CSCD58 Computer Networks - Winter 2016
Programming Assignment 2
IP Forwarding

Due: March 24th, 2016, 5:00PM

In this assignment, you will implement ***VRouter***, a virtual router for basic IPv4 forwarding operations.

Section 1. Specifications

See Section 3 for system I/O files and their formats mentioned in this section.

On start-up, *VRouter* loads its forwarding table and the list of its interfaces from *ForwardingTable.txt* and *Interfaces.txt* respectively. By assumption of this assignment, the set of interfaces and the forwarding table of *VRouter* don't change throughout the execution of *VRouter*. So, load the contents of these two files in a static block in class *VRouter.java*.

It is up to you how to structure and store the entries of *ForwardingTable.txt* and *Interfaces.txt*. You will also decide on a type for *IPaddress* to represent an IP address for its use in the given method signatures (Section 3) and throughout the rest of the code. A suggested implementation for interfaces and forwarding table is as flows:

Interfaces: a HashMap, keys are masked-IP-prefixes, values are MTUs

ForwardingTable: a HashMap, keys are masked-IP-prefixes, values are interface keys.

After start-up, *VRouter* reads *InPackets.txt by* invoking *incomingPackets("InPackets.txt")*. For each IP packet *ip4packet* in *InPackets.txt*, *VRouter* does the following. Note that, *VRouter* stops processing a packet whenever it calls *dropPacket*() for that packet:

1) Runs the checksum (invokes *checksum(ip4packet)*). If the computed checksum doesn't match the checksum on the packet, invokes *dropPacket(sourceAddress, destAddress, ID, "Checksum error").*
2) Else (if the checksum didn't fail), checks the destination address. If it is one of its interfaces or is found in the forwarding table by the longest prefix match, then performs the forwarding operations (Steps A and B below), otherwise invokes *dropPacket(sourceAddress, destAddress, ID, "Destination not found").*

The forwarding operations are as follows:

A. If the destination address is one of its interfaces, then logs to *Messages.txt* "Packet from "+*sourceAddress* +" destined for this router successfully received: "+*ID* and stops processing the packet. *sourceAddress* and *ID* are the corresponding field values of the packets IP header.
B. If the destination address is not one its interfaces,
   a. Decrements time-to-live (TTL). If TTL is less than 0, then invokes *dropPacket(sourceAddress, destAddress, ID, "TTL exceeded.")*
   b. Checks the outgoing link bandwidth for MTU (maximum transmission unit.) If link MTU is less than the packet size, fragments the packet by invoking *fragment(ip4packet, MTU).*
   c. Prepares the packet header (or headers for each frame if fragmented in step (b) above) accordingly and transmits the packet(s) to the outgoing link by invoking *forward(packets, interface)* for each**.**

Assume the following:

- *VRouter* has infinite queue size – there is no queuing delay.
- There are no optional fields in the IP packet headers. Each IP header in *InPackets.txt* is 20 bytes in length.
- No class-D (multicast) or class-E addresses in the source or destination address of any packet in *InPackets.txt*.

Section 2. Components

All your code should be contained in file ***VRouter.java***. You can add other classes, methods and fields to your code as long as they comply with the given specifications. However, you are not allowed to add a public class in addition to *VRouter*.

IP4Packet:

Define a class *IP4Packet* to represent an IPv4 packet header to be processed by your methods. Define field members for each IPv4 header field, according to the fields given in *IPHeader.pdf*. When required, add getters and setters to these fields. You will be using these getter and setter methods for i.) looking up the header fields and ii.) preparing the header of a packet to be forwarded when necessary.

For each packet read from *InPackets.txt*, create an instance of *IP4Packet* (see below the method *incomingPackets()*) and make *VRouter* process on this instance for its operations.

You can make *IP4Packet* an inner class of *VRouter* or a class within the same package as *VRouter*. Recall that all your code should be contained in file *VRouter.java*.

Methods:

VRouter should include the following methods given by their signatures:

static List<IP4Packet> incomingPackets(String fileName)

Reads packets from file *InPackets.txt* passed in as the parameter and puts them into an ordered list for processing. Although parameterized in this method, your code will be tested to read from file *InPackets.txt*.

