Using Simple Per-Hop Capacity Metrics to Discover Link Layer Network Topology

Shane Alcock[1], Anthony McGregor[1,2], and Richard Nelson[1]

[1] WAND Group, University of Waikato[*]
[2] NLANR Measurement and Network Analysis Group, San Diego Supercomputer Center[**]
`(spa1, tonym, richardn)@cs.waikato.ac.nz`

Abstract. At present, link layer topology discovery methodologies rely on protocols that are not universally available, such as SNMP. Such methodologies can only be applied to a subset of all possible networks. Our goal is to work towards a generic link layer topology discovery method that does not rely on SNMP. In this paper, we will present a new link layer topology discovery methodology based on variable packet size capacity estimation. We will also discuss the problems that arose from preliminary testing where different brands of network cards affected the capacity estimates used to detect serializations. As a result, topologically equivalent links fail to be classified as such by the capacity estimation tool. To circumvent this issue, non-VPS methods of capacity estimation that utilise back to back packet pairs have been investigated as a calibration technique.

1 Introduction

Most topology discovery research focuses on the network (or IP) layer, dealing with host machines and routers. At present, there are a number of tools that can successfully perform topology discovery in this capacity. By contrast, there are few effective topology discovery tools that operate at the link layer. Most existing tools utilise the Simple Network Management Protocol (SNMP) to perform link layer topology discovery [1][2][3]. This is an effective and straightforward method but it requires that SNMP agents are running on every node in the target network and that the appropriate access strings are known. This is not always possible. Other tools that operate at the link layer are manufacturer specific and, as a result, are even more restricted than SNMP-based tools. The goal is to create a methodology for generic link layer topology discovery that does not rely on SNMP.

[*] The University of Waikato Network Research Group (WAND) Measurement and Simulation project is supported by the New Zealand Foundation for Research Science and Technology under the New Economy Research Fund, Contract UOWX0206.

[**] NLANR Measurement and Network Analysis Group (NLANR/MNA) is supported by the National Science Foundation (NSF) under cooperative agreement no. ANI-0129677.

Despite the lack of tools to automate the discovery process, knowledge of a network's topology at the link layer is important. Large scale Ethernet networks linked by switches are becoming increasingly commonplace and it is difficult for network operators to accurately keep track of all the devices in their network. A link layer topology discovery tool could form part of a larger network management suite that would create and maintain an accurate picture of the network's topology. The maintenance portion of the suite would be particularly useful for spotting unauthorized devices and machines being added to the network, and for troubleshooting by identifying failing link layer devices. The discovery portion could be used to check for bottlenecks and redundancy (or a lack thereof), or to generate a map of the network, which is very difficult to do manually for a large network.

The lack of link layer topology discovery tools is due to the inherent transparency of the link layer. Link layer devices cannot be communicated with, queried, or interacted with by a remote computer (except via SNMP). This means there is no *direct* way of discovering the existence of these devices. However, this does not rule out the use of a less direct method. An indirect method would rely on the link layer devices affecting the performance of the network in a manner that is detectable and consistent.

Fortunately, one particular class of link layer device imparts a detectable effect upon any link in which such devices are present. Prasad, Dovrolis, and Mah's paper [4] on the effect of store and forward switches on variable packet size (VPS) capacity estimation tools showed that switches caused VPS capacity estimation tools to consistently underestimate link capacity. This effect can be used to detect not only the presence of store and forward switches, but also the quantity of switches and their capacities in the measured link. This effect is limited to store and forward devices so the focus of the remainder of this paper will be on this particular class of devices. In modern Ethernet networks, store and forward switches are the most common type of link layer device so limiting the scope to switches only is not unreasonable. Other link layer devices, such as hubs, require a different method to discover and are the subject of future research.

Initial testing showed that link layer topology can be inferred from VPS capacity estimates, but it also introduced a more practical difficulty with the methodology. Different brands of network interface cards cause VPS capacity estimation tools to provide different estimates for otherwise equivalent links. This is due to each brand of card having a slightly different processing delay for differently sized packets, which appears as a variation in the initial serialization delay at both ends of the link. To alleviate this problem, we have attempted to use non-VPS capacity estimation techniques as a means of calibrating the topology discovery system so that the variation in delay at the network cards is factored out.

