On-line measurement for proactive optimization of bandwidth usage

Laurent Fasnacht

Abstract

This report describes a method for estimating the effective bandwidth of the internet connection between two peers based on packet latency. The method is intended to be used for efficient data transmission in combination with forward error correcting codes.

1 Introduction

Reliable data distribution over the internet is mostly based on TCP (transmission control protocol). With TCP, the rate of data transmission between two nodes decreases as the distance between the two nodes increases. In a globalized internet, it becomes more and more difficult to make optimal use of the broad bandwidth now commonly available in private households. In the case of one-to-many data distribution, large content providers do fix the problem by deploying expensive content delivery networks.

The success of new bandwidth-demanding peer-to-peer applications such as telepresence and high definition video-conference will have to rely on transmission protocols that make more effective use of the available bandwidth. Extensions to the TCP protocol have also been proposed (for instance FAST TCP [1]) showing that monitoring packet latency can improve the effectiveness of bandwidth usage.

Forward error correcting codes (FEC) on top of standard UDP protocol have proved to provide reliable and efficient data transmission over long distance and/or lossy networks. In particular, using fountain codes it is possible to saturate any available bandwidth by flooding the connection with an arbitrary (large) number of repair packets. Although optimal for the two peers, this method is not very fair to the users sharing the same link and will end up wasting resources. In fact, the sender transmission rate should not exceed the effective bandwidth linking the two peers.

This report will focus on the algorithmic and modeling aspects of this project as the software engineering issues are better described directly in the documentation of the computer codes. Nevertheless, the latter is equally (or even more) important because our method requires a reliable measurement of packet latency and precise control over transmission rates that can only be achieved with a very careful design of the programs.

2 Model

In the internet, two peers are connected through a series of physical links. The maximum speed at which the two peers can communicate (capacity of the channel) is at most the speed of the slowest link. As the network is a shared medium, the effective capacity of the connection may change at any time: there can be changes in the concurrent usage of the slowest link, or a different link can be affected by heavy load and hence become the new link with the smallest available bandwidth. The routes could also be reconfigured during transmission.

We model the network connection between the sender S and the receiver R at any given time t as in figure 1 using two parameters.

1. the effective bandwidth $\alpha(t)$ which corresponds to the speed of the slowest link (*bottleneck* in the figure);
2. the size $B(t)$ of a packet buffer located just before the slowest link;

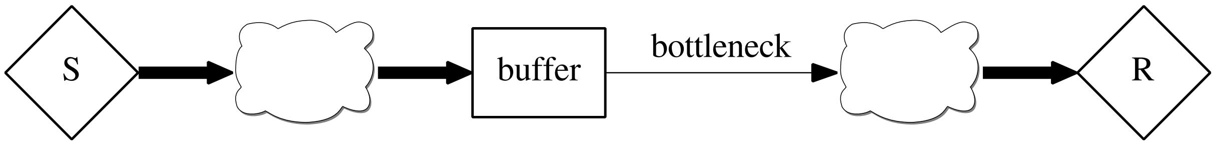


Figure 1: The channel model

The packet buffer is not always present and its size can range from very small (barely measurable) to as large as a few seconds worth of data at the speed of the corresponding link. Large buffers can avoid packet loss when data is transferred in short bursts, but they can also have a negative effect on the performance of TCP for large data transfers and on application requiring real time data transmission (small jitter).

We assume that the channel parameters do not significantly change too often over time. In particular, we assume that they will remain reasonably constant over a period of time that is considerably larger than the time we need to run our measurements. The measurements should also take only a fraction of the transmission time and be repeated periodically. In practice, as we are mostly interested in usages when at least one of the peers is in a private household, the weak link is often the broadband connection of either the sender or the receiver, and the parameters will change only when someone else in the same household is using the same broadband connection.

We also assume that, when the channel is running below capacity, the packet loss rate and latency are roughly constant[1]. Finally, we assume that all network buffers are

[1]More precisely, we assume that their values are normally distributed with small dispersion

simple FIFO queues so that there is no systematic packet reordering. Nevertheless, our implementation is indeed quite robust and will work also in the presence of sporadic packet reordering.

In the remainder of the report we will discuss the case in which there is an observable packet buffer. The case with very small or absent packet buffer can be treated in a similar way using the packet loss rate instead of the packet latency.

3 The method

The idea is to determine at what rate we are sending relative to the channel capacity by observing the latency of received packets. If we are sending at a rate that is below capacity, the buffer queue is kept empty and all packets should arrive with minimal latency. On the other hand, if we are sending at a rate that is higher than the channel capacity, the packets will be queued in the buffer and latency will increase. When the buffer is full, new packets are discarded.[2]

