CNP3-Practice Documentation

Release 0.0

Olivier Bonaventure Mickael Hoerdt, Laurent Vanbever and Virg

November 11, 2010

CONTENTS

This document contains the questions for the practical part of the INGI2141 course during the 2010-2011 academic year. The html and pdf versions will be updated every week. These exercises have been written by Olivier Bonaventure with the help of Mickael Hoerdt, Damien Saucez and Laurent Vanbever and Virginie van den Schriek

CHAPTER

ONE

THE ALTERNATING BIT PROTOCOL

The objective of this set of exercises is to better understand the basic mechanisms of the alternating bit protocol and the utilisation of the socket interface with the connectionless transport service.

1. Consider the Alternating Bit Protocol as described in the book.

- How does the protocol recover from the loss of a data segment ?
- How does the protocol recovers from the loss of an acknowledgement ?

2. A student proposed to optimise the Alternating Bit Protocol by adding a negative acknowledgment, i.e. the receiver sends a *NAK* control segment when it receives a corrupted data segment. What kind of information should be placed in this control segment and how should the sender react when receiving such a *NAK* ?

3. Transport protocols rely on different types of checksums to verify whether segments have been affected by transmission errors. The most frequently used checksums are :

   - the Internet checksum used by UDP, TCP and other Internet protocols which is defined in **RFC 1071** and implemented in various modules, e.g. http://ilab.cs.byu.edu/cs460/code/ftp/ichecksum.py for a python implementation
   - the 16 bits or the 32 bits Cyclical Redundancy Checks (CRC) that are often used on disks, in zip archives and in datalink layer protocols. See http://docs.python.org/library/binascii.html for a python module that contains the 32 bits CRC
   - the Alder checksum defined in **RFC 2920** for the SCTP protocol but replaced by a CRC later
   - the Fletcher checksum

   By using your knowledge of the Internet checksum, can you find a transmission error that will not be detected by the Internet checksum ?

4. The CRCs are efficient error detection codes that are able to detect :

   - all errors that affect an odd number of bits
   - all errors that affect a sequence of bits which is shorter than the length of the CRC

   Carry experiments with one implementation of CRC-32 to verify that this is indeed the case.

5. Checksums and CRCs should not be confused with secure hash functions such as MD5 defined in **RFC 1321** or SHA-1 described in **RFC 4634**. Secure hash functions are used to ensure that files or sometimes packets/segments have not been modified. Secure hash functions aim at detecting malicious changes while checksums and CRCs only detect random transmission errors. Perform some experiments with hash functions such as those defined in the http://docs.python.org/library/hashlib.html python hashlib module to verify that this is indeed the case.

6. A version of the Alternating Bit Protocol supporting variable length segments uses a header that contains the following fields :

   - a number (0 or 1)
   - a length field that indicates the length of the data

- a CRC

To speedup the transmission of the segments, a student proposes to compute the CRC over the data part of the segment but not over the header. What do you think of this optimisation ?

7. On Unix hosts, the `ping(8)` command can be used to measure the round-trip-time to send and receive packets from a remote host. Use `ping(8)` to measure the round-trip to a remote host. Chose a remote destination which is far from your current location, e.g. a small web server in a distant country. There are implementations of ping in various languages, see e.g. http://pypi.python.org/pypi/ping/0.2 for a python implementation of ping
8. How would you set the retransmission timer if you were implementing the Alternating Bit Protocol to exchange files with a server such as the one that you measured above ?
9. What are the factors that affect the performance of the Alternating Bit Protocol ?
10. Links are often considered as symmetrical, i.e. they offer the same bandwidth in both directions. Symmetrical links are widely used in Local Area Networks and in the core of the Internet, but there are many asymmetrical link technologies. The most common example are the various types of ADSL and CATV technologies. Consider an implementation of the Alternating Bit Protocol that is used between two hosts that are directly connected by using an asymmetric link. Assume that a host is sending segments containing 10 bytes of control information and 90 bytes of data and that the acknowledgements are 10 bytes long. If the round-trip-time is negligible, what is the minimum bandwidth required on the return link to ensure that the transmission of acknowledgements is not a bottleneck ?
11. Derive a mathematical expression that provides the *goodput* achieved by the Alternating Bit Protocol assuming that :
    - Each segment contains D bytes of data and c bytes of control information
    - Each acknowledgement contains c bytes of control information
    - The bandwidth of the two directions of the link is set to B bits per second
    - The delay between the two hosts is s seconds in both directions

    The goodput is defined as the amount of SDUs (measured in bytes) that is successfully transferred during a period of time
12. The socket interface allows you to use the UDP protocol on a Unix host. UDP provides a connectionless unreliable service that in theory allows you to send SDUs of up to 64 KBytes.

- Implement a small UDP client and a small UDP server (in python, you can start from the example provided in http://docs.python.org/library/socket.html but you can also use C or java)
- run the client and the servers on different workstations to determine experimentally the largest SDU that is supported by your language and OS. If possible, use different languages and Operating Systems in each group.

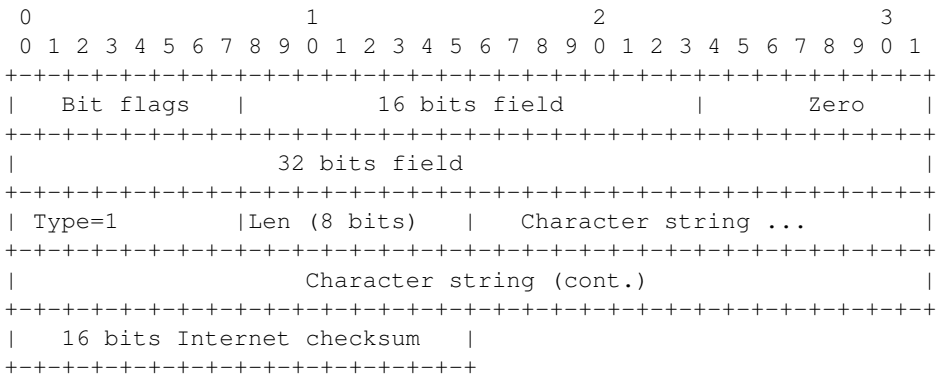
13. By using the socket interface, implement on top of the connectionless unreliable service provided by UDP a simple client that sends the following message shown in the figure below.

    In this message, the bit flags should be set to *01010011b*, the value of the 16 bits field must be the square root of the value contained in the 32 bits field, the character string must be an ASCII representation (without any trailing *0*) of the number contained in the 32 bits character field. The last 16 bits of the message contain an Internet checksum that has been computed over the entire message.

    Upon reception of a message, the server verifies that :
    - the flag has the correct value
    - the 32 bits integer is the square of the 16 bits integer
    - the character string is an ASCII representation of the 32 bits integer
    - the Internet checksum is correct

    If the verification succeeds, the server returns a SDU containing *11111111b*. Otherwise it returns *01010101b*

Inside each group, implement two different clients and two different servers (both using different langages). The clients and the servers must run on both the Linux workstations and the Sun server (*sirius*). Verify the interoperability of the clients and the servers inside the group. You can use C, Java or python to write these implementations.



14. Consider an Alternating Bit Protocol that is used over a link that suffers from deterministic errors. When the error ratio is set to $\frac{1}{p}$, this means that $p - 1$ bits are transmitted correctly and the p^{th} bit is corrupted. Discuss the factors that affect the performance of the Alternating Bit Protocol over such a link.

CHAPTER

TWO

GO-BACK-N AND SELECTIVE REPEAT

Go-back-n and selective repeat are the basic mechanisms used in reliable window-based transport-layer protocols. These questions cover these two mechanisms in details. You do not need to upload the time sequence diagrams on the svn repository, bring them with you on paper.

1. Amazon provides the S3 storage service where companies and researchers can store lots of information and perform computations on the stored information. Amazon allows users to send files through the Internet, but also by sending hard-disks. Assume that a 1 Terabyte hard-disk can be delivered within 24 hours to Amazon by courier service. What is the minimum bandwidth required to match the bandwidth of this courier service ?
2. Several large datacenters operators (e.g. Microsoft and google) have announced that they install servers as containers with each container hosting up to 2000 servers. Assuming a container with 2000 servers and each storing 500 GBytes of data, what is the time required to move all the data stored in one container over one 10 Gbps link ? What is the bandwidth of a truck that needs 10 hours to move one container from one datacenter to another.
3. What are the techniques used by a go-back-n sender to recover from :

- transmission errors
- losses of data segments
- losses of acknowledgements

4. Consider a b bits per second link between two hosts that has a propagation delay of t seconds. Derive a formula that computes the time elapsed between the transmission of the first bit of a d bytes segment from a sending host and the reception of the last bit of this segment on the receiving host.
5. Consider a go-back-n sender and a go-back receiver that are directly connected with a 10 Mbps link that has a propagation delay of 100 milliseconds. Assume that the retransmission timer is set to three seconds. If the window has a length of 4 segments, draw a time-sequence diagram showing the transmission of 10 segments (each segment contains 10000 bits):

- when there are no losses
- when the third and seventh segments are lost
- when the second, fourth, sixth, eighth, ... acknowledgements are lost
- when the third and fourth data segments are reordered (i.e. the fourth arrives before the third)

