How to Identify the Speed Limiting Factor of a TCP Flow

Mark Timmer
University of Twente
Email: m.timmer@student.utwente.nl

Pieter-Tjerk de Boer
University of Twente
Email: ptdeboer@cs.utwente.nl

Aiko Pras
University of Twente
Email: pras@cs.utwente.nl

Abstract—**Despite the end of the Internet bubble, operators continue to increase the capacity of their networks. The question now rises whether these improvements still result in faster communications, or whether most flows are limited by other aspects. In order to answer this question, actual network traffic needs to be analyzed. Therefore, in this paper methods are proposed to identify the factors that limit the speed of TCP flows. Three main categories will be distinguished: the network, the TCP buffers and the application layer. Our proposed methods have been tested on real traces; in many cases it turned out that the network was not the limiting factor.**

I. Introduction

In the mid eighties of the previous century the first version of the Dutch research network, called SURFnet, allowed end users to communicate at speeds up to 9.6 Kbps. Now, with the introduction of the sixth version of SURFnet, users get the ability to communicate at Gigabit speed. In twenty years network capacity thus grew with a factor of a hundred thousand, which is more than the increase of CPU power or computer memory. Whereas in the past bandwidth shortage has always been a problem, the question rises if this still holds. Should bandwidth still be considered a potential performance bottleneck, or is communication speed limited by other factors? Research on extensions of TCP for long-delay paths already indicated that the size of the receive window can limit the speed of a TCP flow too [1]. An interesting question is therefore how to detect if a given flow is limited by the network, or by the receive window. Or, if a flow is not limited by these factors, are there other factors that limit the speed?

The main problem addressed in this paper is how to identify the factors that limit the speed of a TCP flow. Our goal is to find methods that determine these factors in an algorithmic manner, without human intervention. We will also apply these methods to traffic traces previously collected from SURFnet, to get an idea of whether all factors play a role in practice, or only some of them.

The structure of this paper is as follows. Section II identifies the factors that, in theory, could limit the speed of a TCP flow. Then, Section III discusses related work. Section IV develops methods that, for a given flow, detect which of these factors limits the speed in practice. Section V focuses on one of these methods and shows, in detail, how it works. Due to space constraints it is impossible to discuss all methods in detail; the interested reader is referred to [2]. To give an impression of which factors play a role in the current SURFnet, Section VI provides some results of applying our identification methods to an existing repository of Internet traffic. Conclusions and recommendations for further work are presented in Section VII.

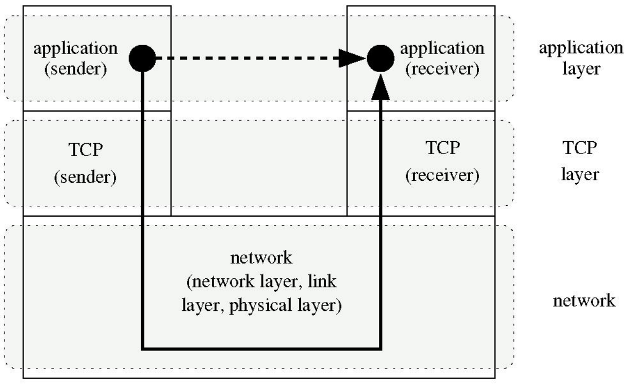


Fig. 1. TCP stack

II. Overview of the Limiting Factors

In order to determine the possible limiting factors of TCP flows (note that by a TCP flow we mean a single TCP connection identified by the IP addresses and TCP port numbers), we have looked at the major entities involved. Five entities have been identified: the sending and receiving application layer entities, the sending and receiving TCP entities and the network (Fig. 1). Each of these entities could possibly limit the speed of a TCP flow. Of course the network could be further decomposed, but for our research we are only interested in whether or not the network in general is limiting a TCP flow; which part of the network causes this limitation could be the subject of further research.

The remainder of this section covers these five entities, grouped by layer, discussing how they can limit the speed of a TCP flow.

A. The application layer

Manual examination of traffic flows revealed that there are three application layer aspects that can possibly limit the speed of a TCP flow. These three factors are discussed below.

1) The slow sender limitation: In this case the sending application doesn't provide data as fast as the network can handle. Two causes for this scenario can be thought of. First, it is plausible that a lot of these flows just don't want to transmit data any faster. One can imagine streaming data or instant messaging applications to exhibit this behaviour. Second, a lack of CPU power could be the cause.

It should be noted that a measurement device located between the sending and receiving host, may not be able to clearly differentiate between a CPU-limited application and a CPU-limited TCP stack. Although the TCP timestamp option could help identifying the source of the limitation, for simplicity this paper assumes that the slow sender limitation covers all CPU limitations at the sending side.