The paper is organised as follows. Section 2 introduces variable packet size capacity estimation and describes the underlying theories and techniques associated with it. The effect of switches on VPS tools and how that effect can be used to generate link layer topology information will also be discussed. Section 3 presents the results of putting the theory into practice. The practical problems that arose from testing are described in Sect. 4 and the solutions we have investigated are detailed in Sect. 5. Section 6 concludes the paper with a discussion of the current state of the methodology and future work in the area of link layer topology discovery.

2 Variable Packet Size Capacity Estimation

Variable packet size (VPS) capacity estimation techniques utilize the relationship between packet size, serialization delay, and capacity. The basic premise is that serialization delay is proportional to the size of the packet being sent. The larger the packet, the longer the serialization delay. The capacity of a link is the rate at which bits can be inserted onto the physical medium and, hence, is directly related to serialization delay. By measuring round trip times for different sized packets, it is possible to calculate the ratio of change in packet size to change in serialization delay. This ratio will describe the capacity of the measured link.

A potential problem with the VPS method is the possibility of other delays, such as queuing, increasing the round trip time by a significant amount. The potential effects of queuing are presented in Fig. 1. Any queuing is going to result in a round trip time measurement that is not solely affected by serialization delay. As such, an accurate capacity estimate cannot be made based on such skewed round trip time measurements. To alleviate this, for each packet size numerous packets are sent and the minimum round trip time is assumed to have been unaffected by queuing or other delays. This technique has been standard in VPS capacity estimation since *pathchar* [5]. VPS capacity estimation tools typically allow the user to specify the number of packets to be sent for each packet size. In situations where it is difficult to observe a minimum RTT, the number of probes may be increased to compensate.

Tools that use the VPS methodology to generate capacity estimates include Van Jacobson's *pathchar* [5], Mah's *pchar* [6], and Downey's *clink* [7]. Each tool uses the same basic algorithm. A packet size is selected at random from a series of possible packet sizes. A packet of that size that will generate a response from the destination machine is created and sent. The round trip time for the packet is then recorded. Once a packet from each possible size has been sent a certain number of times (this is usually specified by the user), the minimum round trip time is calculated for each packet size. Using linear regression, the gradient of a line that shows packet size versus round trip time can be calculated. Inverting that gradient will give the estimated capacity of the link.

One major flaw with variable packet size capacity estimation is the fact that such a method will significantly underestimate the capacity of any link that contains store and forward link layer devices. As described in a paper by Prasad, Dovrolis, and Mah [4], this effect is due to each store and forward device adding an extra serialization into the link that is not accounted for by the capacity estimation tool (see Fig. 2). VPS tools use the TTL field of a packet to determine

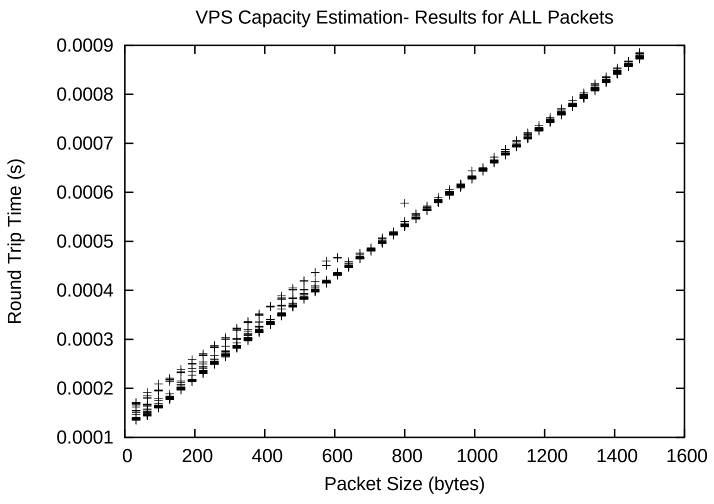

Fig. 1. This graph shows the complete results of a typical *pchar* probe of a link. 32 packets were sent at each packet size and each packet size was 32 bytes apart. Note the differences in round trip times between equally sized packets, often around half a millisecond for many of the smaller packet sizes. These differences are often caused by queuing. Taking the minimum round trip time at each packet size produces a smooth straight line which describes the capacity of the link

the number of hops in a link but link layer devices do not decrement the TTL counter due to their transparency. From the perspective of the VPS tool, each hop only contains one serialization, regardless of how many switches might be present. However, the round trip time is multiplied by the number of extra unnoticed serializations, making the link appear a lot slower than it really is.

