Phase 2 Report

Bartu Akyürek 2594109 - https://github.com/bartuakyurek/middlebox/

1. UDP Checksum Covert Channel

In this project, checksum fields of UDP is used as a covert channel to embed covert data. In [1], Fisk et al. mentions UDP checksum can be used as a flag, resulting in a covert channel with 1 bit/packet bandwidth. Following this notion, the covert channel is designed to use the existence of checksum in UDP datagrams, i.e. if there is a non-zero checksum it indicates a covert bit 1, otherwise 0.

To inform the receiver total number of covert bits to be sent, the first `N=8` bits are used. At each receive, the receiver stores the covert bit based on the existence of checksum. Once the receiver stops, it checks the first `N` bits of the saved bitstream to exactly know the length of the covert flags and extract the covert message from that stream.

Since the bandwith of this channel design is only 1 bit per packet, many number of packets needs to be sent. In the implementation a long carrier message is created, where the sender fragments the message into smaller payloads and sends them to receiver until all covert bits are sent.In order to ensure reliability of UDP to some extent, an ACK mechanism is included in the covert channel design. The sender appends a sequence number in the payload of the packets, with a `[seq_number]` before sending the fragmented carrier packets. Every time the receiver receives a packet, it extracts this sequence number and sends this sequence number back to sender as an ACK.

Sliding windows are also utilized to send multiple packets and asynchronously receive ACKs to avoid one ACK to block other packets to be sent. Once the smallest sequence number's ACK is received, the window is sled forward to send remaining fragments. Additionally, if an ACK cannot reach to the sender within `timeout` then the same sequence numbered packet is sent once again until it reaches to `max_trans` number of transmissions. If no ACK from that sequence number received after sending the same packet `max_trans` times, then the window is again sled to continue with the remaining packets. Note that even if no ACK is received within this time it might be the case the receiver receives the packet eventually.

Covert channel is parametrized by the following variables:

- **Time-out** The amount of seconds to wait for ACK before retransmitting the packet. `--timeout`
- **Maximum transmissions** Maximum allowed number of transmissions of the same packet `--trans`
- **Window size** Number of packets to send in a batch `--window`

2. Tests

Capacity of the covert channel is measured for the channel parameters in Figures 1, 2, and 3 where the legends depict the default parameters used in the experiment campaign with 95% confidence intervals. At this stage, the covert channel achieves maximum bandwith in Figure 2 and 3 for different values. The main capacity drop occurs when the allowed timeout is short, as in Figure 1, which is because sender assumes the packet is lost after maximum retransmissions are achieved. When the timeout is short, the packet is retransmitted multiple times before an ACK is received, such that it is assumed to be lost even if ACK might arrive later on.

Note that the processor from the previous phase is adding random delays from a uniform distribution with mean `--delay=1e-2` seconds before publishing the subscribed packets back. A disadvantage of the current sender is that once a timeout occurs, it leads to timeouts in the following packets as well, which should be handled as an improvement, especially when the processor in the next phase can disturb the timings of the packets. Another improvement is needed for robust transmission, e.g. if the processor changes some of the UDP checksum the covert bits would be corrupted.

The following texts are conducted to send 32 bits of covert data.