2) The slow receiver limitation: In this case the receiving application cannot process incoming data fast enough, resulting in an empty or diminished receive window that throttles the throughput.

Symmetric to the slow sender limitation, the slow receiver limitation includes all CPU limitations at the receiving side.

3) The application protocol limitation: In this case the application appears to have its own kind of acknowledgment or request mechanism. Some flows have been found to have acknowledgment packets that contain a small amount of data, even though these flows did not seem to be transmitting data both ways independently. Because no other phenomenon limited the speed of the flows, the only logical explanation that remained was that the application layer limited the speed, because the sending side wanted to receive an application layer acknowledgment or waited for an application layer request before it provided the TCP entity more data.

B. The TCP buffers

Inside the TCP entities two factors can limit the flow speed: the TCP buffers and the CPU. However, as mentioned in Section II-A, CPU limitations will be considered a slow sender or slow receiver limitation. Therefore, only the buffers are considered here.

At the sending side the send buffer size can limit the throughput of a flow because it must store a copy of each packet until an acknowledgment comes in, which does not happen until one round-trip time (RTT) after the transmission of the packet; thus, the throughput cannot exceed one send buffer size per RTT.

Similarly, the sender is never allowed to have more data unacknowledged than the size of the receive buffer (as given by the receive window field in the TCP header), so the throughput also cannot exceed one receive buffer size per RTT.

C. The network

When the amount of data TCP provides to the network becomes more than the bottleneck link can handle, the network will force the sender to slow down by first increasing the round-trip time as buffers fill up, and finally dropping packets.

III. Related work

Two earlier papers, namely [3] and [4], have dealt with the identification of TCP flow speed limiting factors using trace data, like we do. However, there are significant differences with our work.

In [3], 'flights' of data (corresponding to one RTT) are identified in the trace; comparison of successive flights is used to find out in which state the sending TCP's congestion control is. According to [4], these flights are hard to identify in reality; instead, in [4] average values for quantities such as the receive window are estimated at fixed time intervals, and these quantities are used as indicators for the presence of different limiting factors.

Our approach does not work with averages or flights: it draws conclusions from the individual packets. For example, we compare the receive window to the amount of outstanding data at every 'suitable' (see Section V-A) packet, rather than comparing averages of these quantities once in a while. On the one hand, this approach has the disadvantage that it only works when tracing near the sending side; on the other hand, since it works at a more detailed level, the identification could be more reliable. This approach also forces us to look into minute details of TCP implementations, in particular the block-based sending, discussed in Section V-C.3.

Unfortunately, we were not aware of [3] and [4] until a reviewer for the present paper pointed them out; as a consequence, it is not feasible to include a more thorough comparison of the approaches.

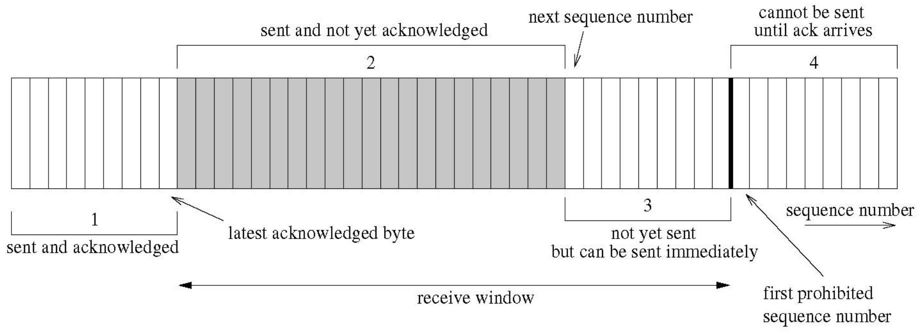
Besides the above two papers, several other papers are related to possible limitations of TCP flow speeds. In [5], it is shown that the TCP processing overhead is typically small; hence, CPU limitations most likely occur at the application layer. The fact that the TCP receive buffer size may limit the speed was identified early: [1] presents TCP protocol extensions to eliminate those limitations, and [6] discusses mechanisms for dynamic tuning of the send and receive buffer. The influence of the network through loss and delay has been modelled mathematically in several papers, including [7] and [8].

IV. Detection of limiting factors

We intend to draw conclusions about the limiting factor of a TCP flow, based on observations made by a measurement device somewhere along the path. For our detection criteria to work, these observations must include the arrival time of each packet, and the complete IP and TCP header. Some quantities needed in the sequel can be read directly from the headers, such as sequence numbers, acknowledgment numbers and the size of the receive window. Others, such as the amount of outstanding data at any moment and the achieved throughput, can be computed straightforwardly.