For example, a link between two machines contains two store and forward switches. Both the switches and the Ethernet adaptors on the machines are operating at the same capacity. This link contains three serialization delays: one at the originating host, and one for each switch. Each serialization delay increases the round trip time of a packet sent across the link. Hence, the round trip time for the link is approximately three times what it would be if there were no switches in the link. This makes the gradient of the packet size versus round trip time line three times what it would normally be. This gradient, when inverted, would produce a capacity estimate one-third the value of the nominal capacity.

Although, this underestimation effect is a problem for capacity estimation, it provides information regarding the presence of store and forward devices. If the nominal capacity is known (or accurately estimated by another method) prior to calculating a VPS capacity estimate, the degree of underestimation can be used to infer the number of extra serializations and, as a result, the number of link layer devices, within the link. The equation to convert a capacity estimate

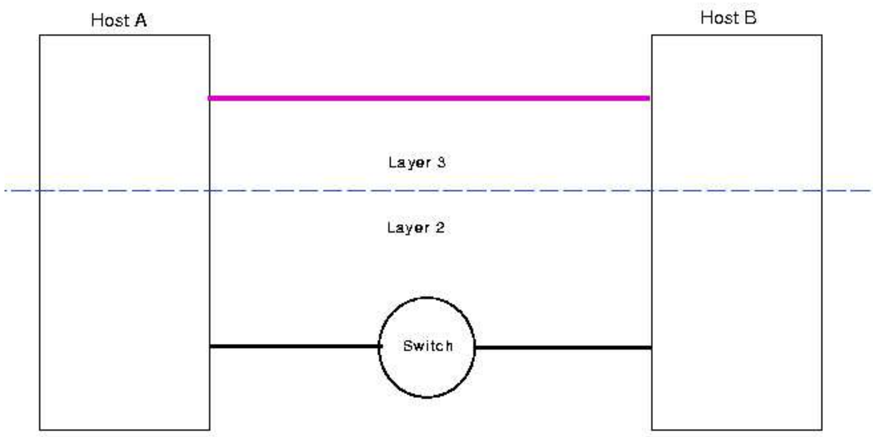

Fig. 2. A demonstration of the effect of a switch on a link between two hosts. At the network layer, it appears that the hosts are directly connected. At the link layer, a switch basically splits the hop into two, creating an extra serialization delay. Since VPS tools operate at the network layer, they do not take that serialization into account when making capacity estimates, resulting in underestimation

into a quantity of serializations is:

$$\text{Serializations} = \frac{\text{nominal capacity}}{\text{estimated capacity}} \tag{1}$$

Using the above equation gives the number of serializations *including* the original serialization at the sending host. This equation works best when all the devices are operating at the same capacity. If some of the serialization delays are of different lengths, it is no longer a simple case of comparing the VPS estimate to the nominal capacity. Fortunately, most Ethernet switches have capacities that make it easier to detect the differing serialization delays.

The serialization delay of a 10 Mbps device is ten times that of a 100 Mbps device. This relationship also holds between 100 Mbps devices and 1 Gbps devices. As it is unlikely that a single hop will contain ten switches of the same capacity, it is reasonable to assume that every ten serializations suggested by a VPS estimate are actually a single serialization for a lower capacity device. By doing this, it is possible to quantify the capacities of the switches in the link, as well as the number of switches. For example, if a VPS estimate suggests that a link contains 11 100 Mbps serializations, it is more likely that the link contains one 100 Mbps serialization and a single 10 Mbps serialization. Such assumptions work in practice because the capacities of Ethernet devices all differ by a factor of 10. This would not be as straightforward if devices had capacities of 5 Mbps, 14 Mbps, 37 Mbps, 96 Mbps, and 112 Mbps, as opposed to Ethernet where 10 Mbps, 100 Mbps and 1 Gbps are the most common device capacities. The assumption that there will be no more than ten switches in any given network layer hop does not always hold true in practice. In such cases, it may be possible to use contextual information such as topology information regarding neighbouring links or prior knowledge of the network layout to correctly classify links with many switches.

