Adaptive congestion protocol: A congestion control protocol with learning capability

Marios Lestas a,*, Andreas Pitsillides a, Petros Ioannou b, George Hadjipollas a

a Computer Science Department, University of Cyprus, Nicosia, Cyprus
b Electrical Engineering Department, University of Southern California, Los Angeles, CA, United States

Received 27 June 2006; received in revised form 15 March 2007; accepted 5 April 2007
Available online 19 April 2007

Abstract

There is strong evidence that the current implementation of TCP will perform poorly in future high-speed networks. To address this problem many congestion control protocols have been proposed in literature which, however, fail to satisfy key design requirements of congestion control protocols, as these are outlined in the paper. In this work we develop an adaptive congestion protocol (ACP) which is shown to satisfy all the design requirements and thus outperform previous proposals. Extensive simulations indicate that the protocol is able to guide the network to a stable equilibrium which is characterized by max–min fairness, high-utilization, small queue sizes and no observable packet drops. In addition, it is found to be scalable with respect to changing bandwidths, delays and number of users utilizing the network. The protocol also exhibits nice transient properties such as smooth responses with no oscillations and fast convergence. In realistic traffic scenarios comprising of a small number of long flows and a large number of short flows, ACP outperforms both TCP and XCP, even in the presence of random packet losses. ACP does not require maintenance of per flow states within the network and utilizes an explicit multi-bit feedback signalling scheme. To maintain stability it implements at each link a novel estimation algorithm which estimates the number of flows utilizing the link. Using a simple network model, we show analytically the effectiveness of the estimation algorithm. We use the same model to generate phase portraits which demonstrate that the ACP protocol is stable for all delays.

© 2007 Elsevier B.V. All rights reserved.

Keywords: High-speed; Congestion control; Adaptive

1. Introduction

TCP congestion control has served the Internet remarkably well as this has evolved from a small scale network to the largest artificially deployed system. Despite its profound success, there are currently strong indications that TCP will perform poorly in future high-speed networks. Simulations
and real measurements indicate that as the bandwidth delay products increase within the network, the slow additive increase and the drastic multiplicative decrease policy of the TCP protocol causes the system to spend a significant amount of time trying to probe for the available bandwidth thus leading to under-utilization of the available resources [1]. It has also been shown analytically that as the bandwidth delay products increase, TCP becomes oscillatory and prone to instability [2]. Moreover, TCP is grossly unfair towards connections with high-round trip delays [3]. Finally, it has been shown that in networks incorporating wireless and satellite links, long delays and random non-congestion related losses also cause the TCP protocol to under-utilize the network [4,3].

These observations have triggered intense research activity on Internet Congestion Control which has led to TCP enhancements and new congestion control protocols. The schemes which have emerged from this research effort have different degrees of complexity in terms of the feedback mechanism with which congestion information is transferred from the network to the end users.¹ The current implementation of TCP [5] utilizes a feedback scheme which is binary and implicit, as it uses packet losses to infer congestion. Floyd suggested a modification to TCP which introduces a single ECN bit in the packet header to enable the explicit notification of the end users of the presence of congestion [6]. Such a modification has been shown to improve TCP performance but fails to deliver schemes which satisfy all the design objectives [7]. Binary feedback schemes have fundamental performance limitations.

These limitations have motivated researchers to explore ways with which to enable explicit multi-bit feedback. One idea has been to use the receiver window field in the packet header to transfer explicit congestion information [8,9]. This approach is attractive since it is transparent to the TCP protocol and requires no standardization. However, a single field in the packet header may not be enough to convey the necessary congestion information. A feedback mechanism which has been recently proposed in [10] introduces multiple fields in the packet header which are used to transfer multiple variables between the network and the end users. The performance implications of these explicit feedback schemes on TCP congestion control have been explored in a number of studies. Some of these studies are based on the assumption that fair rate information is available at the network routers [8,11]. Availability of this information can be materialized by leveraging any of the algorithms which have been proposed in the context of the ABR service in ATM networks (e.g., [12–14]). However, these algorithms require maintenance of per flow states within the network and are thus not amenable for implementation in TCP/IP networks. Attempts to develop algorithms which do not require maintenance of per flow states within the network include the queue length based approach in [9], the XCP protocol presented in [10] and the RCP protocol presented in [15]. All these approaches have distinct disadvantages. The scheme proposed in [9] generates feedback signals using queue length information only. However, it is well known that such an approach offers limited control space and thus leads to significant oscillations and degradation in performance in networks with high-bandwidth delay products. The RCP protocol has been designed with the objective of minimizing the completion time of the network flows. In order to achieve the latter, it applies a rather aggressive policy when increasing or decreasing the sending rate of the network users. However, as shown in this paper, such an aggressive policy can cause under-utilization of the network for large periods of time. XCP constitutes the most promising approach as it achieves high-network utilization, smooth and fast responses, scalability with respect to changing bandwidths, delays and the number of users utilizing the network, small queue sizes and almost no packet drops. However, it has been shown in [16] that the scheme fails to achieve max–min fairness and leads to under-utilization of the network in scenarios with multiple congested links. The deficiencies of the fore-mentioned protocols indicate that the problem of high-speed Internet congestion control still remains open. Our main contribution in this paper is to present a new Internet congestion control protocol which fixes the problems of previous proposals and is shown through analysis and simulations to work effectively in a number of scenarios.

We develop ACP (adaptive congestion protocol) which is a new congestion control protocol with learning capability. This learning capability enables the protocol to adapt to dynamically changing network conditions to maintain stability and good
performance. The protocol does not require maintenance of per flow states within the network. Extensive simulations indicate that the proposed protocol guides the network to a stable equilibrium which is characterized by high-network utilization, max–min fairness, small queue sizes and almost no packet drops. It is scalable with respect to changing delays, bandwidths and number of users utilizing the network. It also exhibits nice dynamical properties such as smooth responses and fast convergence. The fore-mentioned properties are maintained even in the presence of non-responsive traffic. The protocol is shown to overcome the problems encountered by XCP and RCP and in realistic traffic patterns comprising of a small number of long flows and a much larger number of dynamic short flows [17–19] it is shown to outperform both TCP and XCP. Superior performance is reported even in the presence of random packet losses, indicating that the protocol can be used to effectively manage congestion in networks incorporating wireless links where such losses are common. Finally, we address stability issues of the ACP protocol by using phase plane analysis. We use a non-linear network model to generate phase portraits which demonstrate that ACP is stable for all delays.

ACP can be characterized as a dual protocol where intelligent decisions are taken within the network. The main control architecture is in the same spirit as the one used by the ABR service in ATM networks. Each link calculates at regular time intervals a value which represents the sending rate it desires from all users traversing the link. A packet as it traverses from source to destination it accumulates, in a designated field in the packet header, the minimum of the desired sending rates it encounters in its path. This information is communicated to the user which has generated the packet through an acknowledgement mechanism. The user side algorithm then gradually modifies its congestion window in order to match its sending rate with the value received from the network. The user side algorithm also incorporates a delayed increase policy in the presence of congestion to avoid excessive queue sizes and reduce packet drops. The justification for the fore-mentioned design choices, and the ways with which these choices fix the problems encountered by other approaches are discussed in Section 2.

As pointed out above, ACP is characterized by its learning capability which enables the protocol to adapt to the highly dynamic network environment to maintain stability and good performance. This learning capability is materialized by a novel estimation algorithm, which 'learns' about the number of flows utilizing each link in the network. Previous experience in the design of congestion control algorithms [20–24] has shown that at each link, the number of flows utilizing the link is necessary in order to maintain stability in the presence of delays. However, this parameter is unknown and time varying. It is thus necessary to develop algorithms which estimate this parameter using local information only. Algorithms which have been proposed, are based on point wise division in time [25–27], an approach which is known to lack robustness and lead to erroneous estimates. In this work we overcome this problem by developing a novel estimation algorithm, based on on-line parameter identification techniques. These techniques originating from robust adaptive control theory [28] enable designs with analytically verifiable properties. In this work we use a network model comprising of a single bottleneck link to derive the estimation algorithm and show analytically that the generated estimates converge to the desired parameter. The extension of this result to networks of arbitrary topology can be found in [29]. We use simulation results to verify our analytical findings and we demonstrate that the proposed estimation algorithm works effectively in a number of representative scenarios.

The paper is organized as follows. In Section 2 we describe the reasoning behind our design choices, in Section 3 we present in detail the congestion control protocol, in Section 4 we evaluate the performance of ACP and finally in Section 5 we present our conclusions and our future research directions.

