Efficient Bandwidth Estimation for Peer-to-Peer Systems

Richard Süselbeck, Gregor Schiele, Patricius Komarnicki and Christian Becker
University of Mannheim
Mannheim, Germany
{ richard.sueselbeck | gregor.schiele | patricius.komarnicki | christian.becker }@uni-mannheim.de

Abstract—**Many peer-to-peer (P2P) systems require accurate information about their peer's available bandwidth, e.g., for load balancing. Determining this information is difficult, as a suitable approach must address two challenges. First, it must be able to deal with fluctuating bandwidth. Second, it must incur low overhead to prevent interference with the operation of the P2P system. In this paper we present an approach to estimate the available bandwidth of peers in a P2P system, based on a combination of traffic observation and the strategic injection of traffic into the system. We evaluate our approach and show that it is accurate and responsive in settings with variable bandwidth while resulting in limited interference with the system.**

I. Introduction

Many peer-to-peer systems require information about their peers' available network bandwidth, more specifically the total available bandwidth which a peer can contribute to the system. This information is used for a variety of purposes such as coordinator election [15] [16] [17] and load-balancing [15] [2] [3].

Note that this differs from the available bandwidth along specific network paths, e.g. between two individual devices. While the former can be used to determine the general suitability of a peer for a given purpose, e.g. to act as a coordinator, the latter allows to rate communication with a specified set of communication partners. Often, these are not yet known – e.g. when selecting a coordinator – or can change in the future. Therefore, in this paper we focus on estimating the total available bandwidth, independently of specific communication partners.

Determining the available bandwidth of a peer in a P2P system is a challenging task. Existing research work generally assumes that the available bandwidth is known a priori. In practice, such systems must rely on the peers' users to manually specify a suitable value. However, this is not easy for users to accomplish. Internet providers often brand their products with the maximum available bandwidth, while contracts allow to provide substantially less bandwidth at different daytimes [10]. Similarly, mobile peers that roam through different networks or use wireless mobile telecommunication networks may also experience changing Internet connections with varying characteristics. Even if the capacity of a peer's Internet connection can be obtained, this does not necessarily reflect the available bandwidth for a peer, if for example the Internet connection is shared with other computers. Finally, users often intentionally specify incorrect values to avoid their peer being chosen for specific functionalities [1].

In this paper we introduce an approach for determining the available bandwidth of a peer in a P2P-system. Our approach combines low-overhead passive observation of existing traffic with a peer-based injection of traffic to improve accuracy. To reduce overhead, we only use traffic injection when the peer has idle resources or our confidence in the current estimation is too low. The system is configurable by applications to balance the achieved accuracy with the resulting overhead. We evaluate our approach and show that it provides an accurate and responsive estimate of available bandwidth with limited overhead.

The paper is structured as follows. First we discuss the underlying system model and derive requirements. Then we describe our approach in detail. The results are evaluated before the paper concludes with a discussion of related work and an outlook on future work.

II. System Model and Requirements

Our approach targets fully distributed P2P systems. Such systems are composed of a potentially large number of peers, which are connected with each other using a so-called overlay network. Each peer is a software entity that is executed on a networked computer, usually an end-user device that is connected to the Internet via an Internet connection. The Internet connection can be shared between multiple devices.

Peers enter the system from different subnets and may change the subnet over time, e.g., when they are mobile. A peer's performance with respect to the P2P system depends on various properties, e.g., memory, CPU, and available bandwidth. In the context of this paper we are only interested in the total bandwidth a peer can contribute to the P2P system at any given time.

A suitable approach for estimating the total bandwidth of a peer has to fulfil three main requirements that we discuss in the following.

- *R1 Accuracy:* To be useful, the resulting estimate must be accurate. Note that the specific level of accuracy that is required depends on the application using the P2P system. Therefore, the estimation approach should be configurable for different accuracy levels. Inaccurate estimations can be distinguished into underestimation and overestimation. Underestimations report a total bandwidth that is lower than the actual value. Overestimations report a total bandwidth that is too large. While we want to eliminate both kinds of errors, we argue that overestimations are more severe than underestimations and should be avoided. Otherwise, a peer might be rated too good and be overloaded with management functions. This could degrade system performance or even lead to system failure. Therefore, we accept a certain amount of underestimation if this reduces the probability of an overestimation.

- *R2 Responsiveness:* The available bandwidth of a peer is not static. Depending on the peer's context, it can vary frequently, e.g. in case of mobile devices or multiple devices sharing an Internet connection. The bandwidth estimation approach should respond quickly to such variations and always provide a current estimation of the available bandwidth. Note that – just as for accuracy – the specific level of required responsiveness is application dependent and should be configurable.
- *R3 Minimal Interference:* While providing accurate and current information about the available bandwidth, the estimation approach should minimize its interference with the actual application. Otherwise, the estimation could e.g. consume a large percentage of the available bandwidth for its measurements. This should not be the case. When the application needs to send or receive data, the estimation approach should use as little bandwidth as possible and provide as much as possible to the application. On the other hand, if the application currently does not need the bandwidth, the estimation approach can take over and use it to perform measurements.

Having motivated our requirements we continue by describing our approach. After that we evaluate it.

III. Our Approach

Our approach for bandwidth estimation in P2P systems aims to fulfil all three requirements given before (R1 – R3). In the following we first give a short overview and design rationale. Then we discuss different aspects of our approach in more detail. Note that in the discussion we restrict ourselves to the downlink. The same can be applied to the uplink.

