Bandwidth Estimation of End-to-end Path using Optimum Slope Algorithm

Group # 15: Sriharsha Madala, Bolun Huang

Date: 12/07/2012

TABLE OF CONTENT

ABSTRACT

Traffic Analysis and Measurement has long been a research-intensive area especially when the traffic volume within the Internet has increased tremendously in recent decades. Among the characteristics of a physical network link, the available bandwidth is one of the most important metrics, indicating the capacity, latency and practical maximum bit rate that data transmission could achieve of the link. By studying existing tools that are able to estimate available bandwidth of an end-to-end path, especially *pathchar* using Variable Packet Size (VPS) techniques, we wrote an application using a brand new algorithm to adjust the shortcomings of pathchar. Based on VPS probing methodology, we found the best fitting slope of packet size over round-trip time (RTT) using Optimum Slope Estimation (OSE) algorithm to deal with the great variation of RTTs in congested condition. As a result, by conducting experiments, OSE algorithm yields a fair estimation with *pathchar* in light congested link, and performs better than Pathchar in more congested and complex network.

KEYWORDS: available bandwidth estimation, optimum slope estimation, variable packet size, traffic analysis and measurement

I. INTRODUCTION

Nowadays, the Internet modeling and measurement has been carried on world widely not only because the traffic volume of existing Internet is increasing exponentially but also because the Internetwork is becoming more and more complex. Available bandwidth estimation is probably the most underlying measurement effort among them. In this scenario, many tools are developed to measure the network bandwidth metrics, such as capacity, available bandwidth, and throughput. However, the existing tools or methodologies for measuring the network bandwidth are mostly inadaptable to real networks and yields incorrect estimation.

In this project, we propose a more powerful method to measure the available bandwidth from any source to the destination. With the help of existing tool—high resolution PING, We develop a python application to obtain the **shortest-observed round-trip times** (SORTTs) of **variable packet sizes** (VPSs). Based on this data, we designed and implemented an effective algorithm—**optimum slope estimation** (OSE) to determine the optimum $\Delta \mathrm{RTT} / \Delta \mathrm{PacketSize}$ so as to estimate the maximum available bandwidth from local machine to any remote destination. The detailed theory is described in detail in chapter III.

By using the python script and Matlab implementation of OSE, we conducted experiments to testify the performance of OSE. We compared the result between Pathchar and Our application in two cases: 1) High speed link with the source and destination in the same wired LAN 2) Slower link with the source and destination in the different LANs. The results show that OSE's performance is as good as Pathchar in case 1. What's more, OSE is more robust and accurate than Pathchar in case 2.

The remainder of the report is structured as follows. Chapter II provides an overview of the previous research on bandwidth metrics measurements on various kinds of networks. Chapter III introduces the methodologies that our application is using and proposing a new algorithm—OSE to estimate the available bandwidth from local machine to remote destination. The experiment procedures and results are presented in Chapter IV. Finally, conclusions are drawn in Chapter V.

II. Related Work

Recently, many bandwidth-metric estimation methods have been proposed and implemented. Based on these works, many bandwidth estimation tools, such as pathchar, pipechar, pathrate and pathchirp, have been developed and examined in various scales. Pathchar develop by Jacobson, which uses Variable Packet Size (VPS) technique to obtain a good estimation of the link capacity of an end-to-end path in experiment environment is discussed [1] in terms of methodology, performance and accuracy. In [2], Jacobson and Karels initialized another bandwidth measurement technique which is Packet Train Dispersion (PTD). [3], [4] explain and test PTD by implementing tools, and they gave valuable experimental data. In addition, many comparative researches have been conducted recently on comparing different bandwidth measurement algorithms or tools. In [5], the authors conducted comprehensive comparison between pathchirp, pathload and Spruce, verifying their accuracy, measuring time, intrusiveness and so on. Sairam and Barua surveyed on existing bandwidth estimation tools including bing, pathchar, clink in [6], by digging the principles, properties and performance of different tools. Prasad etc in [7] conducted research more thoroughly on the different bandwidth metrics measurement methodologies, concluding valuable suggestions on existing tools and methodologies. CAIDA has also been running a project on the basis of existing bandwidth estimation endeavors, aiming to pilot the bandwidth measurement technology for modern Internet [8]. What's more, some schemes have also been proposed in dealing with bandwidth estimation in more complex networks, such as wireless networks. For example, [9] and [10] proposed new bandwidth measurement schemes so as to accommodate the unique characteristics of wireless network, which is more complicated than wired network.

All these work are valuable and creative. However, what makes our work distinguish is that, we are going to exploit the optimum estimation of available bandwidth using a simple but innovative, efficient algorithm with less cost than previous methods.