3.1 Probe

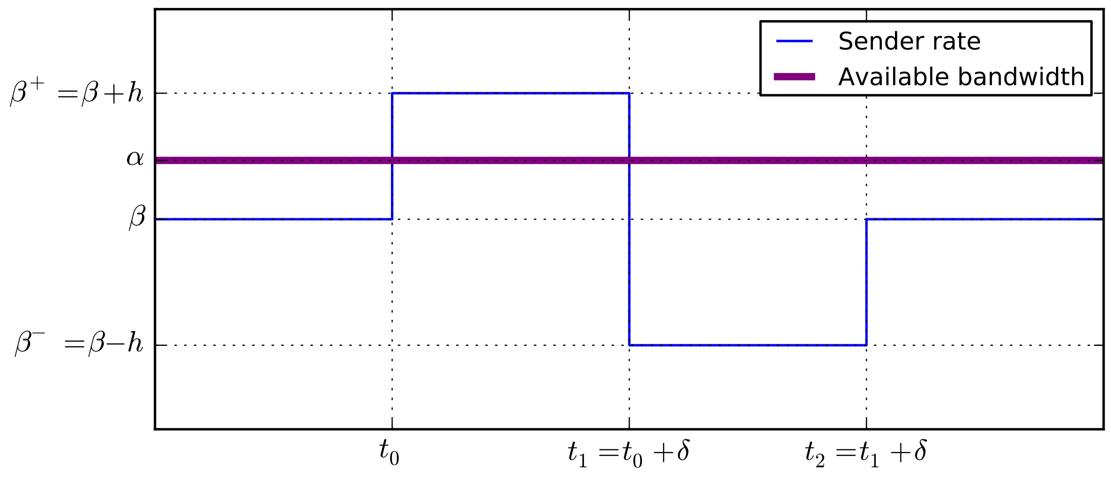


Figure 2: Changing sender rate to probe packet latency

Consider the case depicted in figure 2 where the sender is transmitting at a rate β, lower than the available bandwidth[3] α.

While keeping the average rate unchanged, we probe the packet latency by varying the instantaneous rate as a square wave pulse such that for a short period of time

[2]As a matter of fact, some network appliances are configured to start dropping a fraction of the arriving packets as soon as the buffer starts filling.

[3]The case where $\beta > \alpha$ is essentially symmetrical

δ, we are actually sending at higher rate than the available bandwidth ($\beta^+ > \alpha$). We expect the latency to increase in the first time interval while the buffer is being filled and then decrease when the rate is lowered until the buffer is empty again. Of course, if the rate step h is too small and $\beta^+ < \alpha$ we will not observe any change in the latency and will repeat the experiment with a larger step.

The packet latency is obtained on the receiver by comparing the time instants at which packet where received t_i^r relative to the time at which packets where sent t_i^r:

$$\Delta l \doteq \Delta t^r - \Delta t^s \doteq (t_2^r - t_1^r) - (t_2^s - t_1^s) \tag{1}$$

where all time intervals are relative so that there is no need to synchronize the clocks of the two peers. Similarly, the receiver will communicate back to the sender only rate values that are measured in terms of the known rate β. Therefore, there is no need to negotiate the units used to express β.

3.2 Expected outcome

In the figures in page 15 and 16, we represent how we expect the latency and the packet loss rate to change during the probing experiment in the case $\beta \leq \alpha$, and $\beta > \alpha$ respectively. Since for each experiment we expect two time intervals where the latency is not constant, we have two independent measures that can be combined to increase the precision or to discard the experiment if the two measures are not consistent.

The key idea is that if the link bandwidth remains constant during the experiment, then the buffer will fill (or empty) at a rate equal to the difference between the transmission rate and the available bandwidth α. In the $\beta \leq \alpha$ case, during a time interval Δ_1 after the beginning of the experiment t_0, the amount of data b in the buffer will increase linearly in time at a rate equal to the excess rate $g^+ \doteq \beta^+ - \alpha$:

$$b(t) = g^+(t - t_0) + b(t_0) \tag{2}$$

Assuming that the queue is a simple FIFO, a packet entering the queue will experience an extra latency equal to the time needed by the slow link to dispatch the data already queued:

$$l(t) = \frac{b(t)}{\alpha} = \frac{g^+}{\alpha}(t - t_0) \doteq \frac{\beta^+ - \alpha}{\alpha}(t - t_0) \tag{3}$$

Therefore, with a linear regression on the measured latency, we can estimate g^+ and hence the available bandwidth α. The formulas for the quantities of interest are easily determined and are summarized in the tables on pages 15 and 16 in the appendix as a reference for the implementation.

4 Method details

4.1 Protocol

There are two channels: the data channel, and the service channel.

The data channel is supposed to transport the actual transmission data using some FEC on top of standard UDP. The packets are tagged with a timestamp and with a monotonic increasing packet id that is used to compute blocks and symbol index for FEC and also to eventually avoid packet reordering problems.

The service channel makes use of TCP for reliably transferring short messages (signals) between the two peers. For example, the service channel is used by the sender `S` to communicate β to the receiver `R`. The following is an example session flowing in the service channel (`>` is from `S` to `R`, and `<` is from `R` to `S`)

```
> At <timestamp t in the future>, I'll set the rate to X
> At <timestamp t' in the future>, I'll set the rate to Y
> At <timestamp t'' in the future>, I'll set the rate to Z
[...]
< At <relative timestamp (i.e. +10s)>, set the rate to U
> At <corresponding timestamp>, I'll set the rate to U
```

Note that the sender communicates to the receiver the time instant when the rate β is changed. These are expressed in the sender's internal clock that is the same used for time stamping the packets. In this way the receiver knows what was the actual β when a given packet was sent. On the other hand, the receiver sends instructions to the sender using relative time intervals so that the actual time is always computed by the sender for consistency.