A. Overview and Design Rationale

Our approach combines passive traffic observation with a peer-based active traffic injection. Passive traffic observation allows us to measure the currently used bandwidth. We use this information to estimate the currently available bandwidth continuously and to detect fluctuations in it, fulfilling Requirement R2. As no additional traffic is induced, passive observation also fulfils Requirement R3. However, estimations can be inaccurate, particularly in low traffic scenarios. Therefore, we complement this approach with periodic active traffic injections that allow us to get very accurate measurements at specific points of time (Requirement R1). After an active measurement is finished, the passive estimation can take its result as a new reference point for its continuous estimation. Overall, this results in a system that allows to balance between all three requirements by choosing the frequency of traffic injections. An application can e.g. select many injections and get a highly accurate estimation with good responsiveness but some interference. Another application can choose to use fewer injections, resulting in an estimation with lower interference but less accuracy.

The basic idea of our estimation algorithm is as follows. To determine the available bandwidth, we need two pieces of information: the system's currently used bandwidth uBW and the system's current capacity C (i.e. the maximum available bandwidth). Given these values we can calculate the available bandwidth aBW as $aBW = C - uBW$. We can easily measure uBW, e.g. using system load information provided by the operating system. However, C is a priori unknown and cannot be determined directly. Therefore, the problem of estimating aBW can be reduced to estimating C.

To do so we continuously determine estimates for a lower bound lB and an upper bound uB of C. The real value of C must be between these two bounds. lB is determined by the current (passive) measurement of uBW. uB is set using (active) traffic injections. To avoid overestimation, we add a decay function to calculate the progression of C. The decay function decreases C over time until a given threshold value is met. Then uB is recalculated using another active traffic injection.

Note that the system may have previous knowledge about C. This knowledge can originate from a user setting in our software, information from hardware, e.g., a DSL modem, or from previous measurements. Our approach can use this knowledge as a starting point for its estimations. However, we argue that these values are not reliable and thus cannot replace continuous estimation. As discussed before, users may simply not be aware of their Internet connection's capacity or even have incentives to misreport it [13]. The capacity of an Internet connection can also change, for example during peak hours due to under-provisioning by ISPs [10]. Finally, certain systems such as mobile devices change their type of internet connection regularly.

In the remainder of this section, we describe this approach in more detail. First, we present the basic capacity estimation algorithm. Then we describe how the threshold and the decay rate are computed. After that we show some enhancements to the basic approach and discuss how traffic injections are performed. Finally we bring all the pieces together to compute the available bandwidth.

B. Capacity Estimation

In the following we explain our basic capacity estimation algorithm in more detail (see Algorithm 1). As described before, the algorithm depends on a decay function and a threshold value. The details of their computation (together with the parameters T and L) are described in the next sections and omitted here.

Algorithm 1 Basic Capacity Estimation Algorithm

```
1:  procedure EstimateCapacity($init$, $T$, $L$)
2:      $uB = C = init$; $t = 1$
3:      loop
4:          $lB =$ usedBw()
5:          if $lB > uB$ then
6:              $uB = lB$
7:          end if
8:          $thr =$ threshold($lB$, $uB$, $T$)
9:          $C = uB -$decay($t$, $L$, $uB$, $thr$)
10:         $t = t + 1$
11:         if $C \leq thr$ then
12:             $uB =$ injectTraffic()
13:             $t = 1$
14:         end if
15:     end loop
16: end procedure
```

The algorithm starts by initializing the upper bound uB and the estimated capacity C with the parameter $init$. This parameter is passed to the algorithm by the application and contains all previous knowledge about the network capacity. If no such knowledge is available, $init$ can be set to zero. After that, a variable t is initialized to one. It is used to compute the correct decay rate later. The higher t, the larger the decay. Thus, the initial decay is minimal.

After the initialization, the algorithm loops over the following instructions. First, it sets the lower bound lB to the current network load of the system. Then, it checks whether the current load is higher than the expected upper bound uB. If it is, uB is updated. Now, it computes the current threshold value thr. After that it recomputes the estimated capacity C. C is set to uB minus the current decay function value (decay(..)). Then, t is incremented to increase the decay in the next round. If the resulting capacity is equal or below the threshold, the algorithm measures uB again by initiating a traffic injection and setting uB as the highest traffic observed during the traffic injection. In addition, the algorithm resets the decay rate by setting t to 1. After that, the loop starts again.

This algorithm realizes the basic approach presented in the previous section. It continuously estimates C using passive traffic observation. In order to provide an accurate estimation for C the upper bound has to be determined by traffic injection. Right after an injection, uB is very accurate and we can set $C = uB$. The further our last traffic injection is in the past, the more likely it is that C might have changed, reducing our trust in its estimation. Thus, if we keep C at the upper bound, the likelihood of an inaccurate estimation keeps increasing.

The naïve solution for this is to constantly inject traffic and redetermine uB continuously. Clearly, this would produce prohibitive overhead, violating requirement R3. Thus we use our decay function and the threshold to model the ageing of C over time.

C. Threshold

The threshold thr is the value towards which C falls before a new active traffic injection is initiated. Clearly, thr must be between our two bounds lB and uB. However, choosing the correct value for thr is not trivial, as it influences the accuracy we can expect to get. If we select a high thr, i.e., one close to uB, we increase the risk of overestimation, as the reported C will be higher. On the other hand, if we select a low thr, i.e., one close to lB, we increase the risk of underestimation, as the reported C will generally be lower. The correct choice ultimately depends on the application. The less tolerable an overestimation is for a given application, the lower th should be. We therefore introduce a parameter T with a range from 0 to 1 and define the threshold (and thus the threshold(..) function) as follows:

$$thr = lB + (uB - lB) * T \tag{1}$$

Using T we can select where the threshold will be set between the two bounds. With $T = 0.5$, the threshold is exactly in the middle between lB and uB. With $T = 0$, the threshold is equal to lB. Note that our approach is designed to prefer underestimation even when the threshold is set very high, as the upper bound is selected in such a way that it can be an underestimation even at the moment of selection.