2. Design guidelines

Our objective has been to design an effective window based congestion control protocol assuming availability of a feedback mechanism which allows the explicit exchange of information between the end users and the network. An effective congestion control protocol must satisfy some basic requirements. It must guide the system to a stable congestion point which is characterized by high-network utilization, max–min fairness, small queue sizes and no packet drops [30]. It also needs to be scalable with respect to changing bandwidths, delays, and number of users. Finally, it must exhibit nice dynamical properties such as smooth responses with fast convergence and no overshoot.
XCP constitutes the most notable attempt to fulfill the above objectives. The protocol achieves most of the specifications but fails to achieve max–min fairness and leads to under-utilization of the network resources in the case of multiple congested links [16]. Since XCP has been shown in [10] to outperform all other TCP proposals, our objective has been to develop a new protocol which outperforms XCP.

At each link in the network, the objective of XCP is to match the input data rate to the link capacity and at the same time maintain small queue sizes. It achieves this by explicitly dictating to each flow utilizing the link the amount by which the congestion window must be increased. The incremental increase for each user is calculated by a signal processor which combines an efficiency controller with a fairness controller. In order to prevent convergence stalling the protocol also introduces the concept of bandwidth reshuffling. In [10] the authors claim convergence of the XCP protocol to the desired equilibrium using a simplified model of the protocol in a single bottleneck link network. The model captures only basic functions of the protocol ignoring many of the details of the mechanisms described above. However, in [16], a more accurate model of the protocol in networks of arbitrary topology reveals that XCP does not always achieve max–min fairness at equilibrium. XCP solves a constrained max–min optimization problem which causes flows to receive an arbitrarily small fraction of their max–min allocations. This leads to under-utilization of the network resources. The theoretical findings have been validated using simulations and are confirmed in this paper in Section 4.8. Moreover, the problem cannot be alleviated by suitable choice of the design parameters because for any choice, there always exists a routing matrix which causes the protocol to under-utilize the network. So, our original motivation has been to design a new protocol which fixes this behavior of XCP.

Since the main problem of XCP is its inability to achieve max–min fairness at equilibrium, we have decided to develop a new protocol based on an architecture which provably achieves max–min fairness. Motivated by local and global stability results which have appeared in the literature throughout the years [31–33] we have chosen an “ATM like” architecture where each link in the network calculates explicitly the sending rate it desires from the users traversing the link. Note the difference with XCP where the link calculates the amount by which the sending rate must be increased. Once the desired sending rate is calculated at each link it is made available to the end users through an explicit feedback mechanism. A packet as it traverses from source to destination it accumulates, in a designated field in the packet header, the minimum of the desired sending rates it encounters in its path. This information is communicated to the user which has generated the packet through an acknowledgement mechanism. The user then gradually modifies its congestion window in order to match its sending rate with the value received from the network. The architecture described above has been used in a number of studies to design congestion control protocols which provably converge to max–min fair equilibrium points. Although for ACP, the establishment of global stability in the presence of delays still remains an open problem, what can be shown is that if the protocol does converge, the equilibrium point is max–min fair. The analysis is not shown in this paper but follows directly from the analysis in [33].

The chosen ACP architecture solves another problem of XCP. It has been observed that XCP is rather slow in increasing the sending rate of new flows leading to large completion times [34]. This effect stems from the fact that XCP dictates the amount by which the sending rate must be increased and this amount is proportional to the excess capacity. When the network is fully utilized the excess capacity is close to zero and so new flows increase their sending rate slowly. This problem is slightly alleviated by the bandwidth reshuffling mechanism which, however, does not help much as only 10% of the bandwidth is reshuffled. ACP fixes this problem since new users are directly notified about their desired sending rate and can quickly converge to this value.

Having decided on the control architecture, the next step is to design the link side algorithm which calculates the desired sending rate. At each link, the objective is to match the input data rate to the link capacity and at the same time maintain small queue sizes. So, the algorithm which updates the desired sending rate must use both queue length and rate information. Ignoring a projection operator which imposes hard bounds on the desired sending rate, a continuous time version of the link side algorithm used by ACP is the following:

$$\dot{p} = \frac{1}{N} \left[ \frac{k_i}{\lambda} (0.99 \ast C - y) - \frac{k_q}{\lambda^2} q \right], \quad p(0) = p_0, \quad (1)$$
where $p$ is the desired sending rate, $C$ is the link capacity, $y$ is the input data rate, $q$ is the queue size, $k_i$ and $k_q$ are design parameters, $d$ is the average propagation delay and $\bar{N}$ is an estimate of the number of flows traversing the link. The algorithm is based on the same principles used in [10,15] to design the link side algorithms of XCP and RCP. However, despite the fact that in all approaches Eq. (1) serves as the starting point for the various designs, the end implementations are quite different. In XCP, the integral action is implemented at the sources, leading to the under-utilization problems pointed out above. In RCP $\bar{N}$ is calculated using a rather ad-hoc estimation method leading to a non-linear link side algorithm which is ultimately different from the one shown in (1). Our approach has been to use tools from formal adaptive control theory to implement a certainty equivalent controller. We have used online parameter identification techniques to derive an estimation algorithm for $\bar{N}$, which we integrate in Eq. (1) to derive the desired certainty equivalent controller. Such controllers have proven to be effective in various applications, and one of the main contributions of this paper is to demonstrate their effectiveness in such a complex dynamical system such as the Internet.

The need to estimate the number of flows utilizing each link in the network in order to maintain stability in the presence of delays, has been pointed out in several studies in the past [20–24]. Many estimation algorithms have been proposed in literature with different degrees of implementation complexity. Algorithms which do not require maintenance of per flow states within the network are based on point-wise division in time. Assuming that all users traversing the link, send data with the desired sending rate $p$, the input data rate $y$ satisfies the following relationship:

$$y = Np,$$

where $N$ is the number of flows traversing the link. Since both $y$ and $p$ are known at the link, $N$ can be estimated by dividing $y$ with $p$. However, such a point-wise division in time is known to lack robustness and can lead to erroneous estimates. In this work, we treat (2) as a linear parametric model of the unknown parameter $N$ and we use online parameter identification techniques to derive an estimation algorithm which is shown through analysis and simulations to work effectively. Details regarding the derivation of the algorithm and its properties can be found in Appendix A.

Eq. (1) is a continuous time representation of the algorithm. In practice, each link updates the desired sending rate every control period. The choice of this period needs careful consideration as it affects the stability and transient properties of the congestion control protocol. ACP sets the control period equal to the average propagation delay which is calculated using the same mechanism as XCP. Each user utilizes a designated field in the packet header to inform the routers of its current round trip time estimate. Each router then calculates the average round trip time over all packets traversing the link and sets the control period equal to this value. We choose this particular control period to counter the destabilizing effect of delays. Based on fundamental control theory principles, as delays increase within the network we slow down the response time of our controllers in order to maintain stability.

As described earlier, each user in the network receives, through explicit feedback, the minimum of the desired sending rates a packet encounters in its path. Since we aim at implementing a window based protocol the rate information must be transformed into window information. We do this by multiplying the desired sending rate with a measure of the propagation delay. Such a measure is obtained by calculating the minimum of the round trip time estimates observed throughout the session. In order to avoid the generation of bursty traffic we do not immediately change the congestion window to the desired value calculated. Instead, we make this change gradually in one round trip time by means of a first order filter.

Even with such a gradual increase policy, excessive queue sizes and packet losses may be observed during transients. The problem arises when a particular link is congested (the input data rate is close to the link capacity) and new users traversing the link enter the network. In this case, the new users adopt the desired sending rate of the link in one round trip time. If the desired sending rate is large, the net effect is a sudden increase in the input data rate at the link. Since, the link is already congested, this increase can lead to large instantaneous queue sizes and packet drops. To alleviate this problem we apply a delayed increase policy at the source in the case of congestion. When a link is congested it sets a designated bit in the header of all incoming packets. In this way the users traversing the link are notified of the presence of congestion and react by applying a delayed increase policy. When they have to increase their congestion window they
decrease the smoothing gain of the first order filter by a factor of 10 so that they increase at a much slower rate, thus giving time to the link to detect the new users, adapt its desired sending rate, and avoid packet losses.

3. The protocol

3.1. The packet header

In a way similar to XCP the ACP packet carries a congestion header which consists of 3 fields as shown in Fig. 1. The \( H_{rtt} \) field carries the current round trip time estimate of the source which has generated the packet. The field is set by the user and is never modified in transit. It is read by each router and is used to calculate the control period. The \( H_{feedback} \) field carries the sending rate which the network requests from the user which has generated the packet. This field is initiated with the user’s desired rate and is then updated by each link the packet encounters in its path. At each link, the value in the field is compared with the desired sending rate value and the smallest value is stored in the \( H_{feedback} \) field. In this way, a packet as it traverses from source to destination it accumulates the minimum sending rate it encounters in its path. The \( H_{congestion} \) bit is a single bit which is initialized by the user with a zero value and is set by a link if the input data rate at that link is more than 95% of the link capacity. In this way, the link informs its users that it is on the verge of becoming congested so that they can apply a delayed increase policy and avoid excessive instantaneous queue sizes and packet losses.