4.2 Algorithm overview

The procedure to estimate the available bandwidth consists of five steps.

1. Detect change-point candidates;
2. Merge change-points that are too close;
3. Compute linear regression parameters on all the intervals between change-points;
4. Combine and refine regressions to get the final fit of the response to the probe;
5. Estimate the bandwidth combining the information from the various slopes;

4.3 Algorithm

4.3.1 Data preparation

The probing experiment consists in varying the sending rate as a square wave pulse. We define a *time segment* as a time interval where the sending rate is constant. When the receiver `R` gets the data concerning β, it builds a list of time segments that will be used as a base for all calculations. Received packets are assigned to the *active* time segment and stored together with their *id*s as well as sent and received timestamps. Assuming no (or very little) packet reordering, as soon as a packet belonging to the next time segment is received, the current (*active*) segment is closed, and the algorithm for bandwidth estimation is executed.

Most of the change point algorithms are detecting changes on the mean value. As we are interested in the points where the slope of the latency changes, we use the derivative of the data. Therefore, we first compute the relative latency for each packet, and then do a numerical derivative with respect to the packet id.[4]

We feed the change point detection algorithm with the sequence of derivatives, $\{t\}_{1\leq i\leq n}$ by $(t_1 = 0)$:

$$t_i := \frac{y_{i+1} - y_i}{p_{i+1} - p_i}$$

where $\{p\}_{1\leq i\leq n} \in \mathbb{N}$ the sorted sequence of packet ids, and $\{y\}_{1\leq i\leq n} \in \mathbb{N}$ the sequence of relative latencies.

4.3.2 Change-points detection

Usually, change point algorithms consist of two parts. At first, a list of possible change-points is generated. Afterward, the good change-points are selected by mean of statistical tests. We are only interested in the first step, because we do not have a guess on how the data is distributed around its average value. On the other hand, we have a very strong a-priori information about how the latency should behave, and we can use this knowledge to decide whether a change-point can be considered as "valid" or not. The filtering of change-points is performed on-line and has to run as fast as possible in order to avoid disturbing the latency measurement. Statistical tests can be too expensive to compute.

After testing several alternatives, we have chosen to base our algorithm on the first part of the binary segmentation algorithm [2], with normal distribution. In practice, for the sequence $\{t_i\}_{1\leq i\leq n}$ and $1 \leq k, l \leq n$, we define the following likelihood function:

$$\mathrm{M}(k,l) \mapsto -\frac{1}{2}\left(\sum_{j=k}^{l} t_j^2 - \frac{(\sum_{j=k}^{l} t_j)^2}{l-k+1}\right)$$

[4]we use the packet id instead of timestamp because it avoids floating point exceptions by construction. At this stage, we are not interested in the actual value of the slope.

and we use the following algorithm to return the change-points candidates:

Algorithm 1 Returns at most Q change-points in $\{t_i\}_{1\leq i\leq n}$

```
1: $\tau \leftarrow \{n\}$
2: $\omega \leftarrow 1$
3: while $|\tau| - 2 < Q$ and $\omega \notin \tau$ do
4:   $\tau \leftarrow \tau \cup \{\omega\}$
5:   $\lambda_{\max} = -\infty$
6:   $\omega = 1$
7:   for $i = 1 \rightarrow n$ do
8:     $k^- \leftarrow \max_{j\in\tau, j\leq i} j$
9:     $k^+ \leftarrow \min_{j\in\tau, j>i} j$
10:    $\lambda \leftarrow \mathrm{M}(k^-, i) + \mathrm{M}(i+1, k^+) - \mathrm{M}(k^-, k^+)$
11:    if $\lambda_{\max} < \lambda$ then
12:      $\lambda_{\max} \leftarrow \lambda$
13:      $\omega \leftarrow i$
14:    end if
15:  end for
16: end while
17: return $\tau$
```

The value of Q should be large enough to be sure to find all potentially interesting change-points, but also not too large because we need to keep the running time reasonable even in the worst case. In our case, $Q = 10$ seems to be a good compromise.

It's important to note that it is possible to write a very efficient implementation where line 10 is vectorized by pre-computing the partial sums and redefining M as follows:

$$\mathrm{M}(k,l) \mapsto -\frac{1}{2}\left((v_l - v_{k-1}) - \frac{(u_l - u_{k-1})^2}{l-k+1}\right) \tag{4}$$

where $u_i := \sum_{j=1}^{i} t_i$, $\quad v_i := \sum_{j=1}^{i} t_i^2$, $\quad u_0 := 0$, and $\quad v_0 := 0$.

4.3.3 Filtering change-points

Let's call $\{c_k\}_{1\leq k\leq m}$ the sorted list of change-points (i.e. $c_k \leq c_{k+1}$) including the two time interval boundaries $c_1 = 1$ and $c_m = n$. Each pair of successive change points determines a time interval on which we are going to run a linear regression.

