Active Window Management: Performance Assessment through an Extensive Comparison with XCP

M. Barbera[1], M. Gerla[2], A. Lombardo[1], C. Panarello[1], M. Sanadidi[2], G. Schembra[1]

[1]DIIT – University of Catania
V.le A. Doria, 6 – 95125 Catania, Italy
{mbarbera, lombardo, cpana, schembra}@diit.unict.it

[2]Computer Science Department
Boelter Hall – UCLA
405 Hilgard Ave., Los Angeles, CA, 90024, USA
gerla@cs.ucla.edu medy@ucla.edu

Abstract. The most efficient approaches defined so far to address performance degradations in end-to-end congestion control exploit the flow control mechanism to improve end-to-end performance. The most authoritative solution in this context seems to be the eXplicit Control Protocol (XCP) which achieves high performance but requires changes in both network routers and hosts which make it difficult to deploy. To this aim we have developed a new mechanism, called Active Window Management (AWM), which is able to maintain the queue length in network routers almost constant providing no loss, while maximizing network utilization. The idea at the basis of AWM is to allow network routers to manipulate the *Advertised Window* field in TCP ACKs. In this way no modifications to the TCP protocol are required. The target of this paper is to propose an extensive numerical analysis of AWM to compare it with the XCP protocol, chosen as reference case.

Keywords: TCP performance optimization, XCP, Congestion Control, Advertised Window.

1 Introduction

Internet congestion occurs when the available capacity is insufficient to meet the instantaneous aggregate demand. The resulting effects are long delays in data delivery and wasted resources due to lost or dropped packets. For these reasons congestion control mechanisms have been introduced in the Internet from the early beginning. Congestion control is implemented by transport protocol algorithms that dynamically adjust rate or window size in response to implicit or explicit indications of congestion. In the Internet, the most used transport protocol is TCP which, in the absence of explicit feedback from the gateways, usually infers congestion from packet losses or from other congestion measurements (changes in throughput or end-to-end delay)[1].

Nevertheless, the most effective detection of congestion can occur in the gateways, that is, where the queuing behavior can be monitored over time. For this reason, Active Queue Management (AQM) techniques, such as RED, REM, have been proposed in the last few years to address performance degradations in end-to-end congestion

This work has been partially supported by the Italian PRIN project FAMOUS under grant 2005093971 and by the AIBER project funded by the MIUR.

control. All the proposed AQM techniques are certainly an improvement over traditional drop-tail queues; however, they have several shortcomings and have been demonstrated to be inefficient because they miss the challenging target of both minimizing network delay and keeping goodput close to the network capacity [2]. So the important issue on how to control TCP burstiness while maintaining the stability of the "AQM+TCP congestion control" system is still open.

A different approach exploits the flow control mechanism in order to improve end-to-end performance and maintain network queue stability. There are two different classes of solutions that adopt this approach. The first class includes schemes that are implemented both in the hosts and in the routers [3], and need undesirable modifications to the TCP protocol, or even its replacement, in order to accept, understand and use feedback signals from gateways. On the contrary, the second class of solutions includes algorithms which are implemented in the gateways only.

The most authoritative solution in the first class is XCP [3], a new transport protocol that relies on the explicit cooperation among XCP routers and XCP senders or receivers. This protocol generalizes the Explicit Congestion Notification (ECN) [4] by allowing XCP routers to inform the XCP senders about the degree of congestion at the bottleneck. Moreover XCP uses two different mechanisms to control utilization and fairness: to ensure efficient utilization of network resources, an *aggregate feedback* is calculated according to the spare bandwidth in the network and the feedback delay; to control fairness, the concept of *bandwidth shuffling* is applied and a *per-packet-feedback* is obtained by simultaneously allocating and de-allocating bandwidth such that the total traffic rate does not change while the throughput of each individual flow changes gradually to approach the flow's fair share. To do so, knowledge about flow parameters is needed but in order to avoid per-flow state in routers and ensure scalability as number of flows increases, the control state is maintained in the packet header by introducing new fields. The explicit cooperation among routers and senders allows XCP to achieve better performance than TCP in terms of fairness, high utilization, and small queue size, with almost no packet drops. However, even if XCP seems to be the most valid solution to address performance degradations in end-to-end congestion control, it also requires changes in the whole system (both network routers and hosts) and that makes it relatively hard to deploy.

