社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－DPS－128 (15)
2006－GN－ 61 (15)
2006－EIP－ 33 (15)
2006／9／15

Improvements in TCP Robustness for Asymmetric Bandwidth

DAIGO TANAKA† and HIROSHI SHIGENO††

The Transmission Control Protocol (TCP) proves degraded performance at the environment which has bandwidth asymmetry, since the uplink is filled with ACK packets. In order to solve the issue, we present a TCP option, Pulsed ACK. To apply this option, TCP brings down the number of ACKs and allows receivers to send ACKs at intervals. We consider d, the number of data packets acknowledged by an ACK to achieve these effects, and using this value, receivers adjust the interval of sending them. The senders also control the increment of *cwnd* at the recerpt of an ACK packet so that it increases by 1 every RTT. For evaluation we use computer simulations. The results show that TCP with the Pulsed ACK option gains 1.85 times higher throughput compared to TCP without the option.

1. Introduction

The diversification of telecommunication infrastructure has brought forth numerous models of networks. Among them, links with bandwidth asymmetry such as Asymmetric Digital Subscriber Line (ADSL), satellite and wireless networks have been deployed rapidly these years. Bulk data transfer such as Peer-to-Peer (P2P) connections and multimedia streaming also induces bandwidth asymmetry, since they use large amount of bandwidth in one direction, while the opposite direction is hardly used.

Researchers have pointed out that TCP has difficulty gaining throughput at links with bandwidth asymmetry[1)~7)], either in capacity or due to the connections in the reverse direction, since its ACK packets are delayed or lost because of the saturation of the uplink. Although there are many works addressing this issue, most of them require high performance intermediate nodes, which make them difficult to deploy. Also, in all works, the number of ACK packets increase as the *cwnd* becomes large; just as regular TCP does.

We focus on this characteristic of increasing ACK packets. It is obvious that ACK packets are crucial to the performance of TCP, though regular TCP connections use much more than necessary. When the *cwnd* increases at the sender, it means the number of ACK packets sent from the receiver increases too, which means the connection uses more of the uplink bandwidth. In other words, if the usage of the uplink bandwidth is proportional to that of the downlink bandwidth, the connection can not cope with links that are extremely asymmetric. As mentioned above, asymmetric links are omnipresent, therefore it is necessary for TCP connections to reduce the usage of the uplink in case they transfer such links.

To address the issue, we propose Pulsed ACK, which enables receivers to send a *fixed* number of ACK packets per RTT, making the usage of the uplink small. It also enables receivers to send ACK packets at intervals, to minimize the burst of the increase of *cwnd*. We do this by redefining d, the number of data packets the receivers receive to send an ACK packet, as a function of *cwnd*. Using this d, the receivers adjust the interval of sending ACKs. Also, the senders adjust the increment of *cwnd* at the receipt of an ACK packet so that it increases by 1 every RTT.

We evaluate the performance of Pulsed ACK through computer simulations.

This paper is organaized as follows. We first describe Delayed ACK, which is one of some works to address the issue above in Section 2. We then present the details of Pulsed ACK design in Section 3, and we evaluate the performance of our proposal in Section 4. We then conclude the article in Section 5.

2. Related Work

In this section, we show a TCP option, Delayed ACK, which is one of some approaches to tackle the issue.

There are two standard methods that can be used by TCP receivers to generate acknowledgments. The method outlined in[8)] generates an ACK for each incoming data segment (i.e., $d = 1$). d is the number of TCP data segments acknowledged by a TCP ACK. In other

† Graduate School of Science and Technology, Keio University
†† Faculty of Science and Technology, Keio University

words, a TCP receiver generates an ACK every d packet(s).

[9] states that hosts should use "delayed acknowledgments." Using this algorithm, an ACK is generated for at least every second full-sized segment ($d = 2$), or if a second full-sized segment does not arrive within a given timeout which must not exceed 500 ms[9], and is typically less than 200 ms. Relaxing the latter constraint (i.e., allowing $d > 2$) may generate Stretch ACKs[10]. This reduces the rate at which ACKs are returned by the receiver. An implementer should only deviate from this requirement after careful consideration of the implications[11].

