Explicit Congestion Control Algorithms for Variable Capacity Media

Filipe Abrantes, João Araújo, and Manuel Ricardo, *Member, IEEE*

***Abstract*—Explicit congestion control (XCC) is emerging as one potential solution for overcoming limitations inherent to the current TCP algorithm, characterized by unstable throughput, high queuing delay, RTT-limited fairness and a static dynamic range that does not scale well to high bandwidth delay product networks. In XCC routers provide multi-bit feedback to sources, which in turn adapt throughput more accurately to the path bandwidth with potentially faster convergence times. Such systems however require precise knowledge of link capacity for efficient operation. In the presence of variable capacity media, e.g 802.11, such information is not entirely obvious or may be difficult to extract. We explore three possible algorithms for XCC which retain efficiency under such conditions by inferring available bandwidth from queue dynamics and test them through simulations with two relevant XCC protocols: XCP and RCP. Additionally, preliminary results from an experimental implementation based on XCP are presented. Finally, we compare our proposals with TCP and show how such algorithms outperform it in terms of efficiency, stability, queuing delay and flow-rate fairness.**

***Index Terms*—XCP, RCP, Congestion Control**

I. Introduction

TCP [14] has been responsible for congestion control ever since Van Jacobson's algorithm [8] was embedded into the transport protocol in the late 80's. Despite performing its task remarkably well, it degrades network performance due to increased queueing delay, unstable throughput and limited fairness. Additionally, ever-higher bandwidth coupled with longer delays result in increasingly poor link utilization, slow adaptation to changing link loadings and significant congestion losses.

The root of the problem stems from TCP's reliance on rare events which carry a low resolution of information. As a result, TCP is left to infer too much from too little, being unable to achieve the target share of bandwidth in a timely and accurate fashion. XCC protocols propose a new approach, whereby routers take an active role in informing sources about the state of the network, helping sources to adjust their sending rate by using precise multi-bit feedback. This allows fast and accurate adaptation to network conditions, enabling stable throughput, high utilization, and low standing queues in the network.

In order to calculate the feedback given to sources however, a router needs to compute the current spare bandwidth of the outgoing link, which in turn requires knowing the exact link capacity. In variable capacity media, knowing this value and the fair share of each station is difficult as it depends on a number of factors (e.g. MAC efficiency, data and basic rates used by each station). This unpredictability of the medium capacity causes classic XCC feedback algorithms to perform poorly, inhibiting the use of XCC over variable capacity media.

In this paper we explore the design space for XCC algorithms when the link capacity is unknown or variable, by exploring three alternative algorithms designed to be used under such conditions. The first algorithm measures spare bandwidth from queue speed; the second algorithm tries to infer available bandwidth from the idle/busy times at the

MAC level; the third and final alternative algorithm allows the configuration of a fixed capacity at the router and in turn tries to estimate the difference between the configured capacity and the real medium capacity by monitoring queue build. We apply these three algorithms in simulation to both XCP [11] and RCP [6], two of the most significant XCC protocols.

The paper is organized as follows. Section II refers related work, Section III provides background on XCC operation and feedback algorithms and details the effect that a capacity estimation error has on system performance. In Section IV we present in detail the three alternative algorithms for feedback calculation in variable capacity media. In Section V we present a performance analysis of the algorithms and draw conclusions in Section VI.

II. Related work

The work presented here builds on the approaches taken by XCP-b [4] and WXCP[15]. In these papers two algorithms were proposed to extend the most proeminent XCC protocol at the time, XCP, for wireless variable capacity media. In our work we solve open issues of XCP-b, such as the dynamic configuration of its parameters, and also present modifications to WXCP that make it more practical. We also extend their applicability to RCP, another XCC protocol.

Our work also takes advantage of previous studies and tests [17], [18], [10] which have analysed the impact of a capacity estimation error in the overall performance of XCC algorithms. In the past years further theoretical studies [16], [5], [12] helped to gain a better understanding of the operation of XCC protocols, however they are not directly related with the operation of XCC protocols over variable capacity media.

III. Explicit Congestion Control (XCC) Background

XCC protocols enable queue controllers in a path to inform sources about the state of the network and how they should adapt their sending rate. This communication between sources and the network is enabled by using a header between the network and transport layers. This congestion header typically carries information about the flow to which the packet belongs, such as the throughput or the RTT. The RTT enables the XCC controller to adjust the pace of adaptation to network conditions, while the throughput may be used to decide bandwidth distribution among flows (some protocols do not assign bandwidth directly to individual flows). Based on the values of the congestion header and other local variables the XCC router will calculate a feedback value and insert it into the congestion header. The packet will reach the receiver, which will send an ackowledgement to the sender carrying the feedback inserted by the bottleneck router in the forward path.

A. Aggregate Feedback Calculation at the XCC router

The core of a XCC algorithm is the aggregate feedback calculation performed at the router, that is the amount of bandwidth F that is to be distributed among the flows during a certain control interval d. This calculation is typically a function of the available bandwidth and the standing queue in the router. In this work we focus on the particular case of XCP and RCP, two of the most relevant XCC protocols. We will discuss later how to apply our ideas to any XCC protocol. For the particular case of XCP and RCP, the calculation of F takes the form:

$$F = \alpha \cdot (C - y) - \beta \cdot \frac{q}{d} \tag{1}$$

where C is the capacity of the transmission medium, $y(t)$ is the bandwidth actually used during the last period d and q is the persistent queue or, in other words, the minimum queue length observed during the last d seconds. d is usually set to be the average RTT of the flows traversing this queue. α and

β are constants. The bandwidth allocation algorithm will then distribute F among the currently active flows, however, this is protocol specific as we explain in Section III-C. It is important to note that an XCC router controls each of its output queues individually, meaning that F is calculated for each of those queues.

B. Capacity estimation error

Choosing a capacity value C for the computation of F (Eq. 1) when the underlying medium allows concurrent access from different stations that can use different rates, e.g. IEEE 802.11, is not trivial. The overall medium capacity, C, depends on the data rate used by each station, the number of active stations, the number of collisions, failed transmissions and the handshake mechanisms (RTS/CTS) and their thresholds. Previous studies [17], [4] have shown that XCP is able to compensate an erroneous capacity estimation, up to a certain limit, by building up the queue. The same applies to RCP as they use the same aggregate feedback function. The queue length required to compensate the error is proportional to the error ϵ itself, to the average RTT of the flows d and the ratio $\frac{\alpha}{\beta}$. The error ϵ can be defined as:

$$\epsilon = C - C_{real} \tag{2}$$

where C is the capacity estimate and C_{real} represents the actual medium capacity. The amount of queue build-up required to compensate this estimation error is given by:

$$q = \frac{\alpha}{\beta} \cdot \epsilon \cdot d \tag{3}$$

for a more convenient analysis we decompose d to reflect the effect that queue build-up has on the overall system delay:

$$d = d_0 + \frac{q}{C_{real}} \tag{4}$$

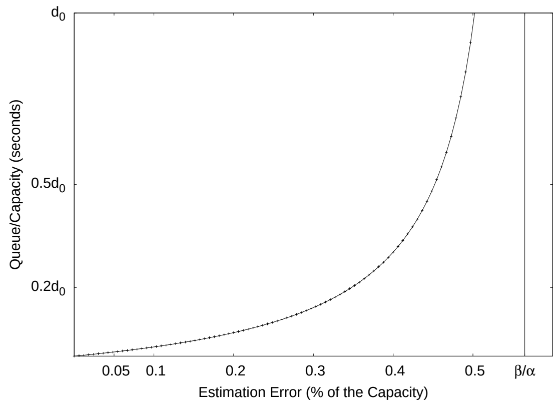


Fig. 1.

where d_0 represents the system base delay, that is the system delay excluding queuing delay. By applying this relation to Eq. 3 we finally obtain the compensation queue as a function of the system base delay d_0 and the estimation error ϵ.

$$q = \frac{\alpha}{\beta} \cdot \epsilon \cdot \frac{1}{1 - \frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}}} \cdot d_0, \quad \epsilon < \frac{\beta}{\alpha} \cdot C_{real} \tag{5}$$

as mentioned above, the system can only compensate estimation errors up to a certain limit $\epsilon < \frac{\beta}{\alpha} \cdot C_{real}$, which, if exceeded, causes the queue to grow indefinetly. These results only have meaning for positive capacity estimation errors. If the medium capacity is under-estimated, then the medium will result under-utilized and of course queue build-up will not occur.

In real scenarios, e.g. an IEEE 802.11 network, the estimation error ϵ may reach the order of magnitude of the medium actual bandwidth or even higher, which will cause large queuing delays or even restrain the queue from stabilizing at all, when $\epsilon > \frac{\beta}{\alpha} \cdot C$. This is observable in Fig. 1, where the queuing delay introduced by the compensation queue is plotted as a function of estimation error, expressed as a percentage of the medium actual capacity.

C. XCP vs. RCP

XCP and RCP are two of the most representative XCC protocols. They share the same aggregate feedback function

(Eq. 1), but differ on how the aggregate of distributable bandwidth, F, is split among flows in each control interval. XCP follows an AIMD rule; if F is negative, each flow is decremented proportionally to its rate, otherwise F is split equally among all flows. The detailed algorithm is described in [7]. RCP on the other hand tries to emulate processor-sharing of bandwidth, meaning that F is split evenly among flows whether F is positive or negative and that all flows use the same common rate R. Additionally, RCP allows new flows to immediately start using the rate that other flows in the network are already using. As a result, RCP has a much simpler job as it only needs to calculate a common rate R - used by all flows - thus not requiring any per-packet calculations. An RCP router calculates the common rate R_n at interval n using:

$$R_n = R_{n-1} \cdot \left(1 + \left(\frac{F}{N}\right)\right) \tag{6}$$

where F refers to the aggregate feedback calculation presented previously (Eq. 1). Processor sharing also has the advantage of providing instantaneous fairness as new flows converge to the target rate in 1 RTT, as well as allowing short flows to complete significantly faster [6]. XCP flows have a smoother adaptation taking tens of intervals to achieve the target rate. RCP gains come at the cost of higher and more frequent queue spikes which can be reflected in higher jitter, or packet loss. Additionally RCP may struggle to perform in the presence of concentrated flow arrivals, i.e. flash-crowds, as seen in [3]. As a final note we would like to stress the fact that none of these protocols requires per-flow state as all flow information required to allocate bandwidth is carried in the congestion header.

IV. Alternative Algorithms for Time-Varying Capacity Media

In this section we present alternative router functions to calculate the aggregate feedback bandwidth F, which allow XCC algorithms to exhibit a behaviour in variable-capacity media similar to that obtained in fixed-capacity networks. The functions proposed remove the need for the router to be configured with the exact medium capacity and they are able to adapt to changing bandwidth conditions over time. The ideas behind the alternatives presented consist of using queue speed as an indicator of available bandwidth (Blind), queue accumulation as an indicator of the capacity estimation error (ErrorS), or relying on direct access to MAC layer information (MAC).