Note that thr is calculated dynamically for each point of time, using the most recent lB and uB. As lB is continuously remeasured, th can change constantly, getting larger and smaller with the rise and fall of lB.

D. Decay Function

After looking at the threshold, we now discuss how the decay function (decay(..)) is computed in more detail. This function determines the behaviour of the bandwidth estimation mechanism and is designed to perform two functions. First, it decreases the estimated capacity over time. This pessimistic approach reduces overestimations. Second, it determines when a traffic injection is initiated.

The first question when designing the decay function is what basic function to use for it. We have experimented with a number of functions. Fundamentally, we could choose between linear decay, decay which decreases over time (getting flatter) and decay which increases over time (getting steeper).

We have experimented with all three types of functions and found that a function which decreases over time leads to significant underestimation as the capacity quickly nears the threshold. This in turn leads to very low reported available bandwidth. While we consider overestimations to be more serious than underestimation, we considered this trade-off too large. Both functions with linear decay and increasing decay showed suitable results, with the linear one again resulting in a more pessimistic approach with lower reported available bandwidth.

In the end we have chosen to use an increasing decay. This type of function models a higher confidence in the value of the upper bound for the near future after a traffic injection followed by decreasing confidence over time.

The second question is the gradient of the decay, i.e., how fast the decay value rises and thus C falls. Modifying the gradient changes how frequently traffic injections are performed. In stable environments, a low gradient can be used, reducing overhead. To be more responsive (see Requirement R2) in very dynamic situations, a high gradient should be chosen. To control the gradient, we introduce a new parameter L which determines how long it takes the decay function to reach the threshold, i.e., how many rounds the loop in Algorithm 1 is executed before a new injection occurs. For now, L is set by the application. We discuss a more sophisticated approach in the next section. The resulting decay function is:

$$decay(t, L, uB, thr) = \log_L(t) * (uB - thr) \qquad (2)$$

Note that as the threshold is computed dynamically and can increase if the current lB is high (i.e. there is a lot of traffic), the result value of the decay function can actually decrease between two consecutive calls, even if t increases. This results in an increase of the estimated capacity C.

E. Dynamic Decay Rate Adaptation

The approach we presented so far relies on a fixed value for L. This introduces two problems. First, what if the application does not know the overall stability of the system and thus cannot set L correctly? Second, what if the system stability varies over time, such that L should be adapted?

To counter these problems, we add an algorithm to let our estimation approach choose L automatically, depending on the current stability. To do so, we use a history of deltas between successive uB measurements. Using this history we scale between a pre-set maximum and a minimum L. If uB remains stable over time, we choose a large L to reduce unnecessary traffic injections. If we detect large changes to uB, we choose a small L to increase the rate of decay, thus improving the accuracy. The history δ is given as:

$$\delta = \alpha * 1 - \frac{min(uB_{old}, uB)}{max(uB_{old}, uB)} + (1 - \alpha) * \delta \qquad (3)$$

It computes a weighted average of past deviations between two consecutive uB measurements. The parameter uB_{old} contains the previous value of uB, α is a smoothing factor between 0 and 1 that can be used to control how quick the history adapts to new values. If α is set to a low value, the history reacts slowly. A large α results in quick history adaptations. For our evaluations we used $\alpha = 0.2$. Using this equation we can now specify our new L (and thus the length(..) function) as:

$$L = L_{min} + (1 - \delta) * (L_{max} - L_{min}) \qquad (4)$$

The equation sets L between a minimal (L_{min}) and maximum (L_{max}) value by adding an amount to L_{min} that depends on the current stability, given by our history δ.

F. Decay Rate Reset

With our new dynamically computed L, we can adapt the traffic injection rate depending on the stability of uB. But there is another possibility to further reduce the number of injections. Our decay function models our decreasing confidence in a previous measurement of uB. A traffic injection is initiated when this confidence is too low. However, there are situations, in which we can increase our confidence in a previous measurement again, specifically with respect to possible overestimations.

The basic idea is to take the amount of current traffic (represented by the lower bound lB) and its distance to the current uB into account. If there is a lot of traffic, i.e. lB gets close to uB, our confidence in uB increases, because it is less likely that we did overestimate uB. As a consequence, we are resetting the rate of decay in this situation. Thus, in case of higher traffic in the system, we are further delaying the next traffic injection, as we have higher confidence in our upper bound.

This has another advantage. We are less likely to initiate a traffic injection in situations where the system is using significant amounts of bandwidth. This reduces the probability to interfere with the operation of the system (see Requirement R3). On the other hand we perform more regular traffic injections in situations where the system has sufficient bandwidth resources available. This dynamic adaptation thus not only reduces overall overhead, but moves the generation of overhead to situations where its effects are less noticeable.

Note that this optimization should not be applied if underestimations are considered very severe. The likelihood of underestimations is not reduced by high traffic. Therefore, using decay rate resets potentially results in more underestimations.

To realize the described behaviour we reset the rate of decay in two cases. First, we perform a reset, if the new estimation of the capacity C is higher than the last estimation of C. This can happen, if lB has increased so much that the result of the decay function decreases, as described in Section III-D. In addition we also reset the rate of decay each time we change the upper bound, either because of a traffic injection or because the current traffic was found to be higher than uB. In both cases we have a reliable new value for uB, which means we can set C equal to uB and restart the decay process.

In some circumstances it may be possible that the rate of decay is constantly reset. In this case it can take very long for C to reach the threshold. While this is intended behaviour, we want to put an upper limit on the duration for which we can delay the next traffic injection. To do so, we also define a maximum interval M for traffic injections, i.e., an interval at which a traffic injection is always performed regardless of whether the decay function has reached the threshold. For our evaluation we have set this as three times the maximum interval for parameter L.