6. Same question when using selective repeat instead of go-back-n. Note that the answer is not necessarily the same.
7. Consider two high-end servers connected back-to-back by using a 10 Gbps interface. If the delay between the two servers is one millisecond, what is the throughput that can be achieved by a transport protocol that is using 10,000 bits segments and a window of

- one segment

- ten segments
- hundred segments

8. Consider two servers are directly connected by using a *b* bits per second link with a round-trip-time of *r* seconds. The two servers are using a transport protocol that sends segments containing *s* bytes and acknowledgements composed of *a* bytes. Can you derive a formula that computes the smallest window (measured in segments) that is required to ensure that the servers will be able to completely utilise the link ?
9. Same question as above if the two servers are connected through an asymmetrical link that transmits *bu* bits per second in the direction used to send data segments and *bd* bits per second in the direction used to send acknowledgements.
10. The Trivial File Transfer Protocol is a very simple file transfer protocol that is often used by diskless hosts when booting from a server. Read the TFTP specification in **RFC 1350** and explain how TFTP recovers from transmission errors and losses.
11. Is it possible for a go-back-n receiver to interoperate with a selective-repeat sender ? Justify your answer.
12. Is it possible for a selective-repeat receiver to interoperate with a go-back-n sender ? Justify your answer.
13. The go-back-n and selective repeat mechanisms that are described in the book exclusively rely on cumulative acknowledgements. This implies that a receiver always returns to the sender information about the last segment that was received in-sequence. If there are frequent losses or reordering, a selective repeat receiver could return several times the same cumulative acknowledgment. Can you think of other types of acknowledgements that could be used by a selective repeat receiver to provide additional information about the out-of-sequence segments that it has received. Design such acknowledgements and explain how the sender should react upon reception of this information.
14. The *goodput* achieved by a transport protocol is usually defined as the number of application layer bytes that are exchanged per unit of time. What are the factors that can influence the *goodput* achieved by a given transport protocol ?
15. The Transmission Control Protocol (TCP) attaches a 40 bytes header to each segment sent. Assuming an infinite window and no losses nor transmission errors, derive a formula that computes the maximum TCP goodput in function of the size of the segments that are sent.
16. A go-back-n sender uses a window size encoded in a *n* bits field. How many segments can it send without receiving an acknowledgement ?
17. Consider the following situation. A go-back-n receiver has sent a full window of data segments. All the segments have been received correctly and in-order by the receiver, but all the returned acknowledgements have been lost. Show by using a time sequence diagram (e.g. by considering a window of four segments) what happens in this case. Can you fix the problem on the go-back-n sender ?
18. Same question as above, but assume now that both the sender and the receiver implement selective repeat. Note the the answer will be different from the above question.
19. Consider a transport that supports window of one hundred 1250 Bytes segments. What is the maximum bandwidth that this protocol can achieve if the round-trip-time is set to one second ? What happens if, instead of advertising a window of one hundred segments, the receiver decides to advertise a window of 10 segments ?
20. Explain under which circumstances a transport entity could advertise a window of 0 segments ?
21. The socket library is also used to develop applications above the reliable bytestream service provided by TCP. We have installed on the *sirius.info.ucl.ac.be* server a simple server that provides a simple client-server service. The service operates as follows :
    - the server listens on port *62141* for a TCP connection
    - upon the establishment of a TCP connection, the server sends an integer by using the following TLV format :
        - the first two bits indicate the type of information (01 for ASCII, 10 for boolean)
        - the next six bits indicate the length of the information (in bytes)

  - An ASCII TLV has a variable length and the next bytes contain one ASCII charater per byte. A boolean TLV has a length of one byte. The byte is set to *00000000b* for *true* and *00000001b* for false.

- the client replies by sending the received integer encoded as a 32 bits integer in *network byte order*
- the server returns a TLV containing *true* if the integer was correct and a TLV containing *false* otherwise and closes the TCP connection

Each group of two students must implement a client to interact with this server in C, Java or python.

CHAPTER

THREE

PROGRAMMING PROJECT

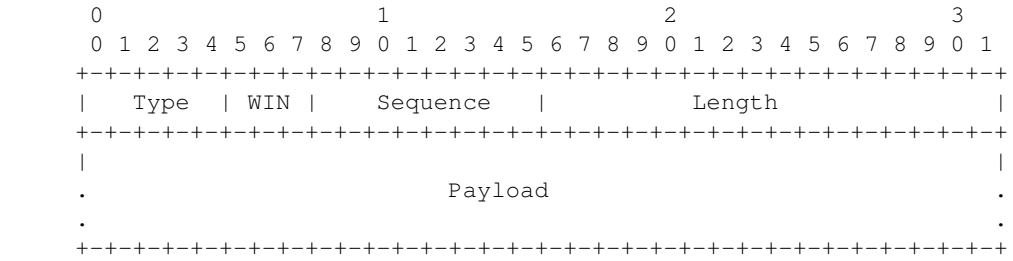
Your objective in this project is to implement a simple reliable transport protocol by groups of 2 students. These groups must be subgroups of the main groups that are registered on icampus. If the number of students in an icampus group is odd, there can be one group of three students.

The protocol uses a sliding window to transmit more than one segment without being forced to wait for an acknowledgment. Your implementation must support variable size sliding window as the other end of the flow can send its maximum window size. The window size is encoded as a three bits unsigned integer.

The protocol identifies the DATA segments by using sequence numbers. The sequence number of the first segment must be 0. It is incremented by one for each new segment. The receiver must acknowledge the delivered segments by sending an ACK segment. The sequence number field in the ACK segment always contains the sequence number of the next expected in-sequence segment at the receiver. The flow of data is unidirectional, meaning that the sender only sends DATA segments and the receiver only sends ACK segments.

To deal with segments losses, the protocol must implement a recovery technique such as go-back-n or selective repeat and use retransmission timers. The project will partially be evaluated on the quality of the recovery technique. Groups of three must implement the selective repeat technique while groups of two can implement a simpler recovery scheme such as go-back-n.

```
Segment format
```



- *Type*: segment type
  - 0x1 DATA segment.
  - 0x2 ACK segment
- *WIN*: the size of the current window (an integer encoded as a 3 bits field). In DATA segments, this field indicates the size of the sending window of the sender. In ACK segments, this field indicates the current value of the receiving window.
- *Sequence*: Sequence number (8 bits unsigned integer), starts at 0. The sequence number is incremented by 1 for each new DATA segment sent by the sender. Inside an ACK segment, the sequence field carries the sequence number of the next in-sequence segment that is expected by the receiver.
- *Length*: length of the payload in multiple of one byte. All DATA segments contain a payload with 512 bytes of data, except the last DATA segment of a transfert that can be shorter. The reception of a DATA segment whose length is different than 512 indicates the end of the data transert.
- Payload: the data to send

3.1 Deliverables

Before October 22nd, 2010 at 23:59 each sub-group must submit its commented source code (with a Makefile) on the SVN and a short report (up to four pages in pdf format) describing the chosen recovery technique, the architecture of the client and server and the tests that have been carried out. Each group must implement both a receiver and a sender. The implementation language can be chosen among C, Java and Python.

The client and the server exchange UDP datagrams that contain the DATA and ACK segments. They must be command-line tools that allow to transmit one binary file and support the following parameters :

```
sender <destination_DNS_name> <destination_port_number> <window_size> <input_file>

receiver <listening_port_number> <window_size> <output_file>
```

A demo session will be organised on Tuesday October 26th. During the demo session, you will be invited to demonstrate that your implementation is operational and is interoperable with another. You also need to perform tests to show that your implementations works well in case of segment losses. For these tests, you can use a random number generator to probabilistically drop received segments and introduce random delays upon the arrival of a segment.

3.2 Demonstration

The demonstration of your project will take place in the intel room at :

- group 1, 2, 3: October 26th 10:45 am
- group 4, 5, 6: October 26th 11:45 am

We will test your implementations. For that, we will link you with each sub-group of the other groups with the same subgroup number. You personally assign the sub-group number inside your group. The receivers will be tested on sirius, the senders from the Intel room. Group 1 (resp. 4) sends to group 2 (resp. 5); Group 1 (resp. 4) receives from 3 (resp. 6); group 2 (resp. 5) sends to group 3 (resp. 6). The sender window size will be set to 6 at startup. The receiver window size will be set to 3 at startup.

We will provide you 6 files to each sender. The receiver must receive all of them and prove the transfer correctness by giving the md5 hashes (digest -v -a md5 <file>.‘).

The receiver port number is computed is defined as 6SGsg, where

- S: sender sub-group number
- G: sender group number
- s: receiver sub-group number
- g: receiver group number

e.g., port 61213 means that sub-group 1 of group 2 sends traffic to sub-group 1 of group 3.

