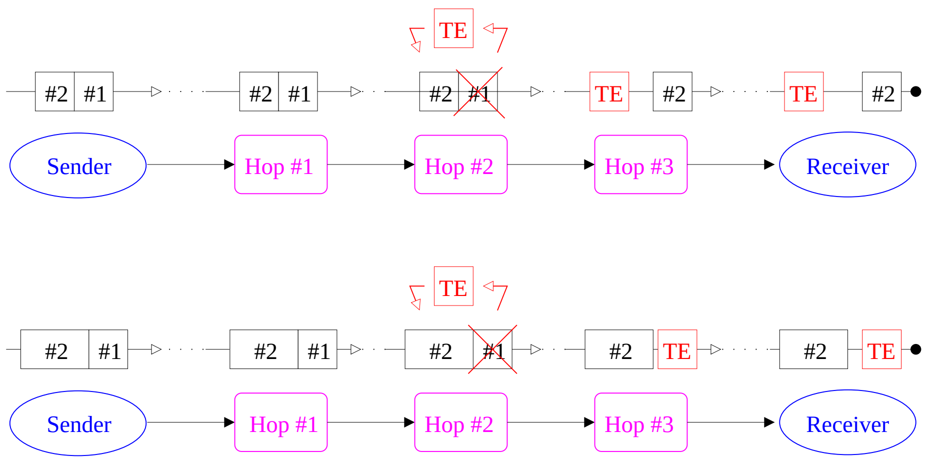

Figure 3.1: Reordering of back-to-back pairs of *spoofed* hop-limited probes followed by direct probes.

where μ^{h-1} is the bandwidth of the link that *arrives* at router h. That's why we should expect to observe less reordering when the second probe is bigger.

3.1.2 What we could expect

To make it more simple to understand, we will first assume the following : (the notations are the usual ones, the 'arrival' event is just the arrival at the router)

- Packets that follow the same route are never reordered.
- The ICMP-TE generation takes a constant time g^h
- The ICMP-TE generation is made separately from the router's packet forwarding process, so that packets can pass through the router meanwhile.
- This treatment starts when the first probe has fully arrived in the router (at time τ_1^*).
- The ICMP-TE is sent before the second probe if, and only if it is ready before the second probe has fully arrived :

$$reordering \iff t^* < g^h \tag{3.2}$$

- The two probes arrive back-to-back at the router :

$$t^* = s_2 \tag{3.3}$$

In such an ideal case, the experiment would produces a result like the one shown in Fig 3.2 :

Indeed, elementary manipulations of the previous equations give us :

$$reordering \iff t^* < g^h \iff s_2 < g^h \iff \frac{p_2}{\mu^{h-1}} < g^h \tag{3.4}$$

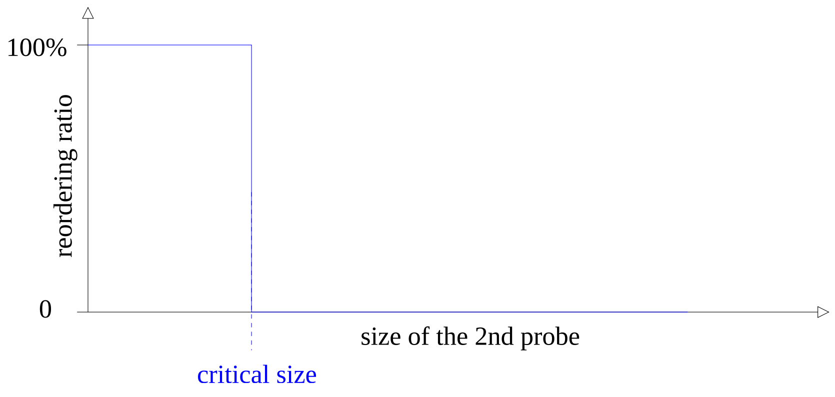

Figure 3.2: The reordering signature, in an ideal case

and finally

$$reordering \iff p_2 < \frac{g^h}{\mu^{h-1}} \tag{3.5}$$

Now, the *critical size* $c_h = \frac{g_h}{\mu^{h-1}}$, that is easily observable on Fig 3.2, can give us the router bandwidth, if we know its ICMP generation time g_h : $bandwidth = \frac{c_h}{g^h}$

If we suppress the first two assertions, and accept that natural reordering can occur, and also that the ICMP-TE generation time can slightly vary; if we also admit that the probes can arrive with any space between them, even a very small one, we could still obtain the same-shaped curve, with some noise. This is represented in Fig 3.3. The important thing is that, hopefully, the critical size could stay visible, if the experimental noise is not too big.

