Studying the Effects of Cross Traffic on Packet Pair Probing Bandwidth Measurement Algorithm

Project Link: https://github.com/X-OppenHeimer-X/Packet-Pair-Probing

Toshitt Ahuja
(ta498)

Jodh Singh
(js2993)

Goal

In this project, we aim to study the effectiveness and robustness of **Packet Pair Probing**, a bandwidth measurement algorithm in the presence of Cross-traffic through its dispersion metric.

About & Pre-existing Work

i. Packet Pair Probing

The packet Pair probing algorithm is a bandwidth measurement algorithm proposed by Paxson [1] to measure a network path's end-to-end capacity or available bandwidth. In this algorithm, a pair of packets are sent back-to-back from one end of the path to the other. The metric that is the backbone of this algorithm is **dispersion**(denoted by Δ).

Dispersion can be defined as the time distance between the last bits of both packets. Dispersion can be visualized through the figure below.

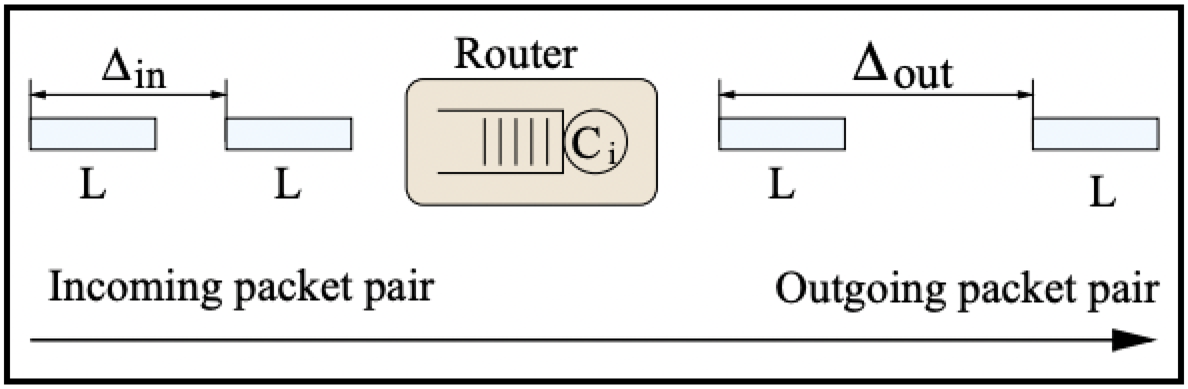


Fig 1: This figure shows the dispersion between the packets before and after it goes through a link of Capacity Ci.

As seen in the above figure, Δin is the dispersion before the packets go through a channel of capacity Ci, and Δout is the dispersion of the packets just at the end of the link. The dispersion at the end of the link can be measured as:

$$\Delta out = max(\Delta in,\ L/Ci) \qquad (1)$$

Where L is the size of the packet, and Ci is the channel's capacity. Considering there are N number of channels and a packet pair has gone through all of them, the dispersion Δr becomes

$$\Delta r = L\ /\ max(C1, C2.... Ci) \qquad (2)$$

Since we can only have one link between host, the equation(2) in our case would simply get reduced to:

$$\Delta r = L\ /\ C \qquad (2a)$$

$$C = L\ /\ \Delta r \qquad (3)$$

Where C is the end-to-end capacity of the link.

Pre-Existing Work

Jacobson has covered the concept of packet dispersion in [2], where he describes it as a burst of the packet traversing through a narrow link of a path. Keshav explores dispersion packet dispersion through congestion control [3], using fair queueing to relate dispersion with the available bandwidth. Before the packet-pair probing algorithm, another algorithm that was used for bandwidth measurement was Variable Packet Size (VPS) probing, in which the game plan is to measure the RTT from the source to each hop of the path as a function of the probing packet size. The TTL(time-to-live) field from the IP header of the packet was used to force the packets to expire at a particular hop. The main limitation of this algorithm was that it had problems with capacity underestimation in case the network had store and forward layer-3 switches.

Novelty in the Project

The authors mention in the paper that the algorithm is based on the hypothesis that no cross-traffic is present. In case it is, it can cause the end-to-end dispersion metric Δr to to either increase or decrease, leading to either capacity underestimation or overestimation. In this experiment, we create a simple network with the help of an open-source network simulator 'mininet' to recreate the experiment in case of cross-traffic so we can study a section of this algorithm that the author didn't cover. We will do a deep comparative analysis of the behavior of Δr(L/C), Δout and Δin under two circumstances: **one,** when the capacity of the link is constant, but the packet size(L) keeps changing, and two, The Capacity of the bandwidth is different, but the packet size is constant.We will run the This will give us more insight into the robustness and reliability of this algorithm because, as the author mentions, circumstances where there are links with no-cross traffic, are almost non-existent.