Reducing the number of ACKs per received data segment has a number of undesirable effects including:

(1) Increased path RTT
(2) Increased time for TCP to open the *cwnd*
(3) Increased TCP sender burst size, since *cwnd* opens in larger steps

The last effect mentioned above does not apply to Delayed ACK, which is a characteristic of it. Since it does not require any changes to the sender, Delayed ACK is a simple method to apply in the network. However, it leads to have the difficulty of gaining the *cwnd* because it requires twice as many ACKs as the method outlined in[8] to increase *cwnd* by 1.

In addition to the effects mentioned above, a TCP receiver is often unable to determine an optimum setting for a large d, since it will normally be unaware of the details of the properties of the links that form the path in the reverse direction.

3. Pulsed ACK

In this section we propose Pulsed ACK, a TCP option so that the issue mentioned above is addressed at end nodes with a simple algorithm. It reduces the number of ACK packets compared to regular TCP and enables receivers to send a *fixed* number of ACK packets per RTT, making the usage of the uplink small and constant, regardless of the throughput it gains.

After the description of redefining d, we explain the detail of our method.

3.1 Redefining d

The original version of TCP receiver sends an ACK packet to every single data packet it receives. This allows TCP connections to use unnecessary uplink bandwidth. Decreasing the frequency of sending ACK packets at receivers will be a way to address this issue.

Table 1 The Value of d used

TCP versions or options	d
Pulsed ACK	$\frac{1}{a}cwnd^{\alpha}$
Original TCP	1
Delayed ACK	2

In order to cope with this, we consider d, which is the number of data packets acknowledged by an ACK, described in Section 2. As shown in Table 1, existing TCP versions or options use fixed number for d, and this makes them use unnecessary uplink bandwidth.

d can be redefined as a function of *cwnd*. Let us redefine d as

$$d = \frac{1}{a}cwnd^{\alpha}. \tag{1}$$

Using this equation, the number of ACKs sent in a RTT is expressed as

$$\frac{cwnd}{d} = \frac{a \cdot cwnd}{cwnd^{\alpha}} = \frac{a}{cwnd^{\alpha-1}}. \tag{2}$$

While the original TCP and Delayed ACK option use fixed number for d, we state d as a function of *cwnd* for Pulsed ACK, so that it will be robust against bandwidth asymmetry. In the later sections, we describe our Pulsed ACK option considering this d.

First of all, we will determine a and α. Since the value of d depends greatly on α, we will determine it first. α is the exponent of *cwnd* in (1), (2). As you can see from (2), it determines whether the number of ACKs per RTT increases or decreases as *cwnd* gets larger.

From Table 1 and equation (1), we can see that the value of α is 0 for both the original TCP and Delayed ACK. This means if $\alpha < 0$ the number of ACKs sent will be larger than the original TCP. Recall that the purpose of our proposal, Pulsed ACK, was to *reduce* the number of ACKs. Therefore α must be greater than 0.

If $\alpha = 1$, (2) will be

$$\frac{cwnd}{d} = \frac{a}{cwnd^{\alpha-1}} = \frac{a}{cwnd^{0}} = a, \tag{3}$$

which means the receiver sends fixed number (a) of ACKs per RTT regardless of the size of *cwnd*.

If $\alpha > 1$, the number of ACKs decreases significantly as *cwnd* becomes larger; the larger the *cwnd*, the less ACKs. The problem is, that if $cwnd^{\alpha-1}$ becomes larger than a, the number

of ACKs sent in a RTT becomes less than 1, which means the sender has to wait for more than a RTT to receive one single ACK. This is too long for TCP to work properly, therefore it is inadequate to choose $\alpha > 1$.

From the inspection above, we reach $0 \leq \alpha \leq 1$. Taking into account that α is an exponent, we should determine α so that the calculation does not overload the sender or the receiver.