For simplicity, this paper only focuses on flows transmitting data in one direction (to cope with irregularities some data heading the sending side is allowed). Section IV-A.3 will show why that is useful. For the measurement data used, only 1 to 2 percent of the data passing through belonged to full-duplex flows; therefore, the simplification isn't a significant restriction.

Moreover, only flows that were measured close to the sender are considered. It appears to be much easier to develop criteria



sent and not yet acknowledged
next sequence number
cannot be sent until ack arrives
2
4
1
sent and acknowledged
latest acknowledged byte
3
not yet sent but can be sent immediately
sequence number
receive window
first prohibited sequence number

Fig. 2. TCP sliding window

for these flows than for flows in general, as will be discussed in Section IV-C and Section V-A.

A. Identifying the application layer limitations

1) The slow sender limitation: To identify this limitation measure the percentage of time that the sending TCP entity doesn't have any outstanding data; something that should occur rarely when the sender is willing and able to provide data fast enough. Therefore, even if a flow has no outstanding data for just a small percentage of its lifetime, that indicates the slow sender limitation applies. We have set the threshold for establishing this limiting factor to 2 percent.

One should, however, be careful when the receive window size is equal to zero; in such a situation the amount of outstanding data *is* equal to zero, but this is not due to the sender and should thus not be considered an indication for the slow sender limitation.

2) The slow receiver limitation: To identify this limitation observe the size of the receive window. When the receiver has no problem keeping up with the incoming data, the advertised receive window may still decrease a little bit, since a packet may be acknowledged immediately by TCP before the application has had a chance to read the data. However, a receive window decrease of more than 2 MSS has been chosen as a sign that the application cannot keep up. Moreover, the receive window should be filled completely, otherwise the flow would not be limited by the slow receiver. The complete filling of the receive windows can be detected by the identification criteria for the receive buffer limitation, discussed in Section V.

If at least at 25 percent of the 'peaks' discussed in Section V-A the window has been decreased by more than 2 MSS, we conclude that the receiver is indeed too slow.

3) The application protocol limitation: To identify this limitation count the number of acknowledgment packets that contain data. We have set the threshold for establishing this limiting factor to a 10 percent occurrence of data-containing acknowledgment packets.

B. Identifying the TCP buffer limitations

1) The receive buffer: The size available in the receive buffer is advertised by the receiver by means of the TCP receive window. Figure 2 illustrates the ranges of sequence numbers that result from using a receive window, pointing out some terms that will be used later on. Sending cannot happen when the receive window is filled completely, i.e. when range 3 of the figure is empty. The filling of the receive window is therefore a useful quantity in the identification of this limitation.

Section V provides an in-depth discussion of the receive buffer limitation identification.

2) The send buffer: A send buffer limited flow shows approximately the same behaviour as a receive buffer limited flow, differing only in that the upper limit on the amount of outstanding data for a send buffer limited flow is less than the receive window (otherwise the receive buffer would have been the limiting factor).

Therefore, the identification of the send buffer limitation works in the same way as the identification of the receive buffer limitation, with the extra burden of having to estimate the size of the send buffer. The highest observed number of outstanding bytes throughout the entire flow seems the best estimation possible; it should be noted, however, that in case of a receive buffer limited flow the send buffer estimation will be equal to the receive window, although the actual size of the send buffer can be higher. Therefore, only flows that were not already identified as limited by the receive buffer should be considered.

Care must be taken when a one-pass implementation is made, because the estimation of the send buffer can sometimes increase during analysis of a repository of measurement data. Then, even though the part of a flow up to the increment possibly appeared to be send buffer limited before, that might *not* be the case with respect to the new send buffer estimation. More details can be found in [2].

C. Identifying the network limitation

The network limitation is probably the most intuitive flow speed limitation. When packets are injected into the network at a higher rate than the bottleneck link in the network can handle, the buffer in front of this link fills up. As a result the round-trip time increases, and loss occurs if the buffer overflows. Both of these effects can be seen in actual TCP traffic traces, and their occurrences can be considered indications that the network is limiting the speed of the flow.

However, packet loss may also be caused by bit errors (particularly in wireless networks), and fluctuations in the round-trip time can be induced by the burstiness of other traffic sharing a link. Therefore, when loss or delay fluctuations are observed, one should estimate to what throughput they would lead, and then compare that value to the actually observed TCP throughput: if the latter is much smaller, then apparently the network is *not* the limiting factor.