III. Measurement Methodology

A. Background

In order to understanding the methodologies that we are going to implement, first we define the capacity C and the available bandwidth A for an end-to-end path. According

to CAIDA's definition, Capacity is the maximum throughput that a path can provide to a flow when there is no competing traffic load; Available bandwidth, on the other hand, is the maximum throughput that the path can provide to a flow, given the path's existing competing traffic load [8]. The formula of C and A of a path could be defined as follows, where N is the number of hops along the path, C_i is the capacity or transmission rate of link i and $C_i(1-\mu_i)$ is the unused capacity in link i.

$$C = \min_{i=0,\ldots,N} C_i \tag{1}$$

$$A = \min_{i=0,\ldots,N} \left(C_i(1-\mu_i)\right) \tag{2}$$

Therefore, the capacity from local machine to remote destination is limited by the bottleneck of the path, which usually is the LAN where the local/remote machines lie in. And the available bandwidth is the maximum transmission rate that I can achieve under the utilization of other traffic μ. In our project, we are going to estimate the available bandwidth of an end-to-end path in a real network.

There are mainly two different bandwidth measurement methodologies: Variable Packet Size (VPS) probing and Packet Train Dispersion (PTD) probing. VPS probing is proposed by Jacobson [11], measuring the hop-by-hop link capacities. PTD probing, originated by Jacobson as well in [2], aims at measuring the capacity of end-to-end link under the circumstance of congestion, by applying complicated variation method, statistical filtering techniques. In our project, particularly, we took advantage of VPS probing method because VPS is much easier to implement than PTD. In the following Part, we are going to discuss the methodologies and techniques that we used in details.

B. Variable Packet Size Probing

Same with Pathchar, our application applies VPS probing method as well. Pathchar is powerful tool to estimate the capacity of intermediary links of an end-to-end path. It takes advantage of ICMP echo request with increased Time-to-Live (TTL) from 1, 2, …, N until it reaches the destination. After each hop the TTL will decrease by 1. When the TTL = 0, the router will respond an error packet indicating the ICMP request is "Time-exceed". This is also how traceroute works.

Unlike Pathchar, our python application implements the ICMP echo request like PING rather than traceroute, without setting the TTL incrementally. What's more, we took advantage of the high resolution time characteristics of hrPING and develop our own python application. The only difference would be that the destination would echo the same packet by only changing the type field to "ICMP reply". Therefore, the

acknowledgement packet would be ICMP echo reply packet rather than ICMP time-exceed error packet. We have the following simplified model for the target link:

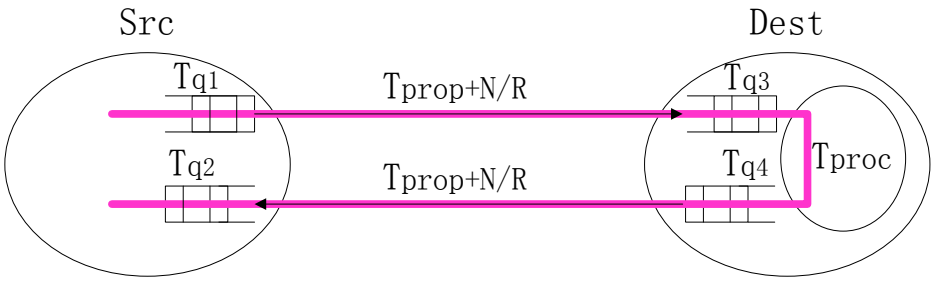


Figure 1: A Simplified Model of Our Estimation Method

According to the model in Figure 1, we have the following equation to represent the RTT:

$$\begin{aligned} RTT &= T_{q1} + \left(T_{prop} + N/R\right) + T_{q2} + T_{proc} + T_{q3} + \left(T_{prop} + N/R\right) + T_{q4} \\ &= T_{q1} + T_{q2} + T_{q3} + T_{q4} + T_{proc} + 2\left(T_{prop} + N/R\right) \end{aligned} \tag{3}$$

Where the packet size is N, the transmission rate of the link is R, T_{qx} is the queuing waiting time for hop interfaces, T_{prop} is one-way propagation delay and T_{proc} is the processing delay in forwarding machines. To simplify equation (3), we have to declare the following two assumptions:

1) By multiple probes we assume that we are able to obtain RTT without queuing delay;

2) The processing time is negligible or could be consider a constant. Here the T_{proc} represents the sum of processing times in all hops along the path.

Under the above two assumptions, we can simplify equation (3) to equation (4):

$$RTT = T_{proc} + 2\left(T_{prop} + N/R\right) \tag{4}$$

Like Pathchar, we choose to use VPS technique to estimate the bandwidth. Assuming there are two packet sizes, N_1 and N_2, we can obtain the following equations:

$$\begin{cases} RTT_1 = T_{proc} + 2\left(T_{prop} + N_1/R\right) & (5) \\ RTT_2 = T_{proc} + 2\left(T_{prop} + N_2/R\right) & (6) \end{cases}$$

Substituting (5) and (6) we obtain:

$$R = 2 \cdot \frac{N_1 - N_2}{RTT_1 - RTT_2} \tag{7}$$

which is the formula of our estimated available bandwidth formula.

C. Shortest Observed Round Trip Time

According to assumption (1), if we are able to find the shortest observed RTT (SORTT), we can fairly say that the SORTT is the RTT without queuing delay throughout the link. Thus equation (7) can be rewritten as:

$$R = 2 \cdot \frac{N_1 - N_2}{SORTT_1 - SORTT_2} = 2 \cdot \frac{1}{slope} \tag{8}$$

In other words, the available bandwidth is nothing but two times of the inverse of the slope of SORTT over Packet Size.