A. The Blind Algorithm

The Blind algorithm has been proposed in [4] as an alternative algorithm for operating XCP in time-varying capacity media. In this paper we refine Blind's design to produce a smoother queue response and extend its applicability to RCP. Our refinements include a method for dynamically adjusting the κ parameter, thus allowing automatic system configuration.

The key concept of the Blind algorithm is that spare bandwidth can be measured from queue speed. In fact, the rate at which the queue is drained, or at which it builds-up, is a fairly accurate estimate of the difference between the incoming bandwidth and the medium capacity within a certain measurement interval. Thus we can replace this difference by the measurement of queue speed. The problem in doing so is that queue speed can only be measured whenever the queue is not empty, when the medium is saturated. To overcome this limitation Blind proposes stabilizing queue length at some positive value, κ so queue variations can be measured around this queue length. The calculation of the aggregate bandwidth feedback, F, becomes:

$$F = -\alpha \cdot \frac{\Delta q}{d} - \beta \cdot \frac{q-\kappa}{d} \tag{7}$$

where Δq is the variation of the persistent queue within a control interval and $\frac{\Delta q}{d}$ represents the queue speed within a control interval. κ is the target queue length, around which queue variations should be measured. Note that the equation we present slightly differs from that of the original Blind proposal [4]. In the original Blind proposal, the controller would switch between the feedback function we present and a fixed feedback value, given the exponential weighted average of the queue would cross a certain threshold. This function allowed the controller to maximize bandwidth distribution during under-utilization periods, while maintaining low queues when the medium reached the saturation point. This technique, however, did not provide an automatic method for adjusting the value of κ as it was considered to be constant, which resulted in non-optimal performance. We propose that this switching function be embedded in the adjustment of the κ parameter and that κ is dynamically adjusted over time. Making κ dynamic allows the queue to have a smoother behaviour while guaranteeing that κ is set according to the instantaneous operating conditions. The detailed instructions and the rationale behind the adjustment of κ are presented later in Section IV-D.

B. *Error Supression Algorithm (ErrorS)*

The idea behind the Error Suppression Algorithm (ErrorS) is to take advantage of queue accumulation to suppress the error present in the capacity estimate. As seen in Section III-B, an error ϵ in the capacity estimate leads to queue build-up, whereas the amount of queue length required to compensate the error, in steady-state, is given by:

$$q = \frac{\alpha}{\beta} \cdot d \cdot \epsilon \tag{8}$$

so, given the current queue length the router is able to infer the value of the error present in its capacity estimate. With this knowledge the router can suppress the error of its capacity estimate. Referring to the error estimate as ξ we can write the calculation of the aggregate bandwidth feedback F as:

$$F = \alpha \cdot (C - y) - \mu \cdot \xi - \beta \cdot \frac{q - \kappa}{d} \tag{9}$$

where μ is a constant gain parameter dimensioned to preserve system stability for any delay, capacity, and number of flows. In Appendix A, we present a stability analysis to assist in the choice of μ. As previously mentioned, ξ is the current estimation of the error and it is calculated over time using:

$$\xi_n = \frac{\beta}{\alpha} \cdot \frac{q - \kappa}{d} + \xi_{n-1} \tag{10}$$

where ξ_{n-1} is our error estimation in the previous control interval and ξ_n is the error estimation in the current control interval. The ErrorS algorithm also adopts the strategy used in Blind of stabilizing queue length at some positive value, which is controlled by κ. This allows the controller to identify negative capacity estimation errors when the queue falls below the threshold κ. In Section IV-D we present the details on how to dynamically adjust κ.

A potential problem of the ErrorS algorithm is that the capacity estimation error may be too large. If that is the case the initial queue lengths required to stabilize the error estimation may be huge. Imagine the real case of 802.11g where a maximum throughput of $\approx 16Mbit/s$ and a minimum of $\approx 1Mbit/s$is available. The maximum is sixteen-fold the minimum, meaning that it can originate sixteen RTTs worth of queue space. We recommend setting the capacity estimate conservatively to avoid large queue build-up and packet loss. One reasonable approach is to set $C = 0$.

C. *MAC-based algorithm*

In [15] another approach to the operation of XCC protocols in time-varying capacity media is presented. This approach relies on information accessible at the MAC layer such as the idle and busy transmission times, as well as the number of active neighbour stations and the number of collisions. The algorithm that we present here is inspired by this idea, but we simplify it a bit so that an XCC router is not required to keep track of the number of active neighbour stations. Using the busy MAC period within an interval and the number of bytes sent and received by the router within that interval, it is possible to calculate the medium bandwidth. Multiplying this bandwidth by the percentage of idle time of the medium, the router is finally able to determine how much spare bandwidth it may grab. The aggregate bandwidth feedback function results as:

$$F = \alpha \cdot \frac{u}{T_{busy}} \cdot \frac{T_{idle}}{d} - \beta \cdot \frac{q}{d} \qquad (11)$$

where u is the number of bytes sent and received by the router within a control interval, T_{busy} is the period during which the medium was sensed to be busy, and T_{idle} is the period during which the medium was sensed as being idle. This formula is only usable if the station was actually active, otherwise $u = 0$ and the router is not able grab any medium bandwidth. If this is the case, we set $F = \frac{Q_{max}}{d}$ which represents the amount of bandwidth that the queue buffer can absorb in one control interval. Q_{max} represents the size of the queue buffer. If $u = 0$, it also means that the router does not have an estimate of d, thus some reasonable value must be used (we recommend using $d = 300$ ms).

The MAC algorithm allows more accurate operation, as the router knows with some degree of certainty how far it is from its objective ($T_{idle} = 0$), something that does not happen when using algorithms based solely on queue dynamics. Queue-based algorithms are not able to determine how far they are from full utilization as that is not reflected on queue build-up. For that reason, queue-based algorithms have to be more conservative when distributing bandwidth when the medium is under-utilized, so that queue overflow when reaching full utilization is avoided or at least kept at a reasonable level. The drawback of a MAC-based approach is that it requires access to layer 2 information, which makes implementation more complex and specific to the underlying layer 2 technology. Additionally, a MAC-based approach may slow the ramp-up of flows from new stations when the medium is already saturated, as those new stations perceive the medium as fully-utilized.

D. The κ Parameter

The κ parameter represents the target queue length at which the system stabilizes and plays an important role in both the Blind and the ErrorS algorithms. During times of under-utilization - when there is no queue build-up - it will control how much bandwidth is distributed in each control interval, while during times of full or over utilization κ determines how much queuing delay is introduced in the path. It is desirable to maximize bandwidth distribution during under-utilization periods and minimize the queuing delay introduced when the medium is saturated, therefore we want κ to be as high as possible during under-utilization, and as low as possible when we have reached full utilization. Additionally, we need to identify these two states - under-utilization and full-utilization - in a robust manner to avoid unnecesary oscillation in κ, and consequently in the queue length. κ should also adapt dynamically to current network conditions, thus not requiring the dimensioning of any network specific parameters. Let us now analyze the three objectives of κ individually:

1) Minimize queuing delay during full-utilization: In order to minimize queuing delay, κ needs to be set as low as possible. However, κ still has to be high enough that it can absorb typical bandwidth fluctuations without draining the queue completely. The question here is how much bandwidth varies over adjacent control intervals. The answer to this question depends on many factors, for example, in a IEEE 802.11 media the variability of the medium total bandwidth will depend on how many stations are using it concurrently, the data rates used by those stations and the current transmission interference signals. It is simply impractical to choose a constant value of κ for all scenarios. One possible approach is to choose a κ that covers queue variability. To do so, we set κ to match the queue standard deviation. There are however two issues in doing so: 1) over which time scale do we calculate the queue standard deviation, and 2) how can we compute it efficiently. The time-scale issue is important because if κ moves too fast, then the system is not able to stabilize queue length - we recall that κ represents the length at which the controller tries to stabilize the queue. The κ signal must move slower than queue length signal. Regarding efficiency, we note that κ is recalculated in each control interval, thus it cannot involve very complex mathematical operations. Having these two aspects in mind, we propose to calculate κ using:

$$\kappa_n = \rho \cdot |q - \bar{q}| + (1 - \rho) \cdot \kappa_{n-1} \tag{12}$$

where κ_n represents the value of κ at control interval n, and κ_{n-1} represent the valu of κ in the previous control interval. q represents the persistent queue length at control interval n and $\bar{q}$ represent the exponential moving average of queue length, calculated using:

$$\bar{q}_n = \rho \cdot q + (1 - \rho) \cdot \bar{q}_{n-1} \tag{13}$$

where ρ has the same meaning as in Eq. 12 and controls the speed of the exponential moving average by defining the weight that is given to the current sample of the moving average. Calculating the queue standard deviation using an exponential moving average consists in a computationally efficient method as only 4 multiplications and 2 sums are required per control interval. The next step of the design process of the calculation of κ is dimensioning the constant ρ so that κ moves slowly enough that the queue length is able to follow it. A practical approach is to set the cut-off frequency of the exponential moving average (the exponential moving average acts as a low-pass pass filter for κ) to be m times lower than the cut-off frequency of the queue length signal. From [11] we know that the cut-off frequency w_{Blind} of the queue length signal of the Blind algorithm is given by:

$$w_{Blind} = \frac{\beta}{\alpha \cdot d} \tag{14}$$

and in Appendix A we show that the cut-off frequency of the queue signal of ErrorS is given by:

$$w_{ErrorS} = 1.75 \cdot \frac{\beta}{\alpha \cdot d} \tag{15}$$

and we know that the cut-off frequency w_κ of an exponential moving average that assigns a weight ρ to the current sample is given by:

$$w_\kappa = \frac{\rho}{d - \rho \cdot d} \tag{16}$$

We performed extensive sets of simulations and concluded that it would be enough to set the exponential moving average cut-off frequency to be 2 times lower than the queue length cut-off frequency for the case of the Blind algorithm ($w_\kappa = \frac{w_{Blind}}{2}$) and 3 times lower for the case of the ErrorS algorithm ($w_\kappa = \frac{w_{ErrorS}}{3}$). Applying the recommended values of α, β of the Blind and the ErrorS algorithms (Appendix B) we finally obtain the value of ρ for both Blind and ErrorS - $\rho_{Blind} \leq 0.22$ and $\rho_{ErrorS} \leq 0.15$.