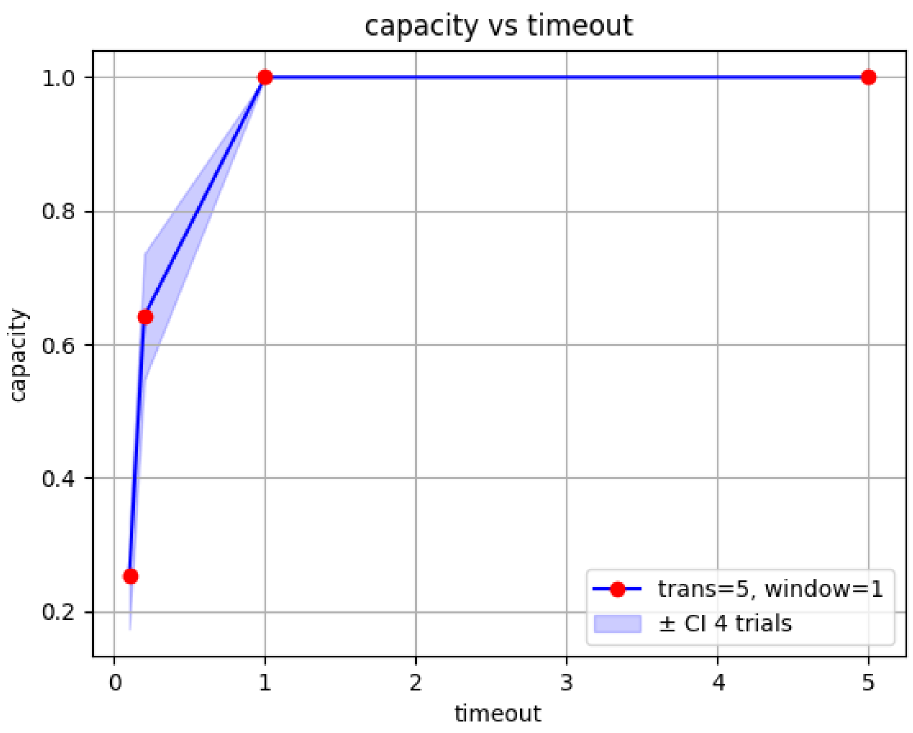

Figure 1. Capacity (bits/packet) vs. timeout (seconds)

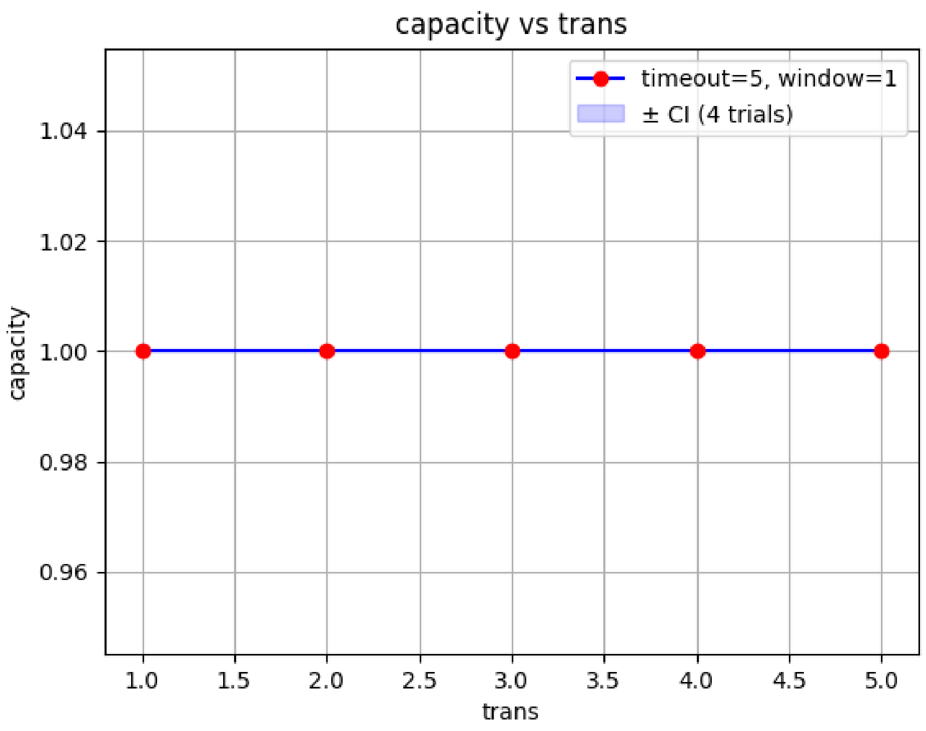

Figure 2. Capacity (bits/packet) vs. maximum transmission (packets)