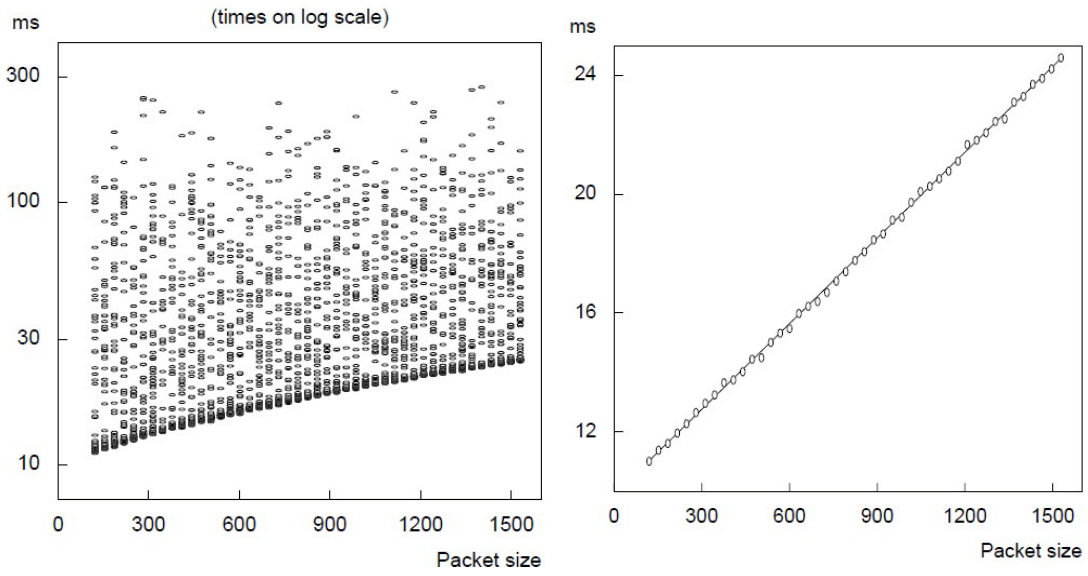
This is a little different from the Pathchar, whose formula of calculating link bandwidth is:

$$R = \frac{N_1 - N_2}{SORTT_1 - SORTT_2} = \frac{1}{slope} \tag{9}$$

because the size of ‘Time-exceed’ error packet is approximated to be ZERO.

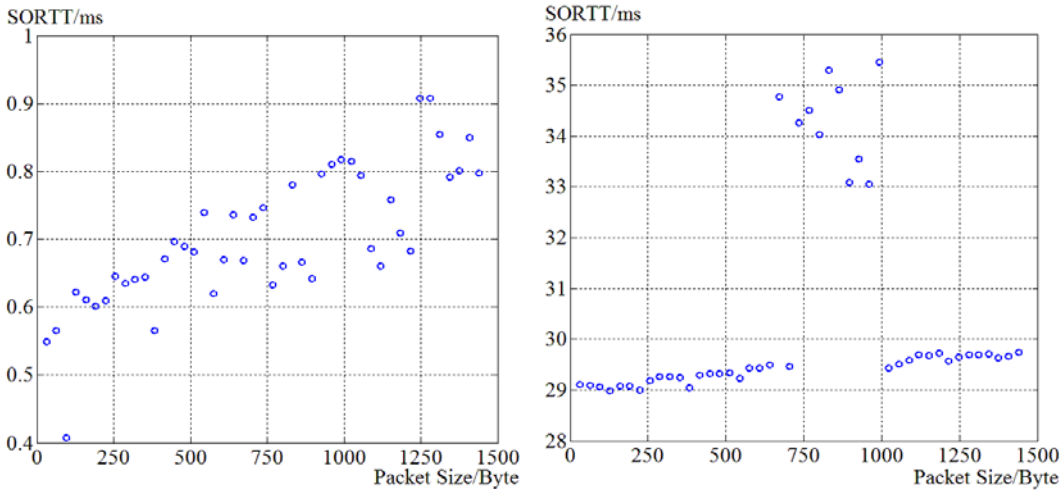
Downey gave a Scatter plot of RTTs against different packet sizes, which is shown in Figure 2(a) and SORTTs against PacketSize [1]. We can see a fine linear relationship between packet size and SORTT. However, in real network, the network congestion is not as ideal as the scenario described in the paper and the SORTT vs. PacketSize plot is not as perfect as Figure 2(b) shows. In reality, the network is usually congested so that the queuing waiting time varies greatly therefore the SORTT plot is not as perfect as linear straight line, which is shown in Figure 3(a). In this case, using the linear regression technique to find the slope is inaccurate since the variation of data samples is too large. In addition, we observed what is called “multipath” problem that causes discontinuity in the SORTT plot, which is show in Figure 3(b). Pathchar does not deal with this problem, which is one of the shortcomings of it.

What traditional method did is to calculate the regression line of the SORTT and use it to estimate the available bandwidth or link capacity. However, because of high congestion and multipath problem, using this method is no longer efficient and adaptable. In this context, we discarded the regression method and proposed a new algorithm, which is Optimum Slope Estimation (OSE) algorithm to obtain the estimated line.