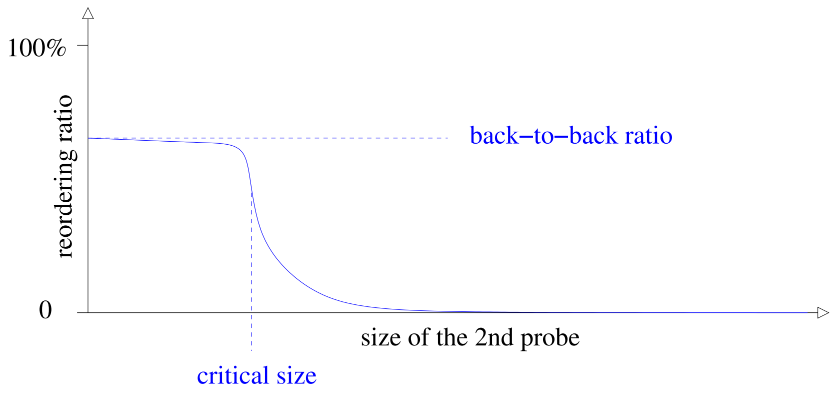

Figure 3.3: The reordering signature, in a less ideal case

3.1.3 Weaknesses

- *Spoofing*-protected gateways don't forward *spoofed* packets. These protections are becoming a standard security requirement : we were able to run experiments that used *spoofing* on only a short part of a single route. We are far from the worldwide 'testbed' that the Internet is. However, if one has enough influence to open the local gateways between ones computer and the international network, he can send *spoofed* packets anywhere, since the international routers cannot detect *spoofing* as easily as local gateways, because they are used to dealing with a much bigger set of source IP addresses.

- Ethernet packet cannot have all sizes : the minimum is 48 bytes and the maximum is (usually) 1500 bytes. If the critical size is outside this (quite narrow) range, we won't be able to determine it.
- Like any active probing method based on reordering, a considerable noise could come from natural reordering, if it occurs on the route.
- The behaviour of the router when it generates the ICMP-TE packet is unknown. The model of a simple delay may be too simple.

3.2 ICMP Echo Request

3.2.1 An alternative to ICMP-TE

Why not taking advantage of the variety of probes we can use ? Sending a *spoofed* ICMP *Echo Request* to the same router h, it will generate an ICMP *Echo Reply* and send it to our destination. This is similar as a *spoofed* hop-limited probe. This similarity allows us to easily copy ICMP-TE method described in the previous section. Though, we don't expect to get the same results, because routers might most probably process *Echo Requests* addressed to them and packets with $TTL = 0$ in a different way. This assertion is strongly encouraged by [8] that advises the routers to separate packets that are destined to them and packets that should be forwarded before doing any other processing (apart from some basic IP header checks). This difference could causes many aspects of the previous experiment to change. First, the *Echo Reply* generation time g'_h might be significantly different from the *Time Exceeded*'s g_h. Moreover, the *Echo Reply* is exactly as big as the *Echo Request*, unlike the ICMP-TE that is most of the time 56 bytes long. This could make the *Echo Reply* generation time vary with the second probe size : $g'_h = f(p_2)$

Having an alternative method is really useful because it gives us a lot more flexibility. And lack of flexibility is one of the big weakness of the ICMP-TE reordering technique.

3.2.2 Combining with ICMP-TE : Get rid of *spoofing*

If the behaviours of ICMP-TE and *Echo Reply* are different enough, we could even combine them in the same experiment. Instead of sending pairs of one direct probe and one *spoofed* hop-limited (or *Echo Request*) probe, we could send pairs of one *spoofed* hop-limited probe and one *spoofed Echo Request* probe. For example, if the first is the hop-limited one, reordering may occur if $g > g' + s_2$. This adds some flexibility again.

But the best part is that this allows us to get rid of this annoying *spoofing* that closes so many routes to us : indeed, if we don't *spoof* the both probes,

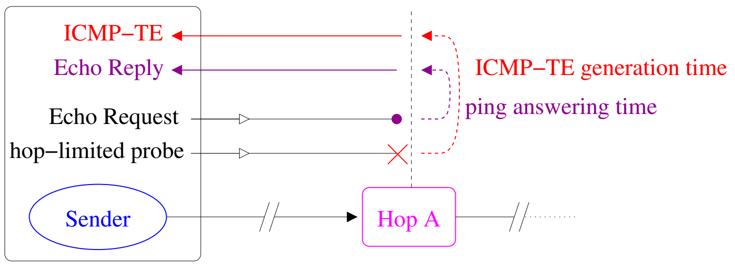

Figure 3.4: Reordering between ICMP *Time Exceeded* and *Echo Reply* packets one a "one way-return" route.

the ICMP-TE and the *Echo Reply* will both return to the sender (see Fig 3.4) Since they will take the same route to go back, this method should work exactly as when using *spoofing*.

3.3 Experimental results

The goal of this project is not to develop a couple of reliable active probing methods. Our approach is more like an exploration of a wide range of possibilities. That's why we didn't force ourselves to deeply analyse the experiments we ran. We only want to see if an experiment seems to work.