Using the method described above, it is possible to create a tool that probes links using a VPS capacity estimation algorithm, infers the number of serializations in the link using the above equation and calculates the number of switches present in those links. Rather than adapt an existing VPS capacity estimation utility, such as *pchar*, a Python implementation of the VPS algorithm called *pychar* was written to perform the probing of links. This has the advantage of providing a tool that supports easy modification and expansion to suit the specific purposes of this project. *pychar* is also designed to be integrated into a future topology discovery suite in an efficient and straightforward manner. *pychar* is capable of using either ICMP or UDP to perform the probing, at least one of which should be available on any Ethernet network.

3 *pychar* Results

Using the WAND emulation network [8], the performance of the *pychar* tool and the validity of the underlying theory has been tested. The test network (see Fig. 3) consists of seven host machines, all running Linux 2.4.20. Three of the host machines are using Mikrotik 4 port Intel Pro100 Ethernet cards, while the remaining four are using single port DSE Realtek 8139 based Ethernet cards. All the cards are operating at 100 Mbps. The machines are connected via three Gigabyte brand 5-port mini switches which are also operating at 100 Mbps. Hence, all the links have a nominal capacity of 100 Mbps. Each machine is within a single transport layer hop of each other, but the number of switches in each link is between one and three. There are no hosts directly connected without at least one switch between them. It is significant that all the devices at both the link and network layers are operating at the same capacity, creating a straightforward situation for initial testing. There is very little traffic operating on this network at any given time, making it easy to gather minimum round trip time data that is free of queuing delays.

Table 1 contains the results of sending probes from Machine 1 to all the other machines in the test network. What these results show is that the basic theory does prove to be correct in practice and that it is possible to approximate the number of switches in any of the links using the technique described earlier. The serialization estimates are not perfect, as there is a fractional component to each of the estimates seen in the table. This is not an issue in this case, as there is prior knowledge that all the devices are operating at the same capacity so any fractional remainders can be ignored. In an environment where device capacities are more varied, this could become a problem. This situation will be discussed in Sect. 4.

It is also important that topologically equivalent links, i.e. links that contain the same number of serializations, produce equivalent capacity estimates when

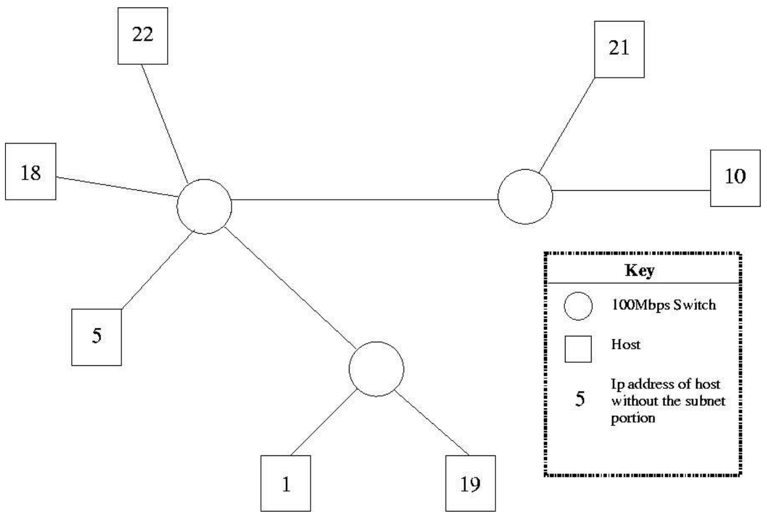

Fig. 3. The topology of the test network

Table 1. The results of sending *pychar* probes from Machine 1 to all the other machines in the test network. When rounded to the nearest whole number, the estimated number of serializations is equal to the actual number of serializations. This is ideal in situations where all the serialization delays are the same length.

