One-Way Delay Measurement: Wired vs Wireless Interfaces

CS 244: Computer Networks

Lamar AlSubhi

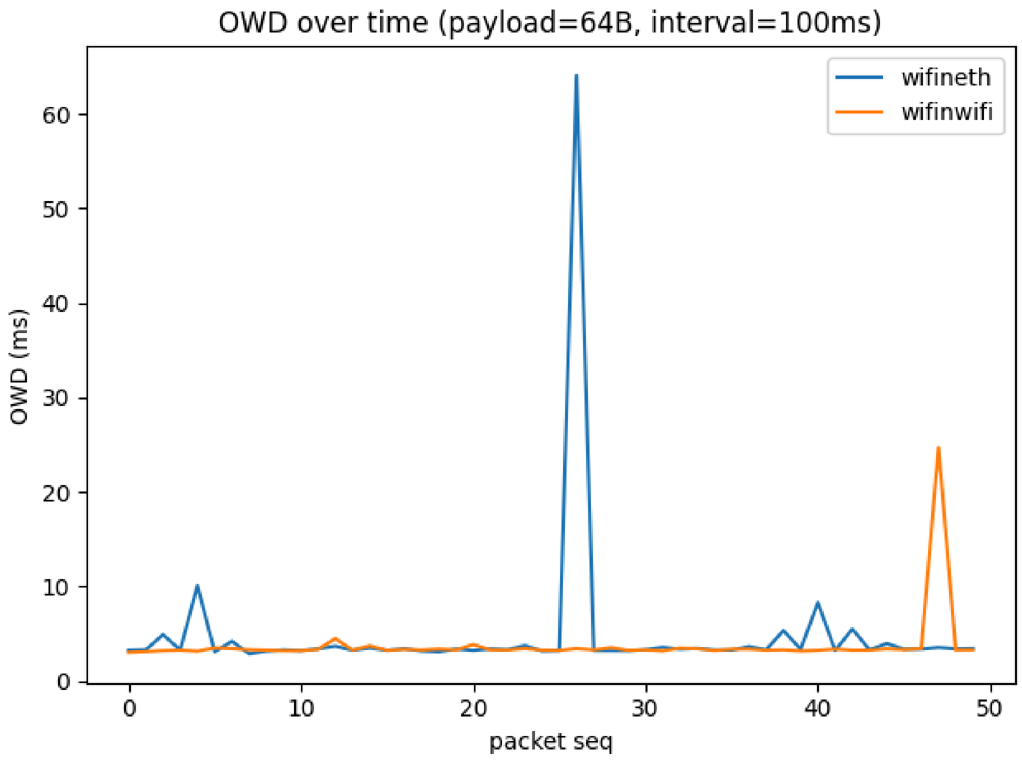
Methodology

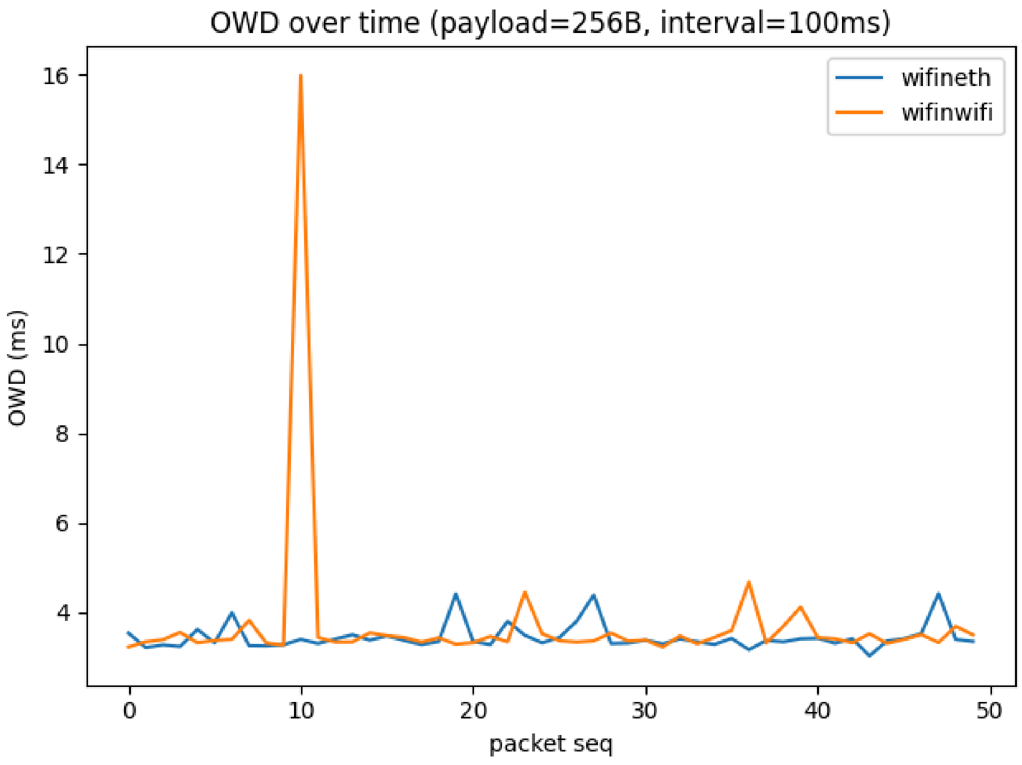
I built a lightweight client–server system in Python using the socket library. The communication was based on TCP with the TCP_NODELAY option enabled to reduce buffering, but I wanted to achieve even more precise measurements. To account for clock differences between devices, I added a synchronization phase at the start of each run. In this phase, the offset between the client and server clocks was estimated and incorporated into the one-way delay calculation. After synchronization, the client began sending "BOOP" messages containing its local send timestamp. Upon receiving each message, the server replied with an acknowledgment that included the server's receive timestamp. All delay computation and processing was performed on the client side, which recorded the sequence number, timestamps, payload size, and computed one-way delay for each packet in a CSV file. These logs were then used for post-processing and analysis. I conducted two categories of experiments: Wi-Fi-to-Wi-Fi and Wi-Fi-to-Ethernet. For both, I varied the payload size, the interval between messages, and the total number of packets transmitted. My implementation used Python's socket, time, and argparse libraries, while the analysis and visualization were carried out with Pandas and Matplotlib.