A suggested implementation of this method is, for each packet: i.) read all header values from *InPackets.txt*, ii.) invoke a constructor of *IP4Packet* to create an instance on all these fields.

static String checksum(IP4Packet ip4packet);

Runs checksum on *ip4packet* and returns the result in format *a-b-c-d* where each of a through d are 4-bit sequences. (See also *Checksum* in *IPHeader.pdf* for this format.) Note: By IPv4 specifications (RFC-791), run the checksum on *ip4packet* as is, except the "checksum" field entry is all 0. So, make sure to reset the checksum field of the input packet to 0 before computing the checksum on the packet header. Also recall: by IP specifications (RFC791), checksum is run only on the packet header and not on payload.

static List<IP4Packet> fragment(IP4Packet ip4packet, int MTU);

Assumes *ip4Packet* is never null, and MTU is always smaller than the size of *ip4packet* (*Total-Length* entry in the header)

If MTU is at least as great as the size of *ip4packet* (*Total-Length* entry in the header), returns *ip4packet* as is since no fragmentation is needed. Otherwise, splits *ip4packet* into fragments of size at most MTU, prepares the packet headers accordingly and returns the fragments in their order of offset in the following steps:

- Checks the DF flag in the IP header. If it is 1, then invokes *dropPacket(sourceAddress, destAddress, ID, "Fragmentation needed and DF set.")* and returns an empty list. Alternatively, you can make your calling method to invoke *dropPacket(sourceAddress, destAddress, ID, "Fragmentation needed and DF set.")* on an empty list returned by this method.
- Splits the data load of the packet into byte sequences so that each fragment, except probably the last fragment has a payload of $8*k$ bytes for maximum integer *k* satisfying $8*k+20 \leq$ MTU of the outgoing link. See also *Fragmentation Offset* and *Flags* in Section *IPHeader.pdf*.
- Sets the *Offset* and *MF* fields of each fragment accordingly.
- Returns the fragments in their order which is the increasing order of *Offset* field of the fragments.

static boolean dropPacket(IPaddress sourceAddress, IPaddress destAddress, int ID, String message)

Writes the following String to *messages.txt* and returns *true*:

“Packet ”+*ID*+” from ”+ *sourceAddress* +” to ”+ *destAddress*+”: ”+*message*

Returns *false* on *IOException*.

Note: The invocation of this method also specifies the end of *VRouter*’s execution of a packet. Whenever *dropPacket()* is mentioned in the specifications to be invoked on a packet, that packet isn’t processed any further by *VRouter*.

static boolean forward(IP4Packet ip4packet, IPaddress interface);

Assumes all fields accept *Checksum* in *ip4packet* are set correctly.

Invokes *checksum(ip4packet)* on the packet passed in and writes the value returned to the *Checksum* field of *ip4packet*.

Than, writes *ip4packet* to *OutPackets.txt* and returns *true*. Returns *false* on *IOException*.

Following is the output format for each packet written to *OutPackets.txt:*

Lines 1-5 lines: the same format as in *InPackets.txt*. (See Section IPHeader)

Line 6: The interface the packet is transmitted on. Is of the form *w.x.y.z* in decimal.

There should be a blank line between consecutive packet entries.

static boolean lookupInterfaces(IPaddress ipAddress);

Looks up the *IPaddress* passed in, returns true if it is one of *VRouter*’s interfaces, false otherwise.

static IPaddress lookupDest(IPaddress ipAddress);

Looks up the *IPaddress* passed in the forwarding table by longest prefix match. If finds a match, then returns the matching interface. Otherwise, returns *null*. Note: The forwarding table entries are in arbitrary order. Do not assume that the entries in *ForwardingTable.txt* are in any specific order for longest prefix match.

Section 3. Input and Output

All I/O files will be read from and written to the default directory.

Input**:** *VRouter* reads the following 3 files for its overall input.

- *ForwardingTable.txt*: The forwarding table of *VRouter*. See ** ***Interfaces*** and ***Forwarding Table*** for format.
- *Interfaces.txt*: The list of *VRouter*'s interfaces. See ***Interfaces*** and ***Forwarding Table*** for format.
- *InPackets.txt*: A sequence of IPv.4 packets. ***See IP Header*** for format.

Output**:** *VRouter* has 2 types of output. The formats of these 2 files are explained in Specifications and Components Sections (Sections 1 & 2) along with their use:

- *OutPackets.txt*: Packets forwarded to other routers, and
- *Messages.txt*: ICMP and other messages logged to a local file.