The second class of solutions that relies on a distributed flow control mechanism among routers and end-users includes algorithms which are implemented in the gateways only. More specifically, the basic idea is that network routers manipulate the *Advertised Window* parameter in the TCP ACKs. By so doing, routers can drive the TCP end-to-end flow control mechanism while TCP is not aware of this occurrence. Our proposed mechanism belongs to this second class. The main challenge of these algorithms is to derive the *Advertised Window* value to be sent to each TCP source from global congestion information such as the available buffer space, and although total compatibility with all versions of the TCP protocol makes this class of algorithms more attractive than the previous one, solutions proposed so far ([5], [6]) are not able to improve the performance of TCP connections enough to be competitive with XCP.

A previous work of the same authors [7] presents an early proposal of a new mechanism, called Active Window Management (AWM), which is able to maintain the queue length in routers implementing it very close to a given target value, while, at

the same time, avoiding packet losses. In this paper we extend the work in [7] with an extensive analysis of the AWM performance. Moreover we compare AWM with XCP and demonstrate that, when the access router implementing AWM is the bottleneck in the network, TCP performs very closely to XCP, providing no loss and very high network utilization. However, since changes required in the whole system (network routers, and hosts) are less than XCP and, differently from XCP, AWM is perfectly compatible with TCP, it is easier to implement and deploy.

By manipulating the *Advertised Window* parameter in the TCP ACKs, AWM gateways minimize the delay jitter and maximize the goodput of TCP sources. In addition, no per-flow state storage is needed in AWM gateways, ensuring scalability as number of flows increases. The TCP sender works as usual, but the AWM gateway action makes the *Advertised Window* usually more constraining than the *Congestion Window*. We propose to implement the AWM technique in the access network routers, that is, in routers through which both incoming and outgoing packets related to the same TCP connections are forced to pass through, whatever the routing strategy used in the network. The remaining gateways in the network may use other AQM technique, if any. Since changes required in the whole system (network routers, and hosts) are less than XCP, AWM is easier to implement and deploy relative to XCP.

The rest of the paper is organized as follows. After a brief description of the AWM mechanism in Section 2, we show simulation results about comparisons between AWM and XCP in different network scenarios. Finally, in Section 3, we draw our conclusions.

2 The AWM Mechanism

2.1 Mechanism description

The goal of AWM is to maximize link utilization, and at the same time avoid losses. AWM is implemented on gateway nodes and interacts with TCP sources without requiring any modification to the TCP protocol. An AWM gateway, that is, a gateway node implementing AWM, acts on ACK packets sent to TCP sources. More specifically, it uses the 16-bit header field in a TCP packet called *Advertised Window* (*awnd*). Let us recall that, according to the TCP flow control, the TCP receiver uses *awnd* to inform the sender about the number of bytes available in the receiver buffer. The TCP sender, on the other hand, transmits a number of packets given by the so-called *Transmission Window* (*twnd*), corresponding to the minimum value between its *Congestion Window* (*cwnd*) and the last received *awnd*.

Note that immediately after the establishment of the connection, *cwnd* is smaller than *awnd*, and therefore *twnd=cwnd*. However, according to the TCP protocol specification, *cwnd* is increased until a loss is detected. Thus, *twnd* increases as well until a loss is detected, unless *cwnd* becomes greater than the *awnd* value specified by the TCP receiver before losses occur. In this way, if the TCP receiver is able to manage more packets than the network nodes in the end-to-end path, *awnd* is greater than *cwnd*, and so losses may occur.