G. Extended Capacity Estimation

To add the additional optimizations described in the last two sections, we have to extend our basic capacity estimation algorithm (see Algorithm 1). Our extended capacity estimation algorithm is shown in Algorithm 2. It changes the original algorithm only slightly. First, we remove the parameter L from the parameter list passed to the algorithm, as L is now computed dynamically, using the length(..) function. We replace L with two new parameters L_{min} and L_{max}, specifying the minimal and maximum length between traffic injections. Second, we add the two cases for resetting the decay rate, $lB > uB$ (Line 7–12) and $C > C_{old}$ (Line 18–20) together with necessary temporary variables to store the old values of uB and C. Third, we add the maximum injection interval M (Line 26–31) and a new counter variable t_m to enforce it.

Algorithm 2 Extended Capacity Estimation Algorithm

```
1:  procedure EstimateCapacity($init$, $T$, $L_{min}$, $L_{max}$)
2:      $uB = C = init$;
3:      $t = 1$; $t_m = 1$;
4:      $L = 0$; $M = 3 * L_{max}$;
5:      loop
6:          $lB$ = usedBw()
7:          if $lB > uB$ then
8:              $uB_{old} = uB$
9:              $uB = lB$
10:             $L$ = length($uB$, $uB_{old}$, $L_{min}$, $L_{max}$)
11:             $t = 1$
12:         end if
13:         $thr$ = threshold($lB$, $uB$, $T$)
14:         $C_{old} = C$
15:         $C = uB-$decay($t$, $L$, $uB$, $thr$)
16:         $t = t + 1$
17:         $t_m = t_m + 1$
18:         if $C > C_{old}$ then
19:             $t = 1$
20:         end if
21:         if $C \leq$ thr then
22:             $uB_{old} = uB$
23:             $uB$ = injectTraffic()
24:             $L$ = length($uB$, $uB_{old}$, $L_{min}$, $L_{max}$)
25:             $t = t_m = 1$
26:         else if $t_m > M$ then
27:             $uB_{old} = uB$
28:             $uB$ = injectTraffic()
29:             $L$ = length($uB$, $uB_{old}$, $L_{min}$, $L_{max}$)
30:             $t = t_m = 1$
31:         end if
32:     end loop
33: end procedure
```

H. Traffic Injection

Traffic injections are a key part of our approach. Whenever the estimation wants to re-evaluate the upper bound uB, it initiates an injection. A traffic injection essentially means that the peer requiring the injection sends or receives as fast as possible to make sure that the uplink or downlink capacity is fully used.

In case of a traffic injection on the uplink, i.e., to measure the uplink capacity, the injecting peer must select a receiver for its traffic. A naïve approach would be to simply use one other peer. However, we may select a peer that has a smaller downlink capacity than our uplink capacity. In fact, given that many Internet connections are asymmetric, this is a likely scenario.Thus we would measure the other peer's downlink instead of our uplink. In addition, we may induce a lot of overhead to the other peer. Therefore, we select a group of peers, which then simultaneously receive data from the injecting peer. We choose the peers in the peer group based on their own available bandwidth, i.e., we prefer peers with high available bandwidth. The details of this election process, how many peers to choose and how to perform the coordination between the peers are beyond the scope of this paper. So far, we rely on a special coordinator peer to control the process. More sophisticated coordination protocols are subject to our future work. For traffic injections on the downlink, i.e., to measure the downlink capacity, we follow essentially the same approach. The only difference is that instead of sending data, the peer initiating the traffic injection is receiving data from the peer group.

Note that there can be a dependency between uplink and downlink traffic injections. High upload speed can negatively influence download speed and vice versa, particularly on certain types of connection (e.g. DSL). This would result in less accurate estimations. Therefore, we delay a traffic injection when another one is currently running.

I. Available Bandwidth

Now that we have determined C, we can simply calculate the available bandwidth aBW by subtracting the current bandwidth from the estimated capacity. However, this value can fluctuate. While one of our requirements is responsiveness (R2), many applications may prefer a somewhat more stable value. By using a simple history-based approach we can reduce the impact of temporary spikes on the reported available bandwidth. The resulting equation for aBW is as follows:

$$aBW := \beta * (C - lB) + (1 - \beta) * aBW \qquad (5)$$

Just as for the history used in the computation of L (see Equation 3), we introduce a smoothing factor β between 0 and 1 to control how fast aBW adapts to new values. In case of a small β value, temporary spikes are smoothed out and aBW reacts slowly to changes. If β is set near 1, the history is mostly ignored, and aBW reacts faster.

IV. Evaluation

In this section we evaluate our approach for bandwidth estimation with a number of experiments. The experiments are performed on five peers that we deployed on different computers in the Internet. Our evaluation setup is depicted in Figure 1. The five peers form an overlay network. Three of them (Peers P_1, P_2, P_3) are executed as micro instances in the Amazon Elastic Computing Cloud (EC2). The other two peers (Peers M and F) run on Intel Core2 Duo (2 GHz, 2GB RAM) computers. They are located in a LAN that is connected to the Internet via DSL. The Internet connection is advertised by the provider with a speed of 16 Mbps. However, in our experiments we found that the actual capacity of the connection is approximately 3.7 Mbps. We have additionally verified this result through several server-based bandwidth tests.