(a) Scatter Plot of RTT (b) SORTT

Figure 2: SORTT vs Packet Size in Ideal Network[1]



(a) SORTT Varies Greatly (tamu.edu) (b) Multipath Problem (google.com)

Figure 3: SORTT vs. Packet Size in Real Network

D. Optimum Slope Estimation Algorithm

Motivation:

The existing Pathchar tool uses linear regression of SORTTs to calculate the slope of the line and then the available bandwidth. In theoretical case, there should be no SORTT lying below the “Optimum line”, since no RTT can be smaller than the RTT for the uncongested link. However, old method did not address the problem that regression line is actually not the optimum case for SORTTs especially when the deviation of data samples is large. Hence we propose what we call the “Optimum Slope Estimation” algorithm.

Principles:

The expected line with “Optimum slope” should be satisfied with the following two principles:

1) All SORTTs must be above the expected line.
2) If there is more than one lines satisfying 1), the line with the minimum slope is considered more likely to be the expected line than other lines that satisfying 1).

In order to describe the OSE algorithm more clearly, we now provide the pseudo code of OSE as follows:

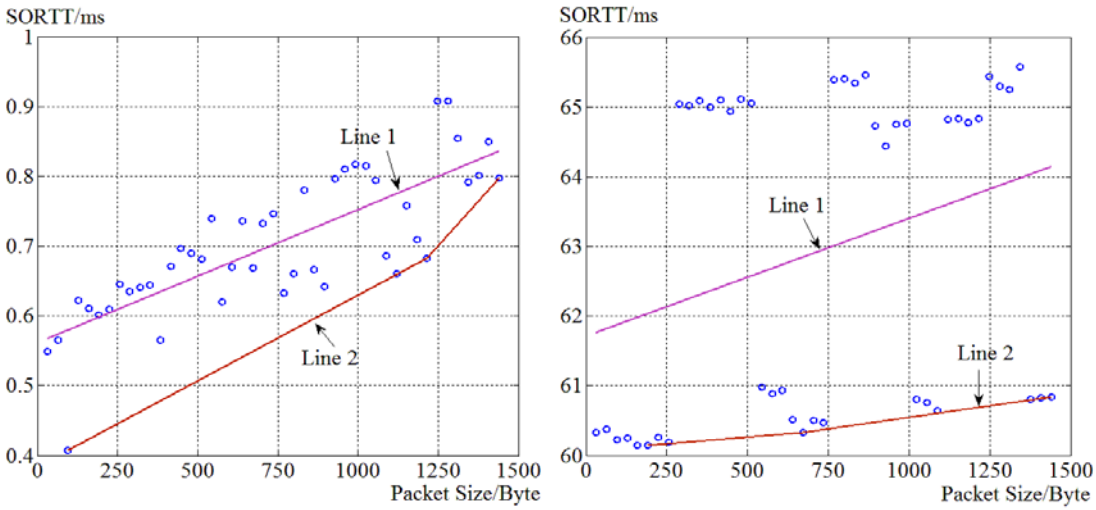
```
//Assuming there are N SORTTs
For every SORTT i {
   Find the slope between i and i+1 to N {
    //Determine the line with the minimum slope
    tmpline(i)=min_slope(line{i, i+1,…,N-i});
    //Determine whether all SORTTs are above the line
    If (TURE) {
       //Take the line as prospective one
       finalline(i)=tmpline(i);
    }
  }
}
if (multiple lines exist) {
   Expectedline = the line with the minimum slope;
}
```

The feature of OSE is to attempt to find the optimum solution in determine the $\Delta RTT/\Delta PacketSize$, which make it distinguished from traditional linear regression methods. When $\Delta RTT/\Delta PacketSize$ is larger, the estimated available bandwidth is smaller; and vice versa. Hence, OSE is exploiting the minimum $\Delta RTT/\Delta PacketSize$, the “slope”, so that we can obtain the largest expected available bandwidth. Saying there is N number of data to be trained, and there are C features of each data. In this case, C is 2. The algorithm complexity of linear regression would be like $O(4N)$.

However, obtaining of multiplicative parameter x_1 and addictive parameter x_o of linear equation would be:

$$x_1 = \frac{\sum_{i=1}^{n} t_i y_i - n \cdot \overline{t}\,\overline{y}}{\sum_{i=1}^{n} t_i^2 - n \cdot \left(\overline{t}\right)^2} \text{ and } x_o = \overline{y} - x_1 \overline{t} \tag{10}$$

Thus, the complexity would at least be $O(N^2)$. The algorithm complexity of OSE would also be $O(N^2)$ since OSE only needs one nested loop to get the expected result.



(a) tamu.edu (b) facebook.com

Figure 4: Comparison between OSE and Tradition Method

Thus theoretically OSE would be able to compete with traditional method. In addition, Figure 4 gives a more direct comparison between OSE and traditional method. In Figure 4, Line 1 is the expected line obtained by traditional method, while Line 2 is the expected line obtained by OSE. We can see that, in Figure 4(a) when traffic is congested, the first red line with three SORTTs going through is chosen to be the expected line. Scenario in figure 4(b) is definitely subjected to multiple paths. In this case, the traditional method gave a poor estimation by using linear regression. While OSE finds the red line with smaller slope, which estimates the optimum available bandwidth better.