If packet loss is the limiting factor of a TCP flow, the maximum throughput achievable under the TCP congestion control mechanism [9] can be estimated by the following 'TCP Friendly formula' [7]:

$$BW = \frac{MSS}{RTT}\frac{c}{\sqrt{p}}. \qquad (1)$$

In this formula c is a constant; it is $\sqrt{3/2}$ according to the simplest derivation, but values differing up to about a factor of 2 are found depending on TCP implementation details [10] and statistical properties of the loss process. Further, p denotes the loss fraction, which can be obtained from traffic traces by counting retransmission events (multiple consecutive retransmitted packets count as one event). RTT is the round-trip time, which is estimated using an approach comparable to Karn's algorithm [11] used in TCP. Such an RTT calculation from a trace is only correct for flows for which the measuring point is near the sending side, which is one of the reasons for only considering such flows.

The actual throughput of a flow can also be obtained from the trace: the total amount of data divided by the total time (actually, as an approximation we subtract the total duration of timeouts since those are not taken into account in (1)). If the actual throughput is at least 50 percent of the value from (1), we conclude that the network is the limiting factor; this 50 percent threshold allows for the uncertainty in c.

If no packet loss occurs but the network is in fact limiting the speed, then apparently the network buffers combined with the delay bandwidth product of the transmission links are so large that the entire receive window worth of packets may be stored in them. When buffer contents increase, the delay increases proportionally to the amount of data in the buffer; thus, the delay should also be proportional to the amount of outstanding data of a flow that is being limited by the outgoing link of that buffer. When buffers are large, this increase may well be measurable, and the increase of outstanding data divided by the increase in delay is an estimate of the transmission rate of the bottleneck link. If the actual speed of the flow is of the same order as this bottleneck bandwidth estimate (the threshold used is 50 percent), we should conclude that the network is the limiting factor (even if no loss occurs and thus (1) cannot be applied).

D. Priorities

In the previous section criteria for identifying the limiting factor of a TCP flow have been proposed. For each factor a threshold has been used to decide whether or not that factor is limiting a flow. These thresholds are educated guesses of the optimal values, and should be subject to further research.

Sometimes, multiple factors could be above their threshold, thereby making it more difficult to establish the actual limiting factor. For example, a flow that appears to be limited by the receive buffer *and* a slow sender is actually only limited by the slow sender, because increasing the size of the receive buffer would not increase the speed if the sender would not provide more data. As it turns out the limitations can be prioritized such that of all established limitations the one with the highest priority applies. For a more in-depth coverage, see [2].

The slow sender limitation is dominant over all other limitations. When there is no more data available to be sent, improving other factors does not improve the overall throughput of the flow but will only increase the length of the idle periods. Therefore, when this limitation is present, all other limitations do not apply anymore.

The least dominant limitation is the application protocol limitation. The identification criterion of this limitation checks whether or not a request mechanism is being used, but does not guarantee this is also the limiting factor. It should therefore be considered the last possibility.

Both the slow receiver limitation and the receive buffer limitation are detected by noting that the receive window is full; the cases are distinguished by an extra condition in the former case, so its priority must be higher. When a flow is limited by one of these factors the estimation of the send buffer size will be equal to the size of the receive window, causing the send buffer limitation to appear to be present too. Since in this case the receive buffer or slow receiver is the real limiting factor, these limitations should be given a higher priority than the send buffer limitation.

This leaves us with the two criteria for detecting the network limitation. If the RTT increases due to buffers filling up without loss, typically the amount of outstanding data will increase until the send or receive buffer are fully used, so one of those criteria also applies; thus, the detection of the network limitation by means of the RTT should have a higher priority than the TCP buffer criteria. On the other hand, detecting the network limitation based on measured loss is rather independent from the other criteria, so its position in the priority stack does not matter much; because of uncertainties regarding c, we choose to put it after the limitations that can be detected with higher confidence.

This brings us to the following priority listing: (1) slow sender, (2) network due to increased RTT, (3) slow receiver, (4) receive buffer, (5) send buffer, (6) network due to loss, (7) application protocol.

V. Detailed discussion: receive buffer limitation

Section IV-B.1 provided an overview of the receive buffer limitation and its identification. This section gives a more in-depth coverage of all the details this identification consists of.

As mentioned before, a completely filled receive window is an indication of the receive buffer limitation, because a larger buffer would have allowed more data to be sent. Figure 3 shows an example flow where the receive window is indeed filled completely. After sending the third data packet, the amount of outstanding data is equal to the size of the receive window; therefore, sending stops until acknowledgments are received.