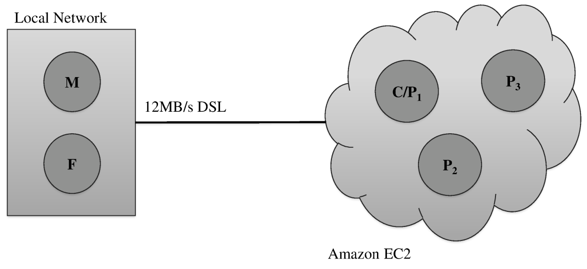

Fig. 1. Evaluation Setup

Using this setup, we implement three scenarios. In the first scenario, we measure the available bandwidth of Peer M without any background traffic, i.e. on an otherwise idle system. Peer F is not used in this scenario. The Peers P_1, P_2, P_3 act as the peer group for traffic injections (see Section III-H). Each has a maximum upstream of 2 Mbps. In addition, Peer P_1 also serves as as coordinator to manage the peer group. This is, however, only of marginal interest to this evaluation. In the second scenario, we add background traffic to Peer M to simulate M actually executing an application that uses bandwidth. To do so, M downloads a 175 MB file via BitTorrent. In the third scenario, we induce background traffic at Peer F, i.e. not at the measuring peer itself, but in the same LAN. Thus, in this scenario M has to share its Internet connection with another active peer. The background traffic is generated – similar to the second scenario – by letting F download a 175 MB file via BitTorrent. The values of additional parameters in our experiments are listed in Figure 2:

Threshold parameter T	0.5
minimal injection L_{min}	30 steps
maximum injection L_{max}	120 steps
L history weight α	0.2
aBW history weight β	0.01
Injection size per peer	1MByte
Step length	1s

Fig. 2. Evaluation Parameters

With these three scenarios we perform a number of experiments, focussing on a different aspect of our approach. First, we want to demonstrate the basic functionality of our approach, including the dynamic adaptation of L (see Figure 3–5). After that, we show the effects of resetting the decay rate under load (see Figure 6). Then, we show the influence of the threshold parameter T on the reported available bandwidth (see Figure 7) before we discuss how a possible overestimation influences our system and the results (see Figure 8). Finally, we show the overhead of our approach, including the effect of the dynamic adaptation of L on it (see Figure 9).

A. No Background Traffic

In our first experiment, we estimate the available bandwidth aBW at Peer M without any background traffic initiated in the local network. Thus, the measured aBW should be close to the capacity of the DSL downlink (3.7 Mbps). Figure 3 shows the resulting values for the lower bound lB, the upper bound uB, the estimated capacity C, and aBW. We can see that our approach reports a constant measured capacity of 3.7 Mbps. The stable measurements lead to an increase in the decay length interval over time and we can see that the reported aBW increases over time with the stability of the measurements. Note that it stays around 3 MB/s and does not reach the upper bound. This is intended behaviour in order to avoid overestimations. Also note that the lack of background traffic is reflected by the fact that lB is 0 during almost the entire measurement. We can also observe the dynamic adaptation of the parameter L. Initially traffic injections are performed at short intervals, as we have no information about the stability of uB. Since uB is almost completely stable, the interval between injections continually increased until it reaches L_{max}. We can also see the effects of the decay function, which continually reduces the estimated capacity C between traffic injections. After an injection, C is once again set to uB.

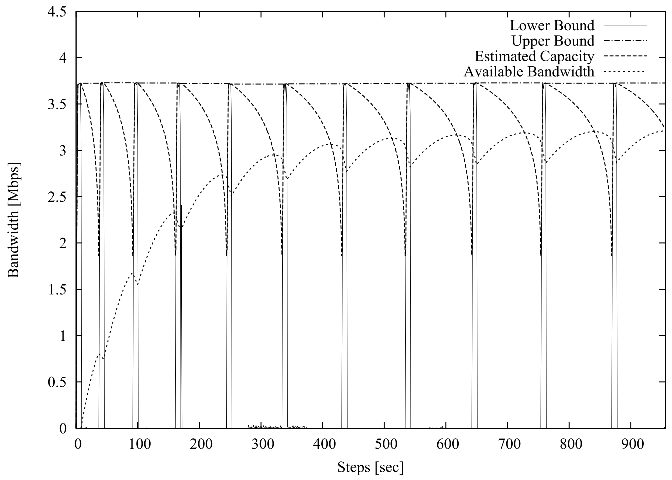

Fig. 3. No background traffic

B. Background Traffic at M

In the second experiment we are generating background traffic on Peer M, the same peer on which we are estimating the available bandwidth aBW. The results are depicted in Figure 4. Initially, there is no background traffic and the results are similar to the first experiment. After approximately 480s, the download of a popular 175 MB file is initiated