2) Maximize bandwidth distribution during under-utilization: In order to maximize bandwidth distribution, κ needs to be set as high as possible. During under-utilization we observe no queue build-up, thus $q = 0, \Delta q = 0$ (remember that we refer to the persistent queue, which represents the minimum queue length observed in a control interval). As a result, during under-utilization, the aggregate bandwidth feedback F is given by:

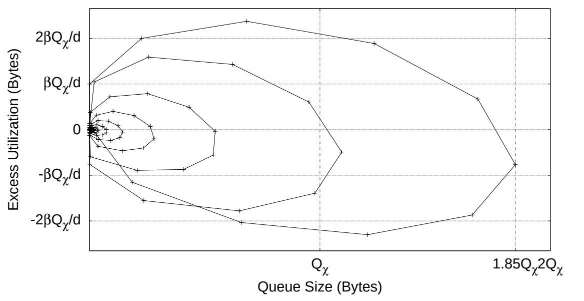


Fig. 2. System trajectory for the worst case scenario when using the Blind algorithm.

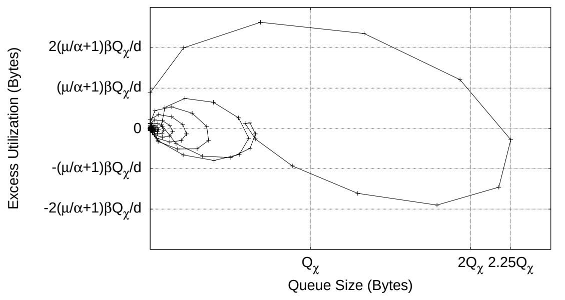


Fig. 3. System trajectory for the worst case scenario when using the ErrorS algorithm.

$$F = \beta \cdot \frac{\kappa}{d} \tag{17}$$

when using the Blind algorithm, and:

$$F = \mu \cdot \frac{\beta}{\alpha} \cdot \frac{\kappa}{d} + \beta \cdot \frac{\kappa}{d} = \left(\frac{\mu}{\alpha} + 1\right) \cdot \beta \cdot \frac{\kappa}{d} \tag{18}$$

when using the ErrorS algorithm. By increasing κ we are indeed increasing the amount of bandwidth distributed within each control interval however we are also increasing the queue spike observed when the system crosses to over utilization. But the question is, how high can κ be so that this spike remains acceptable. We define an *acceptable spike* as a queue spike that does not cause the persistent queue length to exceed a certain limit Q_{max}, which can be for instance the size of the queue buffer. We refer to the maximum value of κ as Q_χ. To help us dimension the parameter Q_χ we draw in Fig. 2, 3 the system trajectory for the worst case scenario. These plots were obtained using a step-by-step simulation of the Blind and ErrorS algorithms in MATLAB/Simulink [2], and provide us the queue peak obtained when using $\kappa = Q_\chi$. We can now apply the maximum peak restriction ($q \leq Q_{max}$) which leads ultimately to the dimensioning of Q_χ. For the Blind algorithm we obtain:

$$1.85 \cdot Q_\chi \leq Q_{max} \rightarrow Q_\chi \leq 0.54 \cdot Q_{max} \tag{19}$$

while for the ErrorS algorithm we can write:

$$2.25 \cdot Q_\chi \leq Q_{max} \rightarrow Q_\chi \leq 0.44 \cdot Q_{max} \tag{20}$$

The formula used for the calculation of κ becomes:

$$\kappa_n = \begin{cases} \rho \cdot |q - \bar{q}| + (1-\rho) \cdot \kappa_{n-1} & \text{if full\_utilization,} \\ \rho \cdot |Q_\chi - \bar{q}| + (1-\rho) \cdot \kappa_{n-1} & \text{if under\_utilization,} \end{cases} \tag{21}$$

.

3) Identify under-utilization and full-utilization robustly: We have seen how to adjust κ during periods of under and full utilization; we shall now discuss how to identify each of these periods robustly. The most naive approach would be to consider the medium under-utilized if the persistent queue is zero, while considering that full-utilization has been reached once the persistent queue starts to build. Although feasible, this approach is not robust as the queue may drain completely in one control interval due to utilization fluctuation during the transient response or other reasons, e.g. temporary interference. For this reason we propose to analyze utilization over a larger time-scale, that is over a certain number of control intervals. The design challenge is to choose the exact number of intervals on which to base the analysis so we can identify under and full utilization accurately. Our proposal is to analyze utilization over a a number of control intervals η that covers at least half the oscillation period of the fundamental frequency of oscillation, i.e. the cut-off frequency of the queue response.

As stated before, we know from [11] that the cut-off frequency of the Blind algorithm is given by Eq. 14, and in Appendix A we show that the cut-off frequency of the ErrorS algorithm is given by Eq. 15. This means that half the oscillation period ($T_{Blind}/2$) of the Blind algorithm is given by:

$$\frac{T_{Blind}}{2} = \pi \cdot \frac{\alpha}{\beta} \cdot d \tag{22}$$

and half the oscillation period ($T_{ErrorS}/2$)of the ErrorS algorithm is given by:

$$\frac{T_{ErrorS}}{2} = 0.571 \cdot \pi \cdot \frac{\alpha}{\beta} \cdot d \tag{23}$$

Now we can use these periods as a reference for medium utilization analysis being that we consider the medium to be under-utilized if the persistent queue has been empty for at least $\frac{T}{2 \cdot d}$ control intervals and fully-utilized otherwise. The final calculation of κ comes as:

$$\kappa_n = \begin{cases} \rho \cdot |q - \bar{q}| + (1-\rho) \cdot \kappa_{n-1} & \text{if } L_{empty} < \frac{T}{2 \cdot d}, \\ \rho \cdot |Q_\chi - \bar{q}| + (1-\rho) \cdot \kappa_{n-1} & \text{if } L_{empty} \geq \frac{T}{2 \cdot d}, \end{cases} \tag{24}$$

where L_{empty} represents a counter of the number of consecutive control intervals during which the persistent queue has been zero. As such, the controller will wait for at least $\frac{T}{2 \cdot d}$ control intervals before entering the under-utilization state; we refer to this method as *late reaction*. In Fig. 4, 5 we plot the evolution of the persistent queue length, its exponential moving average and κ throughout time, for the Blind and ErrorS algorithms respectively.

V. PERFORMANCE

We validate and evaluate the performance of all three of our proposed algorithms - Blind, MAC-based and ErrorS - by applying them to both XCP and RCP. Our goal is to clearly illustrate the dynamics of each algorithm and their respective benefits and drawbacks. Additionally, we attempt to quantify the gains obtained from the use of our proposed algorithms when compared to TCP NewReno.

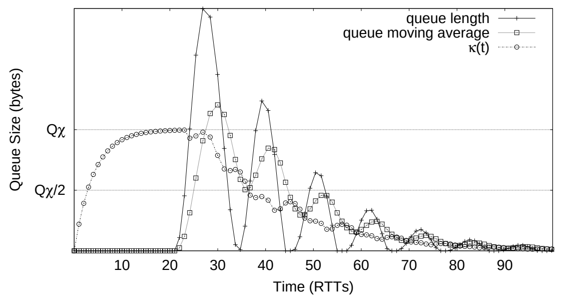


Fig. 4. The evolution throughout time of the persistent queue length, its exponential moving average and κ using the Blind algorithm.

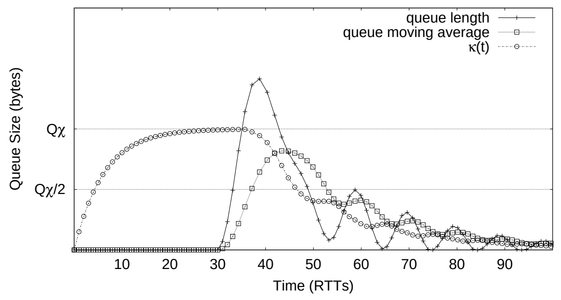


Fig. 5. The evolution throughout time of the persistent queue length, its exponential moving average and κ using the ErrorS algorithm.

Extensive testing of the algorithms was initially performed through simulation, using the ns-2 simulator [1], and subsequent results were then compared to experimental feedback obtained from a physical testbed. As such, we will detail our results separately, first focusing on simulation results and thoroughly evaluating each algorithm in § V-A, and then replicating limited, but representative, simulation tests on a physical testbed in § V-B.

A. Simulation Results

The base scenario used in the simulations is shown in Fig. 6, consisting of a dumb-bell topology with a single IEEE 802.11 bottleneck. Traffic consists of greedy flows (FTP-like) between the wireless nodes $W(i)$ and the wired nodes $N(i)$, in both directions. For the sake of simplicity we consider the flows traversing from nodes $N(i)$ to nodes $W(i)$ as downloads,

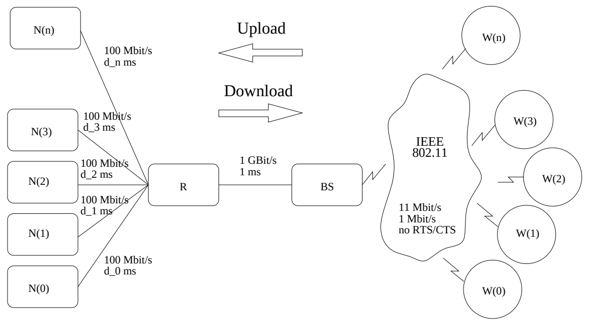


Fig. 6. Simulation scenario.

while flows in the opposite direction are considered uploads. The wired links, with the exception of the link between the router R and the base station BS, have a capacity of 100 Mbit/s and a configurable latency. The IEEE 802.11 medium is configured with no hand-shake mechanisms (RTS/CTS), and stations use a data rate of 11 Mbit/s and a basic rate of 1 Mbit/s, unless stated otherwise. The queue length of the wireless nodes, where the bottleneck occurs, is set to 60 packet if using XCP, while it is set to 80 packet if using RCP. The size of data packets is set to 1300 byte, while acknowledgement packets have 60 byte. We set a higher queue limit for RCP, because RCP causes higher queue peaks. The parameters of the algorithms are set to the recommended values, as listed in Appendix B, and the value of the capacity in ErrorS is configured to 0 Mbit/s. Additionally, we set Q_{max} to 50 packets, slightly below the actual limit of 60 packet of XCP and 80 packet of RCP, to reduce the probability of queue overflow. The simulations do not include packet loss due to corruption, therefore the only causes for packet loss are queue overflow and consecutive collisions or persistent interference at the medium access level.

1) Basic dynamics: Our first experiment explores the dynamics and defining characteristics of each algorithm. During the first 20 seconds, a pair of flows (one download, one upload) enters the network every 5 seconds. These 10 flows have a duration of 40 seconds, which leads to a total experiment duration of 60 seconds. Propagation RTT is set to ≈ 80ms, neglecting the propagation delay in the wireless hop. Albeit simple, this configuration highlights not only how each algorithm performs during stable periods but also how they react to the dynamic arrival and departure of flows.