In sum, OSE can adjust the great variation of SORTTs in congested network, and be able to find the optimum $\Delta RTT / \Delta PacketSize$ regardless of multiple paths in complex network topology.

IV. EXPERIMENTS

A. Description

In order to evaluate the effectiveness of our new algorithm, we designed two sets of experiments. One is between two nodes within one wired LAN. The other one is between two nodes far away from each other and in different LANs. We supposed the second test yields higher congestion probability and higher traffic utilization along the path and we expected OSE to perform better than Pathchar. As a benchmark for the bandwidth, we measured the ftp transfer rate in the two scenarios. Comparing the results of both the tools and the data rate we understand their performance.

By recalling equation (8) in Chapter III, for Pathchar, we have:

$$\frac{1}{R_1} = \frac{SORTT_{11} - SORTT_{12}}{N_1 - N_2} = slope_1,$$
$$\frac{1}{R_1} + \frac{1}{R_2} = \frac{SORTT_{21} - SORTT_{22}}{N_1 - N_2} = slope_2,$$
$$\ldots$$

Thus, the available bandwidth, which is 1 over the slope of last hop, it is nothing but:

$$\frac{1}{R_a} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_N} \tag{11}$$

where $R_i, i = 1, 2, \ldots, N$ is capacity of link i. In this way we can obtain the end-to-end available bandwidth according to Pathchar's hop-by-hop capacity.

B. Case One—Light Congestion

The topology of the tested network is shown in Figure 5.

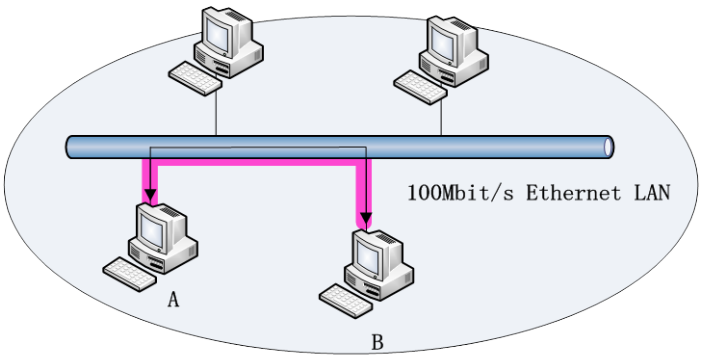


Figure 5: Network Topology in Experiment One

Table 1: Resulting Available Bandwidth in Experiment One

Ideal Bandwidth	FTP download rate	Pathchar	Our tool
100 Mbps	83.3168 Mbps	42 Mbps	41.11 Mbps

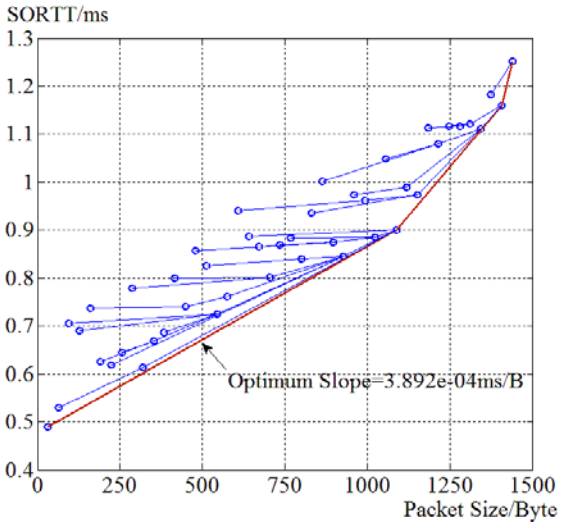


Figure 6: Estimation Result of Our Tool in Experiment One

Since the two machines are in the same Ethernet LAN, there is only one hop along the path, which is the available bandwidth of Ethernet LAN. We do the ftp downloading from A to B, and try both Pathchar and our application from machine A to B. Table 1 gives the result of our experiment. Our testing environment is not ideal because many other machines are using the same LAN at the same time. Hence, none of the trial can reach the ideal maximum of bandwidth, which is supposed to be 100Mbit/s. However, in comparison of the instantaneous available bandwidth by using FTP downloading, we can see that both Pathchar and our tool gave similar bandwidth estimation, which is about half the FTP downloading rate. Compared with the estimating given by Pathchar our result is 2% farther away from the FTP downloading rate than Pathchar's estimation, which is not a big difference. We can fairly say that our tool with new OSE method is performing as well as Pathchar in light congested network. Figure 6 shows the result of OSE on the data set of experiment one.

C. Case Two—Heavy Congestion

The network topology in experiment two is shown in Figure 7. It runs through 7 hops and all the hops are in the campus network of TAMU. In this case, the end-to-end bandwidth is limited by the bottleneck along the path, which is 54 Mbps 802.11 Wireless LAN at the end of Host A.