In order to determine the degree to which the receive window has been filled, we look at the amount of data that is still allowed to be sent. It is possible to calculate the amount directly from the data TCP is providing us, since TCP has the receive window as a mechanism for advertising the maximum allowed amount of outstanding bytes and because the actual number of outstanding bytes can be calculated by subtracting the last received acknowledgment number from the sequence number of the last sent data byte. The difference between the allowed and the actual number of outstanding bytes is equal to the number of bytes still allowed to be sent.

In the remainder of this section, the three major aspects of the receive buffer limitation detection are discussed.

First, we look at the moments in time where the amount of data allowed to be sent should be determined. We show why and when these moments — which are called peaks — occur. Second, the exact calculation of the amount of data allowed to be sent is considered. Third, something called 'buffer utilizations' will be discussed: as it turns out from manual examination of SURFnet traffic traces, some flows do not fill the receive window completely, but should nonetheless be considered receive buffer limited. These flows apply some kind of buffer utilization, describing the way the buffer is filled. The different possibilities are examined and methods for detecting flows that are limited by the receive buffer using these utilizations are discussed.

A. Peaks of outstanding data

To know whether or not the receive window has been filled completely some measurements have to be performed (discussed in Section V-B), aimed at calculating the amount of data allowed to be sent. As acknowledgments are received and data packets are sent, the number of outstanding bytes fluctuates continuously. Therefore, it is important to consider the amount of data allowed to be sent at specific moments: only at peaks of the amount of outstanding data the receive window can be filled completely and the receive buffer limitation can become visible. The decision on whether or not a flow is receive buffer limited in general can be based on the percentage of peaks that were receive buffer limited.

To see how to identify peaks, some understanding of the behaviour of the amount of outstanding data is necessary. Figure 4 shows a simplified illustration of this behaviour for a receive buffer limited flow. Because the receive buffer limits the flow, the sender continues to send data packets until it reaches the threshold prescribed by the receive window, at t_1. During this sending process, the number of outstanding bytes increases gradually. When the threshold has been reached, sending has to halt until acknowledgments arrive, starting at t_2. The new possibility to send data is used to make the amount of outstanding data increase to the size of the receive window again; this occurs at t_3.

Although a flow exhibiting the behaviour described in the figure is clearly limited by the receive buffer, the number of outstanding bytes is not at all times equal to the maximum amount prescribed by the receive window. In order to cope with this, the notion of a *peak* has been defined. In Figure 4 a peak from t_1 till t_2 and from t_3 till t_4 can be observed.

A data burst makes the number of outstanding bytes increase, whereas an acknowledgment packet makes it decrease again. A peak in a trace sending data from A to B is therefore formally defined as a period of time that starts with the last packet of a data burst from A to B and ends with an acknowledgment packet from B to A. The height of the peak under examination is then equal to the sequence number of the last data packet, increased by the length of that data packet and decreased by the acknowledgment number of the last acknowledgment packet seen prior to the data burst.

Unfortunately, at the receiving side acknowledgment packets are sent *during* such a data burst, making it impossible to detect peaks in the way just described. This problem has been solved by only considering flows that were measured at the sending side.

The only moment at which the receive window could be filled completely in case of a receive buffer limited flow is at the time of a peak, since at all moments in time close to it the amount of outstanding data will be lower.

When the receive window has been filled completely at

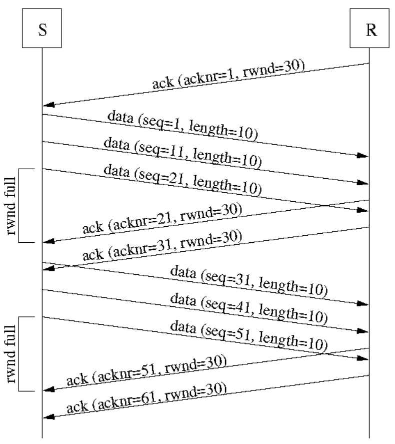


Fig. 3. Receive window limitation

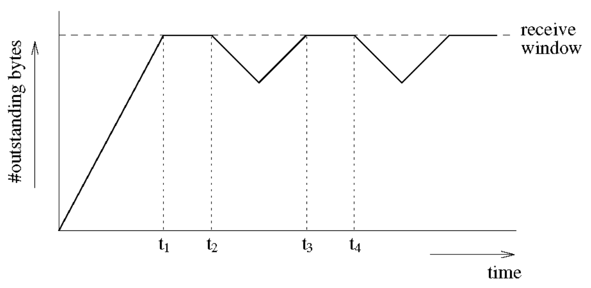


Fig. 4. Peaks of outstanding data

the time of a peak, this peak is called receive buffer limited. The threshold for establishing the receive buffer limitation has been set to a 50 percent occurrence of receive buffer limited peaks, since flows that are not limited by this factor exhibit a percentage close to zero percent and flows that do, exhibit a percentage close to a hundred percent.