The results of the experiment are plotted in Fig. 7, which represents the evolution of the congestion window of all 10 flows, and Fig. 8, which shows the evolution of the instantaneous queue, its moving average and the parameter κ.

We first focus on comparing how each of our proposed algorithms perform. While all three operate in a stable manner, the graphs hint at subtle differences between them. From the congestion window of the flows we can see that the MAC algorithm displays good responsiveness to network changes whilst maintaining low queue spikes, however it also displays the slowest convergence to fairness and the lowest network utilisation as is clear in Table I. This difference is explained by the fact that Blind and ErrorS maintain some degree of occupancy of the queue buffer at all times, allowing them to maintain the network saturated persistently. The Blind algorithm performs well in terms of responsiveness, but reacts slower to newly available bandwidth freed by departing flows when compared to the MAC algorithm. The ErrorS algorithm on the other hand displays some initial delay before fully utilising the available bandwidth, which reflects the time the algorithm takes to estimate the error between the configured capacity (which in this experiment was $C = 0$) and the real capacity. As such, ErrorS is less responsive than the Blind variant, favouring a smoother response to fluctuations in available bandwidth. This is particularly visible as flows depart inducing frequent queue spikes in Blind as the controller agressively probes for available bandwidth. ErrorS on the other hand manages to adapt gracefully to such changes as it maintains an estimate of the total capacity, distributing available bandwidth before under-utilisation is detected. This

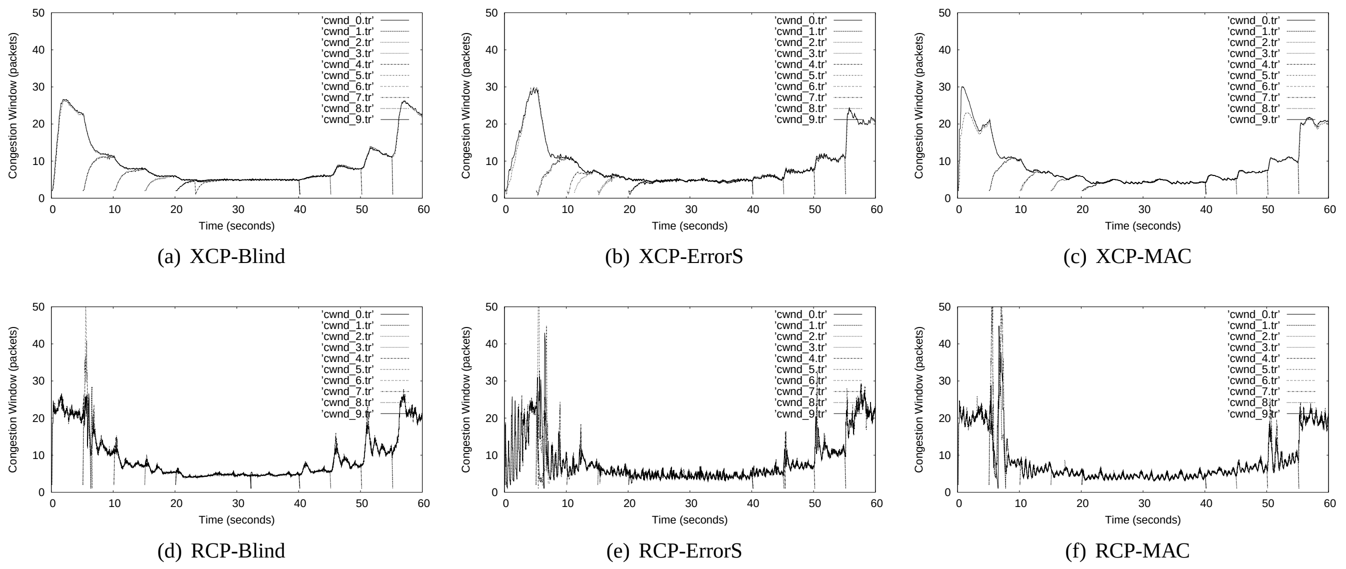


Fig. 7. The evolution of the congestion window throughout time for all algorithm combinations.

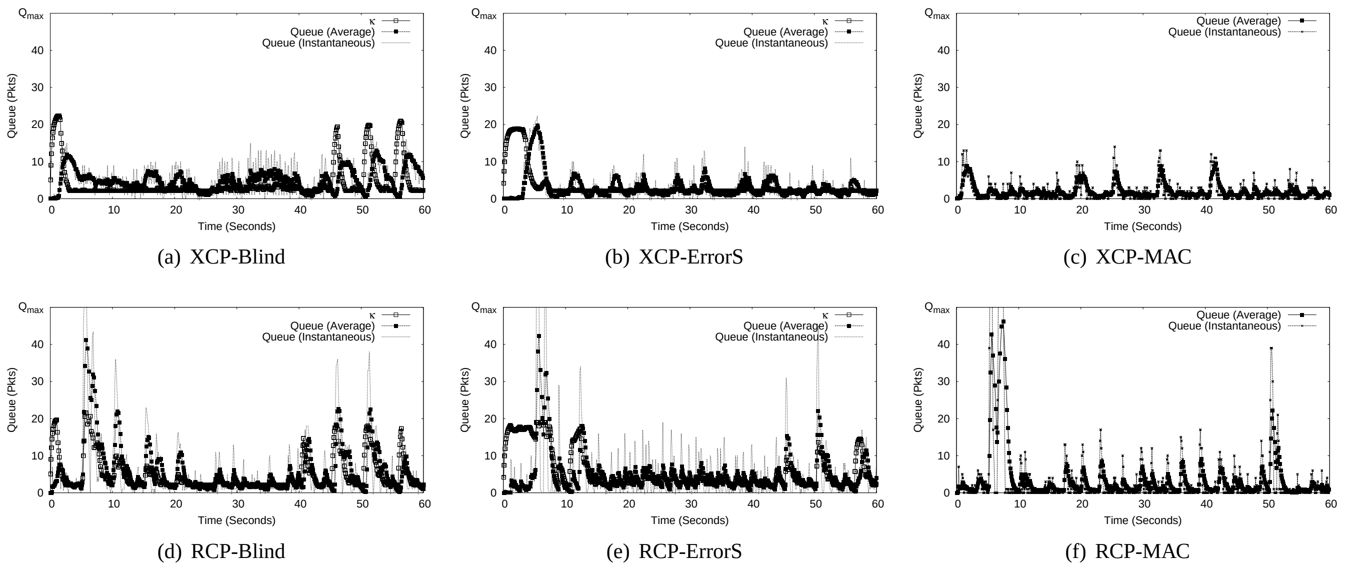


Fig. 8. The evolution of the instantaneous and average queue size throughout time for all algorithm combinations.

is in effect the the fundamental difference between Blind and ErrorS: whereas Blind reacts to changes in both medium capacity and the level of utilisation, ErrorS reacts mostly to capacity changes as it is able to better cope with fluctuations in utilisation.

TABLE I
AVERAGE THROUGHPUT DURING THE INTERVAL $[20 : 40]$ S.

Algorithm	Throughput
XCP-Blind	550.4 KByte/s
XCP-ErrorS	551.7 KByte/s
XCP-MAC	519.3 KByte/s
RCP-Blind	551.9 KByte/s
RCP-ErrorS	548.8 KByte/s
RCP-MAC	517.2 KByte/s

Observing the evolution of the queue helps us to further understand how both queue-driven algorithms work, particularly taking into account the dynamics of the κ parameter present in both the Blind and ErrorS variants. During periods of queue depletion, reflecting under-utilisation, κ rises to its maximum value, which is a fraction of the desired maximum buffer occupancy Q_{max}. With the increase of κ, more bandwidth is distributed amongst sources so as to quickly induce full utilisation. As the system approaches this goal, queue length starts to grow and κ tends to reduce in order to match the standard deviation of the instantaneous queue, so that during regular operation the latency introduced by queue build-up is as low as possible.

Besides algorithm behaviour, many of the differences between RCP and XCP may also be extrapolated from these graphs. For one, RCP produces more jitter than XCP, particularly in the presence of a small number of flows. This is a consequence of allowing flows to jump start to the target rate which in turn provokes abrupt changes to network utilisation. Furthermore, since all flows are kept at a common rate R, even small offsets from the fairshare rate may be magnified to signficant errors, inducing a higher degree of oscillation than in experiments with XCP. This is particularly true if the quantity of new flows is significant when compared to the number of existing flows in the system. The resulting over-allocation of capacity amongst flows leads to a visibly erratic response. Such behaviour should be seen as intrinsic to RCP rather than a consequence of any of our proposed algorithms, and is attenuated as the number of flows in a system increases.

2) Response to abrupt changes in bandwidth: .

We extend our study by analysing how these algorithms respond to abrupt changes of the medium capacity, such as when the data rate used by the stations varies synchronously. Our next experiment consists in setting 10 flows from 10 different stations, which are active throughout the entire simulation. Abrupt capacity changes are cause by modifying the data rate across all stations simultaneously. Stations begin the simulation using a data rate of 54 Mbit/s and a basic rate of 1 Mbit/s. Every 40 seconds the data rate changes: dropping to 11Mbit/s at $t = 40$ s, then 2 Mbit/s at $t = 80$s before rising again to 11Mbit/s at $t = 120$ s and 54Mbit/s at $t = 160$ s. Note that the basic rate is kept unchanged at 1 Mbit/s throughout the simulation, thereby changes in the data rate are not linearly

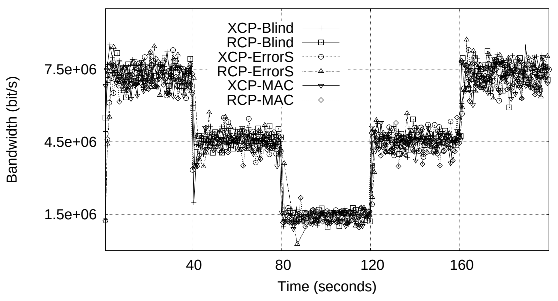
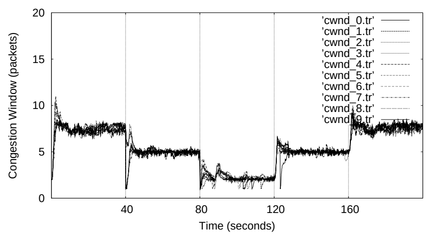


Fig. 9. Throughput measured at the base station (BS). Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

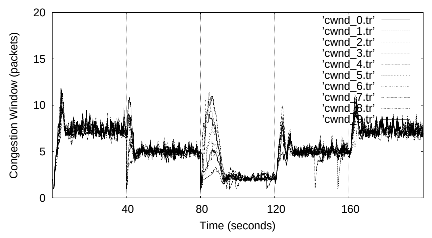
reflected in the overall medium capacity.