Too short time intervals are merged with their respective shortest neighbor hence discarding the change point between them. The algorithm used is the following:

Algorithm 2 Remove change-points from $\{c_k\}_{1\leq k\leq m}$ which interval is smaller than some constant ξ, the number of points

```
1: while $\exists k$ such that $c_{k+1} - c_k < \xi$ do
2:    if $c_{k-1} - c_k < c_{k+2} - c_k$ (assume $c_0 = 1$ and $c_{m+1} = n$) then
3:       Delete $c_k$ from the sequence
4:       $m \leftarrow m - 1$
5:    end if
6: end while
7: return $\{c_k\}_{1\leq k\leq m}$
```

4.3.4 Linear regression

We fit the $n = c_{k+1} - c_k + 1$ latency measures $\{(x_i, y_i), i \in [1, n]\}$ between two successive change points c_k, and c_{k+1} with a straight line $f(x) = mx + h$. In this simple case, the least square method reduces to computing the regression parameters m and h by solving the 2×2 linear system $Ap = b$ where

$$p = \begin{bmatrix} h \\ m \end{bmatrix}, \quad A = \begin{bmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix}, \quad b = \begin{bmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{bmatrix} \tag{5}$$

Note that merging two regressions (*e.g.*, for joining two subsequent time intervals into a single one) is an inexpensive operation if we keep the As and bs in memory because we just have to cumulate the sums by adding the corresponding matrices.

Algorithm 3 Compute $A \in \mathbb{R}^{2\times 2}$ and $b \in \mathbb{R}^2$ such that $A \cdot [h, m]^\mathsf{T} = b$, where $y = mx + h$ is the least square regression line between c_k and c_{k+1}

```
$a_{11} \leftarrow c_{k+1} - c_k + 1$
$a_{12} \leftarrow 0, a_{22} \leftarrow 0, b_1 \leftarrow 0, b_2 \leftarrow 0$
for $i = c_k \rightarrow c_{k+1}$ do
   $a_{22} \leftarrow a_{22} + x_i^2$
   $a_{12} \leftarrow a_{12} + x_i$
   $b_1 \leftarrow b_1 + y_i$
   $b_2 \leftarrow b_2 + x_i \cdot y_i$
end for
return $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$
```

4.3.5 Refinement

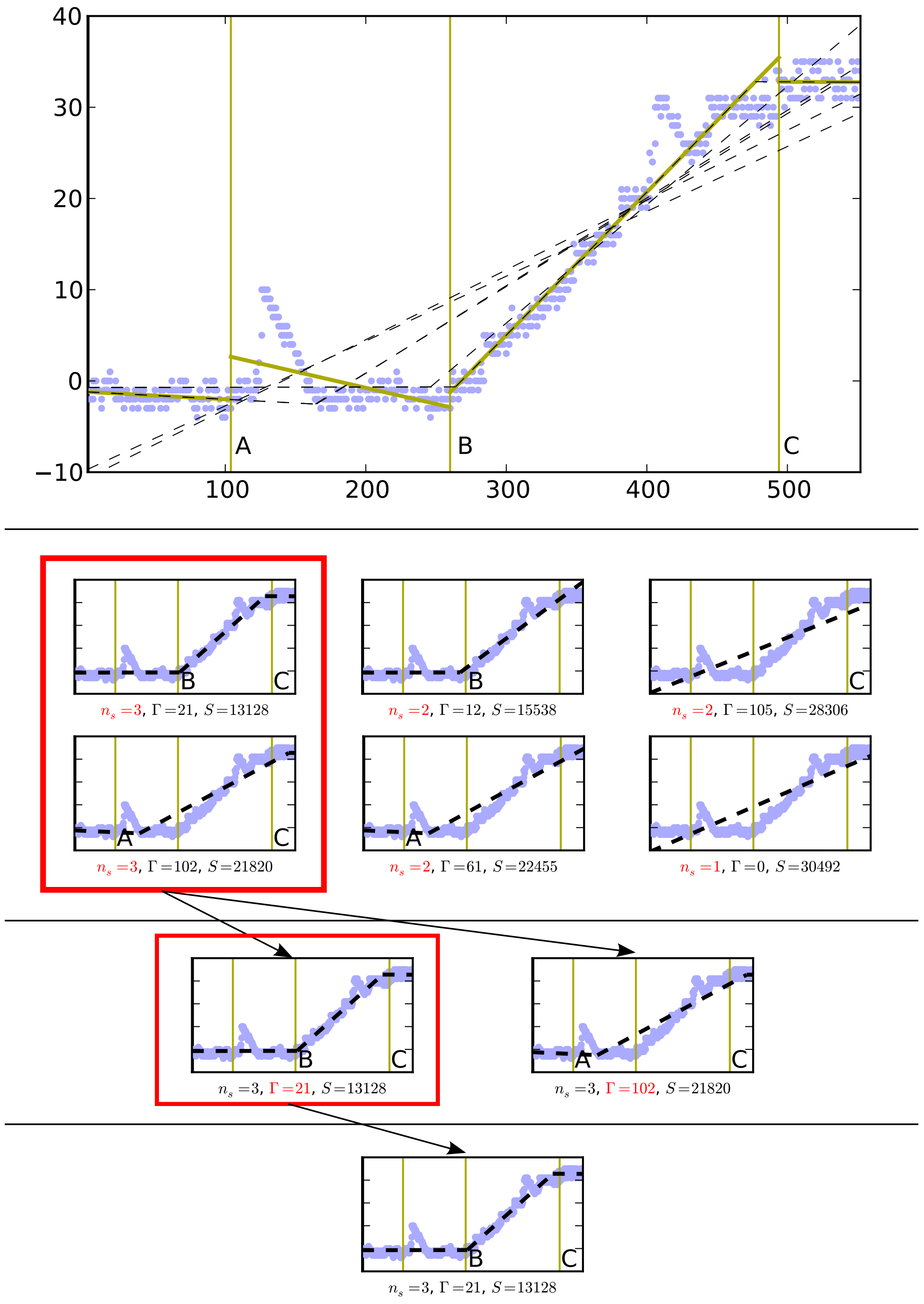


Figure 3: Selection of the line best fitting the latency data. First the two lines with three segments are selected, then the one with the smallest Γ is kept. As expected, the selected line is the one having the smallest sum of square residuals S.