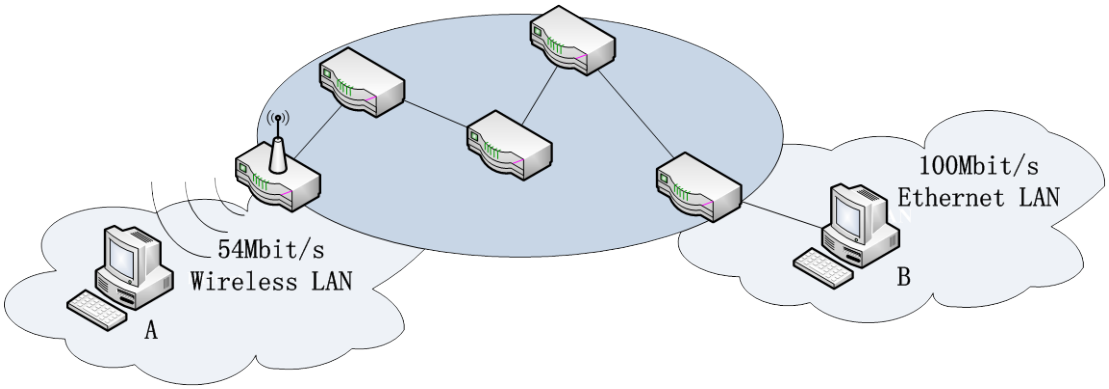


Figure 7: Network Topology in Experiment Two

When trying FTP downloading, it gave 13.342 Mbps downloading rate, which is about one fourth of the maximum bandwidth of Wireless LAN. As we can see in Table 2, our tool gave a better bandwidth estimation than Pathchar did in terms of the available bandwidth given by FTP testing. The available bandwidth estimation is 9.8% closer to the actual available bandwidth of the end-to-end path than what Pathchar gave. We believe that 1Mbps different is pretty big especially when the available bandwidth is only at around 13 Mbps. We can also see that there are some abnormal SORTT samples in Figure 8, which is resulted from the heavy traffic within the Ethernet. It is these extremely large values that result to the lower bandwidth estimation of Pathchar.

We can expect that in business network rather than campus network, this case would be more obvious between Pathchar and our tool because commercial network suffer more congestions and multipath problems. Actually we did some experiments to outer networks such as google.com and our tool gave closer estimation of available bandwidth than Pathchar did. In sum, the results in Table 2 are consistent with what we expected.

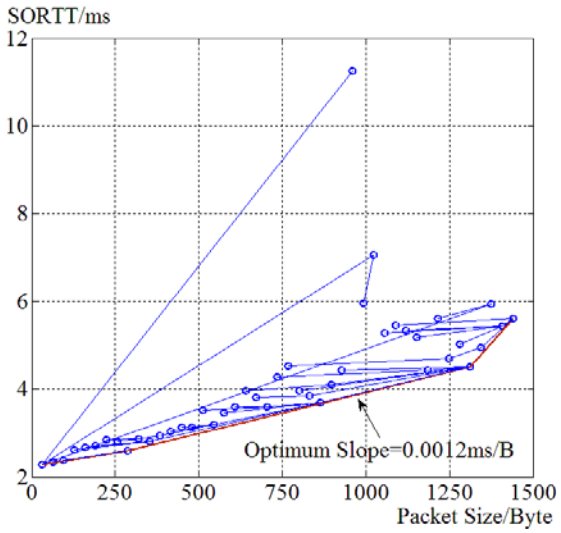


Figure 8: Estimation Result of Our Tool in Experiment Two

Table 2: Resulting Available Bandwidth in Experiment Two

Bottleneck Bandwidth	FTP download rate	Pathchar	Our tool
54 Mbps	13.342 Mbps	12.115 Mbps	13.299 Mbps

D. Performances

According to the above experiments, we can see that, OSE method is performing as well as Pathchar. More importantly, OSE method wins over traditional method in dealing with high congested/slower link network.

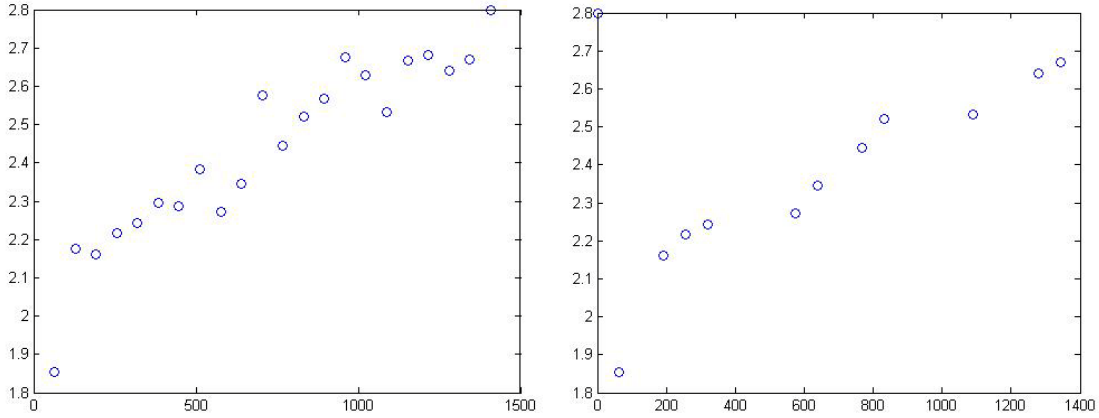
At the same time, it yields a good estimation of end-to-end available bandwidth under high traffic loads as Pathchar, and even better. The advantage of OSE could be summarized as the following points.