The resulting congestion window for each flow and corresponding queue variables are plotted in Fig. 10 and Fig. 11 respectively. The first plot shows that all algorithm combinations are able to adapt to the variations in bandwidth. The ErrorS algorithm however seems to lag behind its counterparts: for both Blind and MAC-based variants, queue overflow caused by bandwidth reduction is quickly drained, while freed bandwidth is rapidly utilised. ErrorS on the other hand is less nimble at distributing available bandwidth and more prone to queue build-up. This behaviour results from a slower convergence to an accurate capacity estimate and is in part a consequence of the use of more conservative design parameters. While it would be possible to design an more aggressive ErrorS controler, the recommended values focus on avoiding greater queue oscillation.

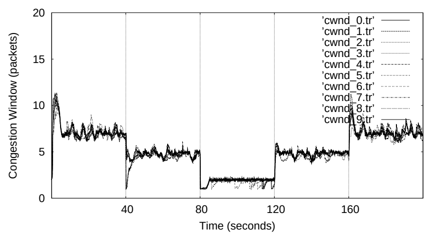
3) Robustness to rate heterogeneity: Another important aspect is to evaluate each algorithm's response when different stations use different data rates. In such cases, bandwidth variation is progressive at a macroscopic level but the data rate may vary by one order of magnitude betwen frames. To perceive how robust the algorithms are to bandwidth fluctuations at a microscopic level, we consider 5 pairs of flows, where each pair consists of one download and one upload flow, with a data rate of either 2Mbit/s or 54Mbit/s, while the data rate of the base station is fixed at 11Mbit/s.



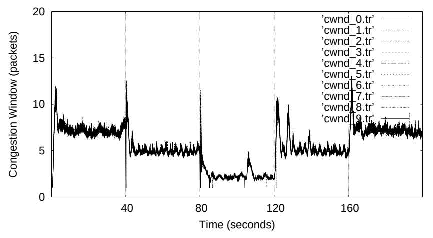
(a) XCP-Blind



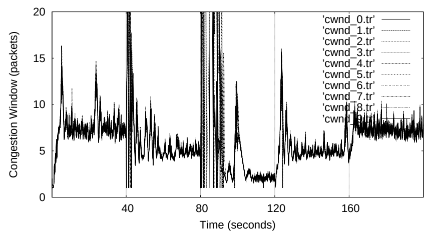
(b) XCP-ErrorS



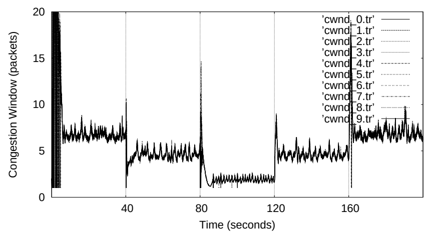
(c) XCP-MAC



(d) RCP-Blind

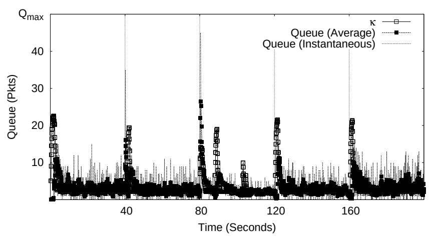


(e) RCP-ErrorS

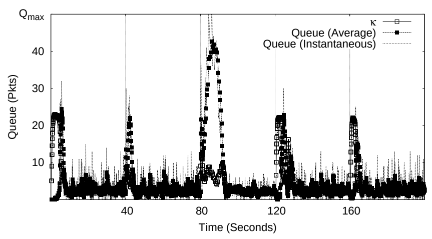


(f) RCP-MAC

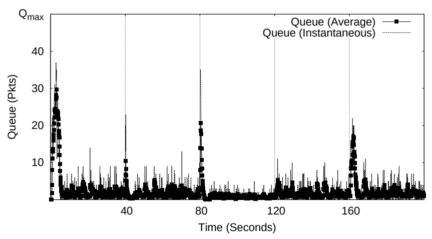
Fig. 10. The evolution of the congestion window throughout time for all algorithm combinations. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.



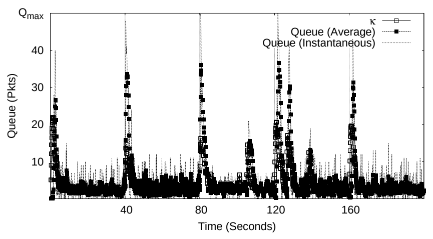
(a) XCP-Blind



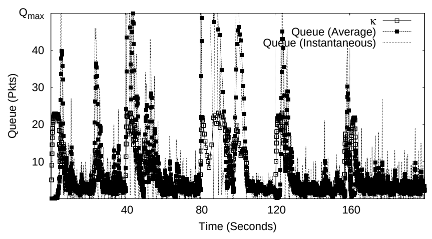
(b) XCP-ErrorS



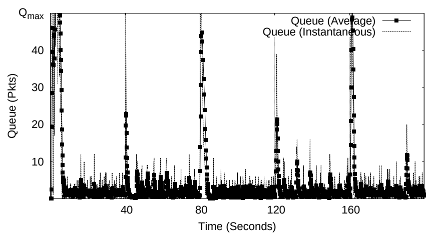
(c) XCP-MAC



(d) RCP-Blind



(e) RCP-ErrorS

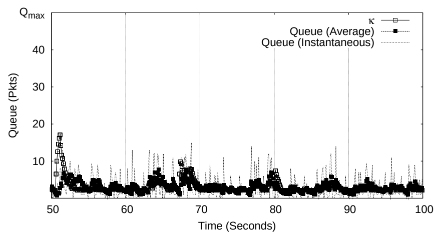


(f) RCP-MAC

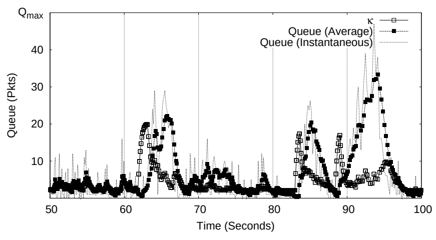
Fig. 11. The evolution of κ and queue dynamics throughout time. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

Every 10 seconds a new pair of flows enters the network as another pair with a different data rate leaves, resulting in 10 active flows at all times. As a result of this substitution policy, one data rate will be used by 6 flows while the other will be used by the remaining 4 flows, with the most popular data rate changing every 10 seconds resulting in an oscillatory overal medium bandwidth. The queue variables plotted in Fig. 13 give yet more insight into how each algorithm behaves. Under these conditions, the MAC algorithm is the most robust, being able to filter most of the noise derived from the rate heterogeneity and maintaining a stable queue. The Blind algorithm is able to maintain a relatively stable queue under XCP, but unable to avoid spikes with RCP. The ErrorS algorithm however performs poorly, once again demonstrating it is unable to adapt to sudden bandwidth variations, producing frequent and severe queue spikes.

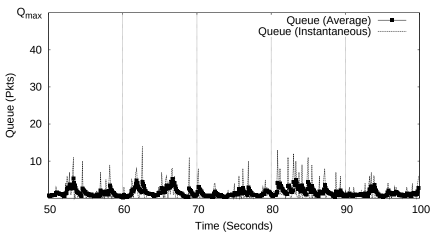
4) Efficiency and scalability: Another important aspect all algorithms must retain is efficiency, with particular emphasis



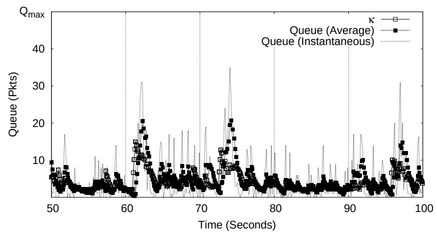
(a) XCP-Blind



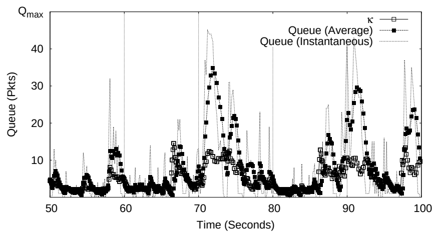
(b) XCP-ErrorS



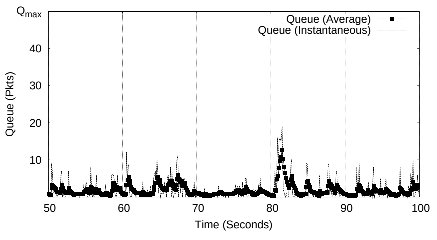
(c) XCP-MAC



(d) RCP-Blind

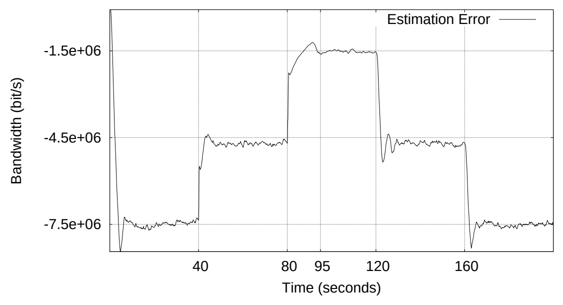


(e) RCP-ErrorS

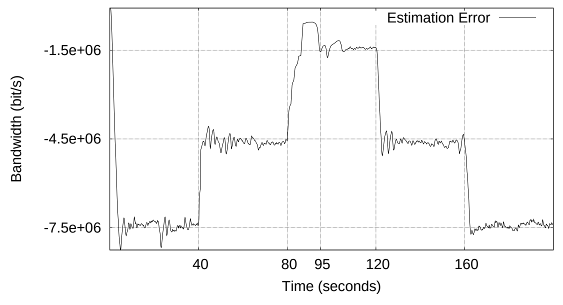


(f) RCP-MAC

Fig. 13. The evolution of κ and queue dynamics throughout time. Every 10 seconds, one pair of flows leaves the network and another pair of flows with a different data rate enters the network.



(a) XCP-ErrorS



(b) RCP-ErrorS

Fig. 12. The evolution of the error suppression throughout time in ErrorS. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

on how network utilisation scales with the bandwidth delay product. Since utilisation is strongly correlated to traffic characteristics, we refrain from focusing much attention on absolute values. Instead, our intention in this set of experiments is to show that a growth in the network BDP has a negative effect on TCP Reno's efficiency whilst bearing no influence on XCC protocols.

In this experiment we set a heterogeneous traffic pattern, compromising both short and long flows. Long flows are active throughout the simulation, while short flows have an exponentially distributed duration with a mean value of 10 seconds and a minimum value of 1 second. A total of 30 sources are set, compromised by an equal number of wired and wireless nodes. Wired paths between end-systems $W(n)$ and the router R were configured with different latencies between 20 ms and 120 ms, resulting that flows will have propagation RTTs within $[40 : 220]$ ms. On simulation start, 6 long flows are initiated. Short flows are spawned throughout the simulation in such a manner that, on average, the number of active short flows is equal to that of long flows.