Note that in this assignment, we simulate ICMP error messages by printing them to our log file, *Messages.txt*. *VRouter* doesn't generate and send any ICMP messages.

Section 3.1 *Interfaces.txt* format:

Each line represents an interface of *VRouter*. Each line contains a triple (*IPaddress*, *subnetMask, MTU*) in format

IPaddress; *mask*; *interface*

where

- *IPaddress* is an IP address of the form *w.x.y.z* where *w, x, y, z* are in decimal format,
- *subnetMask* is the subnet mask for *IPaddress,* is the number of LHS bits to represent the subnetID in the 32bit address given in *IPaddress*. *IPaddress* and *subnetMask* uniquely define one of the interfaces of *VRouter,*
- *MTU* is a decimal integer representing the maximum transmission unit (in number of bytes) for this interface.

Section 3.2 *ForwardingTable.txt* format:

Each line contains a 4-tuple (*IPaddress, subnetMask, interfaceIPaddress, interfaceMask*) in format

IPaddress; *mask*; *interfaceIPaddress; interfaceMask*

where

- *IPaddress* is a destination address, is of the form *w.x.y.z* where *w, x, y, z* are in decimal format,
- *subnetMask* is the network mask for *IPaddress*. is the number of bits that represent the subnetID in the 32bit address given in *IPaddress*,
- *interfaceIPaddress, interfaceMask* are the address and prefix of the *VRouter* interface to transmit a packet towards the destination (*IPaddress, subnetMask).* Is one of the addresses loaded from *Interfaces.txt*

Assume the following on *ForwardingTable.txt* and *Interfaces.txt*:

- Value of *subnetMask* is in range 8-30 inclusive.
- MTU is in range 68bytes-64Kbytes inclusive.
- Each entry of *IPaddress* in *Interface.txt* and *ForwardingTable.txt* is a valid IP address.
- For each entry (*IPaddress, subnetMask, interfaceIPaddress, interfaceMask*) in *ForwardingTable.txt*, there always is a matching (*interfaceIPaddress, interfaceMask*) entry in *Interfaces.txt*.
- The set of *IPprefixes* in interfaces and in forwarding table are exclusive, i.e., an (IPaddress, subnetMask) pair appearing in one of *Interfaces.txt* and *ForwardingTable.txt* never appears in the other.

Section 3.3 *InPackets.txt* format:

Each packet entry is 5 lines, with contents described in text *IPHeader* (See the file *IPHeader.pdf* in assignment pack), in the following format. There is a blank line between consecutive packet entries:

Line 1: *Version; IHL; ToS; Total-Length*
Line 2: *Identification; Flags; FragmentOffset*
Line 3: *TTL; Protocol; Checksum*
Line 4: *sourceAddress*
Line 5: *destinationAddress*

Following are sample entries of *InPackets.txt*. Note: the checksum is accurate in all these packets:

i.) Packet 1:
4; 5; 0; 576
4166; 010; 0
64; 6; 1001-0011-1110-0111
190.16.10.1
171.22.33.99

ii.) Packet 2:
4; 5; 0; 1500
2172; 010; 0
30; 17; 1001-1110-0101-1111
234.54.0.0
182.250.15.5

iii.) Packet 3:
4. 5. 0. 1500
52167; 001; 4
128; 17; 1011-1101-1000-1000
234.72.11.0
150.111.0.5

iv.) Packet 4:
4. 5. 0. 1450
52167; 000; 7
128; 17; 1101 1101 1011 0111
234.72.11.0
150.111.0.5

References

In addition to the material covered in class and in your textbook, refer to the following for further details:

- Fragmentation: RFC791, Section 3.2
- Checksum: RFC1071.
- Packet header entries: RFC791.

Submission Instructions

- Submit on BlackBoard only the file *VRouter.java*.
- The submission deadline is March 24th, 2016, **5:00PM sharp**.
- Write the full name and student ID of both team members as a comment at the top of *VRouter.java* you submit.
- BlackBoard is setup to allow up to 3 submissions. Among multiple submissions—if any, the one with the latest timestamp within deadline will be marked.
- All code must be written in Java 1.7.
- Make sure your code compiles correctly. Classes with compile time errors won't be marked.
- Ensure the quality of your code. In addition to the specifications given here, your code should be readable and well written by Java standards.