The next step consists in refining the set of change-points based on how the response of the latency to the probe is expected to be. An example of our procedure is depicted in figure 3.

Consider, for example, the first time segment of an experiment where $\beta < \alpha$ (the case $\alpha > \beta$ is essentially symmetrical). We expect the latency to immediately start increasing and then eventually stay constant once the buffer is full. In fact, we have observed situations in which the latency remains constant for a short time before starting to increase.[5] Therefore, we model our response on the latency with a continuous line consisting of up to three straight segments:

1. an optional flat segment (when an excess traffic burst is tolerated);
2. a mandatory oblique segment (while the buffer is filling up);
3. an optional flat segment (once the buffer is full).

where a segment is considered flat if its slope is smaller thans a given threshold.

We first enumerate all possible combinations that are compatible with our response model and then we sort them according to the following criteria:

1. those with highest number of segments are selected;
2. those with smallest Γ (see figure 4) are selected;
3. those with steepest slope are selected;
4. the one with largest memory address is selected.

The criteria are applied one after the other till there remains only one candidate. In practice step 3 and 4 are almost never used.

The selection between two lines with the same number of segment is done by comparing the distance Γ between the selected change point and the crossing point of the two segment as described in figure 4.

This procedure gives very good results and is considerably less expensive than computing the sum of square residuals for each candidate.

[5]This is the case when the bandwidth of the slowest link is limited artificially (as for most of the ADSL or cable connections) but the limitation is enforced with a little delay, hence allowing for short traffic bursts.

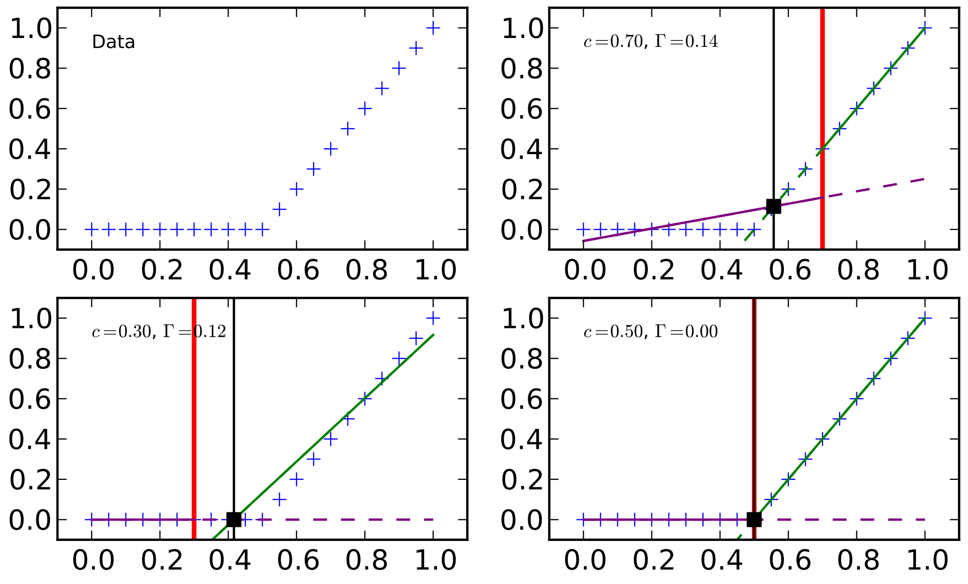


Figure 4: The time interval Γ is defined as the distance between the change point indicated by the red line, and the intersection between the purple and green lines that are obtained fitting the data before and after the change point.

5 Examples

In this section, we present two examples to show the algorithm in action.

5.1 Real-world cable connection

The first example is shown in figure 5. The data was measured on a household cable internet connection providing a nominal upload bandwidth of 7 Mb/s (which corresponds to about 910 packets/s). It is interesting because it presents most of the features discussed above. In particular,

- there is a burst between 50ms and 300ms (this justifies the need of a changepoint detecting algorithm),
- the change-point candidate at 170ms was discarded because of the large Γ value, and
- the bandwidth estimate is in close agreement with the nominal value also verified by a saturation test.