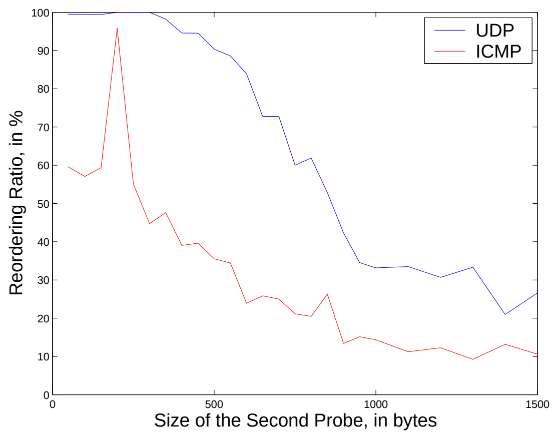

Figure 3.5: ICMP-TE reordering, experimental. Above : reordering signature with UDP probes. Below : with ICMP *Echo Reply* probes

The first experiment we ran was the ICMP-TE reordering method. The results are shown in Fig 3.5. To remember the experimental methodology, see Fig 3.1. The results we could expect are also shown in Fig 3.3. The experiment was ran with two types of probes : UDP probes and ICMP *Echo Reply* probes.

The results are not very encouraging : both curves don't exhibit any critical size, and, even worst, the two curves are very different. This second point proves that ICMP *Echo Reply* and UDP are not processed the same way on every router. This discouraged us of trying to understand the shape of each signature. And because of the very small set of routers we could run this experiment on (because of the *spoofing* protections we described earlier), we weren't able to collect other experimental results like this : the two other routers had respectively 100% reordering with every size for one of them and no reordering at all for the second one. Repetitive failures of other experiments have led to the conclusion that something must be wrong in our simple model for the ICMP-TE reordering. That's why we stopped further investigations to first check what was going on with those ICMP packets, as we describe in the next chapter.

Chapter 4

ICMP Behaviour

In order to use the tools we provided for active probing, we had to know more about ICMP behaviour, since we had clues that ICMP and UDP packets were not processed the same by routers. This is fully part of the project, as the knowledge of ICMP will actually help us to enlarge the active probing playground :

- It will allow to validate or invalidate some active probing method ideas based on ICMP,
- It may give us some indications about new methods based on specific ICMP characteristics.

4.1 ICMP End-to-End Delay

The first thing to check was that ICMP packets and UDP packets have the same end-to-end delay on any route. To compare the end-to-end delays of two types of packets, the experimental methodology is quite simple : we sent independant (well spaced) pairs. In each pair, the first packet is one of the first type, and the second one is of the second type.

These packet pairs are sent with a big IDT t so that they are independant. So, we are not really sending packet *pairs*, we actually send independant packets. We are talking about pairs here because the odd probes are of the first type, the even ones are of the second type, so that, to compare the end-to-end delay of the two types of probes, we 'group' the probe departure and arrival-times τ_i, τ_i^* by pairs : $(\tau_{2i}, \tau_{2i+1}), (\tau_{2i}^*, \tau_{2i+1}^*)$. Now we measure the inter-departure times $t_i = \tau_{2i+1} - \tau_{2i}$ and inter-arrival times $t^* = \tau_{2i+1}^* - \tau_{2i}^*$, and compute the delay variation $\delta_i = t_i^* - t_i$, which is the difference between the two end-to-end delays within pairs.

To suppress the cross-traffic noise that makes a single measurements meaningless, we repeat this several times, and keep the average delay variation $\overline{\delta}$.

4.1.1 Experiments on one route : ENS $\rightarrow$ CUBIN

In Fig 4.1 is plotted a histogram (left) of all the delay variations observed between UDP and ICMP *Echo Reply* probes of the same size, and an histogram (right) of the delay variations observed between identical packets (ICMP *Echo Reply* in this experiment). Both experiment were done on the same route from the École Normale Supérieure (ENS), Paris, to CUBIN.

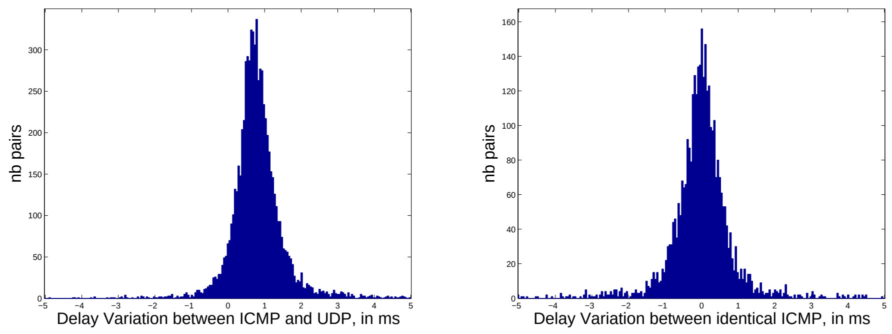

Figure 4.1: Delay variation distribution. Left : between UDP and ICMP *Echo Reply* probes. Right : between identical ICMP *Echo Reply* probes.