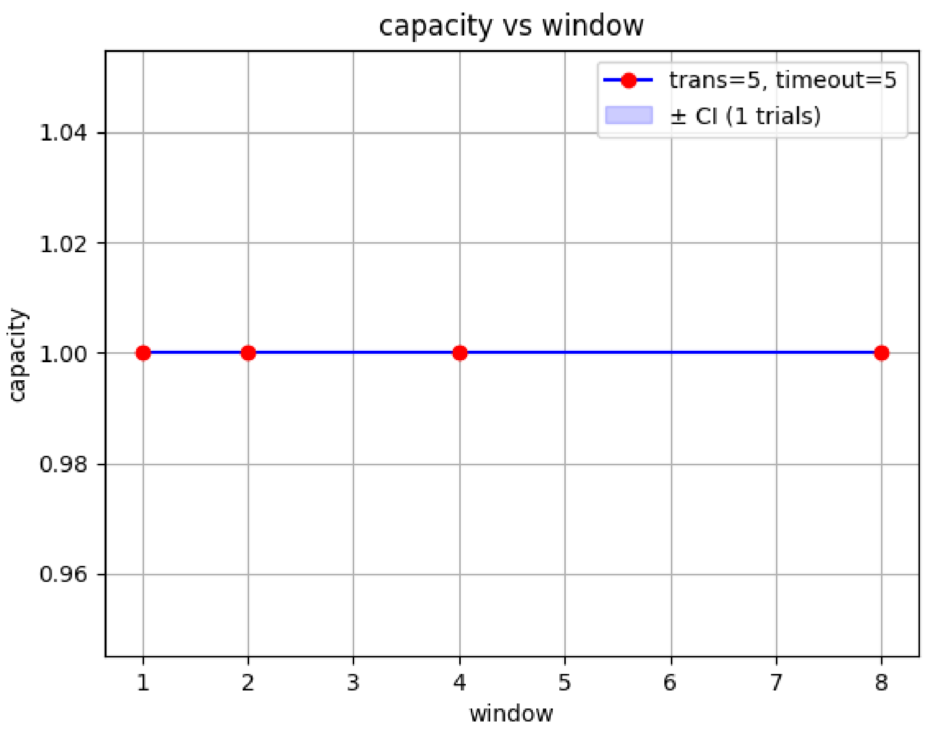

Figure 3. Capacity (bits/packet) vs. window size (packets)

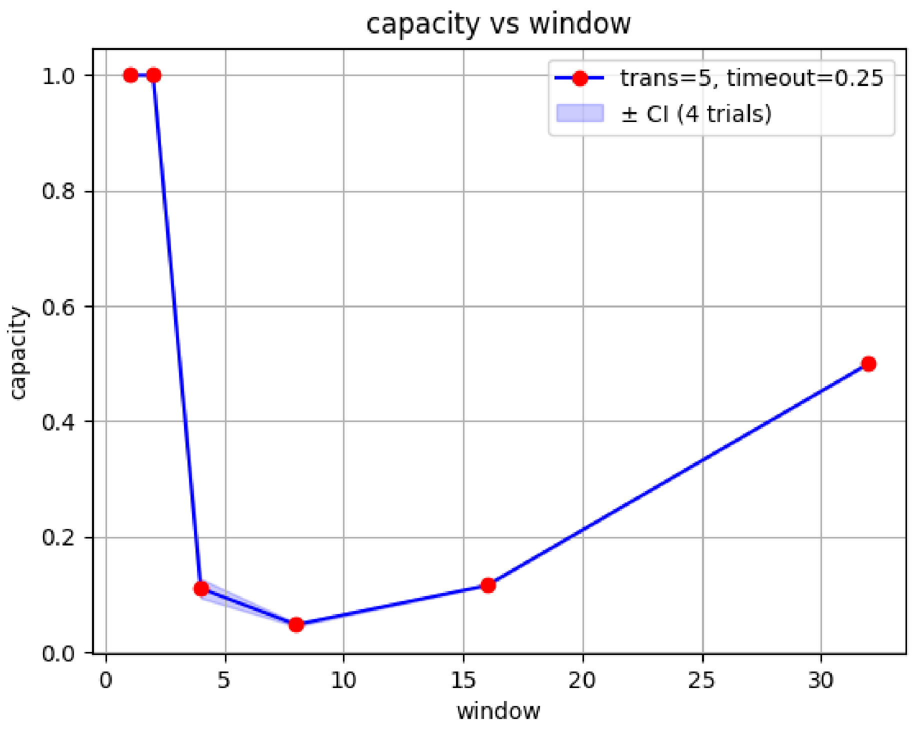

Figure 4. Capacity (bits/packet) vs. window size (packets) in an edge case.

An edge case is tested in Figure 4, where timeout parameter is fixed at 0.25 seconds, together with 0.01 seconds mean delay added in the processor (i.e. 0.02 mean delay added on ACK to be received) the transmission rate suddenly drops and eventually restores for larger window sizes. For a single window size, the packet is successfully sent under 50 ms, resulting in full capacity (1 bit per packet); however as we increase the window size, the packet sending thread locks in longer than the timeout, resulting retransmissions of the packet; hence, it drops the capacity. With smaller window sizes (greater than one) the ACKs are received way later than number of retransmissions, resulting in lower capacity. This capacity is regained gradually with the size of window as the ACKs have more time to be received before retransmitting the maximum allowed transmissions, which result in increased capacity, i.e. sender knows that the packet successfully achieved destination.

References

[1] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, “Eliminating steganography in internet traffic with active wardens”, in *Information Hiding: 5th International Workshop, IH 2002 Noordwijkerhout, The Netherlands, October 7-9, 2002 Revised Papers 5*, Springer, 2003, pp. 18–35.