Since we are interested in evaluating how efficiency scales with the network BDP, we could either increase the path delay or the bandwidth of the wireless medium. We opted for the latter as the more plausible scenario, particularly with the advent of next-generation wireless systems (i.e. IEEE 802.11n). Subsequent IEEE 802.11 rates and related interval variables used are shown in Table II. Each simulation run lasts 200 seconds, and results from this experiment, shown in Fig.

TABLE II
MAC LAYER PARAMETERS USED.

Data Rate	Basic Rate	SlotTime /SIFS
11 Mbit/s	1 Mbit/s	20/10 μs
22 Mbit/s	2 Mbit/s	10/5 μs
55 Mbit/s	5 Mbit/s	4/2 μs
110 Mbit/s	10 Mbit/s	2/1 μs
220 Mbit/s	20 Mbit/s	1/.5 μs

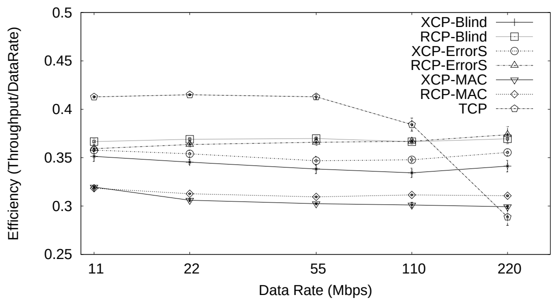


Fig. 14. Efficiency as a function of medium capacity. TCP Reno utilisation decreases with the increase of the medium bandwidth, while XCC mechanisms maintain their properties regardless of the increase of bandwidth.

14, refer to the period $[20 : 180]$ s in order to remove the effect of the convergence periods.

The resulting plot shows how utilisation scales with the increase of the capacity of the wireless medium, normalized to the data rate used in each experiment, and supports the argument that TCP Reno is unable to scale with the network BDP due to its fixed dynamics. Depending on the number of flows and the traffic pattern used, Reno will always be unably to fully utilize network resources beyond a certain bandwidth threshold, which in our case was ≈ 0.44 Mbit/s. On the other hand, it should be noted that for a lower BDP, Reno is able to perform more efficiently than any XCC mechanism. This may be explained in part due to the significant amount of queue build-up employed by Reno, which is able to compensate the bandwidth wasted on flow departure, but also reflects the ACK-loss effect, documented in detail in [4].

Analysing the various combinations of XCC algorithms, we conclude that the MAC-based variants achieve substantially lower utilisation than both Blind and ErrorS. Once again this is a reflection of the lack of queue build-up which results in under-utilisation during periods of flow departure or bandwidth fluctuation. By maintaining small, non-zero queues consistently, the Blind and ErrorS algorithms are able to drive utilisation at all times.

Finally, we observe that for the traffic pattern used, RCP outperforms XCP in terms of efficiency for all algorithm variants. This is due to the use of a *slow start* period for new flows by XCP, while RCP jump starts new flows to the existing fair-share rate R. As such, RCP is able to quickly compensate the departure of flows leading to better overall efficiency.

5) Flow fairness: Next we compare XCC algorithms to Reno in terms of flow fairness. Our objective is to show that XCC algorithms exhibit greater flow fairness than Reno, whilst demonstrating none of the RTT-bias which the latter implies. Our experiment replicates the setup used to evaluate algorithm efficiency in the previous section, this time quantifying flow fairness by using Jain's index [9], given by:

$$J = \frac{\left(\sum_{i=1}^{n} \bar{x}_i\right)^2}{n \cdot \sum_{i=1}^{n} \bar{x}_i{}^2} \tag{25}$$

where $\bar{x_i}$ is the average throughput of source i and n is the number of active sources during the interval considered to calculate the index value.

Fig. 15 plots Jain's fairness index for every combination of XCC algorithms, calculated over periods of 1 seconds and where flows are only considered if active for the entire interval. The results clearly show that XCC algorithms produce much more accurate flow fairness than Reno. Whilst there is not significant difference between the Blind, ErrorS or MAC variants, RCP proves to be more accurate than XCP. This is understandable as RCP provides *instantaneous* fairness allowing new flows to immediately use the rate shared by existing flows. XCP flows, on the other hand, experience an extended convergence period which may span the order of tens of control intervals.

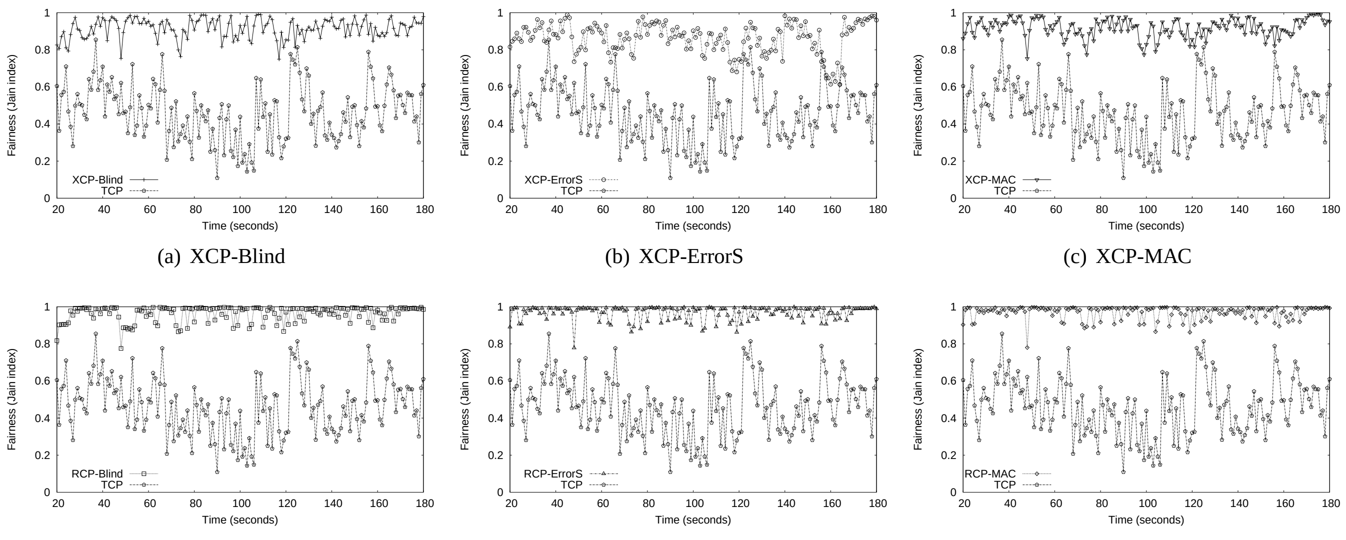


Fig. 15. The Jain's fairness index over time. XCC algorithms vs. Reno.

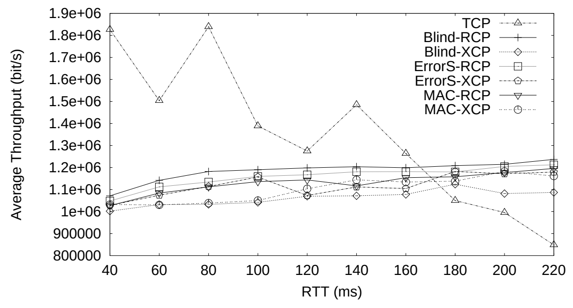


Fig. 16. The amount of traffic transported by flows with different RTTs.

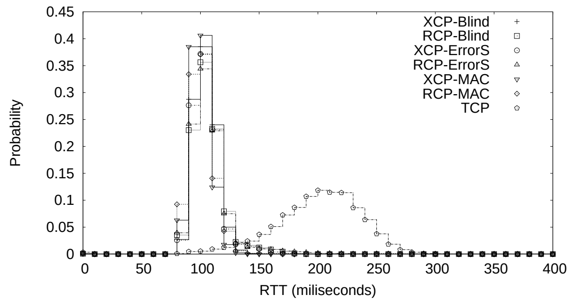
We also test the correlation between flow fairness and the RTT. Using the previous simulation configuration, we only maintain 10 flows active throughout the simulation, with RTTs uniformly distributed within $[40 : 220]$ ms. The results from this experiment are shown in Fig. 16 which shows that the throughput achieved is inversely proportional to the RTT with Reno. This is a well known characteristic of Reno, as its long-term throughput is a function of the inverse of the RTT [13].

XCC algorithms, on the other hand, suffer little impact from RTT heterogeneity. In our results some bias against flows with shorter RTTs is visible, which we believe is caused by the rounding error when using packet units. The smaller the congestion window, the more significant such a rounding error becomes and, as such, smaller RTTs are more affected. Apart from this aspect, XCC algorithms show a clear improvement over Reno regarding RTT-biased fairness.

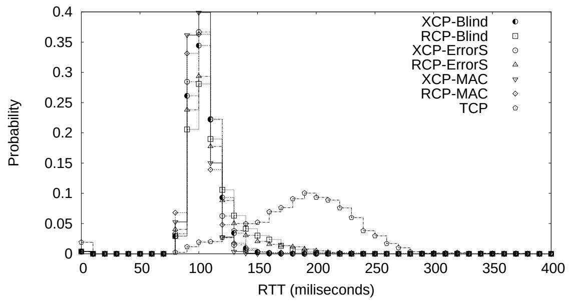
6) Queueing delay: Queue dynamics influence the overall latency perceived by end-systems, which in turn may significantly impact on the interactivity of applications. On our final evaluation we compare queueing delay introduced by XCC algorithms and TCP Reno. With this in mind, we set a fixed propagation RTT of ≈ 80 ms and test two distinct traffic patterns: a first simulation with a total of 12 static flows active throughout the test, and a second simulation which maintains 12 active flows on average, but where half of the flows have an exponentially distributed duration with a mean value of 10 seconds.

The results are shown in Fig. 17 where the probability density function (PDF) of the RTTs for each testcase, as measured by the senders, are plotted. Fig. V-A6 relates to the simulation run with background flows only, while Fig. V-A6 corresponds to the simulation run with a mixture of background and short flows.

As expected, XCC algorithms introduce less queueing delay than Reno by maintaining smaller queues. The queuing delay introduced by Reno is essentially tied to the queue buffer size,



(a) Static Flows



(b) Dynamic Arrival of Flows

Fig. 17. The probability density function of the RTTs measured by the traffic sources. These measurements include the propagation RTT (≈ 80 ms) plus queuing delay.

which for this set of simulations was set to 60 packets. Within XCC algorithms, MAC-based variants introduce less queueing delay when compared to both Blind and ErrorS, which stabilize the queue length at some small positive value and therefore necessarily introduce some additional latency. This is particularly noticeable in the presence of dynamic traffic (Fig. V-A6), as utilisation fluctuations caused by the arrival and departure of flows provoke queue spikes in addition to the constant queue length maintained by Blind and ErrorS. Despite this, the queueing delay introduced by Blind and ErrorS in comparison to the MAC-based algorithm is relatively small in both cases, especially when taking into consideration the overall latency reduction obtained by using XCC algorithms over Reno.