- The second trace (right) is exactly the independance signature (see Fig 1.5) : we see a symmetric-shaped distribution centered at zero. Actually, this experiment was not run to check that two identical packets will be processed identically, but to get an estimation of the noise encoutered by the first experiment.

- The first trace looks roughly the same, but with a big difference : its position. The distribution is centered at a positive value around $0.75ms$: this is our average delay variation $\overline{\delta}$.

Note that in both cases, a small (but not negligible) number of values were far out of the plotted window. This often happens in active probing experiments : exceptional router load may occur at any time, provoking extra queueing delays that result in making one of the two probe significantly late.

Now that we made sure that ICMP were indeed slower than UDP, we can work a little more on that delay variation, let's call it the ICMP *excess delay*. Even if we don't know where it comes from, we can assume that δ is the sum of three terms (see Equation 4.1) :

$$\delta = \delta_{min} + \delta_{noise} + \delta_{size} \tag{4.1}$$

δ_{min} is the part of ICMP excess delay that is directly related to its cause. It is the minimum delay variation that we could observe, in ideal conditions. δ_{noise} is the part of ICMP excess delay that varies, depending on the network conditions : load, congestion.... This term is positive, and equal to zero in the ideal conditions (when ICMP packets are processed the fastest). δ_{size} is the packet size dependance term. This will probably increase with the packet size.

Now, we want to get a good estimation of $\overline{\delta}$. The choice of a good averaging operator deserves some attention. Actually, we had tried a few ones :

- The simple mean is much too sensitive to noise provoked by exceptional queueing delays.

- The robust mean, obtained by filtering out the very big values (like the 1% extreme percentiles), is much better. However, it still suffer from the outliers assymmetry. On the graph (Fig 4.1, left) we can see that the distribution of delay variations at the foot of the main peak are not symmetric : It seems that ICMP can encounter extra queueing more often than UDP. This will cause the robust mean to give inaccurate results.

- The median of the delay variations was a better operator. It is slightly better to take the median of the delay variations between UDP and ICMP than taking the difference between the median of the UDP delays and the median of the ICMP delays : in the first case, we estimate the delay variation with probes that were not very far in time, and thus suppress the error due to long-range variations that occur in the route during the experiment (like people starting to download a movie). However, these two operators always gave almost the same results.

To estimate the number of pairs required to get a good estimation of $\overline{\delta}$, we compared the value obtained by averaging more or less pairs. This is plotted in Fig 4.2 : the graph on the left refers to UDP-ICMP delay variation, and the one on the right refers to the ICMP-ICMP delay variation (noise evaluation).

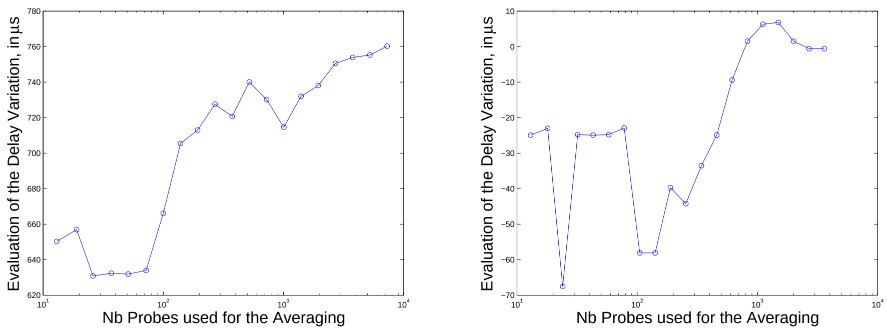

Figure 4.2: Evaluation of $\overline{\delta}$ with variable number of values used for averaging. Left : UDP - ICMP *Echo Reply* delay variation. Right : identical ICMP *Echo Reply* delay variation.

The noise looks not too big : with a couple of thousands of pairs, we can already get a quite reliable estimation : $\overline{\delta} \sim 750\mu s$, with an accuracy of about $20\mu s$. This delay variation looks very small compared to the delay (on this route from France to Australia, packets delays are over $100ms$). But for active probers, it is huge, as we deal with very short events, like packet service times that are $< 100\mu s$ on most routers.

We ran some more experiments, replacing ICMP *Echo Reply* probes by ICMP *Time Exceeded* or *Echo Request* probes. We obtained similar results, which encourages us to believe that different types of ICMP have the same delay. Indeed, running the same experiment but comparing *Echo Reply* and *Time Exceeded* instead of *Echo Reply* and UDP, we obtained traces that looked exactly like the one obtained by comparing identical ICMP *Echo Reply* probes.

We did not try other types of ICMP, as we did not implement them in the package.