Approach

Experimental Setup

For this experiment, we have set up the following network topology:

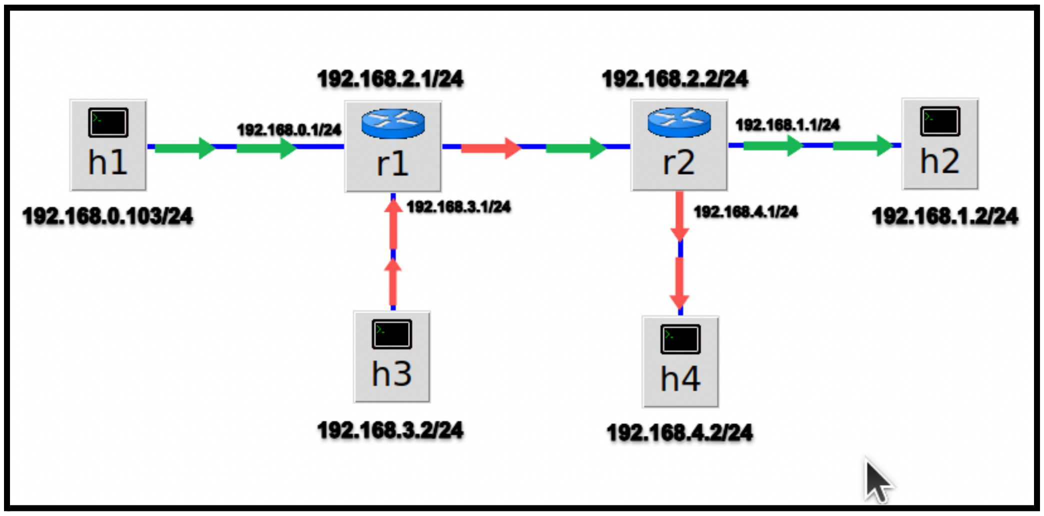


Fig 2: The network topology at glance. The bold black text shows the IPs of the host machines and the routers. The text in small represents their default gateways. The green arrows show the UDP packets going from host1 to host2, and the red arrows show the cross-traffic flowing from h3 to h4.

The topology in the figure above is created with the help of miniedit/mininet GUI. In this topology, all the host machines are reachable and connected with links of the same bandwidth, which can be changed depending on the user's preference. We have set up the routing configuration between the hosts and the routers with the help of Mininet Python API. All of the host machines are in different subnets: (h1: 192.168.0.103/24, h2: 192.168.1.2/24, h3: 192.168.3.2/24 and h4: 192.168.4.2/24). We send the packet pairs from h1 to h2, and h3 and h4 are used for generating the external cross-traffic in the system, which will be developed with the help of iperf.

With the Python struct library's help, we manually created our byte packets and sent them over a custom-made UDP socket(built with Python's socket library). Using our custom-built config module (config.py), we make the appropriate settings and changes(more details in our Repository's README) to select the experiment we want to perform. Once the experiment has started, we record the timestamps using time libraries and dump the lists into a pickle file to make the changes at the modular level. Once done, we plot the dispersion data using matplotlib.

Libraries Used

1. Mininet (Python API)
2. Numpy
3. Pickle
4. Matplotlib
5. Time
6. Socket
7. Struct

Experiments

As discussed in the above sections, we will undertake two scenarios to understand the behavior of the dispersion metric. Here, we have explained the steps to recreate the experiments in the project link.

A) Bandwidth is constant.

For this experiment, after selecting the bandwidth of all our links in the network, we will select a list of packet sizes that will be used. For this experiment, we have taken two one-byte packets: b‘\x00’ and b‘\x01’, and have created their variants of sizes: 3500,3750,4000,4250,5000,5500,6000 and 7500 bytes, respectively. We have created a list of tuples, each containing these two packets but of the sizes mentioned above in the same order.

We send these pairs of packets as UDP packets over a UDP client(with cross-traffic running from h3 to h4). We record the timestamps on the two packets, We then calculate the dispersions Δout and Δin using the following formulas:

$$\Delta in = t_{packet2} - t_{packet1} \qquad (4)$$

$$t'' = t' + len(packet)/(Bandwidth) \qquad (5)$$

$$\Delta out = t''_{packet2} - t''_{packet1} \qquad (6)$$

Where,

=> t is the timestamp when the packets are sent from the client,
=> t’ is the timestamp of when the packet is received.
=> t’’ is the time when the last bit of the packet is received.