B. Experimental Testbed Results

Our final set of experiments aim at providing proof-of-concept for the Blind and ErrorS algorithms in real-world systems. Both algorithms were implemented on top of the existing FreeBSD implementation of XCP, provided and maintained by the Information Sciences Institute (ISI).

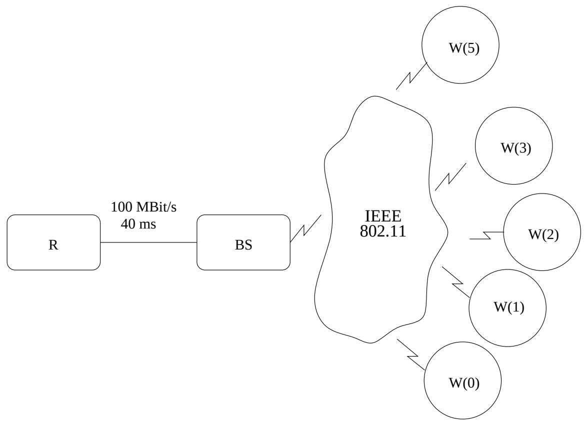


Fig. 18. The test-bed setup. Flows traverse from node R to the wireless clients $W(i)$.

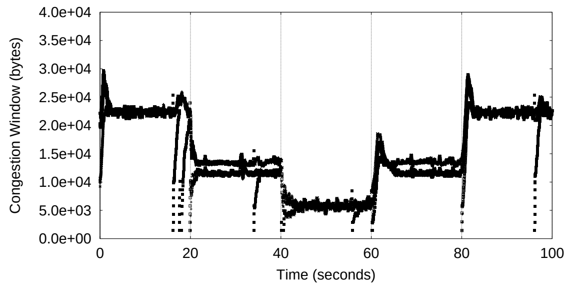
1) Experimental setup: Our laboratory setup (Fig. 18) resembles the scenario used in our single bottleneck simulation. Due to physical limitations we reduce the number of end-systems participating in the experiments to 5 wireless clients $W(i)$, 1 base station BS and 1 wired host R, where all flows are generated. The base station BS is connected to the wired host R directly using a 100 MBit/s Ethernet connection and an induced delay of 40 ms. The wireless clients $W(i)$ are connected to the base station through IEEE 802.11 Network Interface Cards (NICs). The base station itself used an IEEE 802.11 NIC configured in AP mode. The wireless nodes were configured to use a data rate of 24 Mbit/s and configured with virtual queues. The use of virtual queues was required to induce artificial bottlenecks but impedes the saturation of the IEEE 802.11 medium and therefore reduces some of the variability associated to a wireless network. Within the scope of our experiments this drawback is acceptable however, since we are mostly interested in validating previous simulation results. On wireless nodes the virtual queues are configured with a capacity of 5 Mbit/s unless stated otherwise, whilst the virtual queue of the wired host R is set to a capacity of 90 Mbit/s.

C. Basic Dynamics

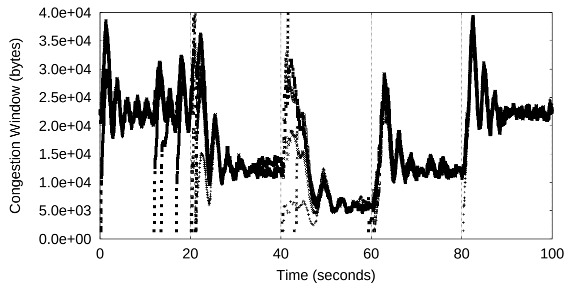
To verify the underlying algorithm properties of both Blind and ErrorS applied to XCP, we set up a total of 5 minute long flows initiated 10 seconds apart. This flow pattern allows us to understand how the network redistributes bandwidth when new flows arrive and also how available bandwidth is allocated as flows depart. The resulting congestion window and queue plots, shown in Fig. 19, and Fig. 20, indicate accurate responses from both algorithms, with slight improvements over simulation results. Such improvement is probably a consequence of not being able to use the wireless medium near its saturation point where bandwidth fluctuations become more significant. Otherwise, both plots are consistent with previous simulations, with ErrorS producing a higher degree of oscillation than Blind but also exhibits a better response to the dynamic arrival and departure of flows. Interestingly, the slower convergence of ErrorS is not immediately apparent in these results. Contrary to our simulation-based results, the initial ramp-up period of ErrorS observable in Fig. 19 was rather short, as the controller had been running for some time before flows started, building up part of the error estimate in the process. Under a real implementation this is unavoidable and, as such, we will leave a more detailed analysis of convergence time to the next section.

D. Response to Variable Bandwidth

To analyze how the queue controller responds to abrupt changes in bandwidth, we run a similar experiment to the simulation present in Section V-A2. In this experiment we run a total of 5 concurrent flows and change the bottleneck capacity every 20 seconds, dropping from an initial capacity of 10Mbit/s, to 5Mbit/s, to 1.5Mbit/s and then throttling the capacity back up in the reverse order. The congestion windows for all flows and queue variables are shown in Fig. 21 and 22, respectively. From the results plotted in Fig. 21 it is clear that ErrorS has the most difficulty dealing with abrupt bandwidth changes, something that is particularly evident by the large queue build-up that the bandwidth reduction from 5

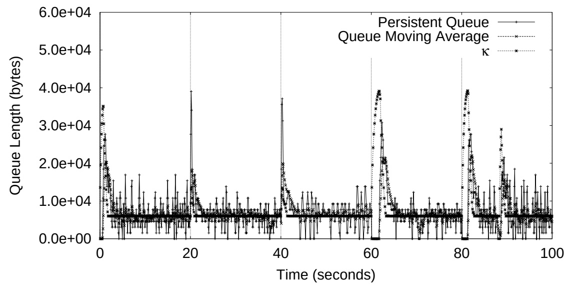


(a) XCP-Blind

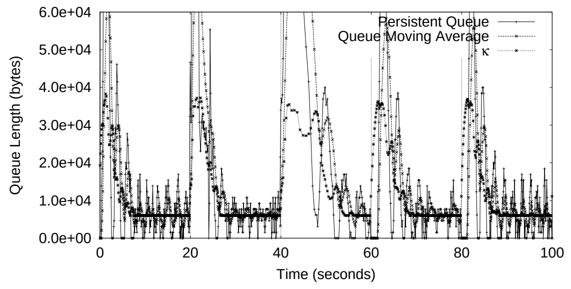


(b) XCP-ErrorS

Fig. 21. The evolution of the congestion window throughout time using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.

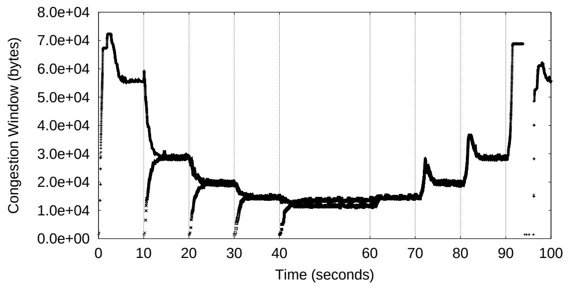


(a) XCP-Blind

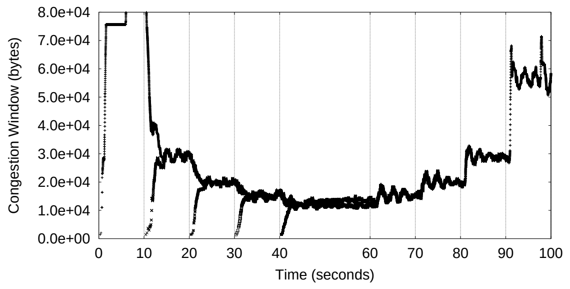


(b) XCP-ErrorS

Fig. 22. The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.

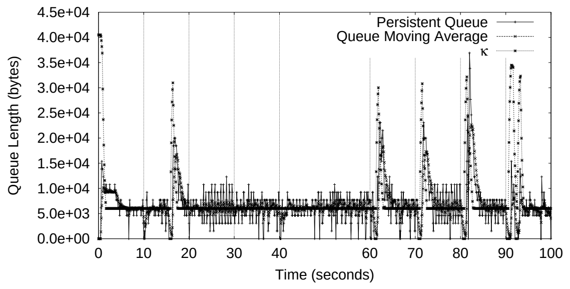


(a) XCP-Blind

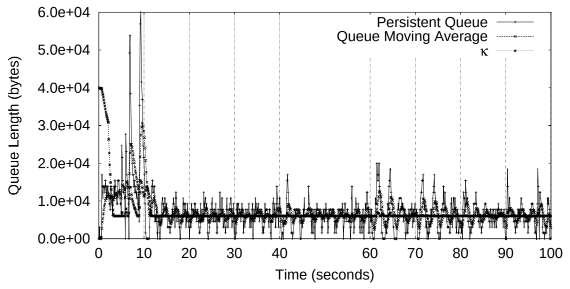


(b) XCP-ErrorS

Fig. 19. The evolution of the congestion window throughout time using our FreeBSD testbed.



(a) XCP-Blind



(b) XCP-ErrorS

Fig. 20. The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using our FreeBSD testbed.

Mbit/s to 1.5 Mbit/s at $t = 40$ s has caused. In comparison, Blind deals with bandwidth reduction effortlessly as only very short queue spikes are noticeable after bandwidth reduction at $t = 20, 40$ s. Both these results are consistent with those obtained in previous simulations, and further emphasize the slower convergence which characterizes ErrorS, leading to increased system delay and potential packet loss on capacity reduction and under-utilization on capacity increase.

VI. Conclusions & Future Work

In this paper we explored the problem of operating XCC mechanisms in transmission media with variable or uknown capapcity. We have proposed three alternative control algorithms: Blind, ErrorS and MAC, which we evaluated both through simulation and in a FreeBSD testbed. Blind and ErrorS use queue properties such as queue speed or queue accumulation to infer the instantaneous capacity of the medium while the MAC algorithm uses information from the MAC layer, such as idle and busy periods. Our evaluation shown that these algorithms maintain most of XCC properties such as stable throughput, low queuing delay, accurate flow-fairness and high efficiency regardless of the network BDP, making these algorithms suitable for multimedia transport in high-speed variable capacity networks, such as IEEE 802.11n.

References

[1] "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/.

[2] "Simulink - simulation and model-based design," http://www.mathworks.com/products/simulink/.