Evaluation

Code: 50%

- no compilation -> 0
- increment of sequence number/ack (1)
- ack processing should be decoupled from data processing (1)
- window management (2)
- timer management (2)
- network byte order (1)
- architecture (2)

- code readability/documentation/synopsis (1)

Report: 25%

- architecture description (2)
- pitfall highlighting (2)
- validation (5)
- further work (1)

Demo: 25%

- correct transfers (6)
- support random loss/delay (2)
- ability to explain the code/events (2)

CHAPTER

FOUR

THE TRANSMISSION CONTROL PROTOCOL

The Transmission Control Protocol plays a key role in the TCP/IP protocol suite by providing a reliable byte stream service on top of the unreliable connectionless service provided by IP. During this exercise session, you will learn how to establish correctly a TCP connection and analyse packet traces that contain TCP segments. Note that some of exercises involve the creation of non-standard TCP segments. These exercises cannot be performed outside the netkit environment that is described below.

4.1 Packet trace analysis tools

When debugging networking problems or to analyse performance problems, it is sometimes useful to capture the segments that are exchanged between two hosts and to analyse them.

Several packet trace analysis tools are available, either as commercial or open-source tools. These tools are able to capture all the packets exchanged on a link. Of course, capturing packets require administrator privileges. They can also analyse the content of the captured packets and display information about them. The captured packets can be stored in a file for offline analysis.

tcpdump is probably one of the most well known packet capture software. It is able to both capture packets and display their content. tcpdump is a text-based tool that can display the value of the most important fields of the captured packets. Additional information about tcpdump may be found in *tcpdump(1)*. The text below is an example of the output of tcpdump for the first TCP segments exchanged on an scp transfer between two hosts

```
21:05:56.230737 IP 192.168.1.101.54150 > 130.104.78.8.22: S 1385328972:1385328972(0) win 65535 <ms
21:05:56.251468 IP 130.104.78.8.22 > 192.168.1.101.54150: S 3627767479:3627767479(0) ack 13853289
21:05:56.251560 IP 192.168.1.101.54150 > 130.104.78.8.22: . ack 1 win 65535 <nop,nop,timestamp 27
21:05:56.279137 IP 130.104.78.8.22 > 192.168.1.101.54150: P 1:21(20) ack 1 win 49248 <nop,nop,time
21:05:56.279241 IP 192.168.1.101.54150 > 130.104.78.8.22: . ack 21 win 65535 <nop,nop,timestamp 2
21:05:56.279534 IP 192.168.1.101.54150 > 130.104.78.8.22: P 1:22(21) ack 21 win  65535 <nop,nop,t
21:05:56.303527 IP 130.104.78.8.22 > 192.168.1.101.54150: . ack 22 win 49248 <nop,nop,timestamp 1
21:05:56.303623 IP 192.168.1.101.54150 > 130.104.78.8.22: P 22:814(792) ack 21 win 65535 <nop,nop
```

You can easily recognise in the output above the *SYN* segment containing the *MSS*, *window scale*, *timestamp* and *sackOK* options, the *SYN+ACK* segment whose *wscale* option indicates that the server uses window scaling for this connection and then the first few segments exchanged on the connection.

wireshark is more recent than tcpdump. It evolved from the ethereal packet trace analysis software. It can be used as a text tool like tcpdump. For a TCP connection, wireshark would provide almost the same output as tcpdump. The main advantage of wireshark is that it also includes a graphical user interface that allows to perform various types of analysis on a packet trace.

The wireshark window is divided in three parts. The top part of the window is a summary of the first packets from the trace. By clicking on one of the lines, you can show the detailed content of this packet in the middle part of the window. The middle of the window allows you to inspect all the fields of the captured packet. The bottom part

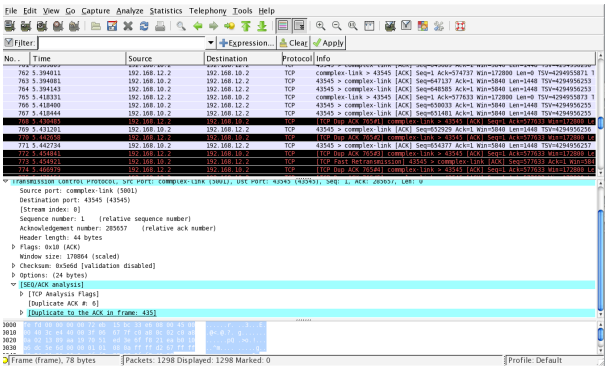


Figure 4.1: Wireshark : default window

of the window is the hexadecimal representation of the packet, with the field selected in the middle window being highlighted.

wireshark is very good at displaying packets, but it also contains several analysis tools that can be very useful. The first tool is *Follow TCP stream*. It is part of the *Analyze* menu and allows you to reassemble and display all the payload exchanged during a TCP connection. This tool can be useful if you need to analyse for example the commands exchanged during a SMTP session.

The second tool is the flow graph that is part of the *Statistics* menu. It provides a time sequence diagram of the packets exchanged with some comments about the packet contents. See blow for an example.

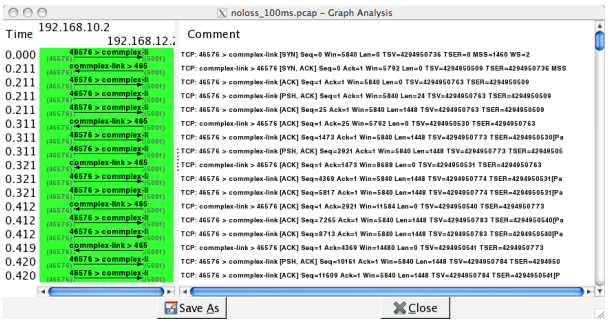


Figure 4.2: Wireshark : flow graph

The third set of tools are the *TCP stream graph* tools that are part of the *Statistics menu*. These tools allow you to plot various types of information extracted from the segments exchanged during a TCP connection. A first interesting graph is the *sequence number graph* that shows the evolution of the sequence number field of the captured segments with time. This graph can be used to detect graphically retransmissions.

A second interesting graph is the *round-trip-time* graph that shows the evolution of the round-trip-time in function of time. This graph can be used to check whether the round-trip-time remains stable or not. Note that from a packet trace, wireshark can plot two *round-trip-time* graphs, One for the flow from the client to the server and the other one. wireshark will plot the *round-trip-time* graph that corresponds to the selected packet in the top wireshark window.

4.2 Emulating a network with Netkit

Netkit is network emulator based on User Mode Linux. It allows to easily set up an emulated network of Linux machines, that can act as end-host or routers.

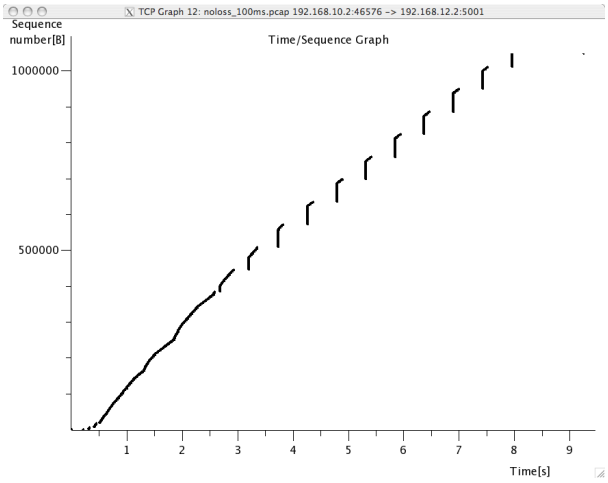


Figure 4.3: Wireshark : sequence number graph

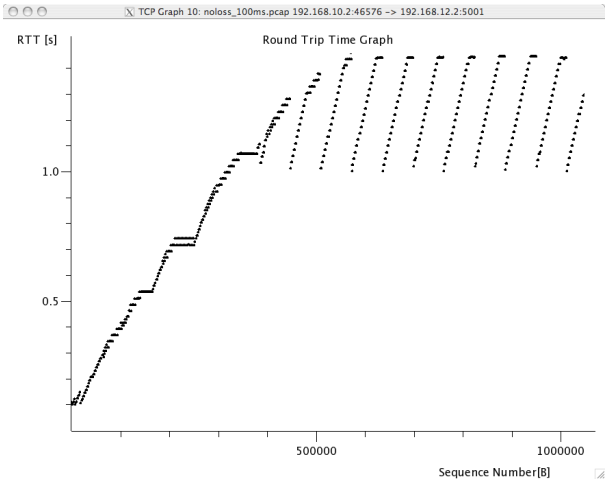


Figure 4.4: Wireshark : round-trip-time graph

4.2.1 Where can I find Netkit?

Netkit is available online. Files can be downloaded from http://wiki.netkit.org/index.php/Download_Official, and instructions for the installations are available here : http://wiki.netkit.org/download/netkit/INSTALL .

Netkit has already been installed in the student labs, in */etinfo/applications/netkit* . All you have to do in order to use it is to set the following environment variables :

```
export NETKIT_HOME=/etinfo/applications/netkit
export MANPATH=:$NETKIT_HOME/man
export PATH=$NETKIT_HOME/bin:$PATH
```

It is usually convenient to put those lines in your shell initialization file.

4.2.2 How do I use Netkit?

There are two ways to use Netkit : The manual way, and by using pre-configured labs. In the first case, you boot and control each machine individually, using the commands starting with a "v" (for virtual machine). In the second case, you can start a whole network in a single operation. The commands for controlling the lab start with a "l". The man pages of those commands is available from http://wiki.netkit.org/man/man7/netkit.7.html

You must be careful not to forgot to stop your virtual machines and labs, using either *vhalt* or *lhalt*.