We then save the results in two different Python lists and dump them in a pickle file(to save the changes at the modular level) for plotting. We will create another list for L/Ci values or the expected dispersion. We will create plots that contain initial, final dispersion, and L/C values. We repeat this same experiment for two scenarios: when cross-traffic is present and when there is no cross-traffic.

B) Packet size is constant.

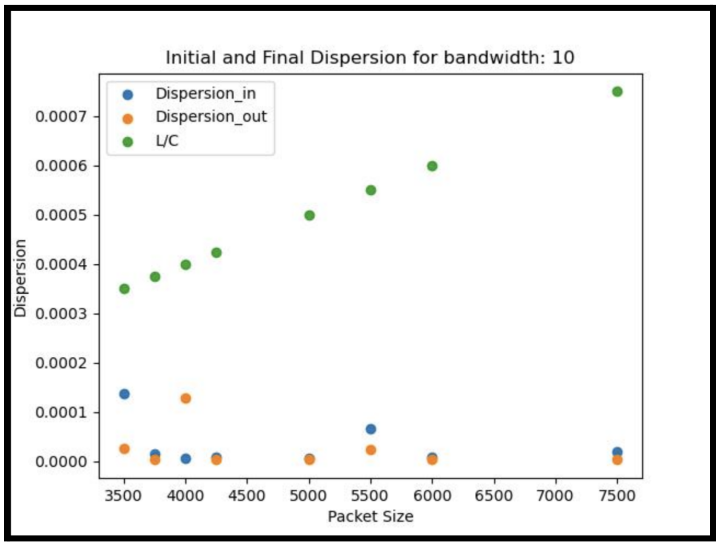
We will set up a pair of fixed packet sizes for this experiment, say 20,000 bytes. We will select a range of capacities of links: 20, 25, 30, 35, 40, 45, 50, and 55Mbits/sec. As done in the equations above, we will individually calculate each link capacity's initial and final dispersion and store them in two lists. We will again calculate the L/C values to compare with dispersion(Δout and Δin) values. Like the experiment earlier, we will run this experiment for two scenarios: when there is cross-traffic present in the system and when there is no cross-traffic present.

Results and Inferences

a) Constant Bandwith and varying Packet Size

i) Without Cross-traffic

For this part, we have repeated the experiment for 3 different bandwidths: 10, 15, and 20 Mbits per second. The results for all three are as follows:



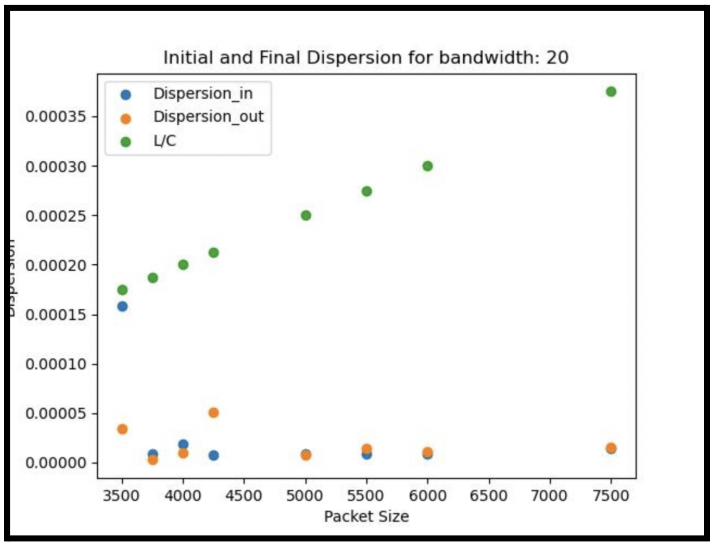
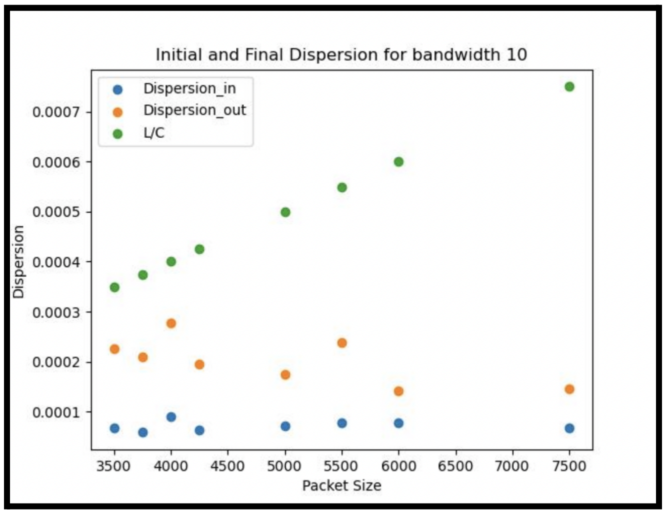