[3] F. Abrantes, J. T. Araujo, and M. Ricardo, "Flash Crowd Effect in RCP," in *Proc. of PFLDnet*, 2008.

[4] F. Abrantes and M. Ricardo, "XCP for Shared-Acess Multi-Rate Media," *ACM Computer Communication Review*, vol. 36, pp. 27–38, 2006.

[5] H. Balakrishnan, N. Dukkipati, N. McKeown, and C. Tomlin, "Stability Analysis of Explicit Congestion Control Protocols," Stanford University Department of Aeronautics and Astronautics Report: SUDAAR 776, Tech. Rep., September 2005.

[6] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor Sharing Flows in the Internet," in *IEEE IWQoS*, June 2005.

[7] A. Falk and D. Katabi, "Specification for the explicit control protocol

(XCP),” Internet Draft (draft-falk-xcp-spec-01), work in progress, October 2004.

[8] V. Jacobson, “Congestion avoidance and control,” in *Proc. of ACM SIGCOMM*, 1988.

[9] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling.* John Wiley and Sons, Inc., 1991.

[10] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin, “Achieving Faster Access to Satellite Link Bandwidth,” in *8th IEEE Global Internet Symposium*, March 2005.

[11] D. Katabi, M. Handley, and C. Rohrs, “Congestion control for high bandwidth-delay product networks,” in *Proc. of ACM SIGCOMM*, 2002.

[12] S. Low, L. Andrew, and B. Wydrowsk, “Understanding XCP: Equilibrium and Fairness,” in *IEEE INFOCOM*, 2005.

[13] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation,” in *Proc. of ACM SIGCOMM*, 1998, pp. 303–314.

[14] J. Postel, “Transmission control protocol,” RFC 793, September 1981.

[15] Y. Su and T. Gross, “WXCP: Explicit congestion control for wireless multi-hop networks,” in *Proc. of IEEE IWQoS*, June 2005.

[16] P. Wang and D. Mills, “Simple Analysis of XCP Equilibrium Performance,” in *Proc. of IEEE CISS*, 2006.

[17] Y. Zhang and M. Ahmed, “A control theoretic analysis of XCP,” in *Proc. of IEEE GLOBECOM*, March 2005.

[18] Y. Zhang and T. Henderson, “An implementation and experimental study of the explicit control protocol (XCP),” in *Proc. of IEEE INFOCOM*, March 2005.

Filipe Abrantes Filipe Abrantes obtained his *Licenciatura* degree in Electrotechnical and Computer Engineering of the University of Porto in 2004. He has since worked as a researcher at INESC Porto and he is currently pursuing his Ph.D. degree. His research focus is on congestion control and wireless routing and forwarding.

João Araújo Filipe Abrantes obtained his *Licenciatura* degree in Electrotechnical and Computer Engineering of the University of Porto in 2004. He has since worked as a researcher at INESC Porto and he is currently pursuing his Ph.D. degree. His research focus is on congestion control and wireless routing and forwarding.

Manuel Ricardo Filipe Abrantes obtained his *Licenciatura* degree in Electrotechnical and Computer Engineering of the University of Porto in 2004. He has since worked as a researcher at INESC Porto and he is currently pursuing his Ph.D. degree. His research focus is on congestion control and wireless routing and forwarding.

APPENDIX A
STABILITY ANALYSIS OF THE ERRORS ALGORITHM

We model the ErrorS algorithm using a fluid model in order to study the stability of ErrorS algorithm as a function of the values of α, β and μ. Abstracting from packet granularity, neglecting queuing delay, and assuming a continuous control loop an ErrorS system is characterized by the following equations:

$$\dot{y}(t) = \frac{F(t-d)}{d} \tag{26}$$

$$F(t) = \alpha \cdot (C - y(t)) - \frac{\mu}{d} \cdot \int_0^t \frac{\beta}{\alpha \cdot d} \cdot q(t)dt - \frac{\beta}{d} \cdot q(t) \tag{27}$$

$$\dot{q}(t) = y(t) - C_{real} \tag{28}$$

$$C = C_{real} + \epsilon \tag{29}$$

where $y(t)$ represents the aggregate incoming bandwidth, C is the capacity value configured at the router, C_{real} is the actual bottleneck link capacity, ϵ is the router capacity estimate error, $q(t)$ represents queue length, and d is the system (average) RTT. α, β and μ are the tunable constants of the algorithm. Now consider the following nomenclature simplification:

$$K1 = \frac{\alpha}{d}; K2 = \frac{\beta}{d^2}; K3 = \frac{\mu \cdot \beta}{\alpha \cdot d^3} \tag{30}$$

which allows us to rewrite Eq. 26 and Eq. 27 as:

$$\dot{y}(t) = K1 \cdot (C - y(t-d)) - K3 \cdot \int_0^t q(t-d)dt - K2 \cdot q(t-d) \tag{31}$$

Applying the Laplace transformation we finally get:

$$s \cdot Y(s) = e^{-sd} \left(K1 \cdot (C - Y(s)) - K3 \cdot \frac{Q(s)}{s} - K2 \cdot Q(s) \right) \tag{32}$$

which is equivalent to the negative feedback control loop represented in Fig. 23. The open-loop transfer function $G(s)$

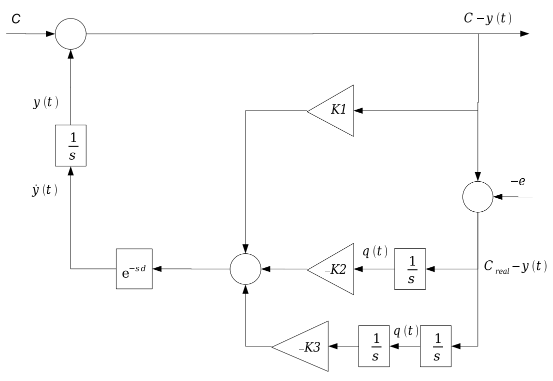


Fig. 23. The ErrorS control loop.

of the the system is given by:

$$G(s) = e^{-sd} \cdot \frac{K1 \cdot s^2 + K2 \cdot s + K3}{s^3} \tag{33}$$

To ensure system stability we have to guarantee that the phase (or gain) margin of the open-loop transfer function is positive. To simplify the analysis consider the following condition:

$$\mu = \beta \tag{34}$$

which implies:

$$K3 = \frac{K2^2}{K1} \tag{35}$$

Using this condition the zeros w_z of $G(s)$ are:

$$w_z = \frac{K2}{2 \cdot K1} \cdot (1 \pm i\sqrt{3}) \tag{36}$$

Now we have to choose the cut-off frequency w_{ErrorS} of the ErrorS system. For the sake of tractability we consider the cut-off frequency as a multiple of the real part of the frequency of the zeros of the open-loop response:

$$w_{ErrorS} = n \cdot \Re\{w_z\} = n \cdot \frac{K2}{2 \cdot K1} \tag{37}$$

Applying the inherent condition of the cut-off frequency (i.e. unitary gain):

$$|G(jw_{ErrorS})| = 1 \tag{38}$$

we get:

$$\beta = \frac{\sqrt{1-\left(\frac{n}{2}\right)^2+\left(\frac{n}{2}\right)^4}}{\left(\frac{n}{2}\right)^3}\cdot\alpha^2 \tag{39}$$

To ensure system stability we must ensure that the open-loop response phase does not exceed $-\pi$ at the cut-off frequency:

$$\angle G(jw_{ErrorS}) > -\pi \tag{40}$$

which results in:

$$\frac{\beta}{\alpha} \le \frac{2}{n}\cdot\left[\arctan\left(\frac{n}{2-\left(\frac{n}{\sqrt{2}}\right)^2}\right)-\frac{\pi}{2}\right] \tag{41}$$

The final step of the design analysis consists in choosing the actual cut-off frequency of the open-loop response by choosing the value of n. We choose n so that the suppression of the error is as fast as possible and the importance given to the erroneous estimation is as low as possible. Considering that the speed of the error suppression is controlled by the parameter μ ($\mu = \beta$), and that α is the weight given to the erroneous capacity estimation, we achieve our design objective by choosing the value of n that maximizes the ratio $\frac{\beta}{\alpha}$. In Fig. 24 we plot this ratio as a function of n, where it is observable that a maximum is obtained for $n \approx 3.5$. Note that using $n = 3.5$ implies a system cut-off frequency w_{ErrorS}:

$$w_{ErrorS} = \frac{3.5}{2}\cdot\frac{K2}{K1} = 1.75\cdot\frac{\beta}{\alpha\cdot d} \tag{42}$$

Applying $n = 3.5$ in Eq. 39, Eq. 41 we get our final set of stability restrictions for the parameters α, β and μ:

$$\beta = 0.5047\cdot\alpha^2; \qquad \frac{\beta}{\alpha} \le 0.4955; \qquad \mu = \beta \tag{43}$$

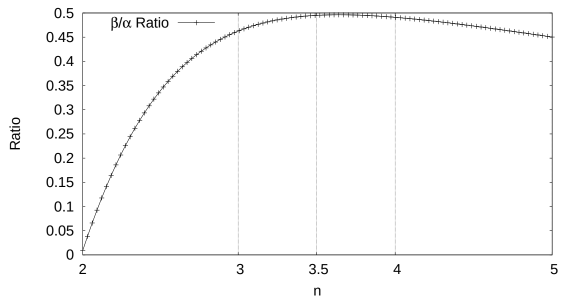


Fig. 24. The maximum value of β as a function of n.

which, simplifying, come as:

$$\beta = 0.5047\cdot\alpha^2; \qquad \alpha \le 0.9818; \qquad \mu = \beta \tag{44}$$

To reduce the oscillation amplitude and to increase system robustness to fluctuations we should add a certain phase margin. Based on simulation results we recommend the utilization of the following values:

$$\beta = 0.1817; \qquad \alpha = 0.6; \qquad \mu = 0.1817 \tag{45}$$

for which the ErrorS algorithm is stable independently of link capacity, delay or number of sources.

APPENDIX B
RECOMMENDED PARAMETER VALUES FOR THE BLIND AND THE ERRORS ALGORITHMS

	Blind	ErrorS
α	0.4	0.6
β	0.226	0.1817
μ	-	0.1817
ρ	0.22	0.15
Q_χ	$0.541\cdot Q_{max}$	$0.444\cdot Q_{max}$
τ	0.225	0.37

TABLE III
RECOMMENDED PARAMETER VALUES FOR THE BLIND AND THE ERRORS ALGORITHMS

We adopt the recommended values of α, β in Blind from [11], as the Blind algorithm maintains the control dynamics of the XCP algorithm. As for the ErrorS algorithms we fix

$\alpha = 0.6$ - allowing a reasonable stability margin - and use the relations found in the previous Appendix that guarantee stability, to dimension β, μ. All the recommended parameter values are shon in B.