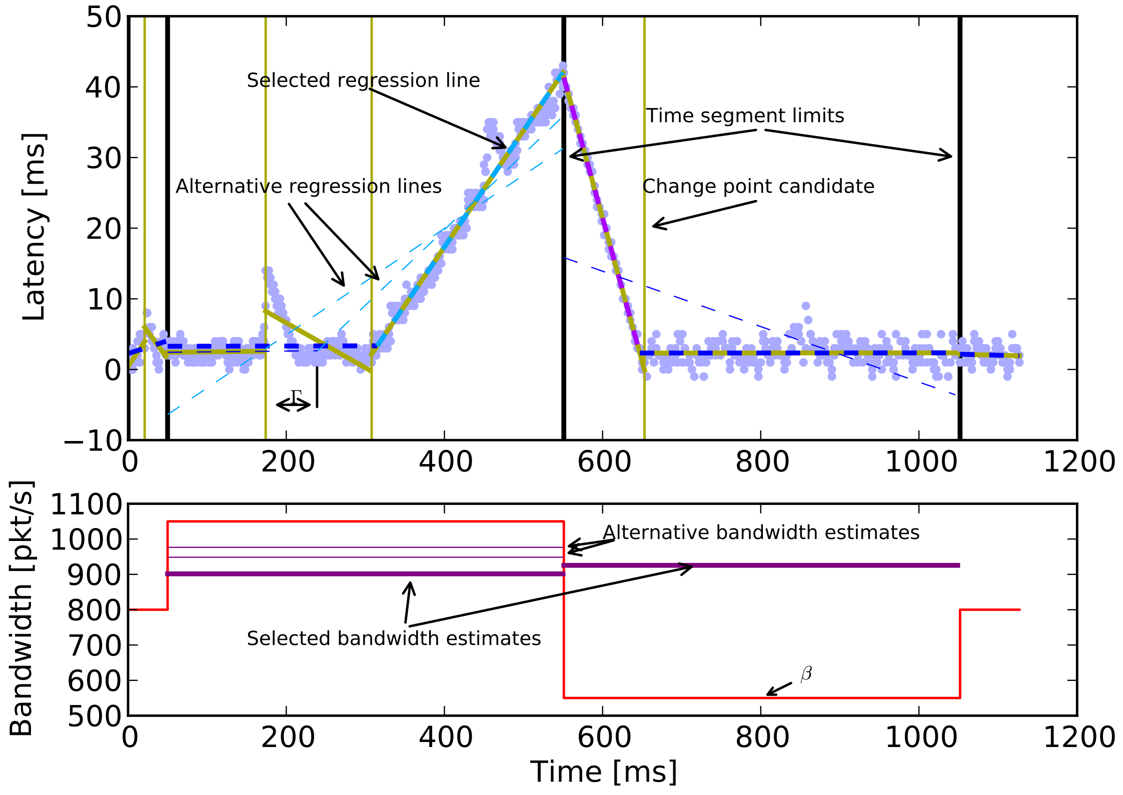


Figure 5: Measurement of the upload bandwidth of a cablecom internet connection

5.2 Controlled case using the simulator

The second example is shown in figure 6 and is based on a test case generated by an ad-hoc simulator where we can control the parameters of the model (bandwidth and buffer size) as well as the noise. With this example we can show how the bandwidth estimation converges to the actual value after few probing experiments.

The channel bandwidth is fixed at 100 KB/s while the initial send rate is set at 110 KB/s. Since in this case $\beta > \alpha$, at the very beginning of the simulation the buffer is already filling and the latency is increasing.

Note that although we obtain a very good estimation of the bandwidth by fitting the very first latency increase, we deliberately keep the initial rate till the buffer is full in order to show (test) how the algorithm behaves in the saturated case.