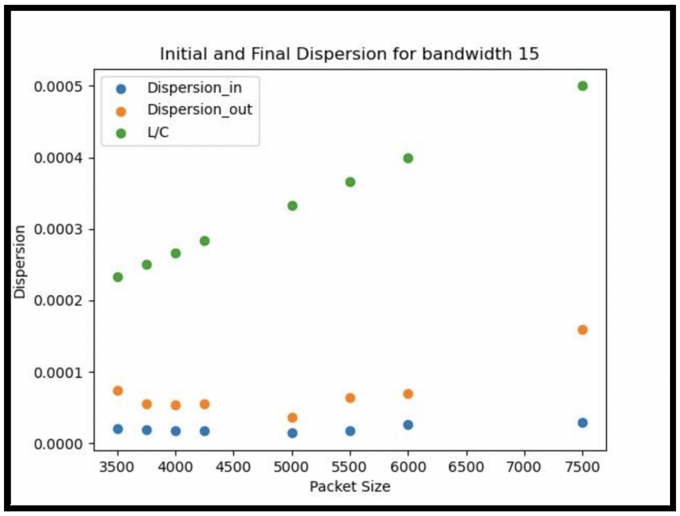
Fig 3: The plots show the dispersions over different packet sizes of the same bandwidth when no cross-traffic is present

We observe that both Δout and Δin are considerably less than L/C for all the packet sizes which have been selected. Also, the Δout and Δin values are almost the same, which was expected because the links are almost ideal, contain zero cross-traffic, and have almost next to zero resistance. Ideally, there would be some difference between the two values in cross-traffic presence because of interruptions caused by other packets. To confirm our hypothesis, we will repeat the same experiment in the case, but with cross-traffic present.

ii) With Cross-traffic

For this part, we have repeated the same experiment on the same three bandwidths(10,15 and 20 Mbits/sec) as above, but only this time we are inducing UDP cross-traffic from h3 to h4 with the helo of iperf. We create the h4 as the server and the h3 as the client. Through h3, we send UDP traffic to h4 for a long period (say 1000 seconds). The results we obtained are as follows:





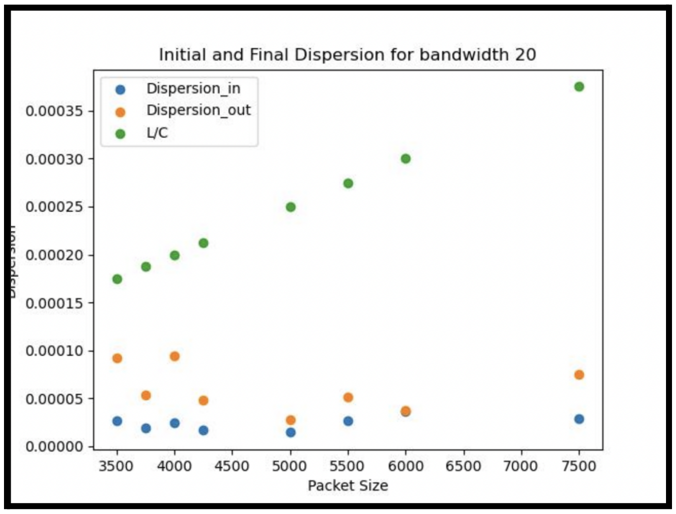


Fig 4: The plots show the dispersions over different packet sizes of the same bandwidth in the presence of cross-traffic.This plot helps make comparisons to L/C

Turns out our hypothesis was correct, Δout did increase from their values than in the case of the no cross-traffic. But we can't say whether there the capacity of the link is being underestimated or not. We must run experiments where packet size is fixed, and bandwidths vary. The individual insight into varying bandwidth will give us more insight into the behavior of Δout and can help us estimate whether the end-to-end capacity is being over or under-estimated.

b) Constant Packet size and varying Bandwidth

i) Without Cross-traffic

We run this experiment on a packet pair of 20,000 bytes for varying bandwidths of 20, 25, 30, 35, 40, 45, 50, and 55 Mbits/sec, respectively. The results can be seen in the plot below.