Destination Machine	Estimated Capacity (Kbps)	Nominal Capacity (Kbps)	Estimated Serializations	Actual Serializations
5	31704	100000	3.15	3
10	24124	100000	4.15	4
18	30708	100000	3.26	3
19	44640	100000	2.24	2
21	23540	100000	4.25	4
22	30764	100000	3.25	3

probed by *pychar*. Table 2 presents *pychar* estimates for a number of different two switch links in the test network. The results show that topologically equivalent links are producing different capacity estimates when probed by *pychar*. A graphical view of this situation is presented in Fig. 4. Again, this is not an issue in this particular case as it is known that all the devices are operating at the same capacity. However, this problem will need to be dealt with to successfully create a generic link layer topology discovery tool.

Table 2. *pychar* estimates for a selection of the two switch links in the test network. The "From" column displays the machine number of the sending host and the "To" column describes the receiving host. The letter in parentheses beside each machine number represents the type of network card used by that host. M represents a Mikrotik card whereas R represents a Realtek card. The rightmost column contains estimates calculated where DAG cards were placed at both ends of the link to perform the timing. This is discussed more in Sect. 5.

From	To	Standard *pychar* Estimate (Kbps)	*pychar* Estimate Using DAG Cards (Kbps)
1 (M)	5 (M)	31726	49970
5 (M)	1 (M)	31860	50023
10 (M)	5 (M)	31689	50023
1 (M)	18 (R)	30721	50010
10 (M)	18 (R)	30707	50036
18 (R)	21 (R)	29786	49984
21 (R)	18 (R)	29738	49931

4 *pychar* Problems

One problem with variable packet size capacity estimation is that the estimates produced are not exactly what one might expect for any given link. For example, given the theory of underestimation due to extra serializations presented earlier in this paper, it is expected that a link with a nominal capacity of 100000 Kbps that contains three serializations would produce a capacity estimate of 33333 Kbps (one third of the nominal capacity). Looking at Table 1, it is apparent that the three links that match this profile give slightly different estimates - 31704 Kbps, 30708 Kbps, and 30764 Kbps respectively.

This becomes significant when there is the possibility of higher capacity devices being present in the link. A 1 Gbps serialization is one-tenth that of a 100 Mbps device. This means that a serialization quantity estimate of 3.25 for a link with a nominal capacity of 100 Mbps, as seen in Table 1, not only suggests three 100 Mbps devices, but also two 1 Gbps devices (and another five 10 Gbps devices, if one has reason to believe devices of such a capacity might be present). As a result, the slight difference between the ideal estimate and the actual estimate can cause *pychar* to erroneously detect high capacity devices. In situations

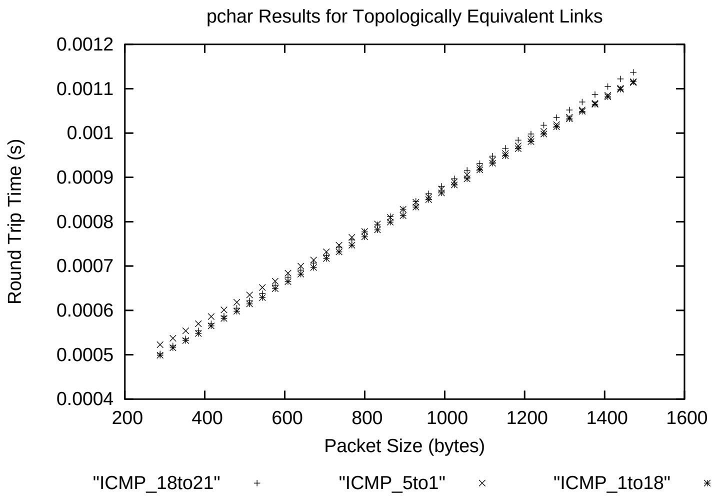

Fig. 4. A graphical illustration of different estimates provided by *pychar* for topologically equivalent links. Each link depicted in the graph contains two switches but the slopes of the lines are different, resulting in different capacity estimates

similar to the test network, where there is prior knowledge that all the devices are the same capacity, this is not a problem and the fractional component can be rounded off and ignored. However, most topology discovery takes place in an *unknown* environment so such an assumption cannot be made.

Table 2 highlights another problem that arose from the initial testing of *pychar*. Some links, despite having the same number of switches present, produce capacity estimates that vary. The links can be divided into three groups based on the capacity estimate given by *pychar*. One group contains links from a host with a Mikrotik Ethernet card to another host with a Mikrotik Ethernet card, one contains links from a host with a Realtek Ethernet card to a host with another Realtek card, and the final group consists of links that have a Realtek card at one end and a Mikrotik card at the other. In the latter case, it does not matter which is the sending host. The important factor is that there are two different cards involved.