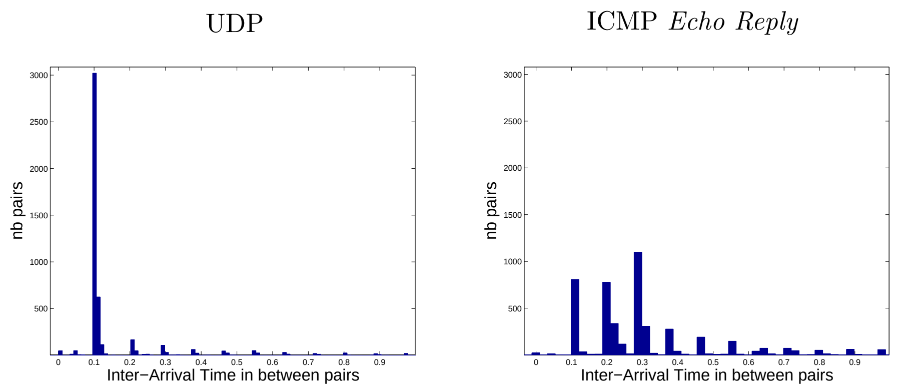

Figure 4.3: Comparison between inter-arrival times of probes sent back-to-back. Left : pairs of UDP back-to-back probes. Right : pairs of ICMP *Echo Reply* back-to-back probes.

We also tried to send back-to-back pairs of ICMP *Echo Reply* probes and back-to-back pairs of UDP probes to see if back-to-back ICMP probe pairs could encounter more spacing than their UDP clones. The results are shown in Fig 4.3. We see that ICMP pairs were indeed more spaced than UDP ones. This implies that ICMP excess delay may be bigger for an ICMP probe sent just after another ICMP probe.

Last but not least, we wanted to estimate the packet size dependance of the ICMP excess delay. So, we ran the same experiments, but with bigger probes. We were careful to keep the sizes of UDP and ICMP identical, as different sizes will cause additional delay variation. (The bigger as packet is, the longer it takes to be serviced at each router). The results are shown in Fig 4.4

Size (bytes)	Excess delay (μs)
56	760
400	990
800	1225
1200	1460
1500	1620

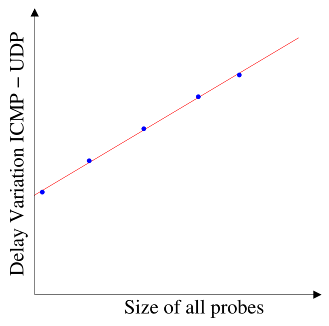

Figure 4.4: Packet size dependance of the ICMP excess delay

4.1.2 Larger Experiments

Now, we wanted to check if our route was a very particular case or if ICMP had extra delaying everywhere. We had to change the experimental methodology, since we cannot afford enough hosts around the world to run experiments that require the control of both sender and receiver. To get rid of the receiver, we used the following trick : instead of sending direct probes, we sent hop-limited probes. When they arrive at the chosen router, the hop-limited packets (UDP or ICMP *Echo Reply*) will generate an ICMP *Time Exceeded*, and it seems very reasonable to suppose that the time taken by the ICMP-TE to return to our sender is the same if they come from a UDP probe or from an ICMP probe. It seems also reasonable to assume that the ICMP-TE generation time is the same for UDP probes and for ICMP probes. So, if we assume that, and consider the sender as the receiver (the round-trip-times will be the delays), the delay variation between UDP and ICMP probes should come from the first part of their trip, i.e. from a delay variation in the transmission between the sender and the receiver.

Now, we can set an experimental methodology : we pick a random host X somewhere on the Internet, and run *traceroute* to obtain the addresses of all the routers on the route from us to X (for more details about the famous *traceroute*, see [11]). If the route is composed of N+1 hops (N routers, plus the host X), we send probes to X with a TTL of N, so that the very last router, just before X, will drop our probes and send us the ICMP-TE. All this is represented in Fig 4.5.

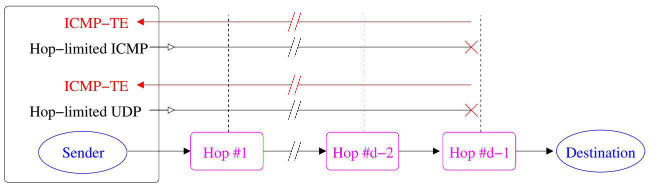

Figure 4.5: Probes sent to estimate the delay variation between UDP and ICMP on a route where we don't control the destination host

We ran this experiment on 18 international routes, all from CUBIN to a random host. The host were www servers from different countries :8 in Weastern Europe, 1 in Russia, 2 in Oceania, 1 in Africa, 3 in the US and 3 in Asia. The results were following :

- On six routes, the targeted router did not send an ICMP-TE when he dropped our ICMP *Echo Reply* probes. So, we had to shorten the route to pick the last router who did.
- On 14 of the 18 routes, we couldn't see any difference between ICMP delays and UDP delays : it seems that our route was indeed peculiar, and that in most of the routes there is no difference between UDP and ICMP.
- However, on 4 of the 18 routes (3 to Europe and one in Oceania), we could observe a significant delay variation : on two of them, ICMP probes were $250\mu s$ slower than UDP, and on two others, this extra delay reached $1ms$.