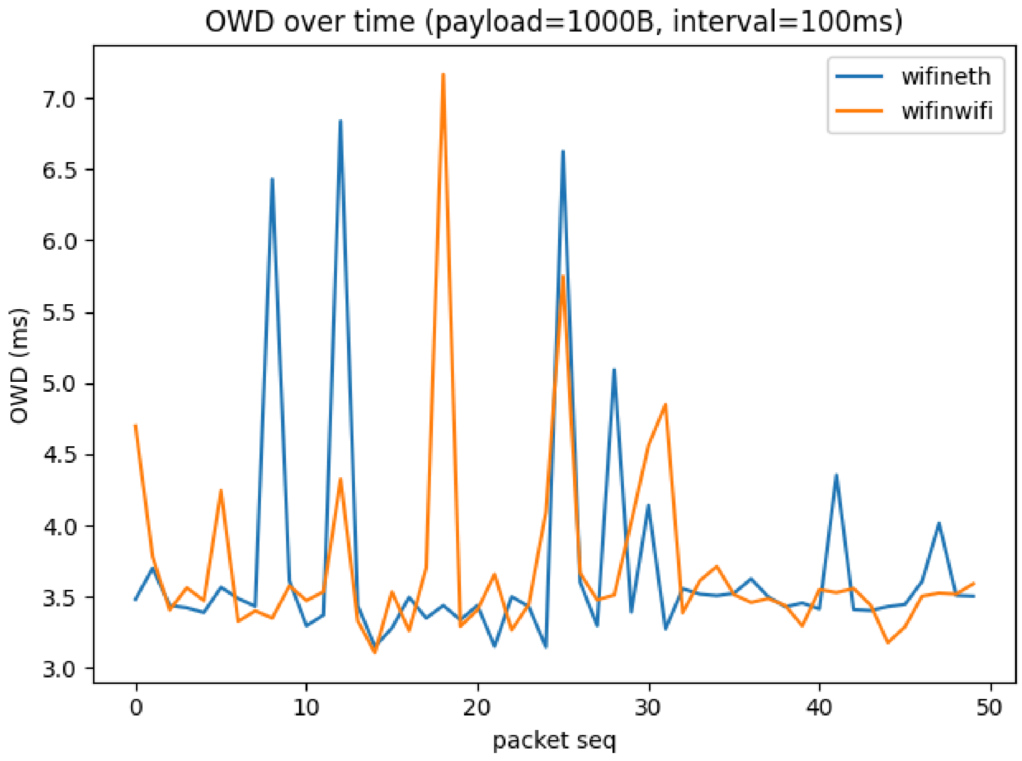
Topology

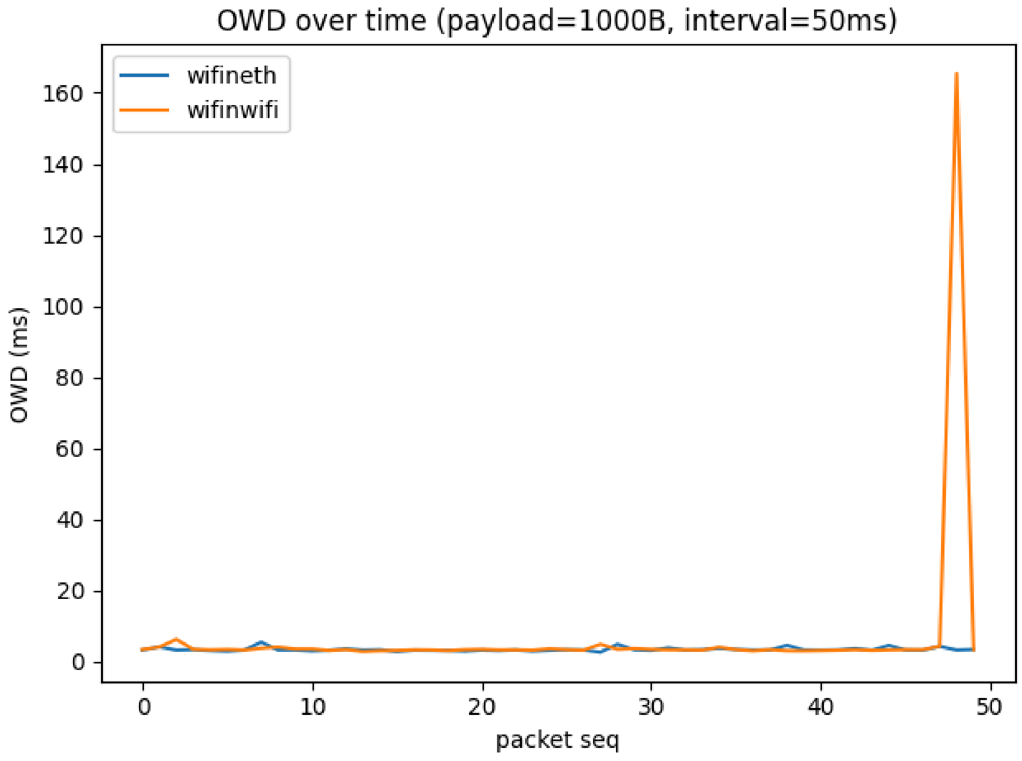
The topology consisted of a MacBook acting as the server and an OMEN HP laptop acting as the client. Measurements were conducted across two types of interfaces: Wi-Fi and Ethernet. During setup, I discovered that my MacBook was unable to function reliably as a client. Even basic connectivity tests such as ping to the Omen's address failed, regardless of whether the MacBook or the Omen was on Ethernet, the KAUST Wi-Fi, or a mobile hotspot connection. This issue persisted even when testing Mac-to-Mac configurations, indicating that the limitation was due to an underlying hardware or network setting on my personal device that I could not disable in time. As a result, I fixed the roles of the devices with the MacBook serving as the server and the Omen laptop serving as the client for all experiments. Although Windows and macOS rely on different underlying network stacks (Windows not being UNIX/Linux-based), the TCP implementation is standards-compliant. I prepared for the possibility of subtle differences in default socket behavior, but no platform-specific issues were observed during the experiments.

