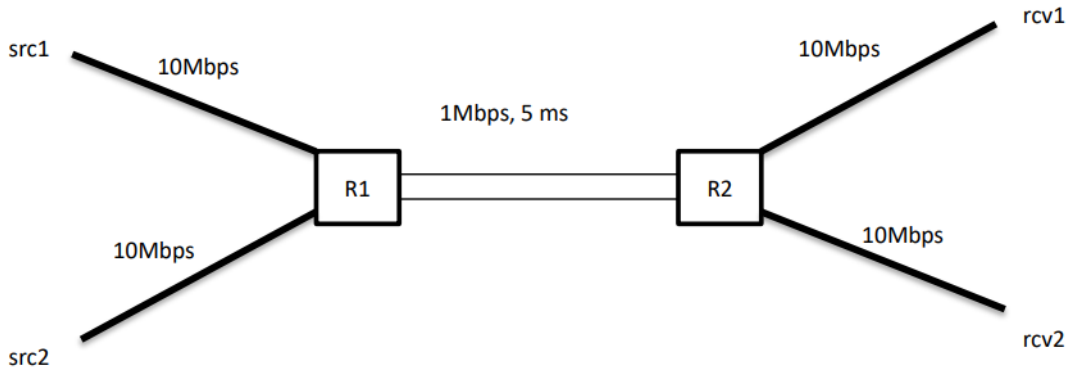
Network Simulation Report:

Test Setup:

The layout of the network is done manually to make topology look like the figure below:



The end-to-end delay from source1 to receiver1 is fixed to 5ms while the end-to-end delay from source2 to receiver 2 is assigned by the user by indicating the test case that the network will simulate. For test case 1, the delay is 12.5ms. For test case 2, it is 20ms. And for test case 3, it is 27.5ms. The TCP implementation (Vegas or Sack) is also assigned by the user in the command line. With these two arguments, we ran each test case for both TCP flavors using the ns2 network simulator.

Procedure:

We followed the standard methodology presented in the TA recitation slides. After creating a simulator object, we create a trace file record the results during the 400s simulation. We created 6 nodes corresponding to the figure shown above and incorporated the delays when defining the links and queuing. We then manually assigned the topology layout as shown above. We established a tcp connection by instantiating the TCP flavor extracted from the arguments in the command line, assigned it to one of the sources and assigned its corresponding sink to the corresponding receiver. We simulate traffic from an FTP application for each source-receiver pair. During the 400s simulation, we start recording throughput for each source at 100s ignoring the first 100s in the simulation while measuring metrics.

Results:

(i)

Flavor: VEGAS

Test Case:	Ratio of AVG. Throughput of src1 to src2:
Case 1	1.4
Case 2	2.2

Case 3	3.0

Flavor: SACK

Test Case:	Ratio of AVG. Throughput of src1 to src2:
Case 1	1.1
Case 2	1.2
Case 3	1.3

Terminal Outputs:

```
ericr@ecen-602:~$ ns ns2.tcl VEGAS 1
Average throughput for src1: 0.58252000000000148 Mbit/s

Average throughput for src2: 0.41599999999999765 Mbit/s

Ratio of average throughput of src1 to src2: 1.400288461538473
```

```
ericr@ecen-602:~$ ns ns2.tcl VEGAS 2
Average throughput for src1: 0.68663999999999692 Mbit/s

Average throughput for src2: 0.31185999999999814 Mbit/s

Ratio of average throughput of src1 to src2: 2.2017571987430289
```

```
ericr@ecen-602:~$ ns ns2.tcl VEGAS 3
Average throughput for src1: 0.99869333333332877 Mbit/s

Average throughput for src2: 0.33264000000000099 Mbit/s

Ratio of average throughput of src1 to src2: 3.0023248356581465
```

```
ericr@ecen-602:~$ ns ns2.tcl SACK 1
Average throughput for src1: 0.52330800000000588 Mbit/s

Average throughput for src2: 0.47523919999999636 Mbit/s

Ratio of average throughput of src1 to src2: 1.1011465384168855
```

```
ericr@ecen-602:~$ ns ns2.tcl SACK 2
Average throughput for src1: 0.54514800000000518 Mbit/s

Average throughput for src2: 0.45339919999999467 Mbit/s

Ratio of average throughput of src1 to src2: 1.2023576574462673
```

```
ericr@ecen-602:~$ ns ns2.tcl SACK 3
Average throughput for src1: 0.565032800000007 Mbit/s

Average throughput for src2: 0.43349359999999548 Mbit/s

Ratio of average throughput of src1 to src2: 1.3034397739667041
```

(ii)

In the cases for source2 shown above, we can deduce that the TCP throughout decrease as RTT delay increase. This is intuitive since delays would cause the network to process packets less frequently which would cause the routers' performance to deteriorate or be bogged down by the delays. The ratio of throughput is greater for TCP Vegas than TCP Sack in the first test case. This is because TCP Vegas takes a more proactive approach in alleviating traffic congestion in the network compared to TCP Sack. TCP Vegas dynamically resize its sending window size based on the RTT delays of packets sent (getting ACK from receiver). TCP also detects congestion early by detecting increase in RTT delays. Whereas TCP SACK is an extension of TCP Reno which only detects congestion when a packet is lost. TCP SACK selectively acknowledges. Overall, TCP Vegas algorithm measures and controls the amount of data packets hosts can transmit considering the available network so it dynamically adjusts based on the current congestion that the network is facing whereas the TCP SACK has a fixed way of alleviating congestion which is keep increasing window size until packet is lost.

Nam file screenshot -