3.2. The ACP sender

As in TCP, ACP maintains a congestion window \( cwnd \) which represents the number of outstanding packets and an estimate of the current round trip time \( rtt \). In addition to these variables ACP calculates the minimum of the round trip time estimates which have been recorded, \( mrtt \). This is a good measure of the propagation delay of the source destination path and is used to transform the rate information reaching the sender to window information.

The initial congestion window value is set to 1 and is never allowed to become less than this value because this would cause the source to stop sending data. On packet departure, the \( H_{feedback} \) field in the packet header is initialized with the desired sending rate of the application and the \( H_{rtt} \) field stores the current estimate of the round trip time. If the source does not have a valid estimate of the round trip time the \( H_{rtt} \) field is set to zero.

The congestion window is updated every time the sender receives an acknowledgement. When a new acknowledgement is received, the value in the \( H_{feedback} \) field, which represents the sending rate requested by the network in bytes per second, is read and is used to calculate the desired congestion window as follows:

\[
\text{desired\_window} = \frac{H_{feedback} \times mrtt}{\text{size}},
\]

where \( \text{size} \) is the packet size in bytes. We multiply with the \( mrtt \) to transform the rate information into window information and we divide by the packet size to change the units from bytes to packets. The desired window is the new congestion window requested by the network. We do not immediately set the \( cwnd \) equal to the desired congestion window because this abrupt change may lead to bursty traffic. Instead we choose to gradually make this change by means of a first order filter. The smoothing gain of this filter depends on the state of the \( H_{congestion} \) bit in the acknowledgement received. If this is equal to 1, which indicates congestion in the source destination path, we apply a less aggressive increase policy. The congestion window is updated according to the following equation:

\[
cwnd = \begin{cases} 
cwnd + \frac{0.1}{cwnd} (\text{desired\_window} - cwnd) & \text{if } \text{desired\_window} > cwnd, H_{congestion} = 1, \\
Pr[cwnd + \frac{0.1}{cwnd} (\text{desired\_window} - cwnd)] & \text{otherwise}, 
\end{cases}
\]

where the projection operator \( Pr[\cdot] \) is defined as follows:

\[
Pr[x] = \begin{cases} 
x & \text{if } x > 1, \\
1 & \text{otherwise},
\end{cases}
\]

Fig. 1. ACP congestion header.
The projection operator guarantees that the congestion window does not become less than 1.

### 3.3. The ACP receiver

The ACP receiver is identical to the XCP receiver. When it receives a packet it generates an acknowledgement in which it copies the congestion header of the packet.

### 3.4. The ACP router

At each output queue of the router, the objective is to match the input data rate $y$ to the link capacity $C$ and at the same time maintain small queue sizes. To achieve the latter, the router maintains for each link a value which represents the sending rate it desires from all users traversing the link. The desired sending rate is denoted by $p$ and is updated every time a control timer expires. The control period is set equal to the average round trip time $d$. The average round trip time is calculated over all $H_{\text{rtt}}$ values recorded in one control period and its initial value is set to 0.05 s. $H_{\text{rtt}}$ values are read from the corresponding field in the packet header upon arrival of a packet.

The router calculates at each output queue the input data rate $y$. For each queue, the router maintains a variable which denotes the number of received bytes. This variable is incremented with the packet size every time the queue receives a packet. When the control timer expires, the router calculates at each queue the input data rate by dividing the received number of bytes with the control period. It then resets the received number of bytes.

The router also maintains at each output queue the persistent queue size $q$. The $q$ is computed by taking the minimum queue seen by the arriving packets during the last propagation delay. The propagation delay is unknown at the router and is thus estimated by subtracting the local queuing delay from the average RTT. The local queuing delay is calculated by dividing the instantaneous queue size with the link capacity.

The above variables are used to calculate the desired rate $p$ every control period using the following iterative algorithm:

$$p(k+1) = Pr\left[ p(k) + \frac{1}{N(k)} \left[ k_i (0.99 \ast C - y(k)) - \frac{1}{d(k)} k_q q(k) \right] \right],$$

$$p(0) = 0,$$

where $k_i$ and $k_q$ are design parameters, $\hat{N}$ represents an estimate of the number of flows utilizing the link and the projection operator is defined as follows:

$$Pr[x] = \begin{cases} 0 & \text{if } x < 0, \\ C & \text{if } x > C, \\ x & \text{otherwise.} \end{cases}$$

The projection operator guarantees that the desired sending rate is non-negative and smaller than the link capacity. Values outside this range are not feasible. The design parameters $k_i$ and $k_q$ are chosen to be 0.1587 and 0.3175 respectively. In Appendix B we show using phase plane analysis that this choice of the design parameters guarantees that the ACP protocol is stable for all delays. Eq. (6) is the discrete time equivalent of Eq. (1). The main idea of the algorithm is to integrate the excess capacity and add a queue size term to guarantee that at equilibrium the queue size converges to zero. Note that we have also chosen to slightly underutilize the link at equilibrium by setting the virtual capacity equal to 99% of the true link capacity. We do this to reserve bandwidth resources which can be used to accommodate statistical fluctuations of the bursty network traffic. This prevents excessive instantaneous queue sizes.

Motivated by previous attempts to design link side algorithms for congestion control, in Eq. (6) we normalize the control parameters of the algorithm with an estimate of the number of flows utilizing the link in order to maintain stability in the presence of delays. A novel part of this work is that we use online parameter identification techniques to derive the estimation algorithm. The derivation is based on a fluid flow model of the network and it is presented in Appendix A together with the properties of the algorithm. Here we present a discrete time implementation of the algorithm:

$$\hat{N}(k+1) = Pr\left[ \hat{N}(k) + \frac{\gamma [y(k) - \hat{N}(k)p(k)] p(k)}{1 + p^2(k)} \right],$$

$$\hat{N}(0) = 10,$$

where the projection operator $Pr[\cdot]$ is defined as follows:

$$Pr[x] = \begin{cases} x & \text{if } x > 1, \\ 1 & \text{otherwise.} \end{cases}$$
The projection operator guarantees that the number of flows traversing the link is never allowed to be less than 1. Values less than one are obviously not feasible. \( \gamma \) is a design parameter which affects the convergence properties of the algorithm. We choose \( \gamma \) to be equal to 0.1. Note that the initial value of the estimated number of flows \( \hat{N} \) is equal to 10. We choose this value to ensure a relatively conservative policy when initially updating the desired sending rate.

The desired sending rate calculated at each link is used to update the \( H_{\text{feedback}} \) field in the packet header. On packet departure, the router compares the desired sending rate with the value stored in the \( H_{\text{feedback}} \) field and updates the field with the minimum value. In this way, a packet as it traverses from source to destination it accumulates the minimum of the desired sending rates it encounters in its path.

The last function performed by the router at each link is to notify the users traversing the link of the presence of congestion so that they can apply a delayed increase policy. On packet departure the link checks whether the input data rate is larger than 95% the link capacity. In this case it deduces that the link is congested and sets the \( H_{\text{congestion}} \) bit in the packet header.

4. Performance evaluation

Our objective has been to develop a window based protocol which does not require maintenance of per flow states within the network and satisfies all the design objectives of congestion control protocols. These objectives have been outlined in Section 2. In this section, we demonstrate through simulations that ACP satisfies these objectives to a very good extent outperforming other protocols such as TCP, XCP and RCP. We conduct our simulations on the ns-2 simulator. In our simulations we mainly consider bulk data transfers but we also evaluate the performance of the protocol in the presence of short web like flows.

4.1. Scalability

It is important for congestion control protocols to be able to maintain their properties as network characteristics change. We thus investigate the scalability of ACP with respect to changing link bandwidths, propagation delays and number of users utilizing the network.

We conduct our study by considering the single bottleneck link network shown in Fig. 2. In the basic setup, 50 users share the bottleneck link through access links. The bandwidth of all links in the network is set equal to 155 Mb/s and their propagation delay is set equal to 20 ms. As mentioned above, the purpose of this study is to investigate the scalability of ACP with respect to changing bandwidths, delays and number of users utilizing the network. When investigating the scalability of the protocol with respect to a particular parameter, we fix the other parameters to the values of the basic setup and we evaluate the performance of the protocol as we change the parameter under investigation. We consider bandwidths in the range 10 Mb/s to 1 Gbit/s, delays in the range 10 ms to 1 s and number of users in the range 1–1000. The performance metrics that we use in this study are the average utilization of the bottleneck link and the queue size of the buffer at the bottleneck link. We consider two measures for the queue size: the average queue size and the equilibrium queue size. The average queue size is calculated over the entire duration of the simulation and thus contains information about the transient behavior of the system. The equilibrium queue size is calculated by averaging the queue length values recorded after the system has converged to its equilibrium state. We do not report packet drops, as in all simulations we do not observe any. In addition, we do not show fairness plots, as in all simulations the network users are assigned the same sending rate at equilibrium, which implies that max–min fairness is achieved in all cases. The dynamics of the protocol and its ability to perform well in more complex network topologies are investigated in separate studies in later sections.