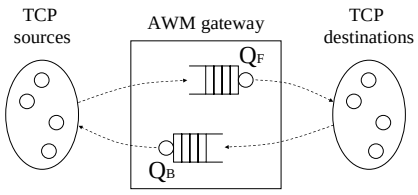


Fig. 1. Buffers considered in the AWM algorithm

With this in mind, the idea at the base of the proposed algorithm is that, before losses occur, the AWM gateway modifies the *awnd* value in ACK packets in order to interrupt the increase in *twnd* values, and therefore control the transmission rate of the TCP sources. By so doing, the *awnd* value received by the TCP sender may be different from that specified by the TCP receiver and the TCP AIMD mechanism is bypassed, providing the network with a traffic shape defined by the AWM gateway.

In order to avoid packet losses and maximize the link utilization, the AWM algorithm tries to maintain the average queue length in the gateway close to a *target* value to be chosen significantly smaller than the buffer size (to have no losses), and at the same time higher than zero (to achieve high link utilization). To this end, based on the state of the queue, the AWM gateway estimates the number of bytes that it should receive from each TCP source. Hereafter we refer to this value as *suggested window* (*swnd*).

The AWM algorithm works on two different buffers in the same gateway (see Fig. 1): Q_F is the buffer loaded by data packets coming from TCP sources; Q_B is the buffer queuing the corresponding ACKs. Therefore, it needs to be implemented in network nodes crossed by both incoming and outgoing packets of the same TCP connections. This is the case of network access routers, which are also very often bottleneck routers. The AWM gateway calculates *swnd* on the basis of the status of Q_F. This is done at each packet arrival and departure in the Q_F buffer, independently of the TCP connection the packet belongs to. In the following the arrival and departure events are called updating events. Every time an ACK packet leaves the buffer Q_B, if its *awnd* value is greater than the last updated value of *swnd*, the AWM gateway overwrites the value of the *awnd* field with the *swnd* value and recalculates the *checksum*. In the opposite case the *awnd* field value remains unchanged in order not to interfere with the original TCP flow control algorithm[1]. Let us stress that this mechanism has been defined in such a way that scalability is not weaken. In fact, the AWM gateway maintains one *swnd* value for the entire set of connections: no per-flow state memory is required, and updates are applied to *awnd* in forwarded ACKs, independently of the particular TCP connection they belong to.

More specifically, the *swnd* value at the generic updating event *k*, $swnd_k$, is evaluated on the basis of its previous value $swnd_{k-1}$, by considering two corrective terms DQ_k and DT_k:

$$swnd_k = \max\left(swnd_{k-1} + DQ_k + DT_k, MTU\right) \quad \textbf{(1)}$$

[1] Clearly the roles of QF and QB are exchanged for TCP traffic in the opposite direction.

The minimum value of the suggested window is set to the Maximum Transfer Unit (*MTU*) in order to avoid the *Silly Window Syndrome* [8].

The term DQ_k in (1) is defined as follows:

$$DQ_k = \frac{1}{N}\left(q_{k-1} - q_k\right) \tag{2}$$

where N is an estimation of the number of active TCP flows crossing the AWM gateway. Its evaluation is discussed in Section 2.2.

The term DQ_k has been defined as in (2) in order to give a negative contribution when the instantaneous queue length q_k is greater than its previous value and a positive contribution in the opposite case. In other words, the AWM gateway detects the bandwidth availability when the instantaneous queue length decreases, and informs TCP sources by proportionally increasing the suggested window value for each of them. When, on the contrary, the queue length increases, the AWM gateway infers incipient congestion and forces TCP sources to reduce their emission rates. If the N TCP sources reduce their transmission window by the term DQ_k, the queue length will go back to the q_{k-1} value.

The term DT_k, on the other hand, has been introduced in order to stabilize the queue length around a given *target* value, T. To this end, DT_k has to give a positive contribution when the queue length is less than T, and a negative contribution in the opposite case. These considerations motivate our choice of evaluating DT_k in the following way:

$$DT_k = \frac{\beta \cdot \delta_k}{N \cdot T} \cdot \left(T - q_k\right) \tag{3}$$

The positive parameter β has to be chosen in such a way that fast convergence to the target is guaranteed, reducing the oscillations as much as possible. In [7], we have presented an analytical fluid model of the AWM gateway to be used for the choice of the appropriate values of β in order to obtain the desired performance.