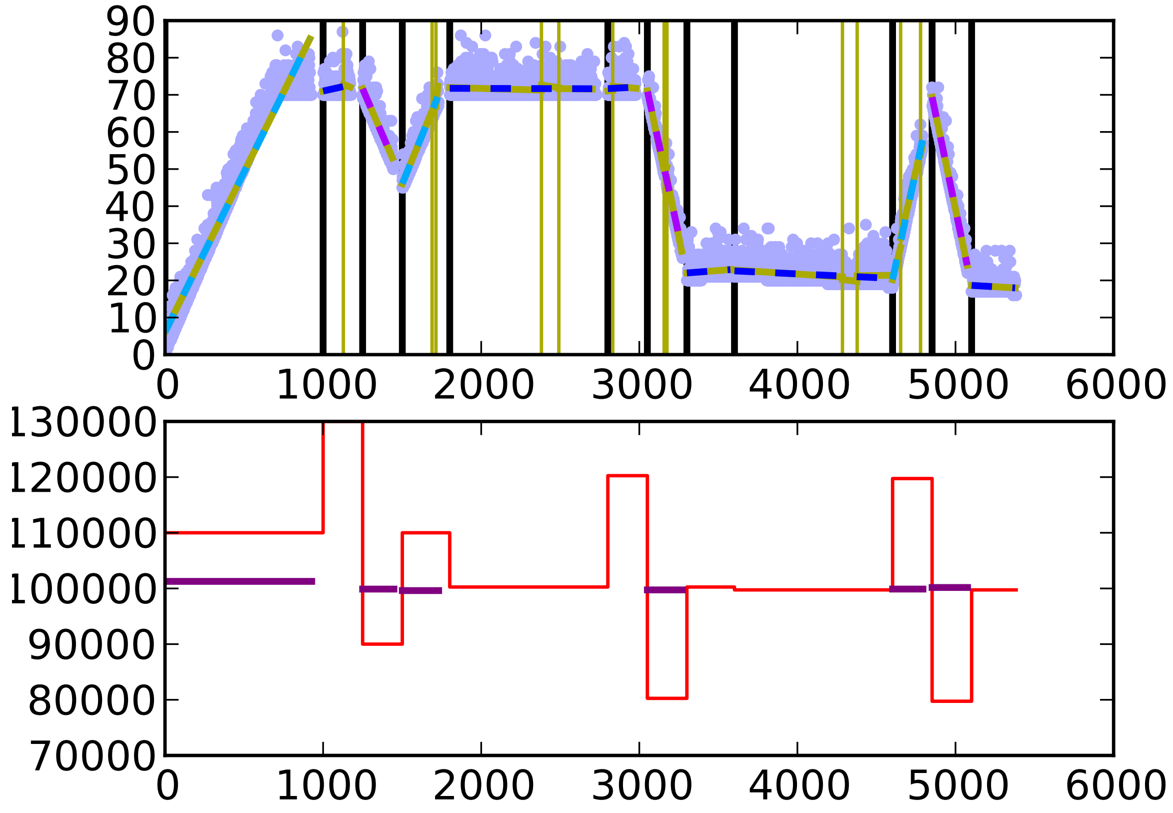


Figure 6: Results of the algorithm run on-line in the simulator

6 Conclusion

We presented an on-line algorithm to estimate the bandwidth effectively available between two peers. The method is based on latency and can be implemented very efficiently. We believe that, combined with other methods (as those based on packet loss), it could make a robust and reliable system.

Future improvement will consist in implementing fallback methods, improving detection of changes between experiments, and defining more precise criteria to detect imprecisions in the results.

We also plan to test our system in many different situations by providing a web service where the software can be downloaded and the measurement results can be accumulated.

Finally, an element that remains to be studied is how our system behaves in presence of other systems for improving bandwidth usage.

References

[1] Wei, David X.; Jin, Cheng; Low, Steven H. and Hegde, Sanjay FAST TCP: motivation, architecture, algorithms, performance In *IEEE/ACM Transactions on Networking*, 14(6):1246-1259, Dec 2006

[2] A. J. Scott and M. Knott. A Cluster Analysis Method for Grouping Means in the Analysis of Variance. In *Biometrics*, Vol. 30, No. 3, (Sep., 1974), pp. 507-512

7 Acknowledgments

The author humbly thanks Dr Giovanni Cangiani for his precious suggestions, especially during the writing of this report.

Appendix

In this appendix we explain some of the notations and formulas used for the implementation of our system.

The time intervals are obtained from the following expressions (note how the two cases considered are in fact very similar):

	case $\beta \le \alpha$		case $\beta > \alpha$	
max fill level:	ω	$= \min(B, g^{+}\delta)$		
min fill level:			ψ	$= \max(0, B + g^{-}\delta)$
time for filling:	Δ_1	$= \min(\frac{B}{g^{+}}, \delta)$	Δ_2	$= \frac{B-\psi}{g^{+}}$
		$= \frac{\omega}{g^{+}}$		
time for emptying:	Δ_2	$= \frac{\omega}{-g^{-}} \le \delta$	Δ_1	$= \min(\frac{B}{-g^{-}}, \delta)$
				$= \frac{B-\psi}{-g^{-}}$

where

δ is the width (duration) of the send rate step;

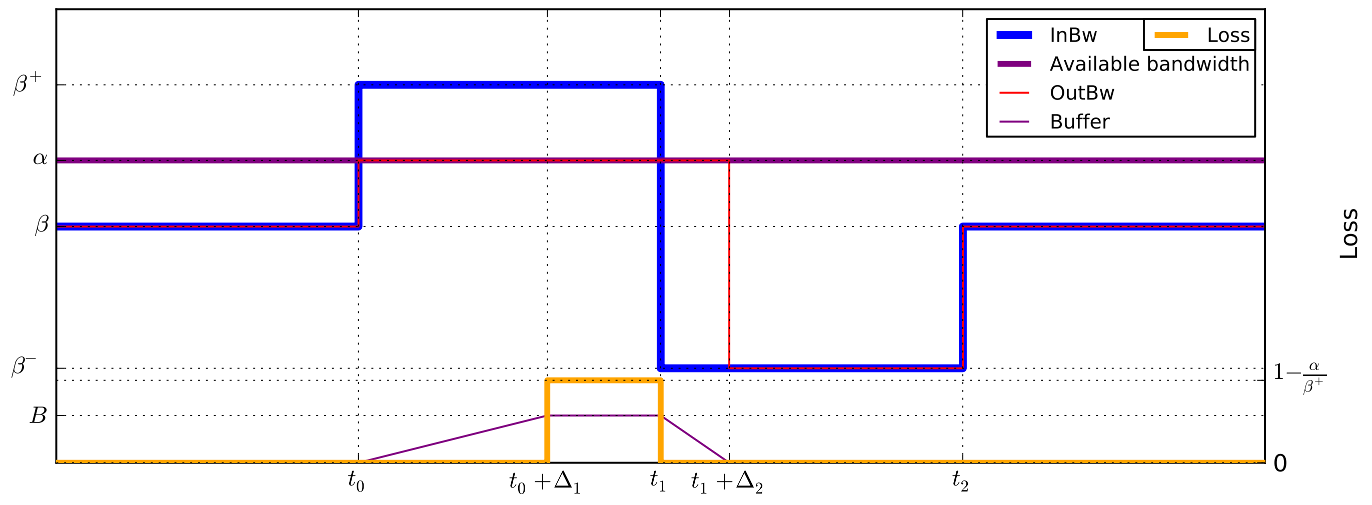
h is the height of the send rate step;