4.2.3 Example of using a lab

A lab is simply a directory containing at least a configuration file called *lab.conf*, and one directory for each virtual machine. In the case the lab available on iCampus, the network is composed of two pc, pc1 and pc2, both of them being connected to a router r1. The lab.conf file contains the following lines :

```
pc1[0]=A
pc2[0]=B
r1[0]=A
r1[1]=B
```

This means that pc1 and r1 are connected to a "virtual LAN" named A via their interface eth0, while pc2 and r1 are connected to the "virtual LAN" B via respectively their interfaces eth0 and eth1.

The directory of each device is initially empty, but will be used by Netkit to store their filesystem.

The lab directory can contain optional files. In the lab provided to you, the "pc1.startup" file contains the shell instructions to be executed on startup of the virtual machine. In this specific case, the script configures the interface eth0 to allow traffic exchanges between pc1 and r1, as well as the routing table entry to join pc2.

Starting a lab consists thus simply in unpacking the provided archive, going into the lab directory and typing *lstart* to start the network.

4.2.4 File sharing between virtual machines and host

Virtual machines can access to the directory of the lab they belong to. This repertory is mounted in their filesystem at the path */hostlab*.

4.2.5 Tools available on Netkit

As the virtual machines run Linux, standard networking tools such as hping, tcpdump, netstats etc. are available as usual.

Note that capturing network traces can be facilitated by using the *uml_dump* extension available at http://kartoch.msi.unilim.fr/blog/?p=19 . This extension is already installed in the Netkit installation on the student lab. In order to capture the traffic exchanged on a given 'virtual LAN', you simply need to issue the command

vdump <LAN name> on the host. If you want to pipe the trace to wireshark, you can use *vdump A | wireshark -i - -k*

In the lab provided in iCampus, you can find a simple Python client/server application that establishes TCP connections. Feel free to re-use this code to perform your analysis.

4.3 Questions

1. A TCP/IP stack receives a SYN segment with the sequence number set to 1234. What will be the value of the acknowledgement number in the returned SYN+ACK segment ?
2. Is it possible for a TCP/IP stack to return a SYN+ACK segment with the acknowledgement number set to *0* ? If no, explain why. If yes, what was the content of the received SYN segment.
3. Open the tcpdump packet trace `traces/trace.5connections_opening_closing.pcap` and identify the number of different TCP connections that are established and closed. For each connection, explain by which mechanism they are closed. Analyse the initial sequence numbers that are used in the SYN and SYN+ACK segments. How do these initial sequence numbers evolve ? Are they increased every 4 microseconds ?
4. The tcpdump packet trace `traces/trace.5connections.pcap` contains several connection attempts. Can you explain what is happening with these connection attempts ?
5. The tcpdump packet trace `traces/trace.ipv6.google.com.pcap` was collected from a popular website that is accessible by using IPv6. Explain the TCP options that are supported by the client and the server.
6. The tcpdump packet trace `traces/trace.sirius.info.ucl.ac.be.pcap` Was collected on the departmental server. What are the TCP options supported by this server ?
7. A TCP implementation maintains a Transmission Control Block (TCB) for each TCP connection. This TCB is a data structure that contains the complete "*state*" of each TCP connection. The TCB is described in **RFC 793**. It contains first the identification of the TCP connection :

- *localip* : the IP address of the local host
- *remoteip* : the IP address of the remote host
- *remoteport* : the TCP port used for this connection on the remote host
- *localport* : the TCP port used for this connection on the local host. Note that when a client opens a TCP connection, the local port will often be chosen in the ephemeral port range (49152 <= localport <= 65535).
- *sndnxt* : the sequence number of the next byte in the byte stream (the first byte of a new data segment that you send will use this sequence number)
- *snduna* : the earliest sequence number that has been sent but has not yet been acknowledged
- *rcvnxt* : the sequence number of the next byte that your implementation expects to receive from the remote host. For this exercise, you do not need to maintain a receive buffer and your implementation can discard the out-of-sequence segments that it receives
- *sndwnd* : the current sending window
- *rcvwnd* : the current window advertised by the receiver

Using the `traces/trace.sirius.info.ucl.ac.be.pcap` packet trace, what is the TCB of the connection on host *130.104.78.8* when it sends the third segment of the trace ?

1. The tcpdump packet trace `traces/trace.maps.google.com` was collected by containing a popular web site that provides mapping information. How many TCP connections were used to retrieve the information from this server ?

2. Some network monitoring tools such as ntop collect all the TCP segments sent and received by a host or a group of hosts and provide interesting statistics such as the number of TCP connections, the number of bytes exchanged over each TCP connection, ... Assuming that you can capture all the TCP segments sent by a host, propose the pseudocode of an application that would list all the TCP connections established and accepted by this host and the number of bytes exchanged over each connection. Do you need to count the number of bytes contained inside each segment to report the number of bytes exchanged over each TCP connection ?

3. There are two types of firewalls [1] : special devices that are placed at the border of campus or enterprise networks and software that runs on endhosts. Software firewalls typically analyse all the packets that are received by a host and decide based on the packet's header and contents whether it can be processed by the host's network stack or must be discarded. System administrators often configure firewalls on laptop or student machines to prevent students from installing servers on their machines. How would you design a simple firewall that blocks all incoming TCP connections but still allows the host to establish TCP connections to any remote server ?

4. Using the netkit lab explained above, perform some tests by using *hping3(8)*. *hping3(8)* is a command line tool that allows anyone (having system administrator privileges) to send special IP packets and TCP segments. *hping3(8)* can be used to verify the configuration of firewalls [1] or diagnose problems. We will use it to test the operation of the Linux TCP stack running inside netkit.

1. On the server host, launch *tcpdump(1)* with *-vv* as parameter to collect all packets received from the client and display them. Using *hping3(8)* on the client host, send a valid SYN segment to one unused port on the server host (e.g. *12345*). What are the contents of the segment returned by the server ?

2. Perform the same experiment, but now send a SYN segment towards port *7*. This port is the default port for the discard service (see *services(5)*) launched by *xinetd(8)*). What segment does the server sends in reply ? What happens upon reception of this segment ? Explain your answer.

1. The Linux TCP/IP stack can be easily configured by using *sysctl(8)* to change kernel configuration variables. See http://fasterdata.es.net/TCP-tuning/ip-sysctl-2.6.txt for a recent list of the sysctl variables on the Linux TCP/IP stack. Try to disable the selective acknowledgements and the RFC1323 timestamp and large window options and open a TCP connection on port *7* on the server by using :manpage:telnet‘(1)‘. Check by using *tcpdump(1)* the effect of these kernel variables on the segments sent by the Linux stack in netkit.

2. Network administrators sometimes need to verify which networking daemons are active on a server. When logged on the server, several tools can be used to verify this. A first solution is to use the *netstat(8)* command. This command allows you to extract various statistics from the networking stack on the Linux kernel. For TCP, *netstat* can list all the active TCP connections with the state of their FSM. *netstat* supports the following options that could be useful during this exercises :

   - *-t* requests information about the TCP connections
   - *-n* requests numeric output (by default, *netstat* sends DNS queries to resolve IP addresses in hosts and uses */etc/services* to convert port number in service names, *-n* is recommended on netkit machines)
   - *-e* provides more information about the state of the TCP connections
   - *-o* provides information about the timers
   - *-a* provides information about all TCP connections, not only those in the Established state

   On the netkit lab, launch a daemon and start a TCP connection by using *telnet(1)* and use *netstat(8)* to verify the state of these connections.

   A second solution to determine which network daemons are running on a server is to use a tool like *nmap(1)*. *nmap(1)* can be run remotely and thus can provide information about a host on which the system administrator cannot login. Use *tcpdump(1)* to collect the segments sent by *nmap(1)* running on the client and explain how *nmap(1)* operates.

[1] A firewall is a software or hardware device that analyses TCP/IP packets and decides, based on a set of rules, to accept or discard the packets received or sent. The rules used by a firewall usually depend on the value of some fields of the packets (e.g. type of transport protocols, ports, ...). We will discuss in more details the operation of firewalls in the network layer chapter.

1. Long lived TCP connections are susceptible to the so-called *RST attacks*. Try to find additional information about this attack and explain how a TCP stack could mitigate such attacks.

CHAPTER

FIVE

TCP CONGESTION CONTROL

The TCP congestion control mechanisms, defined in **RFC 5681** plays a key role in today's Internet. Without this mechanism that was first defined and implemented in the late 1980s, the Internet would not have been able to continue to work until now. The objective of this exercise is to allow you to have a better understanding of the operation of TCP's congestion control mechanism by analysing all the segments exchanged over a TCP connection.

1. To understand the operation of the TCP congestion control mechanism, it is useful to draw some time sequence diagrams. Let us consider a simple scenario of a web client connected to the Internet that wishes to retrieve a simple web page from a remote web server. For simplicity, we will assume that the delay between the client and the server is 0.5 seconds and that the packet transmission times on the client and the servers are negligible (e.g. they are both connected to a 1 Gbps network). We will also assume that the client and the server use 1 KBytes segments.