- OSE can better adjust the scatters of SORRTs when the network seems to be more congested. Usually we have higher estimation of the available bandwidth than Pathchar because we are exploring a line covering the smallest SORTTs.
- OSE filters out the irrelevant SORTT points calculated when the ping message travels by a different path. Thus, OSE always gives the optimum available bandwidth corresponding to the shortest path, while traditional regression method gives only the average of both the paths. Hence OSE solves the "Multi-path problem" as seen in figure 3(b) and 4(b).

E. Observations

1. One of the reasons suggested for poor estimation of the high bandwidth link stated in [1] was the precision of RTT measurement. Since at 100 Mbps, RTTs are usually generally small, their slopes are even smaller and the inaccuracy in minimum filtering and slope differencing are higher. We observed two points to disprove this.
   - **Montecarlo simulation** to allow all possible values to compensate for 0.001 ms precision, only improved the BW estimation by 0.1 Mbps when we were estimating 100 Mbps link and the estimation of both the tools was only 41.1 Mbps.
   - If the poor precision is indeed the cause of poor estimation, it expects a high standard deviation in the BW measurement. But the results we got prove otherwise.
2. In our matlab simulation, we implement one nested loop to obtain the optimum slope. Hence the computation complexity would be $O(N^2)$. To improve the computational efficiency of the OSE algorithm, we propose another technique to prefilter out the SORRTs that are definitely due to congestion before applying OSE to the set of RTTs. We filter out the congested SORTTs so as to make all

SORTTs monotonic increasing. The new order would be $O(N+ C^2)$ where C is the number of filtered samples and C<<N, which is possible an improvement to $O(N^2)$ in many cases. Figure 9 illustrate our modification.



(a) Before pre-filtering (b) After pre-filtering

Figure 9: Monotonic Increasing Pre-filtering

V. CONCLUSION AND FUTURE WORK

In conclusion, by developing a Python script application using our new OSE algorithm, we conducted comprehensive experiments to test and validate the performance of OSE compared with Pathchar. As a result, we have found that OSE yields competitive estimation of end-to-end bandwidth with Pathchar and outweigh Pathchar by giving more accurate estimation when the network is more congested. We also discussed the performance of OSE in real networks. In summary, our newly proposed OSE algorithm is doing a good job on the basis of VPS methodology, especially in dealing with poor network conditions in real world.

Future improvement of OSE is to exploit way to lower the algorithm complexity as we did in Chapter IV Part E. What's more, we are going to do more experiments, especially in more complicated commercial networks, including wireless networks to test our method. Actually we have done experiments from campus to outer network (google.com etc.). Our application yields much larger available bandwidth than Pathchar did. The estimation result of OSE is closer to the bottleneck bandwidth than that given by Pathchar, proving the robustness of OSE. It would be valuable and interesting to put more future work on the improvement and validation of OSE.

REFERENCE

[1] A B Downey (1999). Using pathchar to estimate Internet link characteristics. SIGCOMM, NY, USA (1999). Pp 241-250.

[2] V Jacobson, M J Karels (1988). Congestion Avoidance and Control. SIGCOMM, NY, USA (1988). Pp 314-329.

[3] V J Ribeiro, R H Riedi, R G Baraniuk, J Navratil, L Cottrell (2003). pathChirp: Efficient Available Bandwidth Estimation for Network Paths. Passive and Active Monitoring Workshop (PAM 2003), San Diego, CA, USA.

[4] Constantinos Dovrolis (2004). Packet-Dispersion Techniques and a Capacity-Estimation Methodology. IEEE/ACM Transaction on Networking, Vol. 12, No. 6, Dec 2004. Pp 963-977.

[5] A Shriram, M Murray, Y Hyun, N Brownlee, A Broido, M Fomenkov, K Claffy (2005). Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. 6th Passive and Active Monitoring Workshop (PAM 2005), Boston, MA, USA, Mar 2005. Pp 306-320.

[6] A S Sairam, G Barua. Survey of Bandwidth Estimation Techniques. Retrievd from: http://www.iitg.ernet.in/gb/papers/iacc09_paper.pdf in 12/06/2012.

[7] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy (2003). Bandwidth estimation: metrics, measurement techniques, and tools. Network, IEEE. GIT, Atlanta, GA, USA. Vol. 17, Issue 6, Nov-Dec 2003. Pp 27-35.

[8] Retrieved from CAIDA ISMA 2003 Bandwidth Estimation Workshop BEst: http://www.caida.org/projects/bwest/index.xml in 12/01/2012.

[9] A. Kashyap, S. Ganguly, S. R. Das (2007). A Measurement-Based Approach to Modeling Link Capacity in 802.11-Based Wireless Networks. Mobile Computing and Networking 2007, ACM, NY, USA. Pp 242-253.

[10]M Li, M Claypool, R Kinicki (2008). WBest: a Bandwidth Estimation Tool for IEEE 802.11 Wireless Networks. IEEE Conference on Local Computer Networks, Montreal, Quebec, Canada. Oct 2008.