This effect is explained by the notion that different brands of network interface card take different amounts of time to put the packet onto the wire, depending on the size of the packet. Equally sized packets have slightly different round trip times depending on the network interface cards involved in the link. For Mikrotik card to Mikrotik card links, smaller sized packets have comparatively longer round trip times. Similarly, larger sized packets have longer round trip times on Realtek to Realtek links. As a result, the variation in gradients as seen in Fig. 4 occurs. This variation in slope translates into a variation in capacity estimate. There can be a difference in excess of 1 Mbps between two estimates for topologically equivalent links, which can be enough to suggest the presence of an extra 1 Gbps serialization. The uncertainty is great enough that links that are identical from a topological standpoint can be classified as different by *pychar* simply due to different network interface cards in the end nodes.

5 Possible Solutions

The difficulty in trying to eliminate the variation in capacity estimates for topologically equivalent links is that the variation in minimum round trip times for any given packet size is very small. For example, the difference between minimum RTTs for a 288 byte packet on the two switch links described in Fig. 4 is only 20 microseconds. This is the difference in processing time for the packet on two different Ethernet adaptors. The difference in RTTs is consistent across multiple tests so the problem cannot be resolved by increasing the number of probes. Instead, solutions that eliminate the processing time of the network cards from the serialization estimates must be considered.

The first possible solution investigated involves using DAG passive measurement cards to capture timestamps immediately after the packets have been transmitted. Designed to capture and record details of every packet passing through them, DAG cards are GPS synchronized network monitoring cards capable of timestamping at a better than 100 nanosecond resolution [9]. By placing DAG cards at each end of a link, the network card serializations can be bypassed, eliminating the variability that causes topologically equivalent links to produce different capacity estimates under *pychar*. DAG cards also offer more precise timing than the operating system based timing used by *pychar*.

If two hosts, X and Y, are both connected to DAG cards, as seen in Fig. 5, and a packet is sent between them, the outgoing timestamp is generated at DAG 1 rather than at Host X. When the packet reaches DAG 2, an incoming timestamp is generated. Host Y sends a response packet back to Host X which is timestamped again as it passes through each of the DAG cards on the way back. If the difference between the DAG 2 timestamps (the turnaround time at Host Y) is subtracted from the difference between the DAG 1 timestamps, which is effectively the round trip time minus the initial serialization delay, the result is the round trip time for the packet travelling from DAG 1 to DAG 2. The hosts are removed from the round trip time calculation without altering the link in any significant way.

The results of probing some two switch links using dual DAG cards are presented in the rightmost column in Table 2. Instead of the 1 Mbps difference between estimates for links that had different network interface card configurations, the difference is reduced to less than 100 Kbps. Hence, using DAG cards reduces the problems caused by different network cards to a negligible level. Applying this solution requires that there be a DAG card connected to both ends of every probed link. It is neither practical nor economic to deploy DAG cards on every host in a network of non-trivial size. However, deploying a single DAG card on the host that would be initiating all the *pychar* probes remains practical. This will allow for much more accurate and consistent capacity estimation

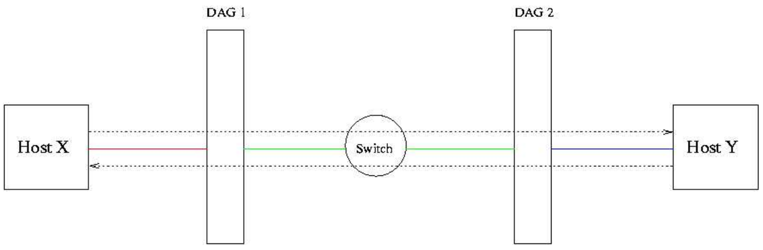

Fig. 5. This diagram demonstrates how DAG cards can be used to eliminate the effects of the network interface cards. Connecting both ends of a link to DAG cards means that timestamps can be captured after the initial serialization delays, removing any hardware variability from the round trip time measurement

due to the increased timing precision. It will also eliminate any variation at the sending end of the link, meaning that only the receiving network card will affect the *pychar* estimates. This means that there will only be two groups of links in the test network instead of three: links that have a Mikrotik card at the receiving end and links that have a Realtek card at the receiving end.