B. Calculating the amount of data allowed to be sent

To calculate the amount of data allowed to be sent keep track of the next sequence number and the first prohibited sequence number of a flow (see Figure 2). As can be seen from the figure, subtracting the next sequence number from the first prohibited sequence number results in that which we are interested in: the amount of data that is still allowed to be sent.

For each data packet observed, if necessary update the value of the next sequence number (calculated by adding the length of the packet to its sequence number). When a retransmission takes place, however, the next sequence number might decrease, even though the section of sequence numbers that have not been sent yet but can be sent immediately has not changed. The value should therefore only be updated in case it is not decreasing.

For each acknowledgment packet observed, if necessary update the value of the first prohibited sequence number (calculated by adding the size of the receive window to the acknowledgment number).

C. Different utilizations and their identification

Thus far, we have assumed that a sender that is not limited by anything but the receive buffer, will always send data when the receive window is not completely full. In reality, however, many senders only send data when there is for example room for at least one full-size packet.

In our data, we found three of such ‘buffer utilizations’. They are described below, together with criteria for detecting whether or not a peak should be considered receive buffer limited with respect to that utilization. To answer the question of whether or not a flow is receive buffer limited in general, the percentage of peaks that were receive buffer limited (regardless of which utilization they were using) can again be considered.

1) The complete buffer utilization: This utilization describes the process thus far assumed to be true for each receive buffer limited flow: even if only a few bytes are permitted to be sent, a sender still transmits a new packet, thereby filling the receive window completely.

To identify a receive buffer limited peak using the complete buffer utilization check whether or not the amount of data still allowed to be sent is equal to zero.

2) The MSS buffer utilization: Some flows seem to only transmit MSS-sized packets, possibly due to the Nagle algorithm preventing the silly window syndrome [12]. For these flows, a receive window that allows less data than the MSS to be sent will already make the sending process halt until acknowledgments arrive.

The identification of a receive buffer limited peak using the MSS buffer utilization is similar to the identification using the complete buffer utilization. Again, look at the amount of data still allowed to be sent. If it is equal to zero the complete buffer utilization would apply and if it is larger than one MSS, another packet could have been sent. Therefore, we establish the MSS buffer utilization when the amount of data still allowed to be sent is between zero and the MSS.

Furthermore, the length of the last data packet before a peak is required to be equal to the MSS. If it was not, the flow is apparently not using the MSS buffer utilization.

3) The block-based buffer utilization: Some flows send their data in fixed-size blocks, consisting of a number of MSS-sized packets followed by one shorter packet. Figure 5 illustrates this utilization by an example flow transmitting in blocks of 4096 bytes. Flows using the block-based buffer utilization do not transmit data when the space left in the receive window is less than the block size.

In order to be able to identify a receive buffer limitation using the block-based buffer utilization for a certain flow, first make sure the flow is using block-based buffer utilization at all. Because blocks are larger than the MSS, the first packets of a block are always equal to the MSS and only the last one is smaller, in order to arrive at the block size.

A block can be defined as a sequence of packets following a packet that is non-full and non-empty, up to and including the next packet that is also non-full and non-empty (so, in case of the example flow, from the first data packet following a 1176-bytes packet up to and including the next 1176-bytes packet).

After a certain amount of blocks has passed by, a check can be performed on whether or not a certain percentage of the blocks had the same size. In our implementation ten blocks are analysed and when nine or ten of them have the same size (which is then or course the block size), the block-based buffer utilization is assumed. In our implementation this check is performed every two hundred packets, in order to cope with changes in the utilization during the course of a flow. Further research could be performed to check whether or not these

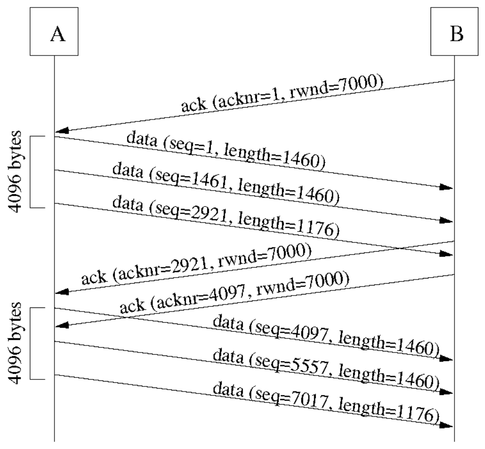

A
B
ack (acknr=1, rwnd=7000)
4096 bytes
data (seq=1, length=1460)
data (seq=1461, length=1460)
data (seq=2921, length=1176)
ack (acknr=2921, rwnd=7000)
ack (acknr=4097, rwnd=7000)
4096 bytes
data (seq=4097, length=1460)
data (seq=5557, length=1460)
data (seq=7017, length=1176)