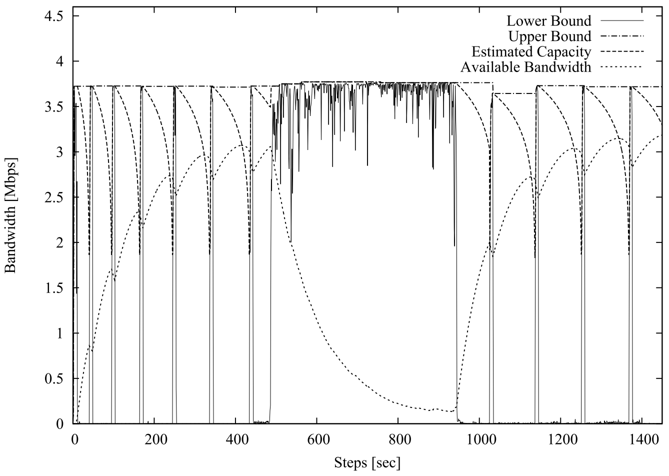

Fig. 4. Background traffic at M

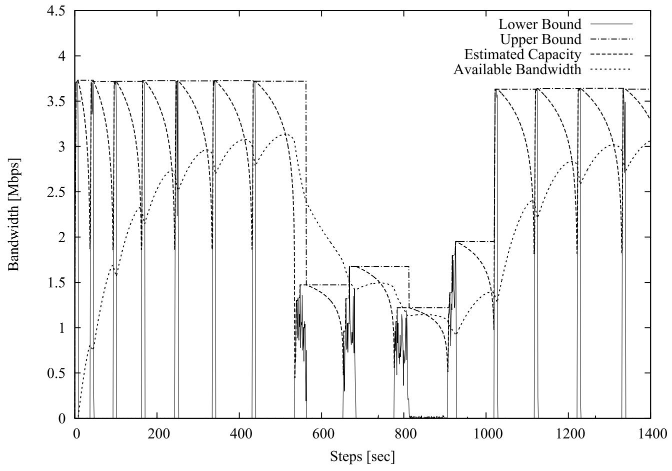

Fig. 5. Background traffic at F

via BitTorrent on M. During the download nearly all of the available bandwidth is used for it. Thus, the passive traffic observation of our algorithm should return a good estimate of the available bandwidth. This can be seen by looking at the lower bound lB. During the download it is near the capacity of the DSL connection with sporadic peaks down to 3 Mbps, in rare cases down to 2.5 Mbps. During this time, the reported available bandwidth aBW decreases to approximated 0.2 Mbps. After the download is finished at approximately 940s , lB – i.e. the observed traffic – is 0 again. Now, the bandwidth estimation requires traffic injections in order to determine aBW. One can see a smooth increase of aBW until it reaches approximately 3 MB/s again at about 1300s.

C. Background Traffic at F

In this scenario we are generating background traffic on Peer F, by downloading a 175 MB file via BitTorrent. This is similar to the second scenario, except that – while the traffic is still generated in the same LAN – this is done on a different host. In the first two scenarios, the upper bound uB remains constant. In the first scenario, this was due to a lack of traffic. In the second scenario, this was the case because the traffic was generated at the same host. By generating traffic on F, which Peer M cannot directly measure, we are reducing the amount of traffic that can be sent by M, thus reducing uB.

Figure 5 shows the measured results over the experiment's time, as in the previous scenarios. Again the experiment starts and the decay length is adapted and increases while uB is increasing to 3 Mbps and the estimated capacity C is 3.7 Mbps. When the download starts at approximately 500s, the available bandwidth aBW decreases. Here, uB drops down to ca. 1.4 Mbps. Additional traffic injections are performed when C reaches the threshold. Note the changes of uB, based on the background traffic generated by F during the traffic injections. Also note the decrease of the injection interval, due to the dynamic adaptation of L based on the decreased stability of uB. Interestingly, our approach reports values for aBW above 0 even in situations where Peer F uses up all its available bandwidth. This is correct, as M and F share the same Internet connection, i.e., F is not entitled to use up all bandwidth. Thus, when M initiates a traffic injection, F's download speed decreases, as part of the DSL connection is used up by the traffic injection. After the download is completed, uB is once again adjusted and the reported available bandwidth returns to previous levels.

D. Capacity Decay

Figure 6 shows an excerpt of 250 seconds from another experiment in the second scenario, i.e., where background traffic is generated on M. This excerpt shows the behaviour of the capacity decay while significant traffic is generated on the peer. We can see that the estimated capacity C is almost constant and very close to the upper bound uB. This is due to two reasons. First, the high volume of traffic leads to a high lower bound lB, which in turn leads to a threshold which is close to uB. That means that C decreases very little every step. Second, due to the high volume of traffic we regularly reset the decay rate. This means that the high volume of observed traffic allows us to keep C almost constant. This in turns allows us to keep delaying the next traffic injection, reducing overhead under high load (when this overhead can impact system performance). Only when the traffic stops, does C start falling at an increasing rate again, finally leading to a traffic injection as C reaches the threshold.

E. Influence of Threshold

Figure 7 depicts the influence of the threshold on the reported bandwidth aBW. We have run three experiments in the first scenario, i.e., with no background traffic generated, each with a different setting for the parameter T. T determines the position of the threshold between the upper and lower bounds, which in turns influences the estimated capacity C. With lower values for T we are expecting lower aBWs and vice versa. As we can see, our experiments confirmed this. The higher the threshold, the higher aBW. However, this in turn results in a higher chance of overestimation. This is a