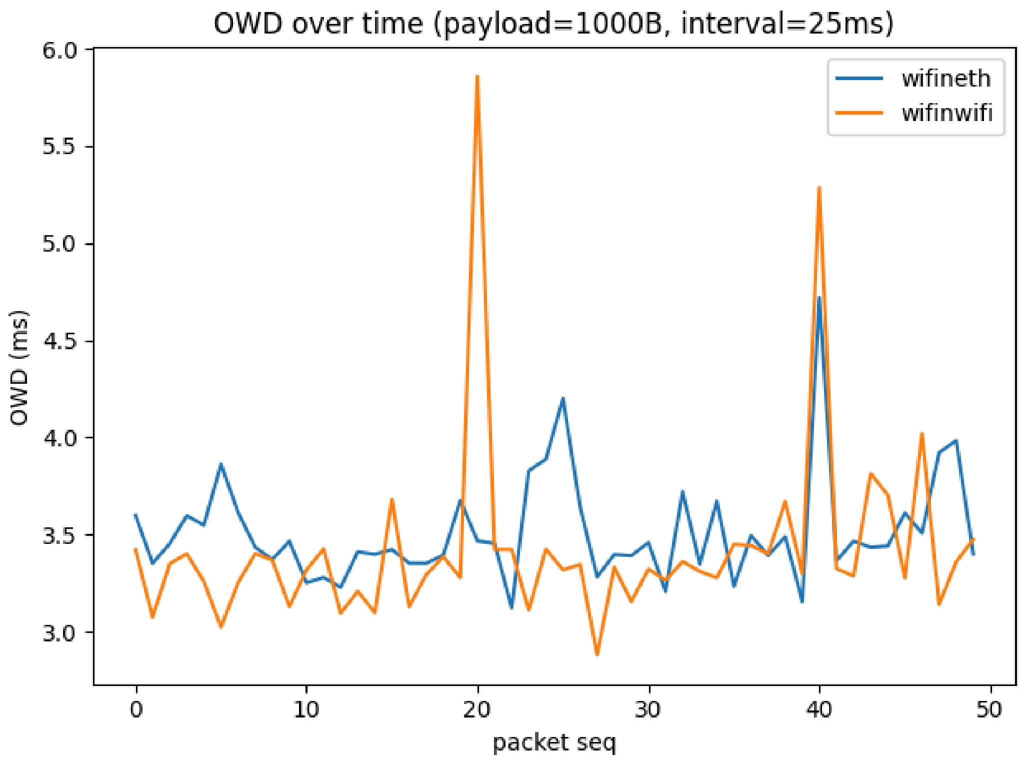
Results

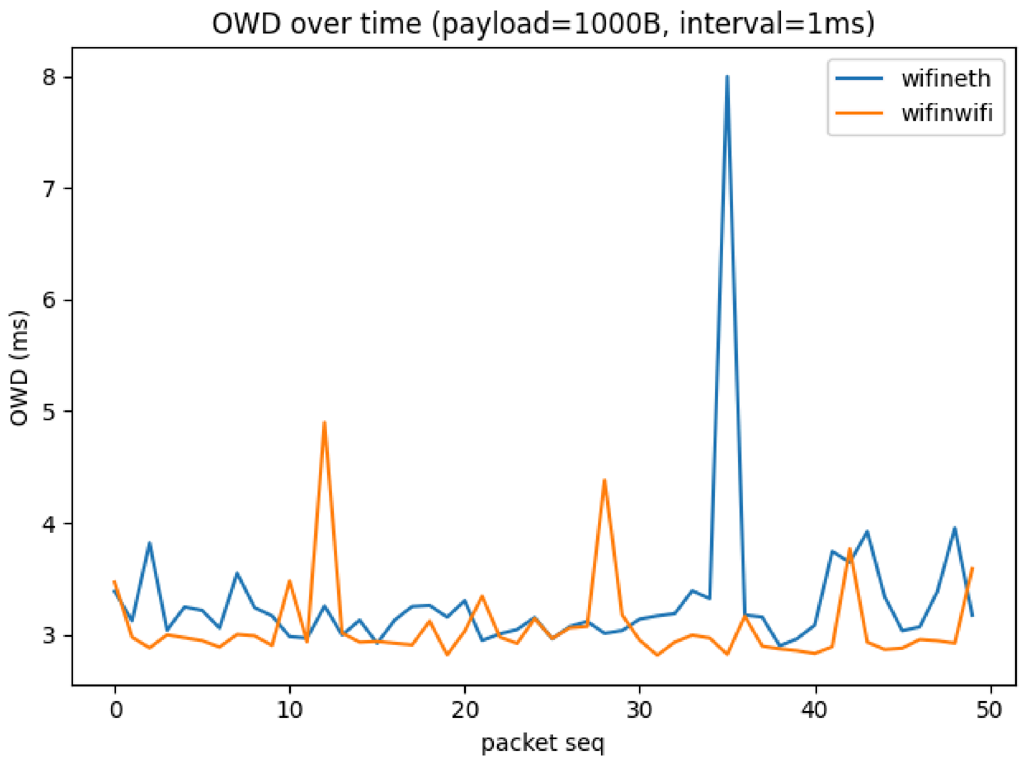

OWD over time (payload=64B, interval=100ms)
OWD (ms)
packet seq
wifineth
wifinwifi
0
10
20
30
40
50
60


OWD over time (payload=256B, interval=100ms)
OWD (ms)
packet seq
wifineth
wifinwifi
4
6
8
10
12
14
16
0
10
20
30
40
50


OWD over time (payload=1000B, interval=100ms)
wifineth
wifinwifi
OWD (ms)
packet seq
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
0
10
20
30
40
50


OWD over time (payload=1000B, interval=50ms)
wifineth
wifinwifi
OWD (ms)
packet seq
0
20
40
60
80
100
120
140
160
0
10
20
30
40
50


OWD over time (payload=1000B, interval=25ms)
OWD (ms)
packet seq
wifineth
wifinwifi
6.0
5.5
5.0
4.5
4.0
3.5
3.0
0
10
20
30
40
50


OWD over time (payload=1000B, interval=1ms)
OWD (ms)
packet seq
wifineth
wifinwifi
8
7
6
5
4
3
0
10
20
30
40
50

Analysis

The analysis of Wi-Fi and Ethernet show that both interfaces maintained average one-way delays in the range of 3–4 ms. Ethernet exhibited generally lower variance but occasional spikes, while Wi-Fi showed a smoother baseline but experienced more severe jitter events, particularly with larger payloads and at mid-range intervals (25–50 ms). The most extreme Wi-Fi spike reached ~160 ms during the 50 ms interval run, compared to Ethernet spikes peaking near ~60 ms. Across all runs, packet delivery was highly reliable. The client logs showed that nearly all transmitted packets were received and acknowledged. Even under high-frequency transmission (1 ms interval), no systematic losses were observed. This indicates that delay variation and jitter, rather than packet loss, were the primary factors differentiating wired and wireless performance.