1. Compute the time required to open a TCP connection, send an HTTP request and retrieve a 16 KBytes web page. This page size is typical of the results returned by search engines like google or bing. An important factor in this delay is the initial size of the TCP congestion window on the server. Assume first that the initial window is set to 1 segment as defined in **RFC 2001**, 4 KBytes (i.e. 4 segments in this case) as proposed in **RFC 3390** or 16 KBytes as proposed in a recent paper.

2. Perform the same analysis with an initial window of one segment is the third segment sent by the server is lost and the retransmission timeout is fixed and set to 2 seconds.

3. Same question as above but assume now that the 6th segment is lost.

4. Same question as above, but consider now the loss of the second and seventh acknowledgements sent by the client.

5. Does the analysis above changes if the initial window is set to 16 KBytes instead of one segment ?

2. Several MBytes have been sent on a TCP connection and it becomes idle for several minutes. Discuss which values should be used for the congestion window, slow start threshold and retransmission timers.

3. To operate reliably, a transport protocol that uses Go-back-n (resp. selective repeat) cannot use a window that is larger than $2^n - 1$ (resp. 2^{n-1}) segments. Does this limitation affects TCP ? Explain your answer.

4. Consider the simple network shown in the figure below. In this network, the router between the client and the server can only store on each outgoing interface one packet in addition to the packet that it is currently transmitting. It discards all the packets that arrive while its buffer is full. Assuming that you can neglect the transmission time of acknowledgements and that the server uses an initial window of one segment and has a retransmission timer set to 500 milliseconds, what is the time required to transmit 10 segments from the client to the server. Does the performance increase if the server uses an initial window of 16 segments instead ?

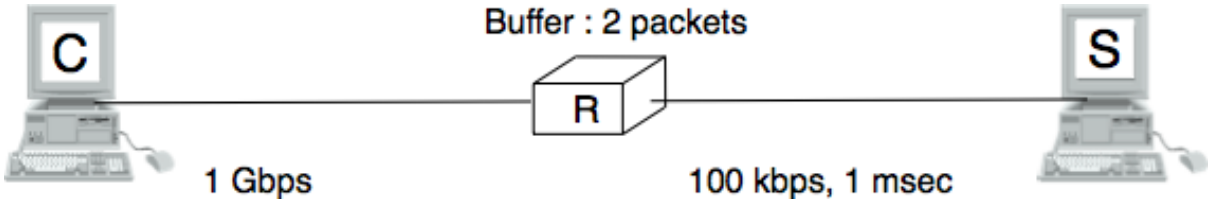


Figure 5.1: Simple network

5.1 Trace analysis

1. For the exercises below, we have performed measurements in an emulated [1] network similar to the one shown below.

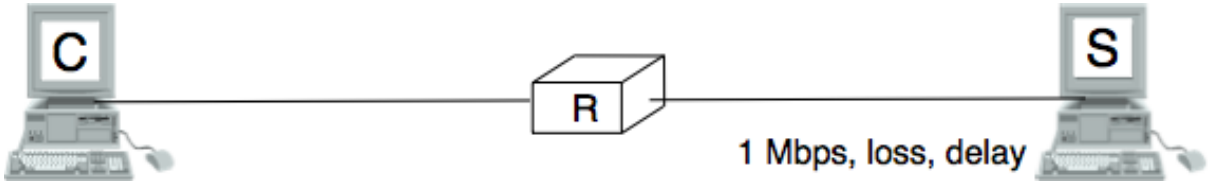


Figure 5.2: Emulated network

The emulated network is composed of three UML machines [2]: a client, a server and a router. The client and the server are connected via the router. The client sends data to the server. The link between the router and the client is controlled by using the netem Linux kernel module. This module allows us to insert additional delays, reduce the link bandwidth and insert random packet losses.

We used netem to collect several traces :

- `traces/trace0.pcap`
- `traces/trace1.pcap`
- `traces/trace2.pcap`
- `traces/trace3.pcap`

Each team of two students will analyse these traces by using wireshark or tcpdump. For each trace, you should

1. Identify the TCP options that have been used on the TCP connection
2. Try to find explanations for the evolution of the round-trip-time on each of these TCP connections. For this, you can use the *round-trip-time* graph of wireshark, but be careful with their estimation as some versions of wireshark are buggy
3. Verify whether the TCP implementation used implemented *delayed acknowledgements*
4. Inside each packet trace, find :

1. one segment that has been retransmitted by using *fast retransmit*. Explain this retransmission in details.
2. one segment that has been retransmitted thanks to the expiration of TCP's retransmission timeout. Explain why this segment could not have been retransmitted by using *fast retransmit*.

[1] With an emulated network, it is more difficult to obtain quantitative results than with a real network since all the emulated machines need to share the same CPU and memory. This creates interactions between the different emulated machines that do not happen in the real world. However, since the objective of this exercise is only to allow the students to understand the behaviour of the TCP congestion control mechanism, this is not a severe problem.

[2] For more information about the TCP congestion control schemes implemented in the Linux kernel, see http://linuxgazette.net/135/pfeiffer.html and http://www.cs.helsinki.fi/research/iwtcp/papers/linuxtcp.pdf or the source code of a recent Linux. A description of some of the sysctl variables that allow to tune the TCP implementation in the Linux kernel may be found in http://fasterdata.es.net/TCP-tuning/linux.html. For this exercise, we have configured the Linux kernel to use the NewReno scheme **RFC 3782** that is very close to the official standard defined in **RFC 5681**

5. wireshark contain several two useful graphs : the *round-trip-time* graph and the *time sequence* graph. Explain how you would compute the same graph from such a trace .
6. When displaying TCP segments, recent versions of wireshark contain *expert analysis* heuristics that indicate whether the segment has been retransmitted, whether it is a duplicate ack or whether the retransmission timeout has expired. Explain how you would implement the same heuristics as wireshark.
7. Can you find which file has been exchanged during the transfer ?

2. You have been hired as an networking expert by a company. In this company, users of a networked application complain that the network is very slow. The developers of the application argue that any delays are caused by packet losses and a buggy network. The network administrator argues that the network works perfectly and that the delays perceived by the users are caused by the applications or the servers where the application is running. To resolve the case and determine whether the problem is due to the network or the server on which the application is running. The network administrator has collected a representative packet trace that you can download from `traces/trace9.pcap`. By looking at the trace, can you resolve this case and indicate whether the network or the application is the culprit ?

CHAPTER

SIX

ROUTING PROTOCOLS

The network layer contains two types of protocols :

- the *data plane* protocols such as IP that define the format of the packets that are exchanged between routers and how they must be forwarded
- the *routing protocols*, that are part of the *control plane*. Routers exchange routing messages in order to build their routing and forwarding tables to forward the packets in the data plane

Several types of routing protocols are used in computer networks. In this set of exercises, you will study intradomain routing protocols. More precisely, you will analyse the operation of routing protocols that use distance vectors or link-state.

6.1 Questions

1. Routing protocols used in IP networks only use positive link weights. What would happen with a distance vector routing protocol in the network below that contains a negative link weight ?

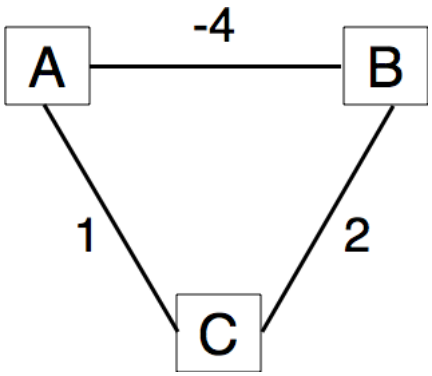


Figure 6.1: Simple network

2. Same question as above with a link weight set to zero.
3. When a network specialist designs a network, one of the problems that he needs to solve is to set the metrics of the links in his network. In the USA, the Abilene network interconnects most of the research labs and universities. The figure below shows the topology [1] of this network in 2009.

In this network, assume that all the link weights are set to 1. What is the paths followed by a packet sent by the router located in *Los Angeles* to reach :

- the router located in *New York*
- the router located in *Washington* ?

[1] This figure was downloaded from the Abilene observatory http://www.internet2.edu/observatory/archive/data-views.html. This observatory contains a detailed description of the Abilene network including detailed network statistics and all the configuration of the equipment used in the network.

Figure 6.2: The Abilene network

- Is it possible to configure the link metrics so that the packets sent by the router located in *Los Angeles* to the routers located in respectively *New York* and *Washington* do not follow the same path ?
- Is it possible to configure the link weights so that the packets sent by the router located in *Los Angeles* to router located in *New York* follow one path while the packets sent by the router located in *New York* to the router located in *Los Angeles* follow a completely different path ?
- Assume that the routers located in *Denver* and *Kansas City* need to exchange lots of packets. Can you configure the link metrics such that the link between these two routers does not carry any packet sent by another router in the network ?

4. In the five nodes network shown below, can you configure the link metrics so that the packets sent by router *E* to router *A* use link *B->A* while the packets sent by router *B* use links *B->D* and *D->A*?