2.2 Estimation of N

In order to evaluate both the terms DQ_k and DT_k in (1), the AWM gateway has to know the number N of active TCP flows passing through the AWM gateway. As a first step we assume that they have the same average round-trip time, $R(t)$. We will subsequently relax this assumption in section 2.3.

Let $q(t)$ indicate the instantaneous queue length in the buffer Q_F of the AWM gateway at the generic time instant t, and $W(t)$ the value of the TCP source transmission window. Thanks to the AWM mechanism, the transmission window is the same for all the sources and equal to the suggested window indicated by the AWM gateway in the ACK packets coming out of the buffer Q_B at the generic time instant t. The change rate of the queue length is equal to the total arrival rate minus the output rate, that is:

$$\frac{dq(t)}{dt} = \frac{N \cdot W(t)}{R(t)} - C \tag{4}$$

where $R(t)$ is the round-trip time calculated as the sum of the round-trip propagation delay D and the queuing delay, i.e. $R(t) = D + q(t)/C$, being C the link capacity of the buffer Q_F.

Since the β parameter is chosen in order to guarantee the convergence of the queue length to the target, we can conclude that at the steady state the derivative of the queue length is zero. As a consequence, we can rewrite (4) as $N \cdot W(t) = C \cdot R(t)$. Moreover, since the queue length is constant, the average round-trip time $R(t)$ does not suffer of variations during the system evolution, so the product $N \cdot W(t)$ is constant as well. This means that if we consider two consecutive updating events at the instants t_{k-1} and t_k, we have:

$$N_{k-1} \cdot swnd_{k-1} = N_k \cdot swnd_k \tag{5}$$

where $swnd_{k-1}$ and $swnd_k$ are the values of $W(t)$ calculated at the instants t_{k-1} and t_k, respectively.

The above relationship states that if the AWM algorithm needs to change the value of the suggested window to be sent to TCP sources according to (1), the reason is that a variation in the number of active sources is occurred. As a consequence, (5) allows us to adaptively evaluate N_k in the following way:

$$N_k = \frac{N_{k-1} \cdot swnd_{k-1}}{swnd_k} \tag{6}$$

The number of sources immediately after the k-th updating event estimated as in (6) should be substituted in (1) for evaluating $swnd_{k+1}$. However, when one or more TCP sources start, because of the Slow Start mechanism, they will probably have a congestion window less than the suggested window, so they will not react to the AWM gateway indications. Therefore, until their congestion window value does not reach the suggested window, only the other sources will have in charge to maintain the queue length constant. For this reason during this phase the number of sources in (1) will be kept constant and equal to the number of sources already active before the start of the new sources.

The number of source will be varied in (1) only when the Slow Start phase is interrupted by a suggested window value less than the congestion window value. This event is detected by the AWM gateway when the number of sources estimated by AWM, which is continuously evaluated by (6), decreases for the first time. On this occurrence the value of N to be used in (1) will be calculated by gradually increasing the value of N from that calculated before the start of the new sources, until it reaches the value estimated by (6).

2.3 AWM model in presence of sources with different round trip time

Let us consider a set of M TCP sources loading the AWM buffer, and divide them in G groups in such a way that in each group there are TCP sources with the same average round-trip time, R_i, where $i = 1, 2, \ldots, G$[2]. Further let us indicate the number of sources in the generic group i as N_i. In this case (4) can be rewritten as follows:

$$\frac{dq(t)}{dt} = \sum_{i=1}^{G} \frac{N_i \cdot W(t)}{R_i} - C \tag{7}$$

Let R_m denote the mean value of the round-trip time of all the groups of sources, and let us define N_{eq} as follows:

$$N_{eq} = R_m \cdot \sum_{i=1}^{G} \frac{N_i}{R_i} \tag{8}$$

The term N_{eq} represents the equivalent number of TCP sources that should produce the same effects on the AWM buffer if all they had the same round-trip time equal to R_m. According to this notation, (4) becomes:

$$\frac{dq(t)}{dt} = \frac{N_{eq} \cdot W(t)}{R_m} - C \tag{9}$$

Since (4) and (9) have the same form, we can repeat all the considerations about the AWM mechanism by substituting N and R with N_{eq} and R_m, respectively.