B is the available buffer size;

α is the available bandwidth;

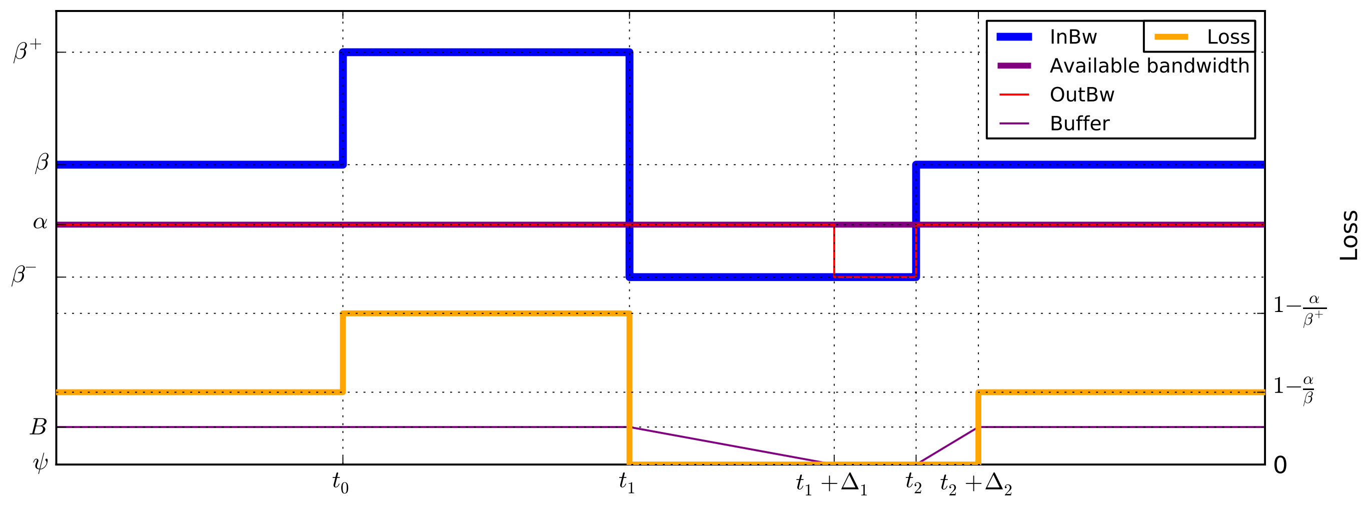
β is the average (steady state) transmission rate and
$\beta^{+} \doteq \beta + h$, $\beta^{-} \doteq \beta - h$, $g^{+} \doteq \beta^{+} - \alpha$, $g^{-} \doteq \beta^{-} - \alpha$

<u>$\beta \leq \alpha$:</u>



t	InBw [pkt/s]	OutBw [pkt/s]	Buffer [pkt]	Latency [s]	Loss rate
$]-\infty; t_0]$	β	β	0	0	0
$[t_0; t_0+\Delta_1]$	β^+	$\min(\alpha, \beta^+)$	$g^+ \cdot (t - t_0)$	$\frac{g^+ \cdot (t-t_0)}{\alpha}$	0
$[t_0+\Delta_1, t_1]$	β^+	$\min(\alpha, \beta^+)$	ω	$\frac{\omega}{\alpha}$	$\left(1-\frac{\alpha}{\beta^+}\right) \cdot (B \overset{?}{<} \delta(g^+))$
$[t_1, t_1+\Delta_2]$	β^-	$\min(\alpha, \beta^+)$	$\omega + g^- \cdot (t - t_1)$	$\frac{\omega+g^- \cdot (t-t_1)}{\alpha}$	0
$[t_1+\Delta_2, t_2]$	β^-	β^-	0	0	0
$[t_2, \infty[$	β	β	0	0	0

<u>$\beta \leq \alpha$:</u>



t	InBw [pkt/s]	OutBw [pkt/s]	Buffer [pkt]	Latency [s]	Loss rate
$]-\infty; t_0]$	β	α	B	$\frac{B}{\alpha}$	$1-\frac{\alpha}{\beta}$
$[t_0; t_1]$	β^+	α	B	$\frac{B}{\alpha}$	$1-\frac{\alpha}{\beta^+}$
$[t_1; t_1+\Delta_1]$	β^-	α	$B+g^-\cdot(t-t_1)$	$\frac{B+g^-\cdot(t-t_1)}{\alpha}$	0
$[t_1+\Delta_1; t_2]$	β^-	$\beta^-\cdot(\psi \overset{?}{=} 0)+\alpha\cdot(\psi \overset{?}{\neq} 0)$	ψ	$\frac{\psi}{\alpha}$	0
$[t_2, t_2+\Delta_2]$	β	α	$\psi+g\cdot(t-t_2)$	$\frac{\psi+g\cdot(t-t_2)}{\alpha}$	0
$[t_2+\Delta_2, \infty[$	β	α	B	$\frac{B}{\alpha}$	$1-\frac{\alpha}{\beta}$