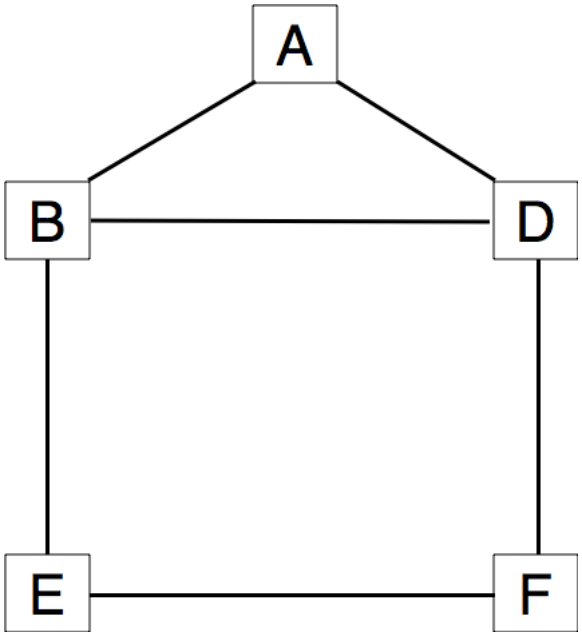


Figure 6.3: Simple five nodes network

5. In the five nodes network shown above, assuming that all link weights are set to *1*, what is the path followed by *E* to reach *D* ?
6. In the five nodes network shown above, can you configure the link weights so that the packets sent by router *E* (resp. *F*) follow the *E->B->A* path (resp. *F->D->B->A*) ?
7. In the five nodes network shown above, can you configure the link weights so that no packet is ever sent on the link between routers *B* and *D* ?

8. In the five nodes network shown above, can you configure the link weigths so that the packets sent by *A* towards *E* follow the path *A->D->B->E* while the packets sent by *B* towards *E* follow the path *B->D->F*?
9. In the five nodes network shown above, assuming unitary link weights, is it possible that the packets sent by *X* towards *Y* follow the reverse path of the packets sent by *Y* towards *X* ?
10. In the network above, is it possible to configure the link weights so that the paths *E->F->D->A*, *E->B->D->A* and *E->B->A* have the same cost ?
11. Consider the network below and assume that it uses distance vectors with the weights shown in the figure. List all the distance vectors that are exchanged until all routers have a route to reach each other router in the network.

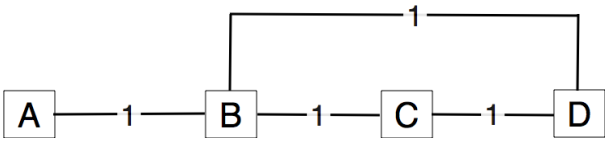


Figure 6.4: A four routers network

12. In the figure above, what are the messages exchanged when the network administrator reconfigures the link *B-D* with a weight of *9* ?
13. In the figure above (all link weights are set to *1*), what are the messages exchanged when link *C-D* fails ?
14. Consider again the four routers network shown above with the link weights shown in the figure, but now with a link state routing protocol.
    - What are the link state packets sent by each router in the network ?
    - What are the routing tables computed by using the Dijkstra algorithm on each router ?
    - Which link state packets are sent when link *B-C* fails ?
    - Which link state packets are sent when the network administrator reconfigures the link *B-D* to use a weight of *9* ? Does this change affects the routing tables ?
15. Consider the six-routers network shown in the figure below.
    - Assume that a distance vector routing protocol is used. What are the routing tables on each router ? Which messages are exchanged and when link *C-F* fails ?
    - Assume now that a link state routing protocol is used. What are the routing tables on each router ? Which link state packets are exchanged and when link *C-F* fails ?

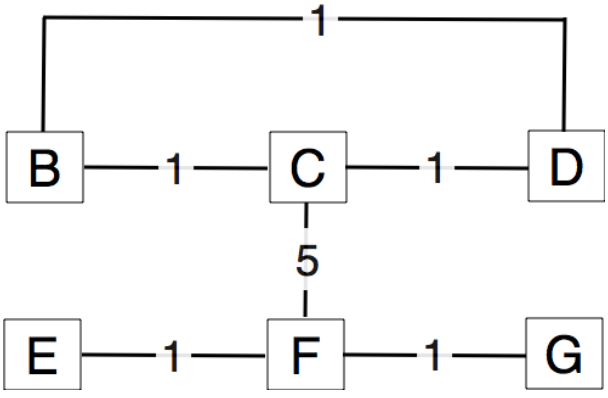


Figure 6.5: A six routers network

CHAPTER

SEVEN

IP ROUTING PROTOCOLS

1. In the previous questions, you have worked on the stable state of the routing tables computed by routing protocols. Let us now consider the transient problems that main happen when the network topology changes [1]. For this, consider the network topology shown in the figure below and assume that all routers use a distance vector protocol that uses split horizon.

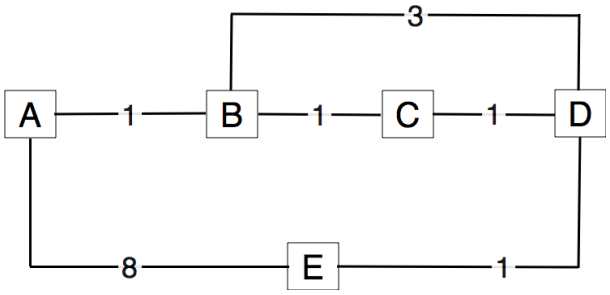


Figure 7.1: Simple network

If you compute the routing tables of all routers in this network, you would obtain a table such as the table below :

Destination	Routes on A	Routes on B	Routes on C	Routes on D	Routes on E
A	0	1 via A	2 via B	3 via C	4 via D
B	1 via B	0	1 via B	2 via C	3 via D
C	2 via B	1 via C	0	1 via C	2 via D
D	3 via B	2 via C	1 via D	0	1 via D
E	4 via B	3 via C	2 via D	1 via E	0

Distance vector protocols can operate in two different modes : *periodic updates* and *triggered updates*. *Periodic updates* is the default mode for a distance vector protocol. For example, each router could advertise its distance vector every thirty seconds. With the *triggered updates* a router sends its distance vector when its routing table changes (and periodically when there are no changes).

- Consider a distance vector protocol using split horizon and *periodic updates*. Assume that the link *B*-*C* fails. *B* and *C* update their local routing table but they will only advertise it at the end of their period. Select one ordering for the *periodic updates* and every time a router sends its distance vector, indicate the vector sent to each neighbor and update the table above. How many periods are required to allow the network to converge to a stable state ?
- Consider the same distance vector protocol, but now with *triggered updates*. When link *B*-*C* fails, assume that *B* updates its routing table immediately and sends its distance vector to *A* and *D*. Assume that both *A* and *D* process the received distance vector and that *A* sends its own distance vector, ... Indicate all the distance vectors that are exchanged and update the table

[1] The main events that can affect the topology of a network are : - the failure of a link. Measurements performed in IP networks have shown that such failures happen frequently and usually for relatively short periods of time - the addition of one link in the network. This may be because a new link has been provisioned or more frequently because the link failed some time ago and is now back - the failure/crash of a router followed by its reboot. - a change in the metric of a link by reconfiguring the routers attached to the link See http://totem.info.ucl.ac.be/lisis_tool/lisis-example/ for an analysis of the failures inside the Abilene network in June 2005 or http://citeseer.ist.psu.edu/old/markopoulou04characterization.html for an analysis of the failures affecting a larger ISP network

above each time a distance vector is sent by a router (and received by other routers) until all routers have learned a new route to each destination. How many distance vector messages must be exchanged until the network converges to a stable state ?

2. Consider the network shown below. In this network, the metric of each link is set to *1* except link *A*-*B* whose metric is set to *4* in both directions. In this network, there are two paths with the same cost between *D* and *C*. Old routers would randomly select one of these equal cost paths and install it in their forwarding table. Recent routers are able to use up to *N* equal cost paths towards the same destination.

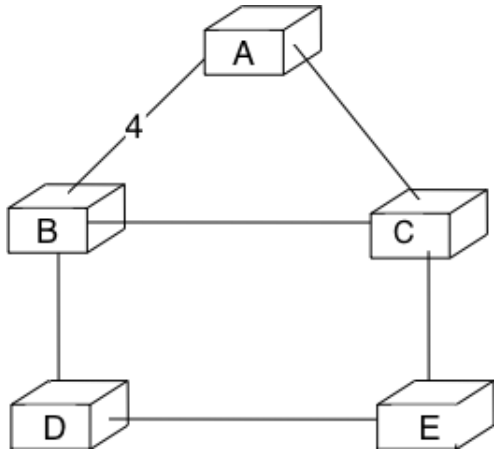


Figure 7.2: A simple network running a link state routing protocol

On recent routers, a lookup in the forwarding table for a destination address returns a set of outgoing interfaces. How would you design an algorithm that selects the outgoing interface used for each packet, knowing that to avoid reordering, all segments of a given TCP connection should follow the same path ?