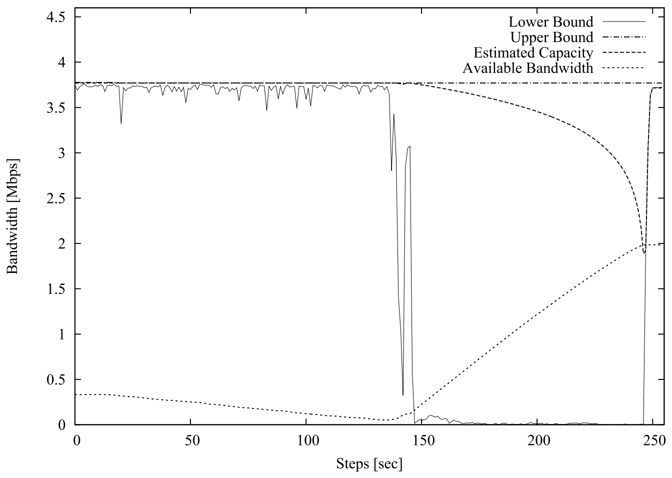

Fig. 6. Behaviour of estimated capacity under high observed traffic

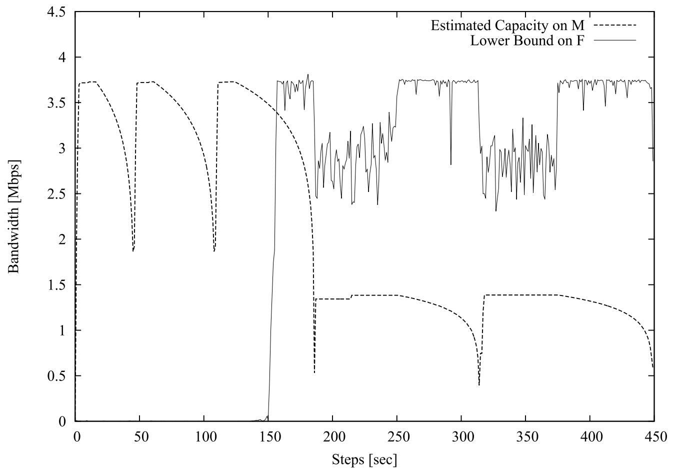

Fig. 8. Overestimation

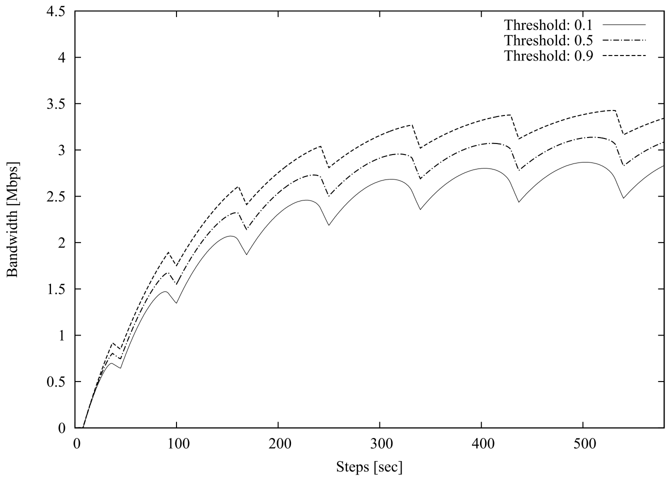

Fig. 7. Influence of threshold on reported available bandwidth

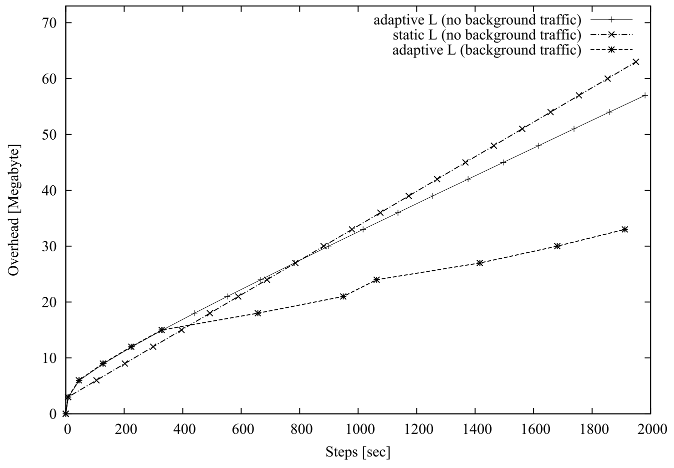

Fig. 9. Overhead of traffic injection

trade-off which must be selected by the application. Note that we are reporting aBW above 0 for all three settings of T. This means that even for very low settings of T our approach does not underestimate the available bandwidth to the point where the result cannot be used any more.

F. Overestimation

Our approach cannot completely prevent overestimations. Specifically, overestimation can still occur in between two traffic injections, when the capacity drops below the current estimate. Our decay function is designed to take this into account and thus reduces the estimated capacity C over time. While this always mitigates a possible overestimation, it cannot always completely prevent it. Figure 8 shows an example for such a situation. It depicts an excerpt from an experiment in third scenario, i.e., where background traffic is generated on F. The figure shows C on M as well as the traffic generated on F, i.e., the lower bound lB on F. At about 140s, a download is initiated at F. From this point until about 180s we can observe that F is using almost all available bandwidth of the internet connection. However, C on M is initially still estimated at about 3.5 Mbps. During this time, the system suffers from an overestimation. The decay function continuously decreases the amount of overestimation, until C is once again below the actual capacity. Shortly afterwards, another traffic injection is initiated and the upper bound uB is adjusted, which is reflected by an estimation of C of about 1.4 Mbps. Note that we can also observe the impact that this traffic injection has on the lower bound of F.

G. Overhead

Finally, we are looking at the bandwidth overhead generated by our approach. Figure 9 shows the total additional bandwidth generated over time. We show the overhead for both the first scenario (no background traffic) and the second scenario (background traffic generated). For the second scenario we have initiated two subsequent downloads of a 175 MB file via BitTorrent.

We can see the reduction in overhead created by the passive observation of the traffic generated in the second scenario. As the decay is reset during this observation, the interval between traffic injections increases, which reduces the overhead.

We also compare our results to a version of the algorithm which uses a static L, set at the middle between L_{min} and L_{max} as $L = \frac{L_{min}+L_{max}}{2}$. Compared to the static approach, the adaptive version initially shows slightly higher overhead, as it starts with a smaller L and thus performs more injections. However, due to stable capacity and high observed traffic both adaptive versions perform better over time as the number of injections is reduced.

Note the result of the static approach does not change when background traffic is introduced, so we omit this result. The result for the adaptive approach in the third scenario (background traffic initiated on F) is also identical to the result for the adaptive approach in the first scenario, so we omit this as well.

V. Related Work

Determining the available bandwidth between two nodes is a well-known problem [11]. However, most approaches focus on estimating the data rate for a specific connection between two hosts (e.g., [8] [9] [7] [6] [12]). In contrast, we are interested in the overall network capacity of a peer, which it can utilize for connections to other peers. This information can be used to determine the peer's suitability to, e.g., host critical data or perform system coordination tasks. For example, we are working on efficient update propagation strategies in P2P networks that require the current available bandwidth for coordinator elections as well as overlay construction [15]. In the following we discuss existing P2P approaches that deal with measurements of a peer's total capacity.