3 Numerical Results

In this section we show that AWM appropriately shapes incoming traffic into the network. More specifically, we will show that, with AWM, TCP performs very closely to a pseudo-constant bit-rate protocol providing no loss, and network utilization becomes very close to one. In addition, we compare performance achieved by using AWM with performance obtained with XCP and when no control is applied in the network. That is, we will consider the following three cases:

- **AWM** - sources use TCP NewReno unmodified as the transport protocol, and gateways in the network implement AWM mechanism over a buffer with Drop-Tail dropping policy;
- **NC** - sources use TCP NewReno unmodified as the transport protocol, but gateways do not implement any particular mechanism; buffer dropping policy is Drop-Tail;
- **XCP** - sources use XCP as transport protocol, and gateways in the network are XCP as well, with a buffer using Drop-Tail dropping policy.

[2] As mentioned above, at the steady state the round-trip time does not suffer appreciable variations. Therefore we can remove its temporal dependence, and refer to it with the constant value R.

Let us observe that, although we are considering TCP NewReno, any TCP version can be used achieving the same results for AWM given that AWM by-passes the TCP control mechanism, which is just the key element making the difference between different versions of TCP.

Comparison is carried out through extensive simulations in different scenarios with the ns-2.29 simulator [9], integrated with a module implementing AWM. For the sake of space we present just a subset of the obtained simulation results. In order to have a fair comparison with XCP, we use the same case study considered in [3], where the XCP authors demonstrated its superiority over other important schemes like RED, REM, AVQ and CSVQ. The used network topology presents a single bottleneck shared by a number of sources (see Fig. 2(a)). The bottleneck capacity, the round-trip propagation delay and the number of flows vary according to the specific goal of each experiment. When unspecified, the link capacity is set to 150 Mb/s, the link propagation delay to 80 ms and the number of flow in the forward direction to 50. All flows have the same round-trip propagation delay and sources are long-lived FTP flows. In order to create a 2-way traffic with the potential for ack compression, we use the same number of flows in the forward and reverse paths. The MTU length is 1000 bytes. The buffer dropping policy is Drop-Tail. The buffer size is set equal to the bandwidth-delay product, while the target value for AWM is half a buffer size.

We present results for link utilization and lost packet percentage. Moreover we introduce a robustness index, defined as the standard deviation of the queue length normalized with respect to the minimum distance between the average queue length and the two buffer limits (zero and buffer size).

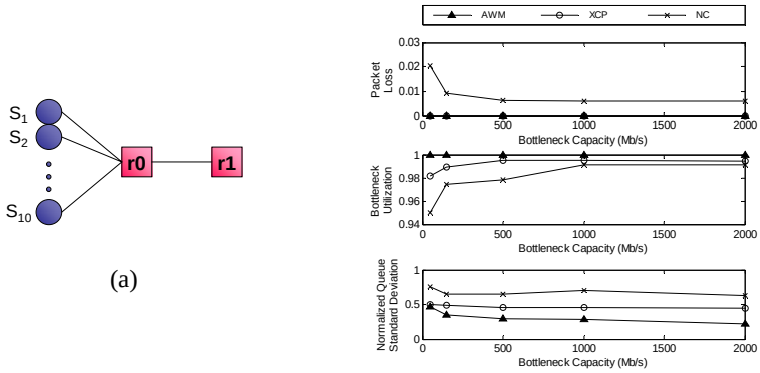
This definition comes from the consideration that the standard deviation is not able to represent by itself the impact of queue length oscillations on buffer performance. In fact, if the average queue length is close to zero, large space is available in the buffer to absorb an increase of the queue length without losses, but an even small decrease may empty the buffer and cause underutilization; on the contrary, if the average queue length is close to the buffer size, relatively large oscillations may lead to buffer overflow.