[11]V Jacobson (1997). Pathchar—a tool to infer characteristics of Internet paths. Network Research Group, Lawrence Berkeley National Laboratory, Berkeley, CA, USA. Could be founded in: ftp://ftp.kfki.hu/pub/packages/security/COAST/netutils/pathchar/msri-talk.pdf.

APPENDIX

A. Python Application

```
import sys
import subprocess
import re
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from pylab import *

#to obtain the scatter plot of the RTTs at all ping sizes
class Test():
    def __init__(self, x, y):
        fig = plt.figure(1)
        ax = fig.add_subplot(111)
        ax.plot(x, y, 'D', color='red')
        ax.set_xbound(64,1472)
        ax.set_ybound(0,400)
        plt.savefig("test1.png")

#to read the output of hrPing and calculate the SORTT
def ping_handle(size):
    #print [2]
    trace_file = open('trace_file','r')
    values = []
    fvalues = []
    for each_line in trace_file.readlines():
        #print [3]
        match = re.search(r'time=(.*)ms',each_line)
        if (match !=None):
            #print [4]
            a = match.groups()
            values.append(a[0])
    #print values
    #for i in range(0,len(values),1):
        #test1 = Test(size,values[i])
    fvalues =   map(float,values)
    #print fvalues
    #print min(fvalues)
    return min(fvalues)
```

```
sortt = []
msgsiz = []
for size in range(64,1472,32):
    subprocess.call("ping -n 64 -l "+str(size)+" 165.91.89.54 > trace_file",shell = True)          # destination is entered here
    #print [1]
    b = ping_handle(size)
    sortt.append(b)
    msgsiz.append(size)
    #print sortt
print msgsiz
print sortt
plot(msgsiz, sortt, 'D')
xlabel('size of ping msg')
ylabel('Smallest observed RTT')
title('slope gives BW, Y intercept gives latency')
grid(True)
savefig("slope.png")  #plot the SORTT vs Ping message size.
show()
```

B. Matlab Implementation of OSE

```
clear all
clc

x = [32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992, 1024, 1056, 1088, 1120, 1152, 1184, 1216, 1248, 1280, 1312, 1344, 1376, 1408, 1440];

y = [2.283, 2.34, 2.38, 2.608, 2.674, 2.697, 2.845, 2.81, 2.591, 2.863, 2.807, 2.947, 3.034, 3.139, 3.132, 3.509, 3.192, 3.467, 3.588, 3.971, 3.81, 3.592, 4.279, 4.524, 3.954, 3.847, 3.689, 4.093, 4.434, 11.246, 5.951, 7.049, 5.28, 5.452, 5.337, 5.187, 4.422, 5.602, 4.7, 5.032, 4.512, 4.945, 5.934, 5.424, 5.601];

figure, plot(x,y, 'o','linewidth',1.5);

m = 100*ones(size(x,2));
Minm = zeros(1,size(x,2));
finalline = zeros(1,size(x,2));
npoints = zeros(1,size(x,2));
tempn = zeros(1,size(x,2));

hold on
for i=1:size(x,2)-1          % for each point calculate the slope to every other line
    for j=i+1:size(x,2)
```

```
            m(i,j) = (y(j)- y(i)) / (x(j) - x(i));
            if(m(i,j) <= 0)          % discard negative slopes
                m(i,j) = 100;
            end
        end

        [Minm(i) index] = min(m(i,:));      % calculate the minimum slope line
        a = [x(i) x(index)];
        b = [y(i) y(index)];
        plot(a,b)

        [c npoints(i)] = isabove2(Minm(i), i, x, y);
        if( c == 1)
            finalline(i) = Minm(i);
            plot(a,b,'Color',[.8 .1 .1],'linewidth',1.5)
        end


end
for i=1:size(x,2)
        if(finalline(i)>0)
            tempn(i) = npoints(i);

        end
end
%calculate the available bandwidth according to eq. (7)
[Maxn ind] = max(tempn)
availbw = 16*(1/finalline(ind))
```

C. Matlab Implementation of Congested SORTTs Pre-filtering

```
clear all
clc
y = [1.854, 2.177, 2.161, 2.216, 2.243, 2.296, 2.286, 2.383, 2.272, 2.344, 2.575, 2.446,
2.52, 2.569, 2.676, 2.63, 2.533, 2.666, 2.682, 2.642, 2.671, 2.8];
x = [64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960, 1024,
1088, 1152, 1216, 1280, 1344, 1408];
figure, plot(x,y, 'o');
flag1 = 0;
while(flag1 == 0) % executed until there is no more change
   ynext = zeros(1,size(x,2));
   xnext = zeros(1,size(x,2));
   count = 1;
   for i=1:size(x,2)-1
      if( y(i+1) >= y(i)) % filter out the congested SORTTs if SORTTi>SORTTi+1
```

```
            ynext(count) = y(i);
            xnext(count) = x(i);
            count = count+1;
        end
    end
    ynext(count) = y(size(y,2));
    if( size(y,2) - count == 0)
        flag1 = 1; % flage is set when there doesnt exists SORTTi>SORTTi+1
    end
    y = ynext(1:count);
    x = xnext(1:count);
    figure, plot(xnext,ynext, 'o');
end
```