Chapter 5
Performance Enhancing Proxy (PEP)
for improvement of TCP Throughput

5.1 Introduction

Today's networks are large and complex. The networks with high variation in bandwidth and large delay are called as TCP unfriendly networks. TCP unfriendly networks cannot be easily incorporated into the present TCP dominated infrastructure transparently. Reasons being they possess new link technologies that have characteristics which can seriously degrade the performance of TCP flows. TCP performs very well in networks with fixed bandwidth and small delays. However the bandwidth available to TCP flow in today’s internet is often variable.

In order to mitigate these problems without modification of TCP specification, deploying a Performance Enhancing Proxy (PEP) has been proposed [94]. It is a known fact that Performance Enhancing Proxy (PEP) are being used in practice to improve the degraded TCP performance caused by characteristics of specific link environments e.g. satellite wireless WAN and wireless LAN environment.

Here we digress and try to extend the concept of PEP in wired network. PEP is implemented by extending ns2 simulator in wired network to see its effect in improving the throughput by varying parameters like buffer size, delay, link capacity, watermark index, PEP node location. Result shows that PEP also holds promise in wired network.

5.2 PEP function

The PEP operates on a router along a TCP connection. It monitors all the packets of a TCP connection through it. When it receives data packet from the source host, it transmits the packet to the destination host, copies the packet into its own buffer (PEP buffer) in case of the retransmission of the packet, and sends an ACK (premature ACK) acknowledging the packet to the source host instead of the destination host. It removes the packets from the PEP buffer which the real ACK acknowledges, and drops the real ACK. When a real ACK for a packet for which a premature ACK has
been returned does not arrive at the PEP from the destination host during a time out value, the PEP retransmits the packet's copy to the destination host.

Performance Enhancement Proxies (PEP) [95] are deployed at the edges of TCP-unfriendly networks shown in figure 5.1 controls the TCP flows passing through them. The PEP element at an edge router monitors the available bandwidth of the network and manipulates the ACK packets of a passing-thru TCP flow. By deliberately speeding up or slowing down TCP ACKs, the PEP element rewrite the behavior of a TCP flow and help achieve better congestion control and improved throughput.

![PEP Node](image)

*Fig 5.1 A Network with PEP Node for Improving TCP Performance*

The job of PEP is to manipulate early acknowledgement based on the current congestion condition so that the TCP sender at the endpoint can respond more effectively. By deploying PEP at the edge router of TCP unfriendly network, acknowledgement manipulation becomes effective because sender to PEP – to sender distance is shortened before it is passed through TCP unfriendly network.

### 5.3 Proposed Implementation of PEP in NS2

#### 5.3.1 Introduction

PEP is implemented using NS2 simulator. Tcl scripts are used for simulation purpose. NS2 has TCP and allows creating new agent. New agent PEP is created which has all functionalities of PEP as well as that of TCP [73-81].

#### 5.3.2 Requirements for PEP implementation

As per the function of PEP the PEP agent needs to work to enhance the performance over TCP. The agent TCP sends the TCP packets to the final destination. The final destination is called as a Sink node. The TCP sends the packets and after receiving the
ack from sink node it accordingly increases its congestion window (cwnd). The sink node accepts the packets received from TCP sender and sends the ack to the TCP sender. The TCP agent handles the packet loss, retransmission of lost packets etc. Sink node handles out of order packets, packet loss etc. For implementation of PEP we need to work PEP as a sender and a receiver too.

The PEP needs to handle the packets which are received from actual TCP sender. The PEP stores the TCP packets received from TCP sender depending upon the free buffer size. Then it sends the ack to the actual TCP sender on behalf of the actual receiver depending upon the watermark index. Now this PEP node forwards the packet to the actual receiver till it gets the ack from the actual receiver. If the packet is lost before reaching the actual receiver then PEP node retransmits the packet. The PEP node stores the packet till it is acknowledged by the actual receiver. When PEP node receives the ack from the actual receiver, the PEP node frees the buffer. If the PEP node doesn’t have free buffer then it just forwards the packet to the actual receiver as if there is no PEP present in between. Hence PEP node works as sender and receiver too.