2. Consider again the network shown above. After some time, the routing protocol converges and all routers compute the following routing tables :

Destination	Routes on A	Routes on B	Routes on C	Routes on D	Routes on E
A	0	2 via C	1 via A	3 via B,E	2 via C
B	2 via C	0	1 via B	1 via B	2 via D,C
C	1 via C	1 via C	0	2 via B,E	1 via C
D	3 via C	1 via D	2 via B,E	0	1 via D
E	2 via C	2 via C,D	1 via E	1 via E	0

An important difference between distance vector and link state routing is that link state routers flood link state packets that allow the other routers to recompute their own routing tables while distance vector routers exchange distance vectors. Consider that link *B*-*C* fails and that router *B* is the first to detect the failure. At this point, *B* cannot reach anymore *A*, *C* and 50% of its paths towards *E* have failed. *C* cannot reach *B* anymore and half of its paths towards *D* have failed.

Router *B* will flood its updated link state packet through the entire network and all routers will recompute their forwarding table. Upon reception of a link state packet, routers usually first flood the received link-state packet and then recompute their forwarding table. Assume that *B* is the first to recompute its forwarding table, followed by *D*, *A*, *C* and finally *E*

1. After each update of a forwarding table, verify which pairs of routers are able to exchange packets. Provide your answer using a table similar to the one shown above.
2. Can you find an ordering of the updates of the forwarding tables that avoids all transient problems ?

7.1 Internet Protocol

1. For the following IPv4 subnets, indicate the smallest and the largest IPv4 address inside the subnet :

- 8.0.0.0/8
- 172.12.0.0/16
- 200.123.42.128/25
- 12.1.2.0/13

2. For the following IPv6 subnets, indicate the smallest and the largest IPv6 address inside the subnet :

- FE80::/64
- 2001:db8::/48
- 2001:6a8:3080::/48

3. netkit allows to easily perform experiments by using an emulated environment is is composed of virtual machines running User Model Linux. netkit allows to setup a small network in a lab and configure it as if you had access to several PCs interconnected by using cables and network equipments.

   A netkit lab is defined as a few configuration files and scripts :

   lab.conf is a textfile that defines the virtual machines and the network topology. A simple *lab.conf* file is shown below

   ```
   LAB_DESCRIPTION="a string describing the lab"
   LAB_VERSION=1.0
   LAB_AUTHOR="the author of the lab"
   LAB_EMAIL="email address of the author"

   h1[0]="lan"
   h2[0]="lan"
   ```

   This configuration file requests the creation of two virtual machines, named *h1* and *h2*. Each of these hosts has one network interface (*eth0*) that is connected to the local area network named *"lan"*. netkit allows to define several interfaces on a given host and attach them to different local area networks. A *host.startup* file for each host (*h1.startup* and *h2.startup* in the example above). This file is a shell script that is executed at the end of the boot of the virtual host. This is typically in this script that the network interfaces are configured and the daemons are launched. A directory for each host (*h1* and *h2* in the example above). This directory is used to store configuration files that must be copied on the virtual machine's filesystems when they are first created.

   netkit contains several scripts that can be used to run a lab. *lstart* allows to launch a lab and *lhalt* allows to halt the machines at the end of a lab. If you need to exchange files between the virtual machines and the Linux host on which netkit runs, note that the virtual hosts mount the directory that contains the running lab in */hostlab* and your home directory in */hosthome*.

   For this exercise, you will use a netkit lab containing 4 hosts and two routers. The configuration files are available `labs/lab-2routers.tar.gz`. The network topology of this lab is shown in the figure below.

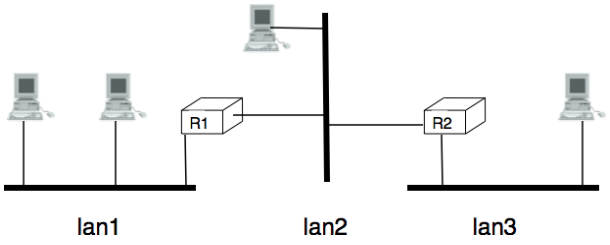


Figure 7.3: The two routers lab

   The *lab.conf* file for this lab is shown below

```
h1[0]="lan1"
h2[0]="lan1"
h3[0]="lan2"
router1[0]="lan1"
router1[1]="lan2"
router2[0]="lan2"
router2[1]="lan3"
h4[0]="lan3"
```

In this network, we will use subnet *172.12.1.0/24* for *lan1*, *172.12.2.0/24* for *lan2* and *172.12.3.0/24* for *lan3*.

On Linux, the IP addresses assigned on an interface can be configured by using *ifconfig(8)*. When *ifconfig(8)* is used without parameters, it lists all the existing interfaces of the host with their configuration. A sample *ifconfig(8)* output is shown below

```
host:~# ifconfig
eth0    Link encap:Ethernet  HWaddr FE:3A:59:CD:59:AD
        Inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::fc3a:59ff:fecd:59ad/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:3 errors:0 dropped:0 overruns:0 frame:0
        TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:216 (216.0 b)  TX bytes:258 (258.0 b)
        Interrupt:5

lo       Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

This host has two interfaces : the loopback interface (*lo* with IPv4 address *127.0.0.1* and IPv6 address *::1*) and the *eth0* interface. The *192.168.1.1/24* address and a link local IPv6 address (*fe80::fc3a:59ff:fecd:59ad/64*) have been assigned to interface *eth0*. The broadcast address is used in some particular cases, this is outside the scope of this exercise. *ifconfig(8)* also provides statistics such as the number of packets sent and received over this interface. Another important information that is provided by *ifconfig(8)* is the hardware address (HWaddr) used by the datalink layer of the interface. On the example above, the *eth0* interface uses the 48 bits *FE:3A:59:CD:59:AD* hardware address.

You can configure the IPv4 address assigned to an interface by specifying the address and the netmask

```
ifconfig eth0 192.168.1.2 netmask 255.255.255.128 up
```

You can also specify the prefix length

```
ifconfig eth0 192.168.1.2/25 up
```

In both cases, *ifconfig eth0* allows you to verify that the interface has been correctly configured

```
eth0      Link encap:Ethernet  HWaddr FE:3A:59:CD:59:AD
          inet addr:192.168.1.2  Bcast:192.168.1.127  Mask:255.255.255.128
          inet6 addr: fe80::fc3a:59ff:fecd:59ad/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:216 (216.0 b)  TX bytes:258 (258.0 b)
          Interrupt:5
```

Another important command on Linux is `route(8)` that allows to look at the contents of the routing table stored in the Linux kernel and change it. For example, *route -n* returns the contents of the IPv4 routing table. See `route(8)` for a detailed description on how you can configure routes by using this tool.

1. Use `ifconfig(8)` to configure the following IPv4 addresses :

- *172.16.1.11/24* on interface *eth0* on *h1*
- *172.16.1.12/24* on interface *eth0* on *h2*

2. Use *route -n* to look at the contents of the routing table on the two hosts.
3. Verify by using `ping(8)` that *h1* can reach *172.16.1.12*
4. Use `ifconfig(8)` to configure IPv4 address *172.16.1.1/24* on the *eth0* interface of *router1* and *172.16.2.1/24* on the *eth1* interface on this router.
5. Since hosts *h1* and *h2* are attached to a local area network that contains a single router, this router can act as a default router. Add a default route on *h1* and *h2* so that they can use *router1* as their default router to reach any remote IPv4 address. Verify by using `ping(8)` that *h1* can reach address *172.16.2.1*.
6. What do you need to configure on *router2*, *h3* and *h4* so that all hosts and routers can reach all hosts and routers in the emulated network ? Add the *ifconfig* and *route* commands in the *.startup* files of all the hosts so that the network is correctly configured when it is started by using *lstart*.

4. Use the network configured above to test how IP packets are fragmented. The *ifconfig* command allows you to specify the Maximum Transmission Unit (MTU), i.e. the largest size of the frames that are allowed on a given interface. The default MTU on the *eth?* interfaces is 1500 bytes.

1. Force an MTU of 500 bytes on the three interfaces attached to *lan2*.
2. Use *ping -s 1000* to send a 1000 bytes ping packet from *h3* to one of the routers attached to *lan2* and capture the packets on the other router by using `tcpdump(8)`. In which order does the emulated host sends the IP fragments ?
3. Use *ping -s 2000* to send a 2000 bytes ping packet from *h1* to *h4* and capture the packets on *lan2* and *lan3* by using `tcpdump(8)`. In which order does the emulated host sends the IP fragments ?
4. From your measurements, how does an emulated host generate the identifiers of the IP packets that it sends ?
5. Reset the MTU on the *eth1* interface of router *r1* at 1500 bytes, but leave the MTU on the *eth0* interface of router *r2* at 500 bytes. Check whether host *h1* can ping host *h4*. Use `tcpdump(8)` to analyse what is happening.
5. The Routing Information Protocol (RIP) is a distance vector protocol that is often used in small IP networks. There are various implementations of RIP. For this exercise, you will use quagga, an open-source implementation of several IP routing protocols that runs on Linux and other Unix compatible operating systems. `quagga(8)` is in fact a set of daemons that interact together and with the Linux kernel. For this exercise, you will use two of these daemons : `zebra(8)` and `ripd(8)`. `zebra(8)` is the master daemon that handles the interactions between the Linux kernel routing table and the routing protocols. `ripd(8)` is the implementation of the RIP protocol. It interacts with the Linux routing tables through the `zebra(8)` daemon.

   To use a Linux real or virtual machine as a router, you need to first configure the IP addresses of the interfaces of the machine. Once this configuration has been verified, you can configure the `zebra(8)` and `ripd(8)` daemons. The configuration files for these daemons reside in */etc/zebra*. The first configuration file is */etc/zebra/daemons*. It lists the daemons that are launched when zebra is started by */etc/init.d/zebra*. To enable `ripd(8)` and `zebra(8)`, this file will be configured as