We are not able to exhibit a nice model or more detailled results that could allow us to understand this phenomenon. This could come from router optimized for TCP and UDP in which ICMP takes a slower path. It could also come from a route change, but the low difference obtained did not encourage this hypothesis. However, it would be easy to check that ICMP and UDP take the same route, by writing a *traceroute* clone that would send ICMP *Echo Reply* instead of UDP packets, and comparing the outputs with the original traceroute. But we didn't do it. The only reliable results that came out from this study are :

- On *Some* routes, ICMP packets are slightly slower than UDP packets. Even if it seldom happens, any active prober must be aware of that.
- However, different types of ICMP seem to be identically processed. But this should be checked for any experiment involving different kinds of ICMP probes.

We did not have enough time to run the other experiments described above (such as packet size dependance, or back-to-back probes spacing). That's also the reason why we have results on so few routes. But getting some accurate measures was not the goal of this project : we only wanted to explore the various possibilities that ICMP offers, as well as the difficulties that active probers may encounter in using ICMP in their experiments.

4.2 ICMP generation time

Althought the few techniques presented in chapter 3 all involved ICMP generation, and needed the ICMP generation times to be known, we have let the topic of ICMP generation time alone so far. But it is understandable since we couldn't really design a reliable experiment to measure this generation time, as long as we didn't know what was going on with ICMP. Now that we are aware of the necessity to be very careful, we might be able to design an experiment whose results we can trust.

4.2.1 ICMP Time Exceeded

The ICMP *Time Exceeded* generation time has been the topic of a much longer study ([7]), done by Ramesh Govindan and Vern Paxson in the year 2001. They sent independant streams of *spoofed* hop-limited ICMP *Echo Reply* and direct ICMP *Echo Reply* probes, as shown in Fig 4.4. Since probes all had the same size (56 bytes), if we assume that ICMP *Echo reply* are forwarded exactly as fast as ICMP *Time Exceeded*, the delay variation δ comes only from the ICMP-TE generation time g : on average, $\delta = g$. For every router, they did several measurements of δ, and then took the median delay variation as an estimate for the ICMP-TE generation time. They obtained results for 200 routers.

In their results, a few things were quite encouraging for the fate of the methods described in chapter 3 :

- Although they used *spoofing*, it seems that they didn't have problem to run experiments on lots of routers.
- The ICMP-TE generation times they obtained were mostly smaller than $1ms$ (for 80% of the routers), and half of them were even smaller than $300\mu s$. This let us hope that the ICMP generation time could match the service time of one probe, or at least of a couple of probes (otherwise, the reordering signature described in section 3.1 couldn't work).

- They also ran a few experiments with back-to-back probes. They obtained a high reordering ratio of 81%. At least we are now sure that artificial reordering is possible on most of the routes (at least 81%).

Since they did not mention anything about the dispersion of the ICMP-TE generation time for a single router, we had to run a few experiments to evaluate it. We used the same experimental methodology as Govindan and Paxson. This methodology was validated by the results obtained in the previous section : on our route ENS → CUBIN, we knew for sure that ICMP-TE and ICMP *Echo* had the same delays (and, therefore, the same route, because it is statistically very unlikely to obtain identical delays with two different routes). So, the delay variation can only come from the ICMP-TE generation time. We could estimate ICMP-TE generation time for 4 routers only (because of the *spoofing* protections). The results are shown in table 4.1.

Route	Router	Gen. Time (μs)
CUBIN → CUBIN	CUBINlab Firewall	< 5
ENS → CUBIN	ENS Gateway	1250
ENS → CUBIN	Router #3	~ 100
ENS → CUBIN	Router #4	-9200

Table 4.1 : ICMP-TE generation times of a few routers

The first three results are consistent with Govindan and Paxson's work, but the last one is hard to believe : it shows that if you send a 56-bytes *spoofed* ICMP *Echo Reply* to CUBIN from the ENS, with a TTL set to 4, the 56-bytes ICMP *Time-Exceeded* generated by the router #4 will arrive more than $9ms$ faster than if we had sent a normal ICMP *Echo Reply* probe. We cannot be sure of why this is happening, but it is probably caused by a route change, as shown in Fig 4.6. This means that *spoofing* doesn't always work the way we think it does : the ICMP-TE answer could actually take another route than the direct probes. Since Govindan and Paxson didn't say a word about this, we supposed it was again a particularity of our route. Anyway, this is again something that active probers should definitely check, if they use *spoofing*.

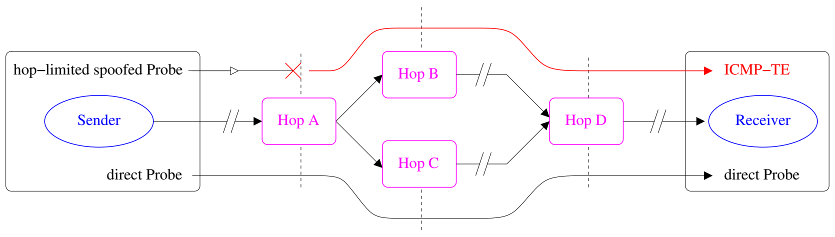

Figure 4.6: ICMP-TE packets that come from *spoofed* probes could take a different route than direct probes.