Fig. 5. Block-based buffer utilization

TABLE I
MEASUREMENT RESULTS

	Location 1 (56 GB)		Location 2 (38 GB)		Location 3 (88 GB)	
Limitation	Perc. of flows	Perc. of bytes	Perc. of flows	Perc. of bytes	Perc. of flows	Perc. of bytes
Undetermined	21.5%	9.3%	39.0%	21.3%	25.2%	10.9%
Network	27.0%	38.7%	20.9%	33.3%	21.8%	36.4%
Send buffer	6.0%	11.1%	0.3%	0.8%	3.1%	6.9%
Receive buffer	12.4%	16.3%	10.9%	15.0%	10.8%	9.6%
Slow sender	28.5%	22.0%	24.2%	26.7%	25.7%	34.8%
Slow receiver	3.3%	1.7%	4.2%	2.9%	4.2%	1.1%
Application protocol	1.5%	0.9%	0.6%	0.2%	9.3%	0.3%

values are optimal.

Once the block-size has been established, it is possible to check if a peak is limited by the receive buffer using the block-based buffer utilization. For each peak three criteria (besides that the flow has to use the block-based buffer utilization) have to be met in order to identify it as receive buffer limited:

(a) The number of bytes left to be sent has to be more than zero, otherwise the complete buffer utilization would apply.
(b) The number of bytes left to be sent should be less than the block size, otherwise another block could have been sent.
(c) The previous data packet should have had the length of the small 'ending packet' that was seen identifying the block-based buffer utilization (1176 bytes in our example), since it should indeed be the end of a block.

VI. MEASUREMENT RESULTS

To get an idea of the occurrence of the potential limiting factors discussed in this report, an implementation of the identification criteria has been made and applied to a repository of actual measurement data.

A. Measurement setting

We have used the traffic repository maintained at our university [13], containing data collected at different locations in the Netherlands in 2002 and 2003. Details can be found in [14].

At each location measurements were performed during intervals of fifteen minutes. Since these intervals were not adjacent, several flows have been recorded partially. Only flows that were recorded from their beginning have been analysed, because the first packet of a flow can contain a receive window scaling option. When the existence of this scaling is unknown, the receive buffer limitation identification cannot be performed correctly. Furthermore, we have restricted ourselves to flows measured at the sending side and larger than 100 kilobyte (to avoid flows in which startup phenomena dominate).

All measurements were collected by devices attached to a switch connecting an access network to the rest of the Internet. Therefore, all traffic going from and to that access network has been recorded.

Unfortunately, the repository turned out to have some problems: not all packets passing through were correctly captured. The interested reader is referred to [2] for a detailed discussion on how to deal with such a repository and still achieve valid results.

B. Measurement results

Since the main goal of our research was to develop identification criteria, extensive statistical analyses of the measurement results have been left for further research. However, Table I gives a first impression of the results. For the three locations that were analysed, results of all examined flows have been summed to get an overall overview. For each limitation the percentage of flows that seems to be limited by it, as well as the percentage of bytes contributed by these flows is shown.

As mentioned before, several aspects of our methods need further research, so the results should be considered preliminary.

C. Observations

We have observed several facts about the resulting numbers. The following discusses each observation.

1) Undetermined: For a large percentage of the flows the limiting factor is still undetermined. It is already indicated by the small percentage of bytes these flows contribute, that a major cause for this is that small flows cannot be identified correctly. Although the smallest flows have already been excluded from our analyses, the results indicate that the lower limit on the amount of bytes of a flow might need to be increased. Another cause for not being able to detect the speed limiting factor is that for some flows this factor changes during the course of the flow, thereby making it impossible to point out a single limiting factor.

2) The network limitation: For only about one *third* of the bytes, and even only approximately a *quarter* of the flows the network is the limiting factor. We can conclude that although bandwidth should still be considered a possible bottleneck, other limiting factors also occur.

3) The send buffer limitation: The send buffer does not occur often in practice, but should not be neglected. See [6] for a suggestion to speed up send buffer limited flows.

4) The receive buffer limitation: Just like the send buffer limitation also the receive buffer limitation turns out to be out in the wild, even in larger numbers. Again, [6] provides methods to speed up these flows.

5) The slow sender limitation: A large percentage of flows is limited by a slow sender. Due to the increasing use of streaming media, Internet games and chat applications, this was not unexpected.