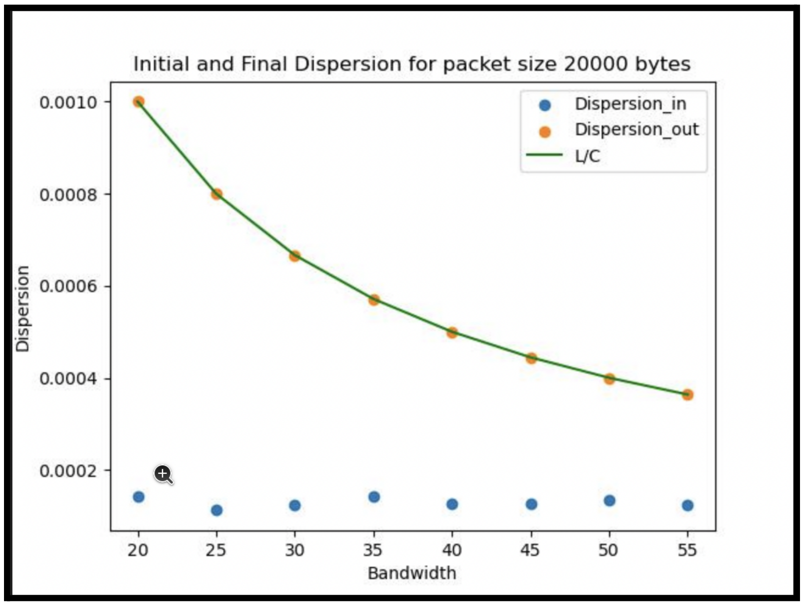


Fig 5: The plots show the dispersions of same packet sizes over different bandwidths

We observe that the Δout values are the same as the L/C values, an inference which matches with the author's inference from eqn(2). We will now observe the behavior of Δout when there is cross-traffic in the system.

ii) With Cross-traffic

We run the exact same experiment above, with the same packet size and same bandwidths, but this time in the presence of cross-traffic, with bandwidth the same as that of the link of the network. The results can be seen in the following plot:

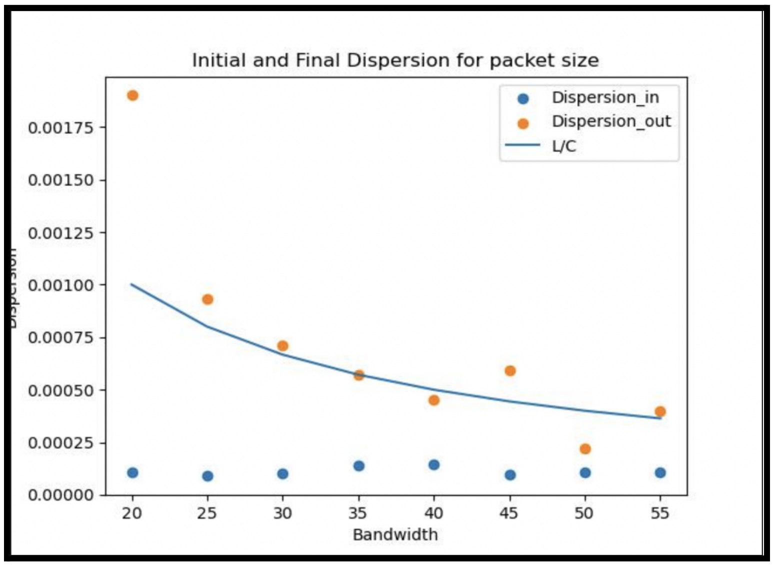


Fig 6: The plots show the dispersions over different packet sizes of the same bandwidth in the presence of cross-traffic

We see that generation of cross-traffic has a significant effect on the Δout value. We see that the Δout is greater than the L/C value, the supposed dispersion value. Although we see anomalies here and there, they can be ignored. It can be correct to say that there was clearly an underestimation of capacity in our system because the Δout values are clearly greater than L/C, and the reason can be what the author mentions, that the cross traffic packets are transmitted between the probing packet pair at a specific link, increasing the dispersion to more than the L/C value.

Conclusion

From our experiments above, we can conclude that the inclusion of cross-traffic causes an underestimation of capacity in the packet-pair probing algorithm. This, in a certain way, creates doubt regarding its robustness and scalability since the real-world networks involve cross-traffic. Underestimation of capacity would only cause blunders and wrong results in the long run, making the algorithm unsafe for industry-level applications.

References

1. V. Paxson, "End-to-end Internet packet dynamics," in IEEE/ACM Transactions on Networking, vol. 7, no. 3, pp. 277-292, June 1999, doi: 10.1109/90.779192.
2. V. Jacobson, “Congestion avoidance and control,” in Proc. ACM SIGCOMM, Sept. 1988, pp. 314–329.
3. S. Keshav, “A control-theoretic approach to flow control,” in Proc. ACM SIGCOMM, Sept. 1991, pp. 3–15.