# Click Exercises

Johan Bergs — Jeremy Van den Eynde

1. Use the existing ping-5.click script and add the following features by changing the configuration of one element:
   (a) Only 20 pings are sent.
   (b) Pings are sent at a rate of 5 packets per second.
2. Use the existing ping-5.click script and add the following features by adding 1 element to the Router compound element:
   (a) Print the packet contents as they pass by.
   (b) Packets received from the network should have their IP time-to-live decreased.
   (c) Packets received from the network with an IP id larger than 10 should be dropped.
   (d) Collect information about the historical packet count and rate and access those data with a handler call.
3. Use the existing ping-5.click script and add the following features by adding several elements outside the Router compound element:
   (a) Echo requests are output at a rate of 5 packets per second and are bandwidth shaped to 2 packets per second.
   (b) Generate a random UDP encapsulated stream that flows over the same virtual network and is discarded at the receiver. Verify that the pings still work.
   (c) Expand the previous solution to give strict priority to ping packets: apply bandwidth shaping again and try to ensure that whenever sending a packet ping packets will be sent if they are available.
   (d) When 5 packets have been generated, the packet interval is set to 10ms.
4. Use the existing ping-5.click script and write new elements to support the following features:
   (a) Add the SimplePushElement (get it from the ZIP file).
   (b) Add your own Null element. So do nothing with packets, just pass them on. Make a push, pull and agnostic version.
   (c) Complete the previous element by making sure users can only specify an empty configuration string when using this element in Click scripts.

(d) Configure the ICMPPingSource element to send the ICMP packets from an incorrect *source* address, e.g. 1.2.3.4. Test the script: replies should not be received. Fix the situation by creating an element that will correct the source address. Mind the IP header checksums, verify with Wireshark!

(e) Configure the ICMPSource element to send the ICMP packets to an incorrect *destination* address. Create an element that will correct the IP destination address. Mind the IP header checksums! Hint: look at SetIpAddress, what does it do?

(f) Write a queue element from scratch. The queue outputs packets on a second port when it overflows, the capacity of the queue must be configurable. Test with a simple script.

(g) Use the existing ICMP generators to generate a PCAP dump file with some Echo Requests and their replies in it. Write your own ICMP Echo Request generator based on that dump, both pull based and timer based. Do not start from existing ICMP generator element. Analyze your generated packets with Wireshark. Mind the checksums!

(h) Simulate your own *traceroute*: generate a random UDP encapsulated stream and set a small IP time-to-live e.g. 10. Forward that packet through a number of compound elements, which decrease the TTL and send back an ICMP TTL exceeded packet if the TTL is 0. For the ICMP packet format do a *traceroute* and study the output in Wireshark.

5. Study the Click IP router in the Click thesis chapters 5.1 and 5.2.