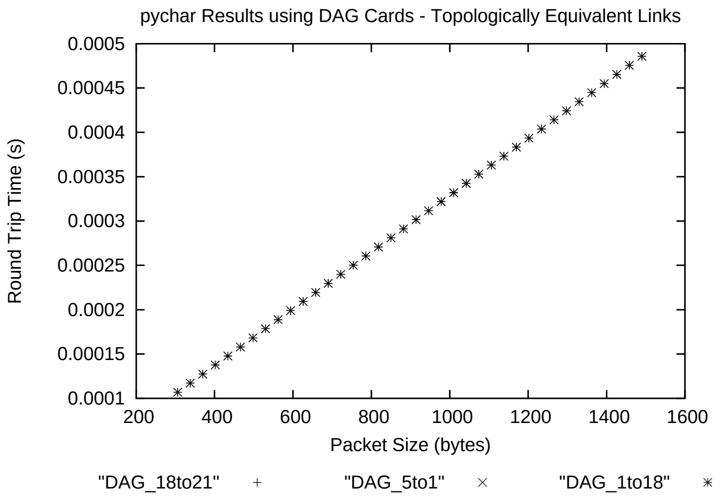

Fig. 6. A graphical illustration of how placing DAG cards at each end of a link can eliminate the variability in *pychar* estimates. The links depicted above are the same links seen in Fig. 4. On this occasion, each link produces exactly the same slope and, as a result, exactly the same capacity estimate

Another approach to factoring out the effects of different network interface cards involves the use of a different type of probe prior to running *pychar* to calibrate the system and provide a reference point that includes the adverse effect of the Ethernet card present at the receiving end of the link. To do this, an estimation technique that does not underestimate capacity due to the presence of switches but is affected by the network interface card is needed to provide a base estimate. This base estimate will act as a replacement for the nominal capacity required for the serialization calculation described above. As a result, this would also remove the need for the nominal capacity of the link to be known in advance.

Capacity estimation techniques that use back to back packets to estimate link capacity are typically unaffected by the presence of switches in the link [10]. Rather than using round trip times as the metric upon which the capacity estimates are based, these techniques involve sending two response-seeking packets back to back to the target machine and using the distance between the two response packets to calculate a estimate of link capacity. Similar to VPS capacity estimation techniques, back to back packet methods generate their estimate by inverting the slope of the packet size versus the distance between response packets line. Different back to back packet techniques apply the varying packet size differently.

The three back to back packet methods that will be investigated are packet pairing, packet tailgating and a leading packets method. Packet pairing involves sending two packets of the same size back to back across a link. The idea is that the trailing packet will queue behind the first packet at the link bottleneck but will not be able to catch up any further, so that upon arrival the gap between the two packets is equal to the serialization delay at the bottleneck. Under packet tailgating, the packet doublet consists of a packet that will vary in size and a trailing packet that is always very small in size. By contrast, a leading packet doublet is made up of a small fixed size leading packet and a variable size trailing packet. Unlike packet pairing and packet tailgating, the leading packets method is not a recognised capacity estimation technique and is a variation on packet tailgating created for the purposes of this project. The numbers produced by this method should be not be seen as capacity estimates. However, the leading packets method may produce results that describe characteristics of the link that the other two methods do not.

A prototype Linux kernel module has been written to enable experimentation with back to back packet theories. A kernel module was used because initial testing with a Python script has shown that a user space application requires too much overhead when sending packets, making it virtually impossible to send packets in an optimal back to back manner. Specifically, the aim is to investigate if there is any useful relationship between estimates produced by back to back packet methods and estimates produced by VPS methods that allows the variable effects of network cards to be factored out. Packet size selection is performed in the same manner as in *pychar*: random selection without replacement from a list of possible sizes. In the case of packet pairing, both packets are created to be the selected size. Under packet tailgating and the leading packets method, the first and trailing packets are set to be the selected size, respectively. To prevent inaccuracies due to queuing, each packet size is used multiple times, as in *pychar*, and the minimum distance between packets is used as the metric for each packet size.

