Lab: TCP Performance, ICMPv6 Echo and Traceroute

In this lab you will use Wireshark to explore pre-collected packet traces and answer three questions:

1. How do the TCP receive window size and optional extensions affect download throughput?
2. When ICMPv6 Echo Requests to several destinations are in flight simultaneously, how does the sender match each reply to its request?
3. When several `traceroute6` probes run in parallel, how does each Time Exceeded reply get matched to the probe that triggered it?

Pre-collected traces are provided for each section. If you wish to collect your own traces, instructions are given at the end of each section.

Prerequisites

- Wireshark (download from https://www.wireshark.org if not already installed)

TCP Window Size and Performance

The TCP receive window limits how much data the sender may have unacknowledged at any moment. Without the Window Scale option (RFC 7323 Section 2) the window field is 16 bits, capping the window at 65 535 bytes regardless of buffer size. Three sysctl knobs control which TCP options the kernel negotiates:

- `net.ipv4.tcp_window_scaling` — enables the Window Scale option
- `net.ipv4.tcp_timestamps` — enables the Timestamps option
- `net.ipv4.tcp_sack` — enables the Selective Acknowledgement option

The following eight traces capture a 1 MB download under different combinations of receive buffer size and sysctl settings:

Experiment matrix

Trace	Window	SACK	Timestamps	Wscale
trace-001.pcap	16 KB	off	off	off
trace-002.pcap	64 KB	off	off	off
trace-003.pcap	16 KB	off	on	off
trace-004.pcap	64 KB	off	on	off
trace-005.pcap	16 KB	off	off	on
trace-006.pcap	64 KB	off	off	on
trace-007.pcap	256 KB	off	off	on
trace-008.pcap	256 KB	on	on	on

The *experiments.log* file records the elapsed download time for each experiment.

Analyse the traces in Wireshark

Open the traces in Wireshark and use the filters below to explore the negotiated options and the window behaviour.

Useful Wireshark display filters

Filter	Selects

`tcp.flags.syn == 1`	SYN and SYN-ACK segments (handshake)
`tcp.options.wscale`	Segments carrying the Window Scale option
`tcp.options.timestamp`	Segments carrying the Timestamps option
`tcp.options.sack_perm`	Segments carrying the SACK-Permitted option
`tcp.window_size_value`	Raw (unscaled) window field value
`tcp.window_size`	Scaled window size as computed by Wireshark

Questions

- Open the SYN and SYN-ACK for each trace. Which TCP options are present in the handshake? How does this differ across the eight traces?
- Look at the Window Scale option when it is present. What shift count does the client advertise? What is the maximum window size that this allows? Read RFC 7323 Section 2.
- Compare the `tcp.window_size` values during the data transfer for traces 001 and 007. What is the largest window you observe in each?
- Read `experiments.log` and compare the elapsed times. Which download is fastest? Which is slowest? Can you relate the elapsed time to the maximum window size that was in effect?
- The Timestamps option adds 10 bytes to every segment. Based on what you observed, does enabling timestamps alone (traces 003–004 vs 001–002) have a measurable effect on throughput? Read RFC 7323 Section 3 to understand what the option provides.

> **Note**
>
> To collect your own traces, run (as root):
>
> ```
> sudo bash run-experiments.sh
> ```
>
> The script requires `tcpdump` and modifies sysctl settings temporarily; original values are restored automatically on exit.

Parallel ICMPv6 Echo

`ping6` sends ICMPv6 Echo Request messages and waits for Echo Reply messages. When several destinations are pinged simultaneously the replies from different destinations arrive interleaved.

The pre-collected traces below were produced by pinging three RIPE Atlas anchors in parallel for five seconds:

- *ping-parallel.pcap* — packet capture
- *ping output — Sydney : ping-au-syd-as1221-anchors-atlas-ripe-net.txt*
- *ping output — Porto Alegre : ping-br-poa-as19200-anchors-atlas-ripe-net.txt*
- *ping output — Hong Kong : ping-hn-sap-as64150-anchors-atlas-ripe-net.txt*

Analyse the capture in Wireshark

Open `ping-parallel.pcap` in Wireshark and apply the following filters:

Useful Wireshark display filters

Filter	Selects
`icmpv6.type == 128`	ICMPv6 Echo Request
`icmpv6.type == 129`	ICMPv6 Echo Reply
`icmpv6.echo.identifier`	Identifier field (column or filter value)
`icmpv6.echo.sequence_number`	Sequence number field

Questions

- Look at the ICMPv6 Echo Request format in RFC 4443 Section 4.1 and the Echo Reply format in Section 4.2. Which fields appear in both the request and the reply?
- When several `ping6` processes run in parallel, each targeting a different destination, how can the sender determine which Echo Reply corresponds to which Echo Request? Examine the identifier and sequence number fields for requests sent to different destinations.
- Within a single `ping6` session targeting one destination, how are successive probes distinguished from each other?

> **Note**
>
> To collect your own traces, run (as root):
>
> ```
> sudo bash ping-parallel.sh
> ```
>
> You can pass alternative destinations as arguments:
>
> ```
> sudo bash ping-parallel.sh 2001:db8::1 2001:db8::2
> ```

Parallel Traceroute6

`traceroute6 -n` discovers the sequence of routers between the source and a destination by exploiting the IPv6 Hop Limit field. Each probe is sent with a small Hop Limit; when the limit reaches zero a router returns an ICMPv6 Time Exceeded message, revealing its address.

The pre-collected traces below were produced by running `traceroute6 -n` to three RIPE Atlas anchors in parallel:

- *traceroute-parallel.pcap* — packet capture
- *traceroute output — Sydney : traceroute-au-syd-as1221.anchors.atlas.ripe.net.txt*
- *traceroute output — Porto Alegre : traceroute-br-poa-as19200.anchors.atlas.ripe.net.txt*
- *traceroute output — Hong Kong : traceroute-hn-sap-as64150.anchors.atlas.ripe.net.txt*

Analyse the capture in Wireshark

Open `traceroute-parallel.pcap` in Wireshark and apply the following filters:

Useful Wireshark display filters

Filter	Selects
`udp`	Outgoing traceroute probes (UDP by default)
`icmpv6.type == 3`	ICMPv6 Time Exceeded (router replies)
`icmpv6`	All ICMPv6 messages

Questions

- `traceroute6` sends UDP probes by default. To which destination port does the first probe go? How does the port change with each successive probe?
- Read RFC 4443 Section 3.3 (Time Exceeded Message). What portion of the original packet is included in the Time Exceeded reply? Which fields of that embedded portion can `traceroute6` use to match a reply to the probe that caused it?
- When several `traceroute6` processes run in parallel to different destinations, how are their probes and replies kept separate? Examine the embedded packets inside the Time Exceeded messages for two different destinations.
- Look at the Hop Limit field of successive probes to the same destination. How does it change, and why? How does this mechanism allow `traceroute6` to discover each intermediate router one at a time?

Note

To collect your own traces, run (as root):

```
sudo bash traceroute-parallel.sh
```

You can pass alternative destinations as arguments:

```
sudo bash traceroute-parallel.sh 2001:db8::1 2001:db8::2
```