Bottleneck capacity – First we investigated how the bottleneck capacity affects performance of the three considered schemes. Fig. 2(b) shows performance in terms of loss percentage, bottleneck utilization and normalized queue standard deviation. From that figure we can observe that no losses occur with AWM and XCP. Moreover, only AWM provides 100% link utilization in all the considered cases. This is because the queue length is always close to the target value therefore greater than zero. In fact the amplitude of oscillations of the queue length around the average is very small for AWM, which means that the queue is stable around its average value. In addition, since the round-trip time is function of the bottleneck queuing time, small oscillations around the average queue length make variance of the round-trip time small as well, implying small values of delay jitter.

Round-Trip Propagation Delay – To study the impact of increasing delays, we vary the propagation delay of each link. The bottleneck capacity is set to 150Mb/s while all the other parameters maintain the same values used in the previous analysis. Fig. 3(a) shows that performance achieved by AWM, as well as XCP, are not affected by the round-trip propagation delay.

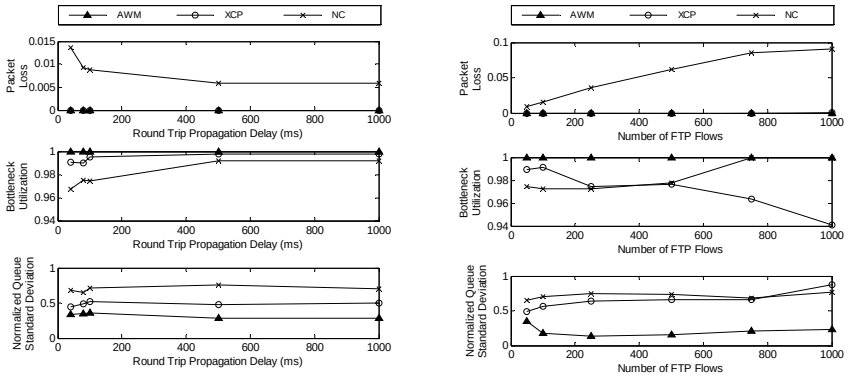
Number of flows – We analyzed the impact of the number of flows on congestion control. Even with a large number of flows, AWM is able to guarantee 100% link utilization and zero loss (see Fig. 3(b)). The amplitude of oscillations around the average queue length, and consequently the delay jitter, grows as the number of flows increases. This behavior is due to the rounding errors introduced when TCP sources convert the number of bytes specified by the suggested window in an equivalent number of packets. As the number of flows increases, the suggested window decreases and the equivalent number of packets decreases as well, making the effects of the rounding errors more evident.

Short Web-Like Flows – Finally we consider the impact of the presence of short web-like sources (mice) on performance of the three considered schemes. To this aim we assume short flows arrive according to a Poisson process, with Pareto-distributed file sizes [10]. Fig. 4(a) demonstrates that AWM is robust in dynamic environments with a large number of flow arrivals and departures. Once again, the utilization is 100% while no losses occur even when the number of short flows grows significantly;



(a) (b)

Fig. 2. (a) Network topology: Single Bottleneck – (b) Performance vs. LinkCapacity



(a) (b)

Fig. 3. (a) Performance vs. Round-Trip Propagation Delay – (b) Performance vs. Number of FTP Flows

