Spring 2014
Computer Networks
CMPE323

Laboratory Experiment No. 5: Introduction to User Datagram Protocol (UDP)

Aims and Objectives:

- Introduce students to UDP, a common layer 4 protocol.
- Allow students to manually craft UDP datagrams (including lower layer protocols that they have learned in past labs).
- Practically verify the correct behavior according to the UDP standard (IETF 768).

Materials Required:

- IP routers,
- Ethernet switches,
- PCs with Ethernet adapters,
- and straight-through/crossover/rollover cables.

Change Log:

- 25-3-2014: original document – mkhonji.
- 26-3-2014: fixing language typos – mkhonji.

1 Introduction

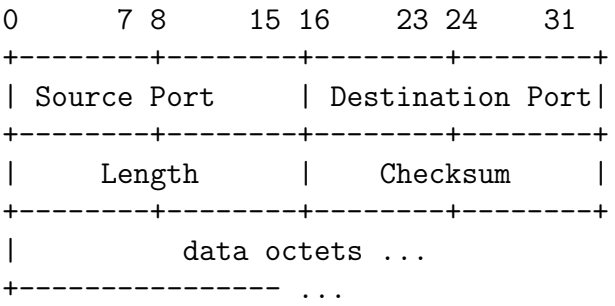

Figure 1: UDP Header Format — Source: RFC768.

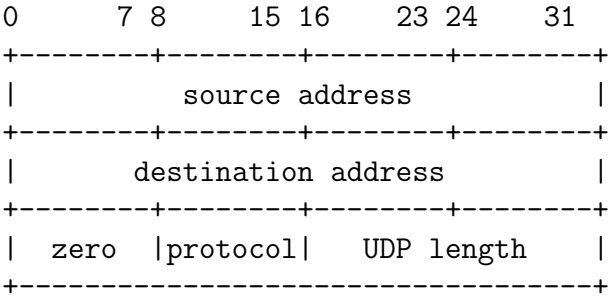

Figure 2: Pseudo IP header used in UDP checksum calculation — Source: RFC768.

In the previous lab, we discussed how the Internet Protocol (IP) and Classless Inter-Domain Routing (CIDR) allow end-to-end reachability across multiple layer 2 networks.

However, what we did not discuss so far is: how can applications know which messages are for them?

For example, a node in a network (e.g. a computer running Linux or Windows) can have multiple applications running on it (e.g. Firefox, IE, IRC clients, ...). When such node receives network datagrams, how can the node know which datagram is for which application? E.g. should the datagram go to (say) Firefox (a HTTP client), irssi (an IRC client), Thunderbird (a mail client), ...?

In this lab, we discuss a simple protocol that does exactly that: allowing sender and receiver nodes determine which application is for which datagram.

The mechanism that is used is quite simple: for every datagram that you send, you add a new header (called the UDP header) which primarily includes to fields:

- Destination port number: a number that destination application is expected to be listening on.
- Source port number: a number that source application is listening on. This is used by the source application to tell the remove application which source the source application is listening on. However, in scenarios were a source application does not expect a reply, this field is set to 0.

Figure 1 presents the structure of the UDP header, which has 16 bit source port number, 16 bit destination port number, 16 bit length (UDP header + UDP payload), and 16 bit checksum (one's complement of one's complement sum of 16 bit words of the pseudo IP header + UDP header + UDP payload + 0 pads if needed). Figure 2 shows the structure of the pseudo IP header. Note: when the checksum in the UDP header is set to 0, it is considered as disabled (RFC768).

For example, consider the two network nodes (PCs) A and B as follows:

- A has the IP address IP_A and runs the applications a_1, a_2 and a_3 which listen on UDP ports p_1, p_2 and p_3 respectively.
- B has the IP address IP_B and runs the applications a_1, a_2 and a_3 which listen on UDP ports p_1, p_2 and p_3 respectively.