We described the fatality of increasing uplink bandwidth in Section 1. Since $0 \leq \alpha < 1$ will have increasing uplink bandwidth and the values other than 0 makes calculation complicated, we will choose $\alpha = 1$ so that the uplink bandwidth will be constant regardless of the usage of the downlink, along with simple calculation.

As you can see from above, a determines the usage of the uplink bandwidth. We want to make it small so that connections using our Pulsed ACK option can take advantage of bandwidth asymmetric links and make good use of them, both downlink and uplink bandwidths, without having its uplink saturated.

The typical size of an ACK packet is 40 bytes, and also some satellite links use dialup connections for ACK channels. In a connection whose RTT is 100 msec in average, 20 ACK packets (or packets having 40 bytes in size) saturate a 64 kbps bandwidth. This means if $a = 20$, a connection using Pulsed ACK can use as much downlink bandwidth as it wants even if dialup link is used as the upward channel. In order to determine a, we focused on connections using dialup links, since they are the most common examples of narrowband links that are provided as an infrastructure today. We chose an even smaller value of $a = 5$, one-fourth of 20, which means 4 Pulsed ACK connections are able to share the dialup link as upward channels. Obviously, $a = 5$ uses 16 kbps of uplink bandwidth. We can say this is small enough for any connection to make efficient use of any link. In order to compare its effect, however, we also use $a = 10$ in Chapter 4 for evaluation.

Therefore we use $\alpha = 1$, $a = 5$ for d, and (1), (2) will be

$$d = \frac{1}{a} cwnd^{\alpha} = \frac{1}{5} cwnd, \tag{4}$$

$$\frac{cwnd}{d} = \frac{a}{cwnd^{\alpha-1}} = 5. \tag{5}$$

3.2 Modifications

3.2.1 The Packet Header

We add 1 bit and 2 fields in the TCP packet header: `pulsed`, `w`, and `a`.

The `pulsed` bit is used to trigger Pulsed ACK option, working as a flag. The good thing is, Pulsed ACK works only if both the sender and the receiver react to this flag. That is, if either of the end nodes has not implemented Pulsed ACK option, it ignores this flag and works as a orginal TCP node. This fact makes it possible for Pulsed ACK option to be implemented gradually, since there will not be any errors or disturb normal data transfers by only one node using this option.

The `w` field is used by the sender to store the size of *cwnd*. This field is used only in the transfer from the sender to the receiver. How this *cwnd* is used is described in the later sections.

The `a` field is used by the receiver to store the value of a it used, to tell the sender how many ACK packets are sent in a RTT. This field is used only in the transfer from the receiver to the sender. Again, how this a is used is described later.

3.2.2 The Sender Algorithm

There are 2 modifications to the sender.

The first modification is to set the `pulsed` bit to 1 when it sends a packet, unless *cwnd* is smaller than *ssthresh*. It also stores its *cwnd* to the `w` field and sends the packet. The reason why the sender does not set the `pulsed` bit during slow start is that since slow start usually takes place either at the beginning of the connection establishment or after a severe congestion indicated by multiple packet losses, both ends must know quickly the condition of the network. Although this leads the connection to use much uplink bandwidth, we can say it does not overload the network, since it lasts only several RTTs.

We add the following codes to realize this modification.

```
if (cwnd < ssthresh) pulsed = 0
else{ pulsed = 1
   w = cwnd}
```

The second modification is to adjust the increment of *cwnd* when the sender receives the ACK packet. Compared to the number of packets sent, incoming ACK packets are much less. It is possible, just like Delayed ACK, to make no modification to this decreased ACKs and make this option simple, but this decrease is too much to ignore. If we leave this as it is, Pulsed ACK senders will suffer from severe unfairness, since the *cwnd* increases by only $5/cwnd$ per RTT, while others increase by at least 0.5 (this is the

value for Delayed ACK) per RTT. Therefore, we modify the sender so that its *cwnd* increases by 1 every RTT.