![Fig. 2. Single bottleneck link topology used to investigate the scalability of ACP with respect to changing link capacities, delays and number of users.](image)
In our simulations we consider persistent FTP sources. The packet size is equal to 1000 bytes and the buffer size of all links is set equal to the bandwidth delay product. The simulation time is not constant. It varies depending on the round trip propagation delay. We simulate for a sufficiently long time to ensure that the system has reached an equilibrium state. It is highly unlikely that in an actual network the network users will enter the network simultaneously. So, in all scenarios, the users enter the network with an average rate of one user per round trip time.

4.1.1. Effect of capacity

We first evaluate the performance of the ACP protocol as we change the link bandwidths. We fix the number of users to 50, we fix the propagation delays to 20 ms and we consider link bandwidths in the range 10 Mb/s to 1 Gbit/s. Plots of the bottleneck utilization and the average queue size versus the link capacity are shown in Fig. 3.

We observe that ACP scales well with increasing bandwidths. The protocol achieves high-network utilization (≈98%) at all bandwidths. Moreover, the queue size always converges to an equilibrium value which is close to zero. The average queue size remains very small but we do observe an increasing pattern. The reason for this, becomes apparent when we investigate the transient properties of the protocol. In the transient period, during which the users gradually enter the network, the queue size at the bottleneck link experiences an instantaneous overshoot, before settling down to a value which is close to zero. As the bandwidth increases the maximum value of this overshoot increases, thus causing the average queue size to increase as well. However, in all cases the queue size at equilibrium is small as required.

4.1.2. Effect of delays

We then investigate the performance of ACP as we change the propagation delay of the links. Any change in the link propagation delay causes a corresponding change in the round trip propagation delay of all source destination paths. We fix the link bandwidths to 155 Mb/s, we fix the number of users to 50 and we consider round-trip propagation delays in the range 10 ms to 1 s. Plots of the bottleneck utilization and the average queue size versus the round trip propagation delays are shown in Fig. 4.

The results are similar to the results obtained when investigating the effect of changing capacities. Fig. 4a demonstrates that the protocol achieves high-network utilization at all delays. The equilibrium queue size remains very small, however, the average queue size increases. This trend, as in the case of capacities, is due to the increasing instantaneous queue size in the transient period. As the propagation delays increase, the maximum of the overshoot observed in the transient period increases, thus causing an increase in the average queue size. Despite this increase in the average queue size, the queue size at equilibrium is always close to zero as required.

4.1.3. Effect of the number of users

We finally investigate the performance of ACP as we increase the number of users utilizing the single bottleneck link network in Fig. 2. We consider

![Fig. 3](image-url)  
Fig. 3. (a) Utilization versus capacity; (b) average queue size versus capacity. ACP achieves high-network utilization and experiences no drops as the capacity increases. The average queue size increases with increasing capacity due to larger instantaneous queue sizes in the transient period. However, at all capacities, the queue size at equilibrium is close to zero.
different number of users in the range 1–1000. Plots of the bottleneck utilization and the average queue size versus the number of users are shown in Fig. 5. We observe that up to approximately 800 users the protocol satisfies the control objectives as it achieves high-network utilization and small queue sizes. However, unlike the previous two cases, the equilibrium queue size is close to the average queue size. The reason for this is that as the number of users increases the queue size experiences oscillations. The oscillatory behavior is caused by the congestion window taking only integer values. When the fair congestion window is not an integer (which is the common case), the desired sending rate at the link is forced to oscillate about the equilibrium value, thus causing oscillations of the input date rate and the queue size. The effect of this rounding error increases as the number of users increases, leading at some point to a slight degradation in performance. We observe in Fig. 5 that when the network is utilized by more than 800 users, the utilization drops to about 90% and the average queue size increases. The natural question that arises is the following: does this behavior imply that ACP will perform poorly in realistic scenarios which involve network routers serving potentially thousands of connections? The answer is no. What the above simulation study demonstrates is that ACP is able to maintain good performance properties as the number of simultaneous users increases, until this num-

---

Fig. 4. (a) Utilization versus delay; (b) average queue size versus delay. ACP achieves high-network utilization and experiences no drops as the round trip propagation delay increases. The average queue size increases with increasing propagation delay due to larger instantaneous queue sizes in the transient period. However, at all delays, the queue size at equilibrium is close to zero.

Fig. 5. (a) Utilization versus number of users; (b) average queue size versus number of users. ACP achieves high-network utilization and experiences no packet drops as the number of users increases. At high-number of users, the utilization drops slightly and the average queue size increases. The reason is that the fair congestion window is small (close to 1). Since the congestion window can only take integer values both the utilization and queue size oscillate thus causing a slight degradation in performance.
ber is so big relative to the link bandwidth that the system reaches its limits. This happens when the max–min fair congestion window of each user is less than 1. Since ACP is a window based protocol, the congestion window cannot be set to a value less than 1 because this would prevent the users from receiving any additional acknowledgements and send more packets. Since the users are forced to send data at a higher rate than their max–min share, the network is over utilized, thus leading to excessive queue sizes and packet drops. This problem is common to all window based schemes. However, in practice the system does not reach its limits very often. Internet traffic does not comprise of a large number of persistent flows. It is characterized by a small number of long flows (elephants) which contribute the biggest percentage of the network traffic and a large number of small flows (mice) which enter and leave the network in a very dynamic and stochastic manner [17]. The performance of ACP in such a realistic scenario is investigated in Section 4 where it is shown to outperform both TCP and XCP.

4.2. Fairness

Our objective in this work has been to develop a congestion control protocol which at equilibrium achieves max–min fairness. In this section we investigate the effectiveness of ACP to achieve max–min fairness in a scenario where the max–min fair sending rates change dynamically due to changes in the network load. Among other things, we demonstrate that ACP is able to avoid instability in the form of “ping–ponging” and converge to the max–min fair sending rates even in the case of switching bottlenecks when the bottleneck link for a particular flow changes.

We consider the three link network shown in Fig. 6. The bandwidth of each link is set equal to 155 Mb/s and the propagation delay of each link is set equal to 20 ms. Seven users utilize the network at different time intervals. At the beginning only users 1 and 2 utilize the network. The path of the first user traverses all three links while the path of the second user traverses the first link only. During the time that only these two users are active, the first link is the bottleneck link of the network and the fair sending rate for the two links is 77.5 Mb/s. At 20 s users 3 and 4 enter the network. Both users traverse the second link which becomes the bottleneck link for users 1, 3 and 4. User 2 is still bottlenecked at the first link since this is the only link that it utilizes. Note that at 20 s, user 2 increases its window to take up the slack created by user 1 sharing the bandwidth of link 2 with the other 2 users. At 40 s users 5–7 start sending data through the third link, which now becomes the bottleneck link for users 1, 5, 6 and 7. User 2 is bottlenecked at the first link whereas users 3 and 4 are still bottlenecked at the second link.

In Fig. 7 we show the time responses of the congestion window of a representative number of users. These responses are compared with the theoretical max–min allocation values at each time. The actual responses are denoted by solid lines whereas the theoretical values are denoted by dotted lines. We observe that at equilibrium, the actual values match exactly the theoretical values which implies that max–min fairness is achieved at all times.

During the simulation, user 1 changes three bottleneck links. In the first 20 s user 1 is bottlenecked by link 1, in the next 20 it is bottlenecked by link 2 and in the last 20 s it is bottlenecked by link 3. Despite these changes in the bottleneck link, user 1 is able to avoid instability in the form of ‘ping–ponging’ and its congestion window converges to the max–min fair allocation value without experiencing any oscillations.

Another thing to notice, is that during the first 20 s, the congestion windows of users 1 and 2 are different, despite the fact that their theoretical max–min sending rates in this period are the same. There is no inconsistency between the two observations. The two users experience different round trip propagation delays as they travel different number of hops. Although their sending rates are identical, the different round trip times generate different congestion windows. This demonstrates the ability of ACP to achieve fairness in the presence of flows with different round trip times and number of hops. Also note that the response of user 4 equals the
response of user 3 and the response of users 6 and 7 are equal to the response of user 5 and are thus not shown.

Another interesting observation is the overshoot in the response of user 3. This is a result of the second link becoming a bottleneck link only when users 3 and 4 enter the network. During the time that only users 1 and 2 utilize the network, the two users are bottlenecked at the first link, and so the input data rate in the second link is consistently less than the capacity. This causes the algorithm which updates the desired sending rate at the link, to consistently demand an increase in the desired sending rate. Basically, the link asks for more data, the users do not comply because they are bottlenecked elsewhere, and the link reacts by asking for even more data. The desired sending rate, however, does not increase indefinitely. A projection operator in the link algorithm causes the desired sending rate at the second link to converge to the link capacity. When users 3 and 4 enter the network the second link becomes their bottleneck link. Their sending rate thus becomes equal to the desired sending rate computed at the link. Since the desired sending rate is originally equal to the link capacity, the congestion windows of the two users experience an overshoot before settling down to their equilibrium value. This can be observed in Fig. 7. Despite this overshoot the system does not experience any packet drops. The above setting can be used to emulate the case where network users cannot comply with the network’s request because they do not have enough data to send. The above shows the ability of ACP to also cope with this case.