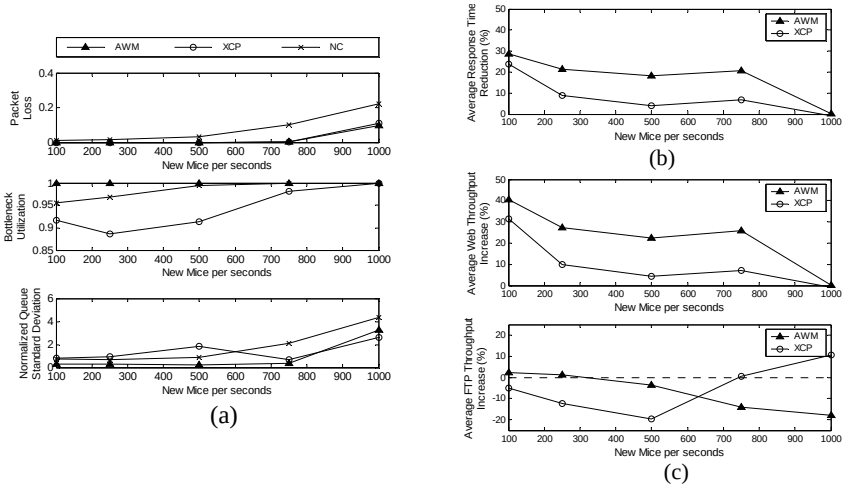


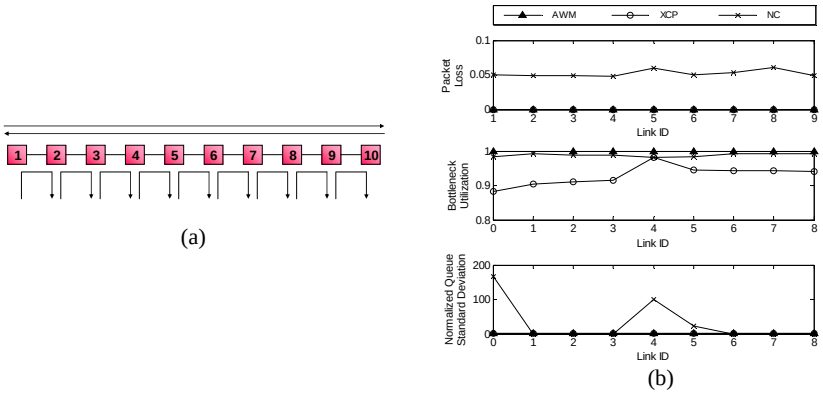
Fig. 4. (a) Performance vs. Mice Arrival Rate – (b) Average Response Time Reduction vs. Mice Arrival rate – (c) Average Throughput Increase vs. Mice Arrival Rate

but as soon as the number of new mice per second becomes higher than 750, the number of simultaneously active flows becomes larger than the buffer size measured in packets: there is no space in the buffer to maintain a minimum of one packet from each flow and both AWM and XCP start dropping packets.

For short web-like sources, as additional performance parameter we consider the Response Time that is, the time interval between the instant in which the users make the web file request and the instant in which its transfer is completed. Fig. 4(b) shows the reduction in the average Response Time among all web connections in AWM and XCP relative to the NC case. When AWM, as well as XCP, is applied, average response time reduction of 20-30% is possible.

Moreover, it is interesting to investigate if this reduction of the response time happens because AWM, or XCP, allows short web-like flows to gather more bandwidth than long FTP-like flows. To this end we show the variation of the Average Web Throughput and the Average FTP Throughput obtained when AWM and XCP schemes are implemented and compare these results with the NC case. Fig. 4(c) shows that AWM is able to substantially increase the throughput of Web sources without wasting the throughput of the FTP sources. In fact, let us consider the case of 500 new mice per second as an example. The average web throughput obtained with AWM is approximately 25% more than the NC case, while the reduction of the average FTP throughput is less than 5%. On the contrary, XCP increases the average web throughput by 5% but the average FTP throughput is approximately 20% less than the NC case.

Topology with multiple bottleneck links – In order to demonstrate that results shown so far are independent from the particular topology, we considered a more complex scenario with multiple bottlenecks. Once again, for comparison we choose to use the same topology used by the authors of XCP. The topology is composed by 9 links as shown in Fig. 5(a). Capacity of link 5 is 50Mb/s, while all other links have a capacity of 100Mb/s. The propagation delay of every link is 20ms. Links are traversed



(a) (b)

Fig. 5. (a) Network topology: Multiple Bottlenecks – (b) Performance vs. Link ID

by 50 flows both in the forward and reverse directions, from node 1 to node 10. In addition, in order to have multiple bottleneck links, 50 flows traverse every link from node i to node $i+1$, with $i = 1, 2, \ldots, 9$.

