<u>Computer Networks Assignment 2</u>

Amirmahdi Ansaripour 810198358
Soroosh Hamedi 810198503

In this assignment, a network, which contains 3 senders (transmitters), 3 receivers, and a load balancer, is implemented. Moreover, its error rate, bandwidth, one way delay, and their probable interactions are analysed.

At first, the topology is explained:

```
# defining node
$ns node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -topoInstance $topo \
                -agentTrace ON \
                -macTrace ON \
                -routerTrace ON \
                -movementTrace ON \
                -channel $channel1\
                -IncomingErrProc HandleErrorRate

#set nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
#balancer
set n3 [$ns node]
#balancer
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

#TCP connections
```

As the figure describes, the topology consists of six nodes, and wireless MAC protocols exist between them.

There are 3 UDP connections from node0,1,2 to node3, and 3 TCP connections between node3 and node4,5,6(In fact, nodes 4,5,6 are the sinks of the TCP connection). After the connections are established, initNetwork() function resets the bytes obtained by sink_links.

Afterwards, the handleSink method calculates the throughput of each link through measuring the amount of bytes on that link each second.

```
#UDP connections
set udp0_3 [new Agent/UDP]
set udp1_3 [new Agent/UDP]
set udp2_3 [new Agent/UDP]

set balancer0_3 [new Agent/Null]
set balancer1_3 [new Agent/Null]
set balancer2_3 [new Agent/Null]

$udp0_3 set packetSize_ $packet_size.Kb
$udp1_3 set packetSize_ $packet_size.Kb
$udp2_3 set packetSize_ $packet_size.Kb

$ns attach-agent $n0 $udp0_3
$ns attach-agent $n1 $udp1_3
$ns attach-agent $n2 $udp2_3
$ns attach-agent $n3 $balancer0_3
$ns attach-agent $n3 $balancer1_3
$ns attach-agent $n3 $balancer2_3

$ns connect $udp0_3 $balancer0_3
$ns connect $udp1_3 $balancer1_3
$ns connect $udp2_3 $balancer2_3

set cbr0_3 [new Application/Traffic/CBR]
set cbr1_3 [new Application/Traffic/CBR]
set cbr2_3 [new Application/Traffic/CBR]

$cbr0_3 attach-agent $udp0_3
$cbr1_3 attach-agent $udp1_3
$cbr2_3 attach-agent $udp2_3

## TCP connections
set tcp3_4 [new Agent/TCP]
set tcp3_5 [new Agent/TCP]
set tcp3_6 [new Agent/TCP]
set sink3_4 [new Agent/TCPSink]
set sink3_5 [new Agent/TCPSink]
```

```
proc handleSink {} {
    global sink3_4 sink3_5 sink3_6
    global throughput4 throughput5 throughput6 time ns

    set link3_4 [$sink3_4 set bytes_]
    set link3_5 [$sink3_5 set bytes_]
    set link3_6 [$sink3_6 set bytes_]

    set currTime [$ns now]
    # The load on sink
    set load4 [expr ($link3_4*8)/$time]
    set load5 [expr ($link3_5*8)/$time]
    set load6 [expr ($link3_6*8)/$time]

    puts $throughput4 "$currTime [expr $load4]"
    puts $throughput5 "$currTime [expr $load5]"
    puts $throughput6 "$currTime [expr $load6]"

    $sink3_4 set bytes_ 0
    $sink3_5 set bytes_ 0
    $sink3_6 set bytes_ 0
    # Repeat the process till time = 100
    $ns at [expr $currTime+$time] "handleSink"
}

proc finish {} {
    global ns tracefd namfile throughput4 throughput5 throughput6
    $ns flush-trace

    close $namfile
    close $tracefd

    close $throughput4
    close $throughput5
    close $throughput6

    exit 0
}

$ns at $val(finish) "finish"

$ns run
```

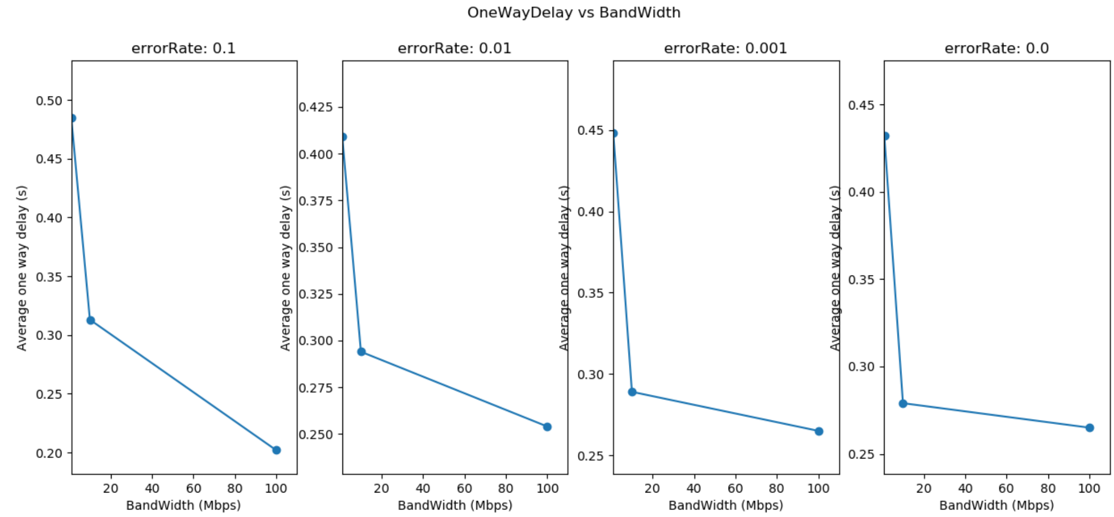
Outputs are generated through running the code below:

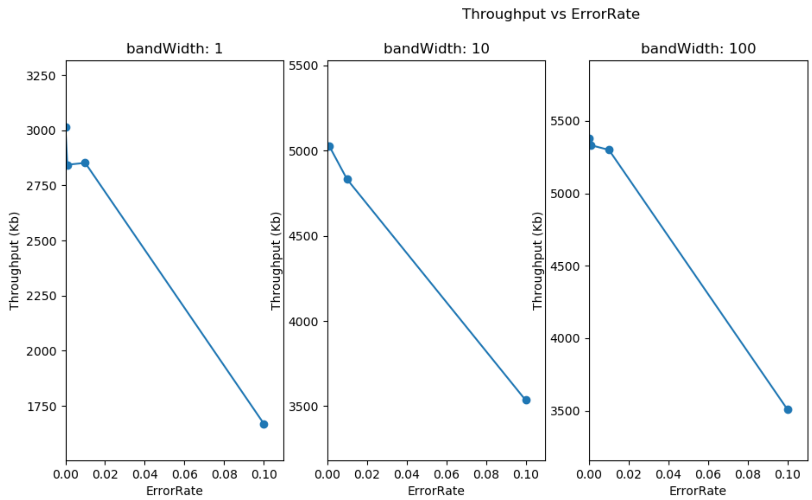
ns wireless.tcl error-rate bandwidth

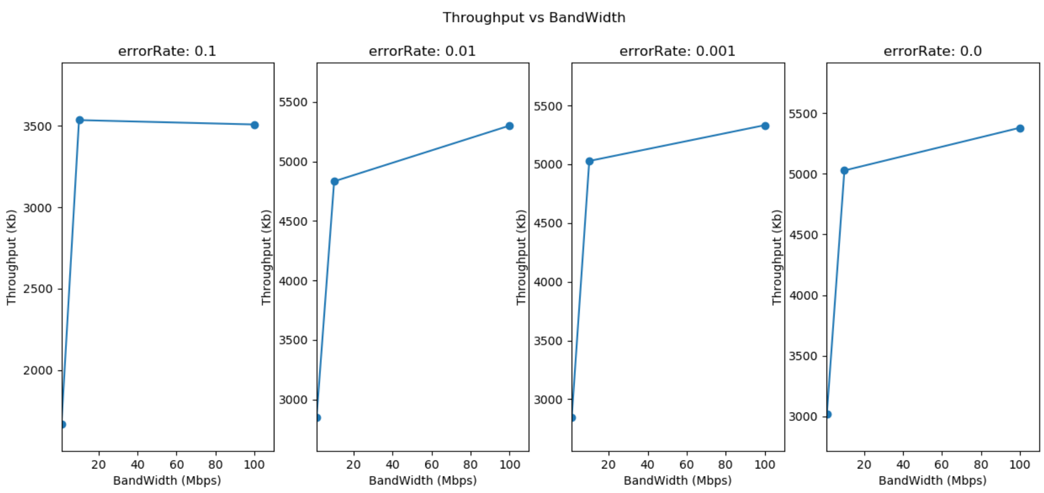
The outputs are some .tcr files containing useful information. They are explored in the network.py file, which is responsible for plotting the results.

```
def runCommand(self, errorRate, bw):
    print("Running for errorRate: " + str(errorRate) + "\tBW: " + str(bw))
    check_call(['ns', self.tclFile, str(errorRate), str(bw)], stdout=DEVNULL,
                        stderr=STDOUT)


def plot(self, data, superTitle, subTitle, x_ax, y_ax):
    plt.figure(figsize = (15, 6))
    keys = list(data.keys())
    values = list(data.values())
    # print(keys)
    # print(values)
    for i in range(len(keys)):
        plt.subplot(1, 4, i + 1)
        plt.suptitle(superTitle)
        plt.title(subTitle + str(keys[i]))
        pointsX = [i[0] for i in values[i]]
        pointsY = [i[1] for i in values[i]]
        # print(pointsX)
        # print(pointsY)
        plt.xlim([float(0.9* min(pointsX)), float(1.1*max(pointsX))])
        plt.ylim([float(0.9*min(pointsY)), float(1.1*max(pointsY))])
        plt.plot(pointsX, pointsY, marker = 'o')
        plt.ylabel(y_ax)
        plt.xlabel(x_ax)

    plt.show()
```

At last, by running the network module, we will have:







The output graphs are generated as expected, because it is natural that by increasing the bandwidth, one-way delay is reduced(the error-rate is also considered). Moreover, error-rate increase causes throughput decline, which can be concluded from the second figure. Finally, an increase in bandwidth leads to throughput rise, which is again demonstrated in the third figure.