4.3. The dynamics of ACP

To fully characterize the performance of the proposed protocol, apart from the properties of the system at equilibrium, we need to investigate its transient properties. The protocol must generate smooth responses which are well damped and converge fast to the desired equilibrium state. To evaluate the transient behavior of ACP, we consider the single bottleneck link network shown in Fig. 2 and we generate a dynamic environment where users enter and leave the network at different times. In such an environment, we investigate the dynamics of the user sending rates, we examine the queuing dynamics at the bottleneck link, and we also evaluate the performance of the estimator which is used to track the number of users utilizing the network. Our principal aim is to examine the responsiveness of the protocol, i.e., its ability to respond quickly without any oscillations to sudden changes in the available bandwidth. Such changes can be observed due to a number of reasons: link failures, routing changes, bursts of competing traffic, mobility of wireless links, etc. In this section we emulate such changes in the available bandwidth by considering multiple users almost simultaneously entering or leaving the network.

To conduct our study we consider the following scenario. Thirty users originally utilize the single bottleneck link network shown in Fig. 2. At 30 s 20 of these users stop sending data simultaneously. So the number of users utilizing the network is reduced to 10. At 45 s, however, 40 additional users enter the network thus causing the number of users to increase to 50.

In Fig. 8 we present the time responses of the congestion window of a representative number of users. User 1 utilizes the network throughout the simulation, user 30 stops sending data at 30 s and user 40 enters the network at 45 s. The transient behavior of the other users is very similar to the ones shown in Fig 8. We observe that the protocol achieves smooth responses which converge fast to the desired equilibrium with no oscillations. However, in some cases, they experience overshoots. When user 1 starts sending data it converges fast to its max–min fair allocation. Since the users gradually enter the network, the max–min allocation gradually decreases. This is why the congestion win-
The congestion window of user 1 experiences a large overshoot before settling down to its equilibrium value. Note, however, that once the desired sending rate calculated at the bottleneck link has settled down to an equilibrium value, a new user, such as user 30, converges fast to the max-min allocation value with no overshoots. When the 20 users suddenly stop sending data at 30 s the flow of data through the bottleneck link drops thus causing an instantaneous underutilization of the link. The link identifies this drop in the input data rate and reacts by increasing its desired sending rate. This causes user 1 to increase its congestion window. The time response in Fig. 8 indicates fast convergence to the new equilibrium value with no oscillations. However, the response does experience a small overshoot before settling down to its equilibrium value. This slight overshoot is caused by the feedback delays and the pure integral action of the congestion controller. It can be avoided by introducing proportional action. However, such a modification would increase the complexity of the algorithm without significantly improving the performance and is thus avoided. When 40 new users enter the network at 45 s, the max-min fair sending rate decreases. The controller at the bottleneck link iteratively calculates this rate and communicates this information to the end users. This causes user 1 to decrease its congestion window and user 40 which has just entered the network to gradually increase its congestion window to the equilibrium value. We observe from Fig. 8 that user 1 converges fast to the new equilibrium value with no undershoots or oscillations. We also observe that the time response of the congestion window of user 40 experiences a small overshoot before settling down to its equilibrium value. This is due to the fact that the user sets its sending rate equal to the desired sending rate calculated at the bottleneck link while the latter is still decreasing.

We observe in Fig. 8 that ACP achieves fairly fast rise times of the order of a few seconds (usually two seconds). Faster response times could have been reported, had we chosen a more aggressive increase policy at the sources; for example at each source to immediately adopt the desired sending rate as the actual sending rate. This approach, which is used by RCP, is shown in Section 4.1 to lead to severe degradation in performance in the case of multiple users simultaneously entering the network. This demonstrates that there is a tradeoff between the speed of convergence and the achieved performance. ACP overcomes the problems encountered in RCP by choosing a less aggressive increase policy, however, the natural question that arises is the following: are the rise times achieved by this policy small enough? Small rise times are required in order to ensure high-goodput and consequently small completion times for the various flows. In Section 4.5 we demonstrate that in realistic scenarios comprising of a small number of long flows and a very large number of dynamic short flows, ACP achieves small enough rise times to ensure higher goodput than both TCP and XCP.

The next thing we investigate is the transient behavior of the utilization and the queue size at the bottleneck link. In Fig. 9 we show the time responses of the utilization and the queue size at the bottleneck link.

We observe that the link utilization converges fast to a value which is close to 1. When the 20 users leave the network, the flow of data suddenly decreases thus causing an instantaneous decrease in the utilization. However, the system reacts quickly by increasing the sending rate of the remaining users, thus achieving almost full utilization in a very short period of time.

The time response of the queue size indicates that the latter converges to a value which is close to 0. This is what is required by the congestion control protocol in order to avoid excessive queueing delays on the long run. However, in the transient periods during which new users enter or leave the network, the queue size experiences an instantaneous increase. It might seem strange that we observe increasing queue sizes when users leave the network.
This is caused by the fact that the remaining users, while increasing their sending rate to take up the slack created, they experience overshoots. It must be noted that the maximum queue size recorded in the transient period, increases as the bandwidth delay product increases. This is why in our study of the scalability properties of ACP, the average queue size increases as we increase the bandwidths and the delays. However, careful choice of the control parameters at the links and the delayed increase policy that we apply at the sources ensure that these overshoots do not exceed the buffer size and thus do not lead to packet drops.

A distinct feature of the proposed congestion control strategy is the implementation at each link of an estimation algorithm which estimates the number of flows utilizing the link. These estimates are required to maintain stability in the presence of delays. Here, we evaluate the performance of the proposed estimation algorithm. In the scenario that we have described in the previous subsection, the number of users utilizing the single bottleneck link network changes from 30 to 10 at 30 s and becomes 50 at 45 s. So, we evaluate the performance of the proposed estimation algorithm by investigating how well the estimator tracks these changes. In Fig. 10 we show the time response of the output of the estimator.

We observe that the estimator generates smooth responses with no overshoots or oscillations. In addition, the estimator tracks the changes in the number of users and produces correct estimates at equilibrium.

4.4. A multi-link example

Until now we have evaluated the performance of ACP in simple network topologies which include 1, 2 or 3 links. Our objective in this section is to investigate how ACP performs in a more complex network topology. We consider the parking lot topology shown in Fig. 11.

The network consists of 8 links which are connected in series. All links have a bandwidth of 155 Mb/s except link 4 which has a bandwidth of 80 Mb/s. The propagation delay of all links is set equal to 15 ms. Twenty users utilize the network by traversing all 8 links. Moreover, each link in the network is utilized by an additional 20 users.
which have single hop paths as shown in Fig. 11. In this way, all links in the network are bottleneck links and link 4 is the single bottleneck link for the 20 users which traverse the whole network. We evaluate the performance of ACP by examining the utilization and the average queue size observed at each link. We do not report packets drops, as we do not observe any. In Fig. 12, we show on separate graphs the utilization achieved at each link and the average and equilibrium queue size recorded at the link.

Since all links in the network are bottleneck links for some flows, we do expect them to be fully utilized. Indeed, we observe that ACP achieves almost full utilization at all links. In addition, both the equilibrium queue size and the average queue size remain small. At link 4 we observe smaller average queue size. This is due to its smaller bandwidth delay product. This is consistent with our observations in previous sections.

4.5. Performance in the presence of short flows

In our performance analysis so far, we have only considered persistent FTP flows which generate bulk data transfers. However, real Internet traffic is different in nature, often characterized by complex statistical properties. The challenge of developing traffic models which mimic closely the observed behavior of Internet wires is immense. However, several stochastic models have been developed whose outputs have exhibited good agreement with real traffic data. It has been observed that Internet traffic is characterized by relatively few long flows (elephants) and a very large number of short flows (mice) [17–19]. Elephants, although smaller in number, account for the largest percentage of the network traffic. Short flows, are stochastic in nature and variables such as the size of the flows and their arrival times can be modeled as random processes.