Its method is rather simple. When it receives the ACK packet, it modifies the default increment of $1/cwnd$, using current *cwnd* and the a stored in the a field of the incoming ACK packet, as

$$\frac{1}{cwnd} \cdot \frac{cwnd}{a} = \frac{1}{a}. \quad (6)$$

This modification makes the sender to increase its *cwnd* by 1 every RTT, theoretically realizing fairness between the original TCP.

The code for this modification is as follows.

```
if (pulsed == 1)
   increment = increment * cwnd / a
```

Again, this works only if the `pulsed` flag of the incoming ACK packet is set. Thus, the sender increments its *cwnd* as the original TCP during slow start or when the receiver is not Pulsed ACK-capable.

3.2.3 The Receiver Algorithm

The receiver also has 2 modifications.

The first one, which is the key modification of Pulsed ACK option, is to control the interval of sending ACKs. The receiver holds 2 variables for this algorithm: a and *receivecount*. *receivecount* is used to count the number of data packets received. This is the process.

(1) When the receiver receives a packet, it sees if the `pulsed` flag is set and calculates $cwnd/a$ using the `w` field of the received packet and a it holds.

(2) If *receivecount* is smaller than $cwnd/a$, it does not send any ACK packets to this received packet, and increments *receivecount* instead.

(3) If *receivecount* is larger than $cwnd/a$, it sends an ACK packet and resets *receivecount*.

This results in a *pulsing flow of ACKs*, sending ACKs at intervals. The name of "Pulsed ACK" derives from this characteristic.

Note that as we described in Section 3.2.2, this process at the receiver also works only if `pulsed` flag is set, and if not, it works as an original TCP receiver.

This process is described with the following codes.

```
if ( pulsed == 1 && receivecount < w
/ a ){
   delay this ACK
   receivecount = receivecount + 1
}else{
   send ACK
   receivecount = 0}
```

Recall that we chose $a = 5$ in the previous section. If the downlink is narrow enough, there will be a point where the *cwnd* becomes less than 5 (roughly 600 kbps with an average RTT of 100 msec), which means the receiver may send more ACKs than the received data packets. However, in this case, this algorithm makes receivers send an ACK every 2 packets received. Therefore this algorithm works even when the downlink bandwidth is narrow.

The second modification is simple. When the receiver generates an ACK packet, it copies the `pulsed` flag of the received packet header to the ACK packet header regardless of its value, and it also stores a to the a field.

3.3 Theoretical Performance Comparison

In this section we compare the performance of Pulsed ACK option with Delayed ACK and the regular TCP (referred to as Regular TCP), theoretically.

We assume a connection of 100 msec link, either symmetric or asymmetric in bandwidth. A data packet is 1500 bytes, and an ACK packet is 40 bytes in size.

At a symmetric link, when a Pulsed ACK connection gains as much as 100 Mbps at downlink, it uses only 0.016 Mbps of uplink bandwidth. On the other hand, the usage of Regular TCP connection is approximately 167 times as large as the usage of Pulsed ACK. A Delayed ACK connection uses less than Regular TCP, though it is still about 83 times as large as that of Pulsed ACK. Uplink usage at a symmetric link is shown in Table 2. The difference becomes much more apparent at 1 Gbps downlink usage.

At an asymmetric link, if the uplink bandwidth is 1 Mbps, Regular TCP connection saturates it when it uses 37.5 Mbps at downlink. Delayed ACK connection saturates it at the use of 75 Mbps. On the other hand, since a Pulsed ACK connection uses only 64 kbps for uplink, it can use as much bandwidth as it wants. Theoretical upper bound of the downlink usage is shown in Table 3. We can see Pulsed ACK shows great performance when the uplink is narrow.