If the application a_1 on node A wishes to send a message to application a_1 on node B, then A has to send an IP packet with the destination IP address IP_B and the destination UDP port p_1

2 Lab Preparation

1. Connect a PC to the routers console port using a rollover cable[1].
2. Erase the configuration of the routers[2].
3. Connect a PC to the switches console ports using rollover cables.
4. Erase the configuration of the switch[3].
5. Run Wireshark on all involved lab PCs as depicted in Figure 3.
6. Physically connect the lab as depicted in Figure 3.
7. Configure the interfaces:
   - Configure[4] `R1`'s interfaces as follows:
     - GigabitEthernet 0/0: IPv4 address 10.0.12.1, subnet mask 255.255.255.0.
     - GigabitEthernet 0/1: IPv4 address 10.0.1.1, subnet mask 255.255.255.0.
   - Configure `R2`'s interfaces as follows:
     - GigabitEthernet 0/0: IPv4 address 10.0.12.2, subnet mask 255.255.255.0.
     - GigabitEthernet 0/1: IPv4 address 10.0.2.1, subnet mask 255.255.255.0.

[1] If using Linux: `screen /dev/ttySx` were `x` is the serial interfaces ID that is connected to the console cable. If using Windows: Use Hyperterminal or Putty to connect to `COMx` ports.

[2] `enable`, `erase startup-config`, `reload`, and make sure to answer `no` to all yes/no questions while hitting *enter* for all `confirm` prompts.

[3] `enable`, `erase startup-config`, `delete vlan.dat`, `reload`, and answer `no` to all yes/no questions while hitting *enter* for all `confirm` prompts.

[4] `enable`, `configure terminal`, `interface Gi0/0`, `ip address 10.0.0.1 255.255.255.0`.

- Configure[5] `PC1`'s interfaces as follows:
  - eth0: IPv4 address 10.0.1.2, subnet mask 255.255.255.0.
- Configure `PC2`'s interfaces as follows:
  - eth0: IPv4 address 10.0.2.2, subnet mask 255.255.255.0.

8. Then add the IP routing tables as follows:
   - For R1: `ip route 10.0.2.0 255.255.255.0 10.0.12.2`
   - For R2: `ip route 10.0.1.0 255.255.255.0 10.0.12.1`
   - For PC1: `route add -net 0.0.0.0/0 gw 10.0.1.1`
   - For PC2: `route add -net 0.0.0.0/0 gw 10.0.2.1`
9. Test connectivity using the `ping` command to send ICMP Echo messages to PCs in other networks and receive the respective ICMP Echo-Reply messages back.

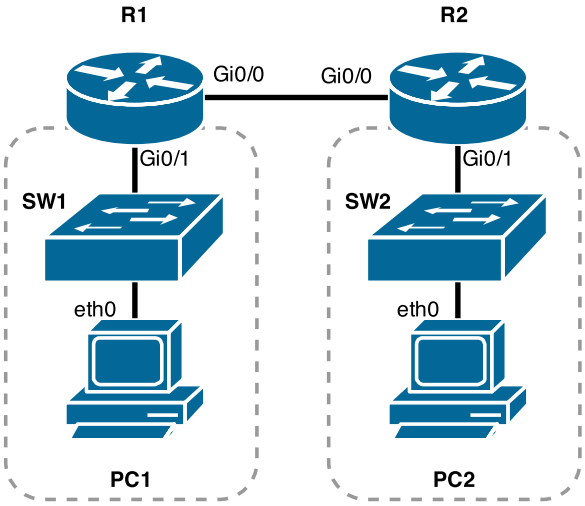

Figure 3: Physical lab topology.

3 Lab experiments

[100 points]

Note: When done, show your work to the lab engineer for grading purposes.

Using the `nc`[6] command run three applications on PC1 that listen on UDP ports as follows:

- Application 1: listen on IP 10.0.1.2 and UDP port 100.
- Application 2: listen on IP 10.0.1.2 and UDP port 200.

[5] `ifconfig eth0 10.0.1.2/24`
[6] `nc -l -p 100 -s 10.0.1.2`

- Application 3: listen on IP 10.0.1.2 and UDP port 300

Then, using the provided tools (macframesender.c, or scapy) send the following UDP messages:

- Message 1: "Hello" from PC2 to PC1's `nc` that is listening on port 100.
- Message 2: "Hello" from PC2 to PC1's `nc` that is listening on port 200.
- Message 3: "Hello" from PC2 to PC1's `nc` that is listening on port 300.

NOTE: the tool `nc` will not accept subsequent connections from *other* source IP addresses and source port numbers after it is used by some source IP and source port. Note that this is only a limitation in `nc` and it is easy to code UDP servers that accept connections from multiple clients.