In this section, we evaluate the performance of ACP in such a realistic scenario. We consider traffic patterns which comprise of a small number of persistent long flows and a much larger number of short flows whose variables are derived from random distributions. This creates a dynamic environment where a large number of flows enter and leave the network at different times. In such a dynamic environment, performance metrics such as fairness and responsiveness are hard to define,
as the sources mainly operate in their transient regime. An appropriate performance metric in this case is the average goodput achieved by each source. The goodput for each source is defined as the number of new data packets sent per unit time and is directly related to the time it takes for a source to complete a particular session. Higher goodput values imply smaller completion times. Minimization of the completion times is a major objective of any congestion control protocol. In the set of experiments that follow, in addition to the average utilization and the queue size, we also include the average goodput in our performance metrics. The goodput for each source is calculated by dividing the total number of new packets sent with the time it takes to send these packets. We differentiate between the goodput of long flows and the goodput of the short flows. The average value for each of these is calculated over a representative number of flows.

We consider the single bottleneck link network shown in Fig. 2. The bandwidth of each link is set equal to 155 Mb/s and the round trip propagation delay is equal to 80 ms. Twenty persistent FTP flows share the single bottleneck link with short web like flows. Short flows arrive according to a Poisson process. We conduct a number of tests where we change the mean number of flows entering the network every second, to emulate different traffic loads. Note that a mean of 500 flows per second is typical in routers experiencing heavy traffic [19]. The transfer size is derived from a Pareto distribution with an average of 30 packets. The shape of this distribution is set to 1.35. We repeat the same experiment using ACP, TCP and XCP sources and we compare the average goodput values recorded for the long and the short flows in Fig. 13. For each congestion control protocol we also plot the average utilization and the average queue size at the bottleneck link and the results are shown in Fig. 14.

Fig. 13 reveals that ACP achieves higher goodput values than both TCP and XCP. In the case of long flows, the three protocols achieve comparable values, with ACP however, consistently achieving higher values at all traffic loads. It is also worth noting that TCP achieves relatively low goodput at small traffic loads but as the load increases it achieves almost the same values as XCP. In the case of short flows, the superiority of ACP is evident with the goodput exceeding in some cases twice the value achieved by both TCP and XCP.

In Fig. 14 we show the average utilization and the queue size of the bottleneck link for the three protocols. We observe that TCP achieves the highest average utilization and queue size, XCP achieves the lowest utilization and queue size while ACP lies in between the two. This highlights the reasons which cause ACP to achieve higher goodput at all traffic loads. TCP is the most aggressive protocol as it attempts to fill the buffer at the bottleneck link and backs off, only in the case of packets losses. It thus achieves high-utilization and queue sizes at the expense of too many packet losses. So, a considerable percentage of the throughput of each source is used for retransmissions thus yielding low goodput values. The primary objective of XCP on the other hand is to keep small queue sizes. In the case of heavy traffic load, it achieves the latter by reducing the average utilization thus causing small goodput values. Finally ACP is able to achieve high-utilization while at the same time maintain relatively
small queue sizes without experiencing any packet losses. This is one of the reasons which cause ACP to achieve higher goodput than XCP. The second reason is the mechanism with which the congestion window opens at the initial stages of each session. The XCP router notifies its users how much to increase their sending rate depending on the available bandwidth at the bottleneck link. When the link is congested the available bandwidth is small and so new users increase their congestion window slowly. ACP users on the other hand, are notified about the desired sending rate which is common to all users traversing the bottleneck link. This serves as a reference value where the users are asked to converge. ACP users converge to this value relatively quickly thus ensuring high goodput.

4.6. Performance in cross traffic

In all our previous simulation experiments we considered traffic generated only by responsive ACP users. In this section, we investigate the performance of ACP in the presence of non-responsive traffic. We consider the single bottleneck link network shown in Fig. 2. The bandwidth of each link is set equal to 155 Mb/s and the round trip propagation delay is equal to 80 ms. Fifty ACP users share the bottleneck link with 2 UDP sources which inject CBR traffic into the network. We change the sending rate of these sources to account for different percentages of CBR traffic relative to the bandwidth of the bottleneck link. For each sending rate, we evaluate the performance of ACP with reference to the average utilization, the average queue size and the equilibrium queue size achieved at the bottleneck link. Plots of the fore-mentioned metrics as a function of the percentage of the CBR traffic are shown in Fig. 15.

We observe that the performance of ACP is not affected by the CBR traffic, independent of its load. In all cases ACP achieves almost full utilization and the queue size remains small. No packets drops are observed. Although not shown here in all cases we observed that ACP sources allocate the remaining bandwidth fairly between them and that the protocol generates smooth congestion window responses with convergence properties similar to the ones reported in the case of homogeneous ACP traffic.

4.7. Performance in the presence of random losses

In our simulations so far, we have observed that ACP does not cause any packets to be lost. However, packets may incur random loss due to a number of reasons, especially in networks incorporating wireless links. Since such links are expected to be abundant in the future Internet it is important to evaluate the performance of ACP in the presence of random packets losses. ACP, just like XCP, is implemented over TCP which implies that when packets are inferred as lost, the response is identical to that of TCP. Since, however, we know priori that if packets are lost this is due to reasons other than congestion, we have chosen to set the decrease parameter of the AIMD policy equal to 1.

We consider the single bottleneck link network shown in Fig. 2. The bandwidth of each link is set equal to 155 Mb/s and the round trip propagation
delay is equal to 80 ms. We consider a realistic traffic scenario consisting of a small number of long flows and a large number of short flows. Twenty persistent FTP flows share the single bottleneck link with short web-like flows. Short flows arrive according to a Poisson process with a mean of 200 flows per second. The shape of the distribution is set to 1.35. Any packet traversing the bottleneck link may incur random loss with a probability \( p \). The random losses are independent. We conduct a number of tests where we change the loss probability to account for different mean rates of packets losses. We consider loss rates in the range of 0.01–5%. We repeat the same experiments using ACP, TCP and XCP sources and we compare the average goodput values recorded for the long and the short flows in Fig. 16. For each congestion control protocol we also plot the average utilization and the average queue size at the bottleneck link and the results are shown in Fig. 17.

Fig. 16 reveals that independent of the rate with which packets are lost, ACP outperforms both XCP and TCP in terms of the goodput achieved. It is also interesting to observe that XCP achieves higher goodput values than TCP in the case of long flows but lower goodput values in the case of short flows. The main reason which causes ACP to achieve higher goodput than the other two protocols is evident in Fig. 17 where it is shown that it manages to achieve higher utilization of the bottleneck link at all loss rates. The lower utilization achieved by TCP and XCP results in smaller average queue sizes.
at the bottleneck link. Note also, that at low loss
rates, TCP achieves high-network utilization but
large queue sizes which cause packet losses and thus
low goodput. The fact that ACP outperforms both
XCP and TCP in the presence of random packet
losses indicate that the protocol constitutes a good
candidate for deployment in wireless networks where
such random losses are typical. Modifications of the
protocol which will ensure even better performance
in such networks is currently under investigation.

4.8. Comparison with XCP

Our objective in this work has been to develop a
congestion control protocol which does not require
maintenance of per flow states within the network
and satisfies all the design objectives. An explicit
congestion control protocol (XCP) which has been
recently developed in [10], satisfies most of the design
objectives but fails to achieve max–min fairness in
the case of multiple congested links. It has been
shown through analysis and simulations that when
the majority of flows at a particular link are bottle-
neced elsewhere, the remaining flows do not make
efficient use of the residual bandwidth [16]. In this
section, we consider a topology where the above
problem is evident and we demonstrate that ACP
fixes this problem and achieves max–min fairness.

We consider the two link network shown in
Fig. 18. Link 1 has a bandwidth of 155 Mb/s
whereas link 2 has a bandwidth of 80 Mb/s. Eighty
users access the network though 155 Mb/s access
links. The access links of the first 60 users have a
propagation delay of 15 ms, the access links of the
next 10 users have a propagation delay of 100 ms
and the propagation delay of the last 10 users are
set to 2 ms. We have chosen a variety of propaga-
tion delays to investigate the ability of ACP to

![Fig. 18. A two link network, used to investigate the ability of ACP to achieve max–min fairness at equilibrium. We consider a simulation
scenario which involves users with heterogeneous round-trip times.](image-url)
achieve fairness in the presence of flows with multiple round trip times. The first 10 users of the network have connection sinks at the first router and the rest of the users have connection sinks at the second router. This has been done to ensure that both links are bottleneck links for some flows. The first 10 users are bottlenecked at link 1 whereas the remaining users are bottlenecked at link 2.

We simulate the above scenario using both XCP and ACP users. In Table 1 we compare the theoretical max-min congestion window values with the equilibrium values achieved by ACP and XCP. We observe that ACP matches exactly the theoretical values, whereas XCP does not. XCP fails to assign max-min sending rates to the first 10 users which utilize link 1 only. This is consistent with the findings in [16]. The other users traversing link 1 are bottlenecked at link 2 and so the 10 users which are bottlenecked at link 1 do not make efficient use of the available bandwidth. This inefficiency causes underutilization of link 1. This is demonstrated in Fig. 19 where we plot the time response of the utilization achieved at link 1 by the ACP and the XCP users. Obviously XCP causes underutilization of the link, whereas ACP achieves almost full utilization of the link at equilibrium. This example demonstrates that ACP outperforms XCP in both utilization and fairness. Another thing to note in Table 1 is the ability of ACP to achieve max-min fairness despite the presence of flows with a variety of round trip times.