```
# This file tells the zebra package
# which daemons to start.
# Entries are in the format: <daemon>=(yes|no|priority)
# where 'yes' is equivalent to infinitely low priority, and
# lower numbers mean higher priority. Read
```

```
# /usr/doc/zebra/README.Debian for details.
# Daemons are: bgpd zebra ospfd ospf6d ripd ripngd
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

The second configuration file is the */etc/zebra/zebra.conf* file. It defines the global configuration rules that apply to *zebra(8)*. For this exercise, we use the default configuration file, i.e.

```
! -*- zebra -*-
!
! zebra configuration file
!
hostname zebra
password zebra
enable password zebra
!
! Static default route sample.
!
!ip route 0.0.0.0/0 203.181.89.241
!
log file /var/log/zebra/zebra.log
```

In the zebra configuration file, lines beginning with *!* are comments. This configuration defines the hostname as *zebra* and two passwords. The default password (*password zebra*) is the one that must be given when connecting to the *zebra(8)* management console over a TCP connection. This management console can be use like a shell on a Unix host to specify commands to the *zebra(8)* daemons. The second one (*enable password zebra*) specifies the password to be provided before giving commands that change the configuration of the daemon. It is also possible to specify static routes in this configuration file, but we do not use this facility in this exercise. The last parameter that is specified is the log file where *zebra(8)* writes debugging information. Additional information about quagga are available from http://http://www.quagga.net/docs/docs-info.php

The most interesting configuration file for this exercise is the */etc/zebra/ripd.conf* file. It contains all the parameters that are specific to the operation of the RIP protocol. A sample *ripd(8)* configuration file is shown below

```
!
hostname ripd
password zebra
enable password zebra
!
router rip
 network 100.1.0.0/16
 redistribute connected
!
log file /var/log/zebra/ripd.log
```

This configuration files shown the two different ways to configure *ripd(8)*. The statement *router rip* indicates the beginning of the configuration for the RIP routing protocol. The indented lines that follow are part of the configuration of this protocol. The first line, *network 100.1.0.0/16* is used to enable RIP on the interface whose IP subnet matches *100.1.0.0/16*. The second line, *redistribute connected* indicates that all the subnetworks that are directly connected on the router should be advertised. When this configuration line is used, *ripd(8)* interacts with the Linux kernel routing table and advertises all the subnetworks that are directly connected on the router. If a new interface is enabled and configured on the router, its subnetwork prefix will be automatically advertised. Similarly, the subnetwork prefix will be automatically removed if the subnetwork interface is shutdown.

To experiment with RIP, you will use the emulated routers shown in the figure below. You can download the entire lab from `labs/lab-5routers-rip.tar.gz`

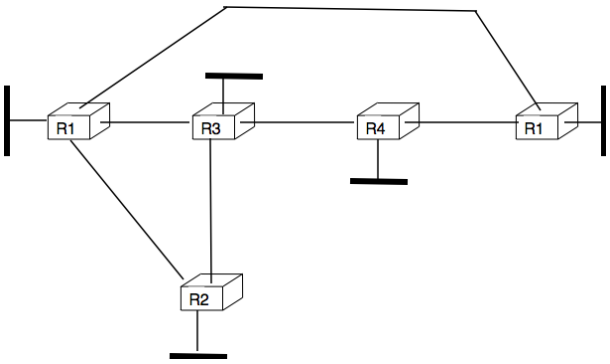


Figure 7.4: The five routers lab

The *lab.conf* describing the topology and the interfaces used on all hosts is shown below

```
r1[0]="A"
r1[1]="B"
r1[2]="F"
r1[3]="V"
r2[0]="A"
r2[1]="C"
r2[2]="W"
r3[0]="B"
r3[1]="C"
r3[2]="D"
r3[3]="X"
r4[0]="D"
r4[1]="E"
r4[2]="Y"
r5[0]="E"
r5[1]="F"
r5[2]="Z"
```

There are two types of subnetworks in this topology. The subnetworks from the *172.16.0.0/16* prefix are used on the links between routers while the subnetworks from the *192.168.0.0/16* prefix are used on the local area networks that are attached to a single router.

A router can be configured in two different ways : by specifying configuration files and by typing the commands directly on the router by using `telnet(1)`. The first four routers have been configured in the provided configuration files. Look at *r1.startup* and the configurations files in *r1/tmp/zebra* in the lab's directory for router *r1*. The *r?.startup* files contain the `ifconfig(8)` commands that are used to configure the interfaces of each virtual router. The configuration files located in *r?/tmp/zebra* are also copied automatically on the virtual router when it boots.

1. Launch the lab by using *lstart* and verify that router *r1* can reach *192.168.1.1*, *192.168.2.2*, *192.168.3.3* and *192.168.4.4*. You can also `traceroute(8)` to determine what is the route followed by your packets.
2. The `ripd(8)` daemon can also be configured by typing commands over a TCP connection. `ripd(8)` listens on port *2602*. On router *r1*, use *telnet 127.0.0.1 2602* to connect to the `ripd(8)` daemon. The default password is *zebra*. Once logged on the `ripd(8)` daemon, you reach the > prompt where you can query the status of the router. By typing *?* at the prompt, you will find the list of supported commands. The *show* command is particularly useful, type *show ?* to obtain the list of its sub options. For example, *show ip rip* will return the routing table that is maintained by the `ripd(8)` daemon.
3. Disable interface *eth3* on router *r1* by typing *ifconfig eth3 down* on this router. Verify the impact of this command on the routing tables of the other routers in the network. Re-enable this interface by typing *ifconfig eth3 up*.
4. Do the same with the *eth1* interface on router *r3*.

5. Edit the */etc/zebra/ripd.conf* configuration file on router *r5* so that this router becomes part of the network. Verify that *192.168.5.5* is reachable by all routers inside the network.

6. The Open Shortest Path First (OSPF) protocol is a link-state protocol that is often used in enterprise IP networks. OSPF is implemented in the *ospfd(8)* daemon that is part of quagga. We use the same topology as in the previous exercise. The netkit lab may be downloaded from `labs/lab-5routers-ospf.tar.gz`.

   The *ospfd(8)* daemon supports a more complex configuration that the *ripd(8)* daemon. A sample configuration is shown below

   ```
   !
   hostname ospfd
   password zebra
   enable password zebra
   !
   interface eth0
     ip ospf cost 1
   interface eth1
     ip ospf cost 1
   interface eth2
     ip ospf cost 1
   interface eth3
     ip ospf cost 1
   !
   router ospf
     router-id 192.168.1.1
     network 172.16.1.0/24 area 0.0.0.0
     network 172.16.2.0/24 area 0.0.0.0
     network 172.16.3.0/24 area 0.0.0.0
     network 192.168.1.0/24 area 0.0.0.0
     passive-interface eth3
   !
   log file /var/log/zebra/ospfd.log
   ```

   In this configuration file, the *ip ospf cost 1* specify a metric of *1* for each interface. The *ospfd(8)* configuration is composed of three parts. First, each router must have one identifier that is unique inside the network. Usually, this identifier is one of the IP addresses assigned to the router. Second, each subnetwork on the router is associated with an area. In this example, we only use the backbone area (i.e. *0.0.0.0*). The last command specifies that the OSPF Hello messages should not be sent over interface *eth3* although its subnetwork will be advertised by the router. Such a command is often used on interfaces that are attached to endhosts to ensure that no problem will occur if a student configures a software OSPF router on his laptop attached to this interface.

   The netkit lab contains already the configuration for routers *r1* - *r4*.

   The *ospfd(8)* daemon listens on TCP port *2604*. You can follow the evolution of the OSPF protocol by using the *show ip ospf ?* commands.

   1. Launch the lab by using *lstart* and verify that the *192.168.1.1*, *192.168.2.2*, *192.168.3.3* and *192.168.4.4* addresses are reachable from any router inside the network.
   2. Configure router *r5* by changing the */etc/zebra/ospfd.conf* file and restart the daemon. Verify that the *192.168.5.5* address is reachable from any router inside the network.
   3. How can you update the network configuration so that the packets sent by router *r1* to router *r5* use the direct link between the two routers while the packets sent by *r5* are forwarded via *r4* ?
   4. Disable interface *eth3* on router *r1* and see how quickly the network converges ? You can follow the evolution of the routing table on a router by typing *netstat -rnc*. Re-enable interface *eth3* on router *r1*.
   5. Change the MTU of *eth0* on router *r1* but leave it unchanged on interface *eth0* of router *r2*. What is the impact of this change ? Can you explain why ?

6. Disable interface *eth1* on router *r3* and see how quickly the network converges ? Re-enable this interface.
7. Halt router *r2* by using *vcrash r2*. How quickly does the network react to this failure ?

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *search*

INDEX

R