Table 2 Theoretical Uplink Usage at a Symmetric Link

	10 Mbps link	100 Mbps link	1 Gbps link
Pulsed ACK	0.016 Mbps	0.016 Mbps	0.016 Mbps
Regular TCP	0.267 Mbps	2.67 Mbps	26.7 Mbps
Delayed ACK	0.133 Mbps	1.33 Mbps	13.3 Mbps

Table 3 Theoretical Upper Bound of Downlink Usage at an Asymmetric Link

	64 kbps uplink	1 Mbps uplink
Pulsed ACK	no limitation	no limitation
Regular TCP	2.4 Mbps	37.5 Mbps
Delayed ACK	4.8 Mbps	75 Mbps

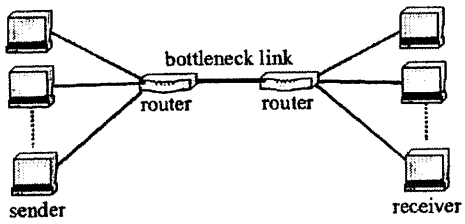

Fig. 1 Simulation topology.

Table 4 Common Parameters

Data Packet Size	1500 bytes
ACK Packet Size	40 bytes
Queuing Algorithm	DropTail

4. Evaluation

4.1 Simulation Environment

We evaluate Pulsed ACK using Network Simulator Version 2.27[12]. The protocol we use is TCP Reno. We also use Delayed ACK option, which is the only method that addresses the issue by end nodes, and, which has become the standard today.

In all simulations, we use TCP Reno with no option (Regular TCP) and TCP Reno with Delayed ACK option (Delayed ACK), TCP Reno with Pulsed ACK option (Pulsed ACK). Figure 1 shows the simulation topology. It is a single bottleneck link, and each flow has a RTT of 100 msec in average. In each simulation, we change the number of flows, bottleneck bandwidths, and send bulk data from the senders to the receivers. The parameters we used in common are shown in Table 4.

We evaluate the performance at steady state, and use the average value of 10 same simulations.

4.2 Bandwidth Usability

In this section we conduct simulations to evaluate the usage of the bandwidth.

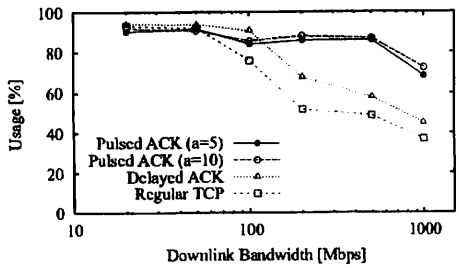

Fig. 2 Downlink usability at asymmetric links.

We change the bottleneck bandwidth and start flows at the same time. We conduct simulations at asymmetric links, at which we use a fixed value of 1 Mbps for the uplink. For the number of flows, we use 5, 10, and 50.

Figure 2 shows the usability of the downlink of 10 flows. We can say from this figure that Pulsed ACK flows have much greater robustness to asymmetric links than Regular TCP or Delayed ACK flows. While Regular TCP begins to degrade its throughput at 100 Mbps, Pulsed ACK keeps a high range of almost 90 % until 500 Mbps. Furthermore it gains 68.5 % of the 1 Gbps downlink bandwidth, which is 1.85 times as much as Regular TCP. However all three protocols do not gain much throughput at 1 Gbps. The reason for this is the underlying issue of TCP Reno that it can not use wide bandwidth efficiently because of its congestion control algorithm[13]. In fact, most of the simulations end before the aggregated throughput reaches 1000 Mbps.

Figure 3 shows the usability of Pulsed ACK flows, with the same condition of Figure 2. The filled circle shows the result of 10 flows of $a = 5$, which is the same as that shown in Figure 2. This figure shows that the greater the number of flows, obviously, the more efficient use of the bandwidth they make. Taking a look at the value of 50 flows at 1000 Mbps, they use over 82 % of the bandwidth, which demonstrates Pulsed ACK is capable of using bandwidth this wide, even when the uplink is very narrow. On the other hand, the graph of 5 flows shows they have difficulty gaining much bandwidth at wide links of over 500 Mbps, again,

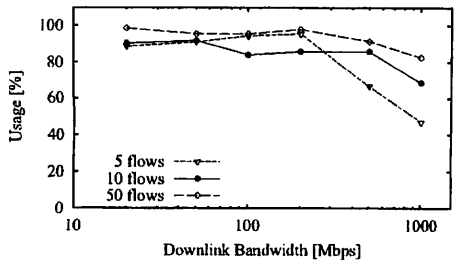

Fig. 3 Downlink usability of Pulsed ACK at asymmetric links.