4.9. Comparison with RCP

RCP and ACP were developed independently based on similar design principles. However, the design objectives of the two protocols are different. The main objective of RCP is to minimize the duration of the network flows whereas the main objective of ACP is to optimize network centric performance metrics such as fairness, utilization, queue sizes and packet drops. Although RCP and ACP were motivated by the same design ideas, they implement different algorithms at both the sources and the links. At each source RCP applies a rather aggressive increase policy where it immediately adopts the desired sending rate received from the network as the current sending rate of the source. This is done to ensure that flows with small file sizes finish their sessions quickly. However, such an aggressive decrease policy in the case of congestion, to avoid packet losses. However, as we will see later such an aggressive decrease policy can cause RCP to underutilize the network for a significant time period. ACP on the other hand, applies a more conservative policy both when increasing and when decreasing the source sending rate. This conservative policy ensures no packet losses and high-network utilization. However, it does take several round trip times for each source to converge to its max-min fair sending rate and this can cause larger duration of flows with small file sizes.

RCP and ACP also have fundamental differences in the implementation of the algorithm which updates the desired sending rate. RCP implements a non-linear congestion controller whereas ACP implements a certainty equivalent controller. The
properties of the RCP controller have been established by linearizing the non-linear equations in a small neighborhood about the stable equilibrium point. However, the linear model is a poor approximation of the non-linear model in some regions of the state space. These model inaccuracies can cause the RCP algorithm to deviate significantly from the predicted behavior and perform poorly in some scenarios. Specifically, when the desired sending rate experiences a large undershoot, the controller is very slow in recovering thus causing underutilization of the network for large time intervals. ACP on the other hand, implements a certainty equivalent controller at each link. The controller is designed assuming that the number of users utilizing the link is known. In practice the latter is an unknown time varying parameter. We utilize online parameter identification techniques to estimate this parameter online. We then replace the known parameter in the control algorithm with its estimate to yield the certainty equivalent controller.

RCP performs poorly when the network experiences sudden changes in the traffic load. Such sudden changes can cause RCP to underutilize the network for significant time periods. In this section we demonstrate this behavior of RCP and we show that ACP continues to perform well in the scenarios where RCP performs poorly. We consider the single bottleneck link network of Fig. 2. The bandwidth of each link is set to 155 Mb/s and the round trip propagation delay is set equal to 80 ms. The network is initially utilized by only one user. At 15 s a second user enters the network. This represents a 100% increase in the traffic load at the bottleneck link. We simulate both ACP and RCP networks.

In Fig. 20 we show the time responses of the congestion window of users 1 and 2 for ACP and RCP. We observe that ACP generates smooth responses which gradually converge to their equilibrium values. The sending rate of user 1 converges to the bandwidth of the bottleneck link and then gradually decreases to half of this value when user 2 enters the network. It takes several round trip times for the congestion windows to converge to their equilibrium values. RCP on the other hand adopts a more aggressive response policy. Note how quickly user 1 originally converges to its equilibrium value. However, when user 2 enters the network its sending rate is set equal to the sending rate of user 1. This causes excessive queue sizes at the bottleneck link. The aggressive decrease policy which RCP adopts, then causes the desired sending rate calculated at the link to decrease to the minimum value allowed by the control algorithm, which is one packet. When this happens, the desired sending rate does not recover quickly. It remains close to 1 for ≈ 5 s and converges to the equilibrium value in 15 s. This slow response is a result of the non-linear control algorithm which RCP utilizes to calculate the desired sending rate. The non-linearity causes slow responses when the desired rate experiences large undershoots. This problem is exacerbated as we increase the link bandwidths. This behavior of RCP can cause underutilization of the network for significant time periods. In Fig. 21 we show the time responses of the utilization of the bottleneck link achieved by ACP and RCP.

We observe that RCP underutilizes the network for a significant amount of time when the second user enters the network, whereas ACP achieves almost full utilization in that period.

![Fig. 20. (a) ACP (b) RCP. Time responses of the congestion window of the network users for ACP and RCP. Observe that RCP users converge very slowly to their equilibrium value when user 2 enters the network at 15 s.](image-url)
5. Conclusions

Our main contribution in this paper is to develop an Adaptive Congestion control Protocol (ACP) which is shown through simulations and analysis to satisfy all the design requirements, outperforming previous proposals such as TCP, XCP and RCP. ACP is a window based protocol which does not require maintenance of per flow states within the network. It utilizes an explicit multi-bit feedback signalling scheme to convey congestion information from the network to the end users and vice versa. A distinct feature of the protocol is the implementation at each link of an estimation algorithm which is derived using on line parameter identification techniques. The algorithm generates estimates of the number of flows utilizing the link which are used to tune the control parameters in order to maintain stability. This feature enables the protocol to adapt to dynamically changing network conditions. Extensive simulations indicate that the protocol is able to guide the network to a stable equilibrium which is characterized by max–min fairness, high-utilization, small queue sizes and no observable packet drops. In addition it is found to be scalable with respect to changing bandwidths, delays and number of users utilizing the network. The protocol also exhibits nice transient properties such as smooth responses with no oscillations and fast convergence. Apart from its practical significance, this work also demonstrates the effectiveness of formal control theory techniques in general and adaptive control techniques in particular in delivering efficient solutions in a highly complex networked system such as the Internet. Our next objective is to verify the properties of ACP analytically in networks of arbitrary topology.

Appendix A. Derivation of the estimation algorithm

We consider the single bottleneck link network shown in Fig. 22. It consists of N users, which share a common bottleneck link through high-bandwidth access links. At the bottleneck link we assume that there exists a buffer, which accommodates the incoming packets. The rate of data entering the buffer is denoted by \( y \), the queue size is denoted by \( q \) and the output capacity is denoted by \( C \). At the bottleneck link, we implement a signal processor, which calculates the desired sending rate \( p \). This information is communicated to the network users, which set their sending rate equal to \( p \). We ignore time delays and the queuing dynamics and we assume that the desired sending rate \( p \) is larger than some positive constant \( \lambda \). This is true in practice because the congestion window of the users is never allowed to be less than one packet. Since the sending rate of all users is equal to \( p \), the input data rate is given by the following equation.

\[
y = Np.
\]  

(10)

The number of users \( N \) is the unknown parameter that needs to be estimated. The input data rate \( y \) and the desired sending rate \( p \) can be measured at the bottleneck link and so Eq. (10) constitutes a linear static parametric model of the unknown parameter. Using online parameter identification techniques [28], we derive the following estimation algorithm:

![Fig. 22. Single bottleneck link network used for analysis.](image-url)
The above equation establishes that \( m_b \) where \( e \) to get:

\[
\dot{x} = P[x] = \begin{cases} 
  x & \text{if } x > 1, \\
  x & \text{if } x = 1 \text{ and } x \geq 0, \\
  0 & \text{otherwise.}
\end{cases}
\]

We use the projection operator since we know priori that the number of users cannot be less than 1. The properties of the above estimation algorithm are summarized in the following theorem:

**Theorem 1.** Eqs. (10)–(12) guarantee that \( \hat{N}(t) \) converges exponentially fast to the unknown parameter \( N \).

**Proof.** When \( N = 0 \), no data is traversing the link and \( \hat{N} \) is not updated. So, we only consider values of \( N \) greater or equal to 1. When \( \hat{N} = 1 \), \( \hat{N} \hat{N} \). Since we choose the initial condition of \( \hat{N} \) to be greater than 1, the above implies that \( \hat{N} \) cannot become less than 1. We can thus ignore the projection operator in our analysis.

Substituting Eqs. (10) and (11) in Eq. (12), we can establish that:

\[
\frac{d\hat{N}}{dt} = -\gamma\hat{N} + \frac{p^2}{1 + p^2},
\]

where \( \tilde{N} = \hat{N} - N \). We can solve the above equation to get:

\[
\hat{N}(t) = \hat{N}_0 e^{-\gamma t} \int_0^t e^{\gamma \tau} d\tau \leq \hat{N}_0 e^{\frac{-\gamma}{1+p^2}}.
\]

The above equation establishes that \( \hat{N}(t) \) converges exponentially fast to 0 which implies that \( \hat{N}(t) \) converges exponentially fast to the desired parameter \( N \).