In this scenario, XCP provides lower link utilization than the other considered schemes (Fig. 5(b)). As XCP authors explained, this is because, at links 1, 2, 3 and 4, XCP tries to share the bandwidth between long-distance and short-distance flows. Nevertheless, throughput of long-distance flows is limited by the same mechanism at the most stringent bottleneck link 5. To overcome this drawback, XCP could be modified in order to keep the queue length around a *target* value and improve the link utilization, but the effect of this choice would be a fluctuation in the queue. However XCP authors chose to trade "*a few percent of utilization for a considerably smaller queue*". The opposite choice (i.e. to trade a greater average queue length for the maximum link utilization) can be made for AWM without affecting the stability of the queue around the target. This is demonstrated in Fig 5(b) where the normalized standard deviation is very small. Results show that packet losses are zero for both AWM and XCP schemes.

Fairness – Simulation results are obtained considering 30 flows with a bottleneck link of 30Mb/s. As regards the fairness metric, we use the Jain fairness index [11]. Considering N flows, with flow i receiving a fraction x_i of the given link bandwidth, the Jain fairness index is defined as $\left(\sum x_i\right)^2 / \left(N \sum x_i^2\right)$. Simulation results are presented in Table I. In case 1, flows have the same round-trip propagation delay equal to 40ms and performance of three considered schemes are quite the same. In case 2, different round-trip propagation delays are considered. In particular round-trip propagation delays of each flow are chosen as $D_i = D_{i-1} + 10$ms, where $i = 2, 3, \ldots, 30$ and $D_1 = 40$ms. Simulation results demonstrate that AWM is fairer than the non-controlled case, but as expected, thanks to the explicit collaboration between XCP users and XCP network nodes, XCP is fairer than AWM. In fact, an XCP router receives, from each XCP source, information about the estimated round-trip time and on the base of this value calculates a different feedback for each source. On the con-

trary, since the goal of AWM is to preserve the compatibility with TCP, AWM does not receive any information regarding the estimated round trip time and calculates the *swnd* regardless of the specific TCP connection would receive it.

Table 1. Jain's Fairness Index

Case	AWM	XCP	NC
1	1.0000	1.0000	0.9937
2	0.7007	0.9954	0.4392

4 Conclusions

In this paper we have proposed an extensive numerical analysis to compare the AWM mechanism, intended to enhance TCP congestion control, with the eXplicit Control Protocol (XCP), chosen as usual as reference case. Results demonstrate that AWM performs very closely to XCP. It achieves high performance as it stabilizes the queue length in the network access routers by manipulating the *Advertised Window* parameter in TCP ACKs. Moreover it is scalable as the number of flows increases. However, AWM application is easier than XCP since it requires implementation only at access gateways and no modification at internal routers or at hosts is required.

References

1. L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, No. 8 (October 1995) pages 1465-1480.
2. A. Bitorika, M. Robin, M. Huggard, C. Mc Goldrick, "A Comparative Study of Active Queue Management Schemes," in *Proc. of ICC2004* (June 2004)
3. D. Katabi, M. Handley and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks", in *Proc. of ACM Sigcomm 2002*
4. S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, Vol. 24 No. 5, pages 10–23, Oct. 1994.
5. L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, Explicit window adaptation: A method to enhance TCP performance. *IEEE/ACM Transactions on Networking*, 10(3):338–350, June 2002.
6. M. Savoric, Fuzzy Explicit Window Adaptation: Using router feedback to improve TCP performance, Technical Report TKN-04-009, July 2004
7. M. Barbera, A. Lombardo, C. Panarello, G. Schembra, "Active Window Management: an efficient gateway mechanism for TCP traffic control", in Proc. of IEEE ICC 2007, Glascow (Scotland), June 2007
8. R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122, October 1989.
9. The network simulator ns-2. http://www.isi.edu/nsnam/ns
10. P. Barford, M. Crovella, "Generating representative Web workloads for network and server performance evaluation", in *Proc. of ACM SIGMETRICS 199*
11. Jain R. Chiu D. and Hawe W. 1984, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems", DEC Research Report TR-301.