Another thing we would like to know is the packet size dependance of the ICMP-TE generation time. This is possible, but difficult, because it implies that we will send probes bigger than 56 bytes. Therefore, the *spoofed* hop-limited probe will not have a constant size : it will be big until its "transformation" in ICMP-TE, and then it will be 56 bytes long. We can actually make the

ICMP-TE be bigger using the IP header options, but it will add a maximum of 40 bytes, which is really small compared to the 1500 bytes of the biggest packets. This non-constant size will causes some delay variation, because the direct probe has a fixed size, and therefore cannot have the same size as the hop-limited probe during all the route. We could correct the delay variation caused by this variable size, using tools like *pathchar* or methods inspired of it, but it would increase the experiment's complexity a lot.

4.2.2 ICMP Echo Request

As described in chapter 3, we can make use of *spoofed* ICMP *Echo Requests* instead of *spoofed* hop-limited probes. The only difference is that the size of a *spoofed* ICMP *Echo Request* will stay constant, even when it becomes an ICMP *Echo Reply* answer generated by the targeted router. Therefore, we can measure the ICMP *Echo Reply* generation time exactly as the ICMP-TE generation time, and we can even evaluate the packet size dependance.

We ran these experiments on the same routers as the ICMP-TE generation time experiment. The routers #2 and #4 had a generation time $< 10\mu s$ that was too small too estimate. The CUBINlab firewall did not answer to our *spoofed Echo Requests*. But we had interesting results with the router #3, that are shown in Fig 4.7. The probe size dependance looks linear, but of course we won't assert such a thing with only three points of measurement. The important thing is that this generation time can actually vary significantly with the probe size. That confirms the hopes we had in Chapter 3.

Size (bytes)	Gen. Time (μs)
56	116
800	130
1500	148

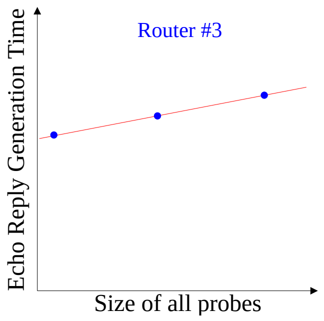

Figure 4.7: Packet size dependance of the ICMP *Echo Request* generation time on router #3

Something we didn't have time to do was to compare the ICMP *Echo Request* and *Time Exceeded* generation time on more routes. The methodology is the same as for the comparison between UDP and ICMP delays : sending hop-limited probes and *Echo Request*, we will receive an ICMP-TE and an *Echo Reply*, coming from the same router (if we were smart enough to address the *Echo Request* to the router that will drop the hop-limited probe). Then, we can compare the round-trip times, and the average delay variation should be what we are looking for : the difference between ICMP-TE generation time and ICMP *Echo Reply* generation time.

4.3 ICMP and Natural Reordering

Natural reordering is the nightmare of any active prober playing with the reordering measurements. Fortunately, it seldom occurs. What we usually call natural reordering causes only a very small amount of packets to be reordered. One source of that kind of natural reordering is the multiplicity of links between two routers, for example to support load sharing. If a link is faster, packets that take this link could pass packets in the other one. But some strange results forced us to check if some exceptionnal reordering was happening on one of our test routes. So, we sent back-to-back UDP probes, on the route ENS $\rightarrow$ CUBIN again, looking for reordering. The results were very surprising (two of them are plotted in Fig 4.8)

- A small (56 bytes) probe sent just after a big one (i.e. $>$ 500 bytes) will *always* (i.e. with a probability $>$ 99%) pass it (Fig 4.8, Left).
- A probe *almost never* (i.e. with a probability $<$ 1%) pass a probe that is much smaller. But it may pass a probe of equal size (if they are sent back-to-back) with a probability over 10%.
- A probe, even very small, will *almost never* (i.e. with a probability $<$ 1%) pass two probes at a time, even if they are very big (Fig 4.8, Left).
- Many (more than 10 and up to 20) small probes, sent just after a big one, can all pass it (Fig 4.8, Right).