6) The slow receiver and application protocol limitation: These limitations do not occur often in practice. Since CPU power increased significantly in the past decades, the low percentage of slow receiver limited flows seems logical.

VII. Conclusions and Recommendations

A. Conclusions

In this paper we identified which factors can limit the speed of TCP flows, and presented some methods to analyze TCP flows to detect these limitations. In addition, we have applied these methods on a traffic repository to get an impression of which factors play a role in practice, and whether or not bandwidth shortage is still the main limiting factor.

In general there are three possible categories of factors that can limit the speed of a TCP flow: the application, the TCP buffers and the network.

Application-limited flows can be further subcategorized into flows where the sending application doesn't provide data fast enough (slow sender), flows where the receiving application cannot handle the data any faster (slow receiver) and flows that appear to have their own acknowledgment or request mechanism at the application protocol level.

TCP-buffer limited flows can be subcategorized into flows that are limited by the size of the TCP receive buffer and flows that are limited by the size of the TCP send buffer.

Network limited flows have not been further subcategorized in this paper; this is subject of future research.

For all these factors it is possible to derive methods to detect the limitations; Section IV and Section V demonstrated how this can be done. It should be noted, however, that the identification of these factors is much easier — or maybe even only possible — when measurements were performed at the sending side of a flow (see Section IV-C and Section V-A).

Section VI gave an impression of which factors play a role in practice. For twenty to forty percent of all flows it turned out to be impossible to determine which factor is limiting its speed; this is particularly true for short flows. Approximately one third of the flows seemed to be limited by the application; network managers need not worry about these flows. Ten to twenty percent of all flows is limited by TCP buffer settings; changing operating system parameters may speed up these flows. Finally, twenty to forty percent of all flows is limited by the network; it seems therefore that in many cases a further increase in network capacity is still useful. It should be noted, however, that these findings are obtained on research networks, which may not be representative for the whole range of networks. These figures should therefore be used with care; further study is needed to provide more reliable figures.

B. Recommendations

Obviously, the present method should be compared more thoroughly to [3] and [4]; combining ideas from all three methods might be advantageous.

It would be of interest to extend our method for flows with a measurement device located at the receiving side; at the moment this seems very difficult (if not impossible), but detection criteria that produce useful estimations might exist.

Furthermore, full-duplex flows have not been addressed yet and network reordering is assumed not to exist. Further research could try to incorporate these issues.

Finally, the thresholds used to identify each limiting factor were chosen based on manual examination of measurement data in a quite ad hoc manner. Further research could possibly improve these thresholds.

Acknowledgment

The authors would like to thank the reviewers for their comments, particularly for bringing [3] and [4] to our attention.

References

[1] V. Jacobson and R. Braden, "TCP Extensions for Long-Delay Paths," RFC 1072, Oct. 1988.

[2] M. Timmer, "How to Identify the Speed Limiting Factor of a TCP Flow," Bachelor's Thesis, University of Twente, Aug, 2005. [Online]. Available: http://dacs.cs.utwente.nl/assignments/completed/Timmer_B-assignment.pdf

[3] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA: ACM Press, 2002, pp. 309–322.

[4] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and T. En-Najjary, "Root cause analysis for long-lived tcp connections," in *CoNEXT'05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology.* New York, NY, USA: ACM Press, 2005, pp. 200–210.

[5] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications*, vol. 27, pp. 23–29, June 1989.

[6] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication.* New York, NY, USA: ACM Press, 1998, pp. 315–323.

[7] M. Mathis, J. Semke, and J. Mahdavi, "The macroscopic behavior of the tcp congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 303–314, 1998. [Online]. Available: citeseer.csail.mit.edu/padhye98modeling.html

[9] V. Jacobson, "Congestion avoidance and control." in *SIGCOMM*, 1988, pp. 314–329.

[10] J. Mo, R. J. La, V. Anantharam, and J. C. Walrand, "Analysis and comparison of TCP reno and vegas," in *INFOCOM (3)*, 1999, pp. 1556–1563. [Online]. Available: citeseer.ist.psu.edu/mo99analysis.html

[11] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in *SIGCOMM '87: Proceedings of the ACM workshop on Frontiers in computer communications technology.* New York, NY, USA: ACM Press, 1988, pp. 2–7.

[12] J. Nagle, "Congestion control in IP/TCP internetworks," RFC 896, Jan. 1984.

[13] "University of Twente - Traffic Measurement Data Repository," http://traffic-repository.ewi.utwente.nl/.

[14] R. van de Meent, "M2c measurement data repository," http://arch.cs.utwente.nl/projects/m2c/m2c-D15.pdf, 2003.