6 Future Work and Conclusion

None of the back to back methods described in the previous section have proven effective in eliminating the turnaround time at the receiving end of the link. Although there is not enough space for a detailed discussion of the results produced by those techniques here, they were either too inconsistent or failed to exhibit the effects of the receiving network card. As a result, the major problem with this technique remains unresolved. Until it is, it is not possible to produce a generic link layer topology discovery tool that uses variable packet size capacity estimation techniques to gather information about each link.

However, the progress that has been made up to this point still has some more specific uses. If a network is known to contain switches that are all the same capacity (our test network being a prime example, see Fig. 3), then simply rounding the serialization estimates will produce the correct results. Such a network is usually small enough that link layer topology discovery is not necessary but there may be some occasions where *pychar* could prove useful, especially for troubleshooting.

The results produced when using dual DAG cards show that a tool such as *pychar* can be used for link layer topology discovery in more heterogenous environments, provided the variation in Ethernet adaptor serialization delay can be factored out. While placing DAG cards at the end of every link in a network is impractical, it may still be possible to create a viable hardware-based solution. A device that can be inserted at each end of link could run *pychar* and discover the link layer topology between two such devices. Because the devices will all have the same hardware (and possibly DAG cards doing the timing) variation in network interface card is non-existent. The device can simply account for the known effects of the particular brand of card it uses, if necessary. The only drawback to such an approach is that only a single link can be dealt with at any given time, rather than an entire network. Probing multiple links will require manual movement of the devices to the appropriate endpoints.

A number of further practical issues with the *pychar*-based technique not addressed in this paper will also be the focus of future work. This includes finding a method for detecting cut-through devices such as hubs. Also, this paper has not detailed how the link layer information will be combined to create a topology map. Some rudimentary thought has been given to this problem without settling on a comprehensive solution. Finally, the emulated network that *pychar* has been tested on is very homogeneous with regard to operating systems, host hardware, and both the capacity and manufacturer of the switches. Further testing on more varied networks will be required to reveal problems similar to the network interface card problem. However, all these outstanding issues are irrelevant if the problem with differing serialization delays on the network interface cards cannot be resolved.

The next step for the VPS-based link layer topology discovery project is to investigate other measurement techniques that might be able to provide information that will allow the effects of the network interface cards to be factored out. At this stage, given the failure of back to back packet capacity estimation methods to provide such information, we do not know of any techniques that might be of use for this purpose. However, that seems the only way forward for a software-based solution that utilizes VPS capacity estimation.

Although variable packet size capacity estimation appears to be a viable tool for inferring link layer topology information, it is susceptible to the effects of different varieties of network interface cards. Back to back packet capacity estimation techniques have proven ineffective in factoring out these effects. As a result, although *pychar* can provide link layer topology information under specific conditions, it is currently not a viable link layer topology discovery technique in a generic environment.

References

1. D. T. Stott, "Layer-2 Path Discovery Using Spanning Tree MIBs," Avaya Labs Research, Avaya Inc, March 7 2002.
2. Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi, "Physical Topology Discovery in Large Multi-Subnet Networks," IEEE Infocom 2003.
3. B. Lowekamp, D. R. O'Hallaron, and T. R. Gross, "Topology Discovery for Large Ethernet Networks," in Proceedings of ACM SIGCOMM, San Diego, California, Aug. 2001.
4. R. S. Prasad, C. Dovrolis, and B. A. Mah, "The Effect of Layer-2 Store-and-Forward Devices on Per-Hop Capacity Estimation," http://citeseer.ist.psu.edu/prasad03effect.html/, 2003.
5. V. Jacobson, "Pathchar: A Tool to Infer Characteristics of Internet Paths," ftp://ftp.ee.lbl.gov/pathchar/, April 1997.
6. B. A. Mah, "pchar: a Tool for Measuring Internet Path Characteristics," http://www.employees.org/~bmah/Software/pchar/, February 1999.
7. A. Downey, "clink: a Tool for Estimating Internet Link Characteristics," http://allendowney.com/research/clink/, 1999.
8. http://www.wand.net.nz/~bcj3/emulation/
9. Endace Measurement Systems, http://www.endace.com/networkMCards.htm
10. K. Lai and M. Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay," SIGCOMM 2000, pp. 283-294.