Eigenspeed [14] uses passive observation only. Each peer observes the traffic to its neighbours in the overlay and stores the maximum bandwidth it has achieved with each of them. This information is then aggregated across multiple nodes and combined into a consensus view using principal component analysis. The primary contribution of Eigenspeed is on the issue of trust. It assumes that peers cannot be trusted to report their own bandwidth to the system. Thus the focus is on preventing attackers or colluding groups of attackers from intentionally reporting incorrect bandwidth. From the viewpoint of bandwidth estimation, Eigenspeed has a number of drawbacks. First, little traffic between two peers can lead to significant underestimations. This is mitigated somewhat by combining observations from multiple peers. However, this does not work if a peer is experiencing low traffic volume to all its neighbour peers. Second, Eigenspeed assumes stable network capacities. In contrast to this, our approach gets more accurate results in low traffic scenarios by injecting traffic periodically and is able to handle fluctuating network capacities.

ThunderDome [4] uses active measurement only. Its key concept is the use of pairwise active measurements between peers, called bandwidth probes. As the result of such a measurement is always limited by the slowest member of each pair, ThunderDome proposes an algorithm that schedules pairwise bandwidth probes for all peers in the system. Using a tournament system, peers whose bandwidth has not yet been determined are repeatedly paired and execute a bandwidth probe. This approach takes some time to converge. The authors first prove a logarithmic lower bound for the tournament's runtime, but also show that improvements are possible under certain assumptions about the bandwidth distribution in the system [5]. However, as with Eigenspeed, the results may already be outdated once the tournament has finished. Also, the system relies on existing connections between peers. Thus the bandwidth is depending on the underlay characteristics of the connections. Alternative connections might show different behavior. Also, the estimation may change if the background traffic changes during measuring the different connections of a peer.

In contrast to these two approaches, our bandwidth estimation is a combination of passive traffic observation and active traffic injection. Hence, results are immediately available. We do not assume static capacity of the traffic/channels and adaptation to fluctuations in capacity can be done quickly.

VI. Conclusion and Future Work

Bandwidth determination is important in P2P systems. e.g., in order to determine coordinators or for content placement. Due to the potentially changing and unknown topology, approaches that measure the bandwidth of a connection between peers are not suitable. Thus we aim to determine the overall bandwidth that a peer can contribute to the system. We have presented an adaptive approach that combines passive traffic observation and active traffic injection. By doing so, we have created a system which provides accuracy and responsiveness while minimizing system interference through overhead.

We discussed three major challenges for bandwidth estimation, i.e., imprecise specification of connectivity, sharing of connections and mobility. Our approach is evaluated in fixed network settings coping well with the first two challenges. In future work we want to integrate mobile peers. Mobility will lead to rapid changes in the topology. Peers will change the subnet encountering a different capacity and possibly a transition delay. After changing the subnet, a new bandwidth estimation has to be determined. Mobility prediction can be used to determine when new estimations have to be done.

We have also assumed that we can select a peer group with sufficient upload speed to provide meaningful traffic injections. So far, this is done with a fixed number of peers in a peer group. In the future we want to develop techniques to select peer groups dynamically, which are both large enough to guarantee sufficient bandwidth to correctly determine the upper bound, and small enough to reduce unnecessary overhead.

References

[1] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, ser. WPES '07. New York, NY, USA: ACM, 2007, pp. 11–20.

[2] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 389–400.

[3] B. Biskupski, R. Cunningham, J. Dowling, and R. Meier, "High-bandwidth mesh-based overlay multicast in heterogeneous environments," in *Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, ser. AAA-IDEA '06. New York, NY, USA: ACM, 2006.

[4] J. R. Douceur, J. W. Mickens, T. Moscibroda, and D. Panigrahi, "Thunderdome: discovering upload constraints using decentralized bandwidth tournaments," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 193–204.

[5] J. Douceur, J. Mickens, T. Moscibroda, and D. Panigrahi, "Collaborative measurements of upload speeds in p2p systems," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –9.

[6] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *Proceedings of the Passive and Active Measurement Workshop*, March 2002.

[7] K. Lai and M. Baker, "Nettimer: a tool for measuring bottleneck link, bandwidth," in *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, ser. USITS'01. Berkeley, CA, USA: USENIX Association, 2001.

[8] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, "Bandwidth estimation in broadband access networks," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 314–321.

[9] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE.*

[10] Ofcom, "Uk broadband speeds: The performance of fixed-line broadband delivered to uk residential consumers," Tech. Rep., May 2010.

[11] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *Network, IEEE*, vol. 17, no. 6, pp. 27 – 35, nov.-dec. 2003.

[12] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in *In Proceedings of the Passive and Active Measurement Workshop*, 2003.

[13] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia Systems*, vol. 9, pp. 170–184, 2003, 10.1007/s00530-003-0088-1.

[14] R. Snader and N. Borisov, "Eigenspeed: secure peer-to-peer bandwidth evaluation," in *Proceedings of the 8th international conference on Peer-to-peer systems*, ser. IPTPS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 9–9.

[15] R. Sueselbeck, G. Schiele, S. Seitz, and C. Becker, "Adaptive update propagation for low-latency massively multi-user virtual environments," in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, aug. 2009, pp. 1 –6.

[16] Z. Xu and Y. Hu, "Sbarc: A supernode based peer-to-peer file sharing system," in *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, june-3 july 2003, pp. 1053 – 1058 vol.2.

[17] B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin Heidelberg, 2002, vol. 2429, pp. 34–44.