For the implementation of PEP in ns2 we need to combine the working of the TCP Agent as well as Sink Agent. PEP node sends the packets as the TCP agent transmits. It acknowledges the packets as the TcpSink agent. The cwnd of PEP varies depending upon the acknowledgments it receives from actual receiver. As PEP receives the acknowledgments, the value of free buffer increases and it sends the acknowledgment of unacknowledged packets to the actual sender.

5.3.3 NS2 changes

In NS2 the PEP agent is inherited from TCP agent class. The TCP agent class is inherited from class Agent. Fig 5.2 shows the partial class hierarchy of NS2 with new added agent PEP. Since PEP is inherited from class TCP, it inherits all the properties and functions of class TCP agent. Class TcpAgent does not maintain a buffer in TCP to store the packets. But the PEP element maintains a buffer to store the TCP packets received from the actual TCP sender till it gets the acknowledgment from the actual receiver. To maintain the buffer, in the inherited class TcpPepAgent, the required
parameters pepqueue, congqueue, free_buffer, max_buffer, watermark_index, PEPdelay are added.

- The max_buffer stores the value of the maximum buffer.
- The free_buffer stores the value of the free buffer.
- The watermark_index is used to control the flow of the sender to avoid congestion.
- The PEPdelay is the assumed processing time for the arrived TCP packet from the sender, at the PEP node before it can be retransmitted to the actual receiver.
- The queue pepqueue is used for the storage of packets which are received from the actual sender.
- The queue congqueue is maintained to store the unacknowledged packets.

These four variables and two queues control the flow from TCP sender to the PEP node and from PEP node to the actual receiver.

![Diagram of NS2 Class Hierarchy (Partial) with new agent PEP](image)

Fig 5.2 NS2 Class Hierarchy (Partial) with new agent PEP

We need to work PEP as both sender and receiver. To work it as a sender we need all properties of TCP. We have overridden some of the functions of TCP so as to work
PEP as per our requirement. The functions which are overridden are recv(), output(), send_much().

To work PEP as a receiver we need it to inherit properties of class TCP Sink. Hence we have introduced the class PepAcker which takes care of acks to be sent to the original sender by the PEP node. In class PEP agent we have added all Acker functions and an ack function. We have overridden the functions as ack(), recv() so as to work as a receiver for the packets which are sent by original TCP sender and ack those packets. This PEP agent also takes care of acks which are received from final destination node.

To simulate PEP following changes are made to the ns2 source code. Two new files are added for the TcpPepAgent class.

1. ptcp.h – This file declares a class “TcpPepAgent” for creating new agent (Pep Agent).
2. ptcp.cc – This file contains the definition of the “TcpPepAgent” class. It defines actual send and receive functions of pep agent.

5.3.4 Flow chart

The function recv(), which receives the data from system is the most important function in TcpPepAgent which actually plays the important role in implementation of PEP. This function is modified to receive data from actual sender as a TCP packet and as an ACK packet from actual receiver. Fig 5.3 gives the flow chart of function recv().

The flow of the function recv() is:

- If the received packet type is TCP then check for buffer space. If free buffer is available then check if free buffer value is greater then watermark index. If so then send ack to the sender. If not then store the packet in congestion queue i.e. congqueue. Decrement the free buffer value. Store the packet in pepqueue and schedule it with delay equal to PEPdelay to send to the actual receiver. If
the buffer space is not available then do nothing and just forward the packet as if there is no PEP present in between.

- If the received packet type is ACK then check if the packet is not from previous incarnation. If its sequence number is greater than last acknowledgment then increment the buffer size equal to the number of acknowledgments received from the actual receiver. Also send those many acknowledgments of the unacknowledged packets in congestion to the actual TCP sender. If the received packet sequence number is not greater than last acknowledgment then check for duplicate packet and take duplicate packet action accordingly.
Start

PKT Type is TCP

Y

Free buffer > 0

Y

Free Buffer > watermark