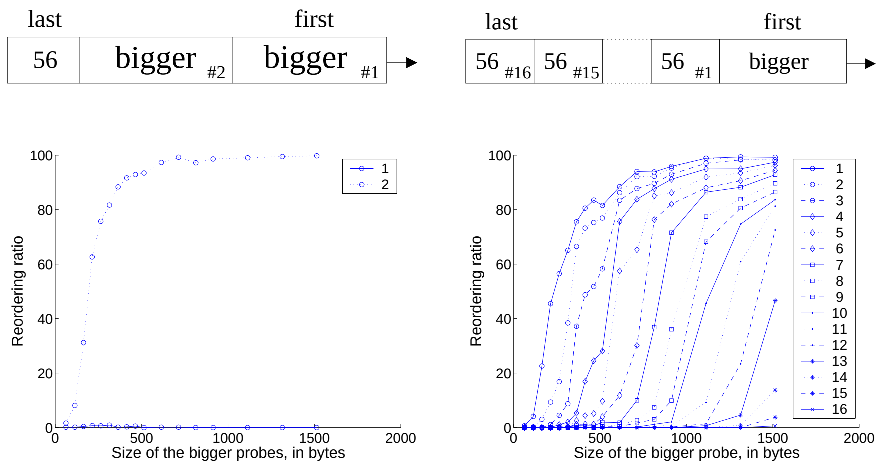

Figure 4.8: Two examples of natural reordering. We send groups of back-to-back probes : one or several 56-bytes probe(s) behind one or two bigger probes. We make the size of the bigger probe vary. *Left* : reordering ratios for the two big probes (i.e. the small probe passing them). *Right* : reordering ratios for the 16 small probes (i.e. they pass the bigger one).

In fig 4.8, on the right, we can see the ratio of 1, 2, . . . 16 probes passing the bigger one at the same time.

We didn't want to analyse these results, since this appeared to be very specific to our route : none of the few experiments we made on other routes exhibit this reordering behaviour. Moreover, the packet size dependance of the reordering ratios wasn't easily understandable at all. The only thing we could have expected is that, the bigger a packet is, the more easy small probes will pass it. However, we tried to redo these experiments with ICMP packets, and the results were quite interesting : we had no natural reordering any more, apart from a very small ratio (less than 1%). This is a nice trick to know, as that kind of natural reordering make active probing based on reordering much more difficult.

Chapter 5

Conclusion

Here, we will present what we learned with ICMP. We first designed quite a few experiments based on reordering caused by *spoofed* ICMP. Unfortunately, we didn't have time to focus on a given experiment and make it work properly. But this failures led us to the general study of ICMP behaviour, that appears to be not exactly what one usually expects. So, we found lots of ICMP specificities that finally helped us to understand why our first experiment (with the reordering signature) didn't work : The combination of ICMP extra delay and natural reordering made it impossible to work. Now, we would like to sum up all the tools or methods that could be used for active probing, but also some difficulties or traps for the active prober.

5.0.1 Traps

- Natural reordering may be a source of big errors when probing the reordering signature.
- ICMP delay might be different than UDP (and probably TCP)
- An ICMP-TE generated by a *spoofed* hop-limited probe may take another route than the one we expect.
- ICMP *Echo Reply* packets do not always generate ICMP-TE answers when their TTL reach zero. This is also true for UDP, but less likely.
- ICMP back-to-back probes may have more difficulties to stay back-to-back than UDP probes.

5.0.2 New Methods

- The packet size dependance of ICMP excess delay, or of ICMP *Echo Reply* generation time, could give us some information about the router internal bandwidth (CPU bandwidth ?)
- Reordering signature, with all its variants : *spoofed* probes or not, ICMP *Time Exceeded* or *Echo Reply*, or both, usage of the packet size dependance of the *Echo Reply* generation time . . .

- UDP packets can pass ICMP packets because of the slower processing of them in some routers. This could give us another way of creating reordering.

5.0.3 Future Work

We will have to do a lot of work if we want to try all the methods described above, especially using the reordering signature, that look very promising because of its great flexibility. But the first thing to do is to learn more about all the traps discovered during this project. It has been useful to point out their existence, as they could all be the sources of great annoyance for the unwarned active prober, but it would be even more useful to make some larger studies about them.

Bibliography

[1] W. Richard Stevens, “TCP/IP Illustrated, Volume 1 : The Protocols”, 1994

[2] Attila Pásztor and Darryl Veitch, “A precision infrastructure for acting probing”, *PAM2001, Workshop on Passive and Active Networking*, Amsterdam, The Netherlands, 2001

[3] Attila Pásztor and Darryl Veitch, “Active probing using packet quartets”, 2002

[4] Active Probing Testbed, http://www.cubinlab.ee.mu.oz.au/probing/
Software Documentation, . . . /probing/software.shtml#intro

[5] Endace Measurement Systems : http://www.endace.com
DAG project webpage : http://dag.cs.waikato.nz

[6] `tcpdump` webpage : http://www.tcpdump.org

[7] Ramesh Govindan and Vern Paxson, “Estimating router ICMP generation delays”, in *Proceedings of Passive & Active Measurement : PAM-2002*, Fort Collins, Colorado, USA, 2002

[8] “Requirements for IP version 4 Routers”, *RFC 1812*, 1995
http://www.faqs.org/rfcs/rfc1812.html

[9] “Internet Protocol Specification”, *RFC 791*, 1981
http://www.faqs.org/rfcs/rfc791.html

[10] “Internet Control Message Protocol”, *RFC 792*, 1981
http://www.faqs.org/rfcs/rfc792.html

[11] `traceroute` manual page, http://www.zytek.com/traceroute.man.html