CENG 519

# 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 mainly 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

For this system, capacity is calculated as the number of successfully received covert bits divided by the total number of packets sent, including retransmissions. Measured capacities for different channel parameters are given 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.

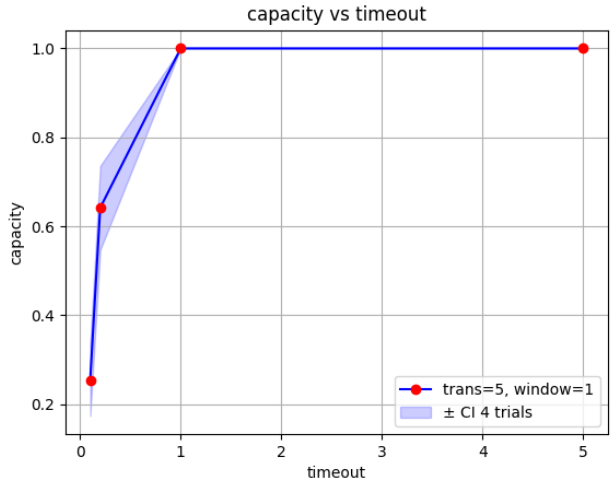


**Figure 1.** Capacity (bits/packet) vs. timeout (seconds)

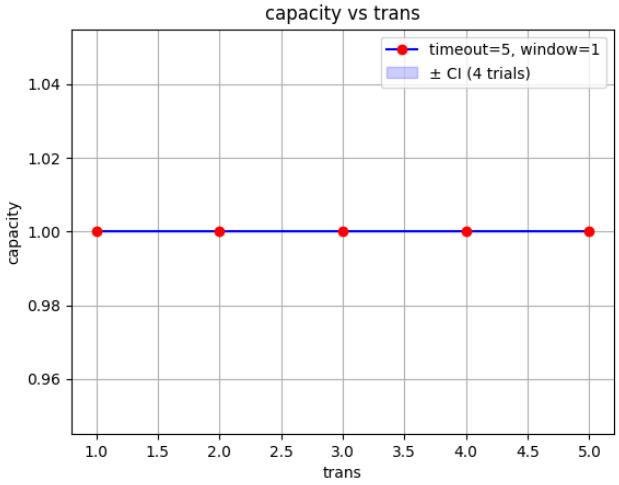


**Figure 2.** Capacity (bits/packet) vs. maximum transmission (packets)

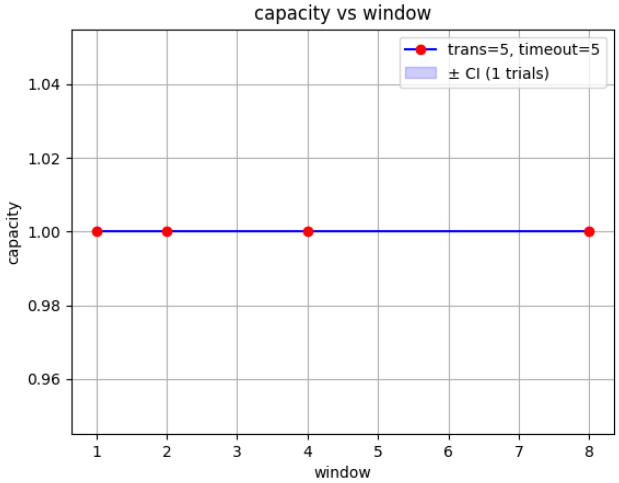


**Figure 3.** Capacity (bits/packet) vs. window size (packets)

Some edge cases are tested for timeout parameter, together with 0.01 seconds mean delay added in the processor (i.e. 0.02 mean delay added on ACK to be received) in Figures 4 and 5. In the capacity measurements, it is assumed that the packet is dropped if

sender does not receive an ACK for that packet. Furthermore, retransmission of a packet lowers the capacity when we define capacity as `number_of_successful_covert_bits / total_packets_sent` i.e. total packets sent increases with retransmission without increasing number of successful covert bits sent. Therefore, even if the receiver fully gets the covert message, the capacity is lower compared to sending all covert bits only once. When timeout parameter is too low, sender will keep retransmitting the packet, lowering the overall capacity as in Figure 4 (left). On the other hand, if sufficient time is given, no retransmission is necessary and capacity stays at 1 (right). Note that with a transmission parameter of 5, capacity can at most be 0.2 in the case of all packets time-out, i.e. every packet will be transmitted 5 times $capacity = N/5N = 0.2 \ \ bits/packet$ for $N$ covert bits which is the case in Figure 4.

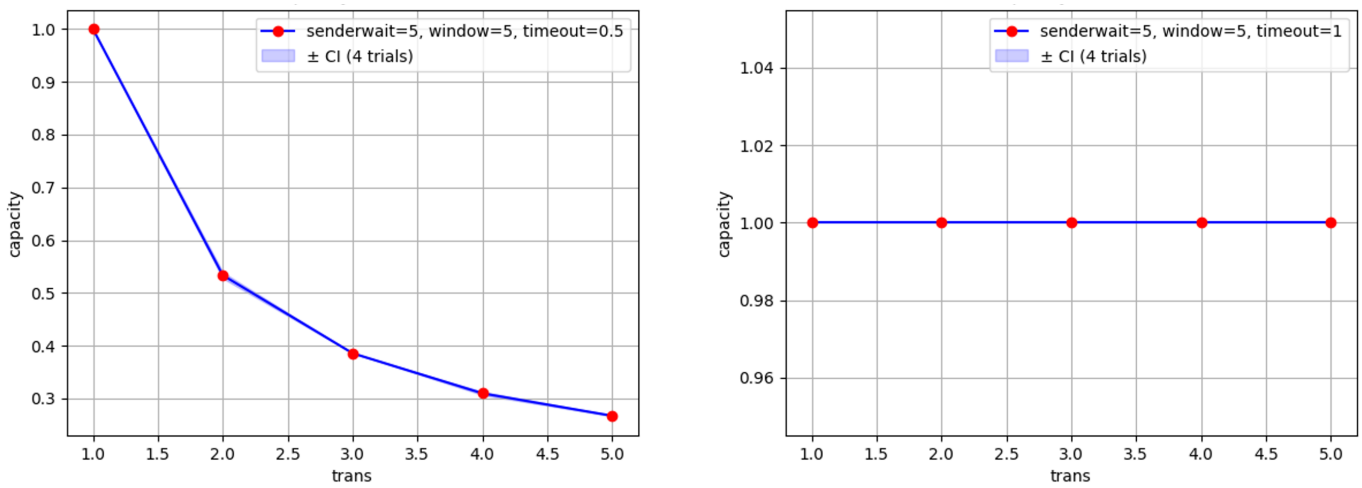


**Figure 4.** Capacity vs. number of transmission of the same packet in an edge case with timeout of 0.5s (left) and timeout of 1s (right).

For the window size experiments (Figure 5) with a critical timeout value, i.e. almost all ACKs are received later than timeout, 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 to the destination. In the previous case sender is not sure what happened to the packet so it marks it as dropped and moves onto the following packets.

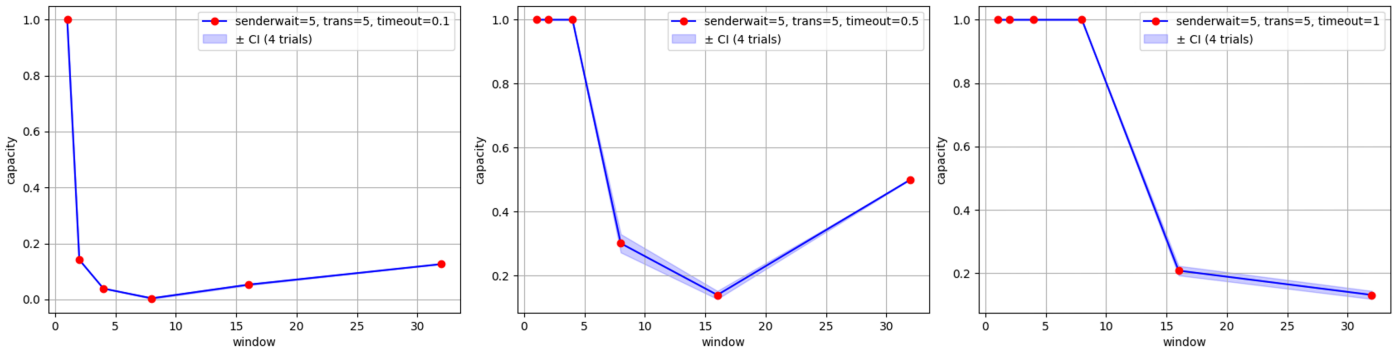


**Figure 5.** Capacity vs. window size in an edge case. Effects of changing timeout is observed with 0.1s (left), 0.5s (middle) and 1s (right) for window sizes.

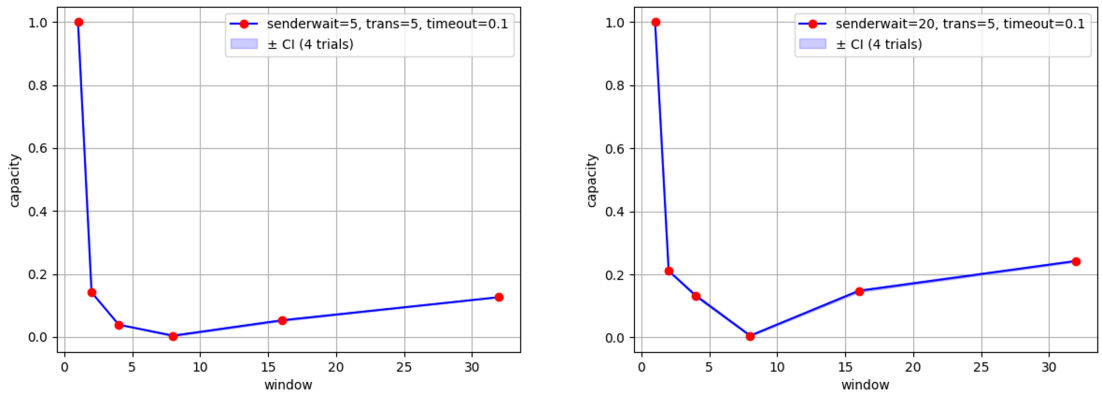


**Figure 6.** Capacity vs. window size graphs under parameters waiting 5 seconds (left) and 20 seconds (right) before closing communication.

From the experiments, it is observed that window size can become a bottleneck of the communication (Figure 5) when sufficient timeout parameter is not given; otherwise, capacity is 1 as in Figure 3. This is because sending packets within a window is a blocking operation in the system. In the absence of sufficient timeout, the ACKs are queued and assumed to be dropped when sending side finishes. To collect more ACKs before calculating the capacity, `senderwait` parameter is used to wait that many seconds before closing the communication. The effects of this waiting time is demonstrated in Figure 6. For the edge cases where capacity measurements drops severely, i.e. small timeout and larger window sizes, additional waiting time improves overall calculations as more ACKs gets to be received.

## 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.