Scripts

client.py

- required arguments: --host
- optional arguemnts:
    - --port: port used
    - --label: Wifi or eth
    - --payload: an estimate of the message size you want
    - --interval: how often to send messages in milliseconds
    - --count: total number of messages to send
- example: python3 server.py --host 192.168.8.30 --port 5001 --label wifi --payload 64 --interval 100 --count 50

```
import socket, time, argparse, csv, os


def parse_args():
   ap = argparse.ArgumentParser()
   ap.add_argument("--host", required=True, help="server IP (Wi-Fi or Ethernet)")
   ap.add_argument("--port", type=int, default=5001)
   ap.add_argument("--label", default="wifi", help="run label: wifi or eth")
   ap.add_argument("--payload", type=int, default=0, help="extra bytes to append")
   ap.add_argument("--interval", type=int, default=100)
   ap.add_argument("--count", type=int, default=10)

   return ap.parse_args()

# this estimates clock offset
def sync(s, seq=20):

   print("------------------- SYNC PHASE -------------------")
   offsets = []

   for i in range(seq):

       # t0: client time before send
       t0 = time.time_ns()
       msg = f"SYNC,{i},t0={t0}"
       s.sendall((msg + "\n").encode())

       # read one full reply line
       reply = s.recv(1024).decode().strip()

       # t2: client time right after recive
       t2 = time.time_ns()
       # expect "SYNC_ACK,{i},t1={t1}"
       parts = reply.split(",")

       # t1: server time when recieved
       t1 = int(parts[2].split("t1=", 1)[1])

       roundtrip = t2 - t0
       offsets.append(t1 - (t0 + roundtrip // 2))

       # just to control pacing
       time.sleep(0.005)

   offsets.sort()
   median = offsets[len(offsets) // 2]

   print(f"------------------- SYNC DONE offset={median} -------------------")

   return median



def run():
   args = parse_args()
   HOST = args.host
   PORT = args.port
   COUNT = args.count
   INTERVAL = args.interval
   PAYLOAD = args.payload
   PADDING= "A" * PAYLOAD if PAYLOAD > 0 else ""
   LABEL = args.label


   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
       s.connect((HOST, PORT))
       print(f"client connected. host:{HOST} port:{PORT}")

       # SYNC HERE
       offset = sync(s)

       # initialize log
       os.makedirs("logs", exist_ok=True)
       LOG = f"logs/client_{LABEL}_p{PAYLOAD}_i{INTERVAL}_c{COUNT}.csv"
       f = open(LOG, "w", newline="")
       w = csv.writer(f)

w.writerow(["seq","time_sent","time_received",f"delay(offset={offset})","payload_bytes
"])

       next_send = time.time()
```

```
        for seq in range(COUNT):   # 0 to COUNT-1
            now = time.time()

            # to avoid loop running faster than its supposed to
            if now < next_send:
                time.sleep(next_send - now)

            t0 = time.time_ns()

            msg = f"BOOP,{seq},time_sent={t0},{PADDING}"

            # SEND
            s.sendall((msg + "\n").encode())
            # print(f"[CLIENT] sent: {msg}")
            payload_bytes = len(msg.encode())

            # RECIEVE
            reply = s.recv(1024).decode().strip()
            # print(f"[CLIENT] recv: {reply}")

            # parse time_receieved from reply and compute OWD = time_recieved -
(time_sent + time_desync)
            parts = reply.split(",", 2)
            seq = int(parts[1])
            t1 = float(parts[2].split("t1=", 1)[1])
            OWD = (t1 - offset) - t0

            # append a CSV row here for analysis
            w.writerow([seq, f"{t0}", f"{t1}", f"{OWD}", f"{payload_bytes}"])
            # print(f"[CLIENT] seq={seq} OWD={OWD:.3f} ns")

            next_send += INTERVAL/ 1000.0

        print("[CLIENT] done")
        f.close()

if __name__ == "__main__":
   run()
```

server.py

- required arguments: --host
- optional arguemnts: --port
- example: python3 server.py --host 192.168.8.30 --port 5001

```
import socket, time, datetime, argparse

# I will need to use 2 different connection types to compare their delays
# "both wired and wireless interfaces (e.g., eth0 vs wlan0)"
def parse_args():
   ap = argparse.ArgumentParser()
   ap.add_argument("--host", required=True, help="server IP")
   ap.add_argument("--port", type=int, default=5001)
   return ap.parse_args()


def run():
   args = parse_args()
   HOST = args.host
   PORT = args.port

   # AF_INET: address family for IPv4, SOCK_STREAM: TCP
   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

       s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
       s.bind((HOST, PORT))
       s.listen(1)
       print(f"[SERVER] bound host:{HOST} port:{PORT}")

       conn, addr = s.accept()
       print("[SERVER] connection accepted from:", addr)
       conn.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)

       # since TCP is a data stream, we gotta listen for whole line
       buffer = ""
       while True:
           data = conn.recv(1024)
           if not data:
               break  # connection closed
           buffer += data.decode()

           # process all complete lines
           while "\n" in buffer:
               line, buffer = buffer.split("\n", 1)
               line = line.strip()
               if not line:
                   continue

               # client should send BOOP,{seq},t0={t0},{padding}
               # or SYNC,{round},t0={t0}
               parts = line.split(",", 2)
               t1 = time.time_ns()  # server receive time
               seq = parts[1]

               if parts[0] == "SYNC":
                   reply = f"SYNC_ACK,{seq},t1={t1}"

               else:

                   reply = f"ACK,{seq},t1={t1}"
               conn.sendall((reply+"\n").encode())
               print(reply)


if __name__ == "__main__":
   run()
```