**Appendix B. ACP stability analysis**

We consider the network model described in Appendix A. In this section we also account for the link propagation delays and the queuing dynamics. We assume that all network users have the same round-trip propagation delay which we denote by \( \tau \). Assuming that the sending rate of all users is equal to the same delayed value of the desired sending rate, the input data rate at the link is given by the following equation:

\[
y = Np(t - \tau).
\]

To account for the queuing dynamics we model the queue as a simple integrator with saturation as follows:

\[
\dot{q} = \begin{cases} 
  y - C & \text{if } y - C \geq 0, \\
  y - C & \text{if } y - C < 0 \text{ and } q > 0, \\
  0 & \text{otherwise.}
\end{cases}
\]

We assume that the estimation algorithm at the bottleneck link generates accurate estimates of the number of users utilizing the link. If in addition, we ignore the projection operator which bounds the desired sending rate, a continuous time version of the link side algorithm is as follows:

\[
\dot{p} = \frac{1}{N} \left[ k_i (C - y) - k_q q \right].
\]

We define the variable \( x(t) = y(t) - C \). We substitute the latter in Eqs. (16)–(18) to obtain the following set of differential equations:

\[
\dot{x} = -\frac{k_i}{\tau} x(t - \tau) - \frac{k_q}{\tau^2} q(t - \tau), \quad x(0) = x_0, \quad (19)
\]

\[
\dot{q} = \begin{cases} 
  x & \text{if } x \geq 0, \\
  x & \text{if } x < 0 \text{ and } q > 0, \quad q(0) = q_0, \\
  0 & \text{otherwise.}
\end{cases}
\]

These equations describe the dynamics of the ACP system and have been also used to describe XCP [10] and RCP [15]. In [10], the authors linearize Eq. (20) and use Nyquist analysis, to obtain a set of values for \( k_i \) and \( k_q \) which guarantee that the linear system is stable. The values that we have chosen for ACP lie outside this set. Linear analysis thus predicts that ACP is unstable. Extensive simulations, however, demonstrate that ACP is stable. In this section we show using phase plane analysis, that if we describe ACP using the non-linear Eqs. (19) and (20), the system is stable.

A phase portrait is a plot that shows a representative sample of trajectories for a given system. For the ACP system, we generate trajectories by simulating on Matlab Eqs. (19) and (20). We use the phase portraits that we obtain to demonstrate that the ACP system is stable for a variety of initial conditions \( x_0, q_0 \) and round trip propagation delays \( \tau \). We first investigate the stability properties of ACP as we change the initial conditions \( x_0, q_0 \). We set the propagation delay \( \tau \) equal to 1 s and we choose
the design parameters $k_i$ and $k_q$ to be equal to 0.1587 and 0.3175, respectively, as in ACP. We consider a family of initial conditions in the range 0–100 and for each initial condition we generate one trajectory. In Fig. 23 we show the phase portrait that we have obtained. We observe that all trajectories converge to the origin which is the unique equilibrium point of the system.

We then investigate the stability properties of the ACP system as we change the time delays. We fix the initial conditions of $x$ and $q$ to 100 and 0 respectively, we set the design parameters $k_i$ and $k_q$ equal to 0.1587 and 0.3175, respectively and we consider values of $\tau$ in the range 100–600 ms. For each value of $\tau$ we generate a state trajectory. The resulting phase portrait is shown in Fig. 24. We observe that all trajectories converge to the equilibrium point. We also observe that as we increase the delays the maximum value of the queue size increases.

We have obtained state trajectories for a much larger set of initial conditions and round trip propagations delays which we do not show here for clarity of presentation. In all cases the ACP system was found to be stable. We thus conjecture that the ACP system described by Eqs. (19) and (20) is globally asymptotically stable for all delays when $k_i = 0.1587$ and $k_q = 0.3175$. Analytical proof of this conjecture using Lyapunov–Krasovskii functionals is the topic of current research.

References


Fig. 23. ACP trajectories when the propagation delay $\tau$ is set equal to 1 s. Each trajectory corresponds to a different set of initial conditions for the state variables. All trajectories converge to the origin.

Fig. 24. ACP trajectories for time delays in the range 100–600 ms. All trajectories converge to the origin.

Marios Lestas received the B.A. and M.E. degrees in Electrical and Information Engineering from the University of Cambridge UK and the Ph.D. degree in Electrical Engineering from the University of Southern California in 1999, 2000 and 2005, respectively. He has worked for the EC funded SEACORN, the RPF VIDEO and the UCY ADVIDEO projects. He is currently a visiting lecturer at the University of Cyprus. His research interests include application of non-linear control theory and optimization methods in Computer and Transportation Networks.

Andreas Pitsillides received the B.Sc. (Hons) degree from University of Manchester Institute of Science and Technology (UMIST) and Ph.D. from Swinburne University of Technology, Melbourne, Australia, in 1980 and 1993 respectively. He is a Professor, Department of Computer Science, University of Cyprus, and heads the Networks Research Laboratory (NetRL). He is also a Founding member and Chairman and Scientific Director, of the Cyprus Academic and Research Network (CYNET) since its establishment in 2000. Prior to that he has worked in industry for six years (Siemens 1980–1983, Asea-Brown Boveri, 1983–1986), and from 1987 till 1994 had been with the Swinburne University of Technology (Lecturer, Senior Lecturer 1990–1994, and Foundation Associate Director of the Swinburne Laboratory for Telecommunications Research, 1992–1994). In 1992, he spent a six month period as an academic visitor at the Telstra (Australia) Telecom Research Labs (TRL). His research interests include fixed and wireless Networks (ad-hock and sensor networks, VANETS, WLANs, UMTS Third Generation mobile networks and beyond), flow and congestion control, resource allocation and radio resource management, and Internet technologies and their application in Mobile e-Services, e.g. in Tele-Healthcare and security issues. He has a particular interest in adapting tools from various fields of applied mathematics such as non-linear control theory and computational intelligence, and recently nature inspired techniques, to solve problems in computer networks. He has published one edited book (with Petros Ioannou) on Modelling and Control of Complex Systems, CRC Press, 2007, over 180 research papers and book chapters, presented invited lectures at major research organisations, has given short courses at international conferences and short courses to industry. His work has been funded by
the European Commission IST program, the Cyprus National Research Promotion Foundation (RPF), the Cambridge Microsoft Research Labs, the University of Cyprus, the Swinburne University of Technology, and the Australian Government research grants board. He also serves/served on the executive committees of major conferences, as e.g. INFOCOM, WiOpt, ISYC, MCCS, and ICT. He is a member of the International Federation of Automatic Control (IFAC) Technical Committee TC 1.5 on Networked Systems and TC 7.3 on Transportation Systems, and of the International Federation of Information Processing (IFIP) working group WG 6.3: Performance of Communications Systems. He is also a member of the Editorial Board of Computer Networks (COMNET) Journal.

Petros Ioannou received the B.Sc. degree with First Class Honors from University College, London, England, in 1978 and the M.S. and Ph.D. degrees from the University of Illinois, Urbana, Illinois, in 1980 and 1982, respectively. In 1982, he joined the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California. He is currently a Professor in the same Department and the Director of the Center of Advanced Transportation Technologies which he founded in 1992. He also holds a courtesy appointment with the Department of Aerospace and Mechanical Engineering. He has been the Associate Director of Research for the University Transportation Center METRANS at the University of Southern California since 2006. He was visiting Professor at the University of Newcastle, Australia and the Australian National University in Canberra during parts of Fall of 1988, the Technical University of Crete in summer of 1992 and Fall of 2001 and served as the Dean of the School of Pure and Applied Science at the University of Cyprus in 1995. He has been an Associate Editor for the IEEE Transactions on Automatic Control, the International Journal of Control, Automatica and IEEE Transactions on Intelligent Transportation Systems. He also served as a Member of the Control System society on IEEE ITS Council Committee and his center on advanced transportation technologies was a founding member of IVHS America which was later renamed ITS America. He is currently Associate Editor at Large of the IEEE Transactions on Automatic Control and Chairman of the IFAC Technical Committee on Transportation Systems. He is a member of the Board of Governors of the IEEE Intelligent Transportation Society. He was one of the founders of the University Transportation Center, METRANS, at the University of Southern California and California State University Long Beach. His research interests are in the areas of adaptive control, neural networks, nonlinear systems, vehicle dynamics and control, intelligent transportation systems and intelligent flight control systems. In 1984 he was a recipient of the Outstanding Transactions Paper Award by the IEEE Control System Society and of the 1985 Presidential Young Investigator Award for his research in Adaptive Control. He is a Fellow of IEEE, Fellow of the International Federation of Automatic Control (IFAC) and the author/co-author of 8 books and over 150 research papers in the area of controls, neural networks, nonlinear dynamical systems and intelligent transportation systems.

George Hadjipollas has completed his undergraduate and graduate studies at the University of Cyprus (B.Sc. and M.Sc. in Computer Science, 2001 and 2003, respectively). He is a research associate at the department of Computer Science of University of Cyprus, where he is involved in the IST funded SEA-CORN project, and Cyprus National Research Funded ENDIKTIS project. Currently, he is a Ph.D. Candidate and his research interests include application of formal methods in networking, and resource management and congestion control in IP networks.