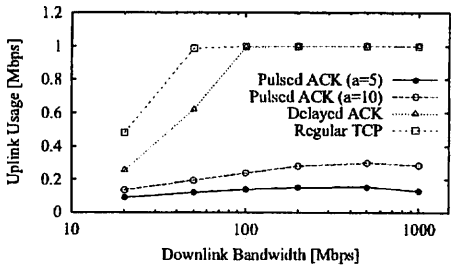

Fig. 4 Uplink usability at asymmetric links.

due to the issue of TCP Reno.

Figure 4 shows the usage of the uplink (in Mbps) of the simulation shown in Figure 2. As we described in Section 3.3, the uplink usage of Regular TCP and Delayed ACK becomes 1 Mbps at early stage of 50 or 100 Mbps, saturating the link. This leads to the degradation of the downlink throughput, as seen in Figure 2. On the other hand, Pulsed ACK stays at a range of less than 0.3 Mbps.

5. Conclusion

Researchers have pointed out that TCP has difficulty gaining throughput at links with bandwidth asymmetry, either in capacity or due to the connections in the reverse direction.

In order to address this issue, we proposed Pulsed ACK. It reduced the number of ACKs compared to Regular TCP and enabled receivers to send a fixed number of ACK packets per RTT, making the usage of the uplink small and constant, regardless of the throughput it gains. It also enabled receivers to send ACKs at intervals, to minimize the burst of the increase of *cumd*. Also, the senders adjusted the increment of *cumd* at the receipt of an ACK packet so that it increased by 1 every RTT.

We conducted computer simulations and compared the performance of Pulsed ACK with Regular TCP and Delayed ACK. Simulation results showed the significant performance of Pulsed ACK at asymmetric links: it gained 68.5 % of the 1 Gbps downlink bandwidth, which was 1.85 times as much as Regular TCP. Therefore, Pulsed ACK is an effective TCP option for bandwidth asymmetry.

Acknowledgment

This work was supported by a special grant from the 21st Century COE(Centers of Excellence)Program.

References

1) H. Balakrishnan, V. N. Padmanabhan, and R.H. Katz. The effects of asymmetry on TCP performance. In *Proceedings of the 3rd MOBICOM Conference*, pp. 77–89, Sep. 1997.

2) H. Balakrishnan, V. N. Padmanabhan, and R.H. Katz. The effects of asymmetry on TCP performance. *ACM Mobile Networks and Applications*, Vol.4, No.3, pp. 219–241, 1999.

3) N.Ghani and S.Dixit. TCP/IP enhancements for satellite networks. In *IEEE Communications Magazine*, pp. 64–72, 1999.

4) I. T. Ming-Chit, D. Jinsong, and W. Wang. Improving TCP performance over asymmetric networks. *Proceedings of ACM SIGCOMM, ACM Computer Communications Review*, Vol.30, No.3, pp. 45–54, Jul. 2000.

5) G.Fairhurst etal. Performance issues in asymmetric TCP service provision using broadband satellite. In *IEE Proceedings on Communications*, Vol. 148, No. 2, pp. 95–99, Apr. 2001.

6) H. Balakrishnan et al. TCP Performance Implications of Network Path Asymmetry. RFC 3449, IETF, Sep. 2002.

7) Y. N. Lien. Performance issues of P2P file sharing over asymmetric and wireless networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 850–855, Apr. 2005.

8) M. del Rey. Transmission Control Protocol. RFC 793, IETF, Sep. 1981.

9) R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, IETF, Oct. 1989.

10) M. Allman et al. Ongoing TCP Research Related to Satellites. RFC 2760, IETF, Feb. 2000.

11) M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, IETF, Apr. 1999.

12) The network simulator - ns-2. URL: http://www.isi.edu/nsnam/ns/.

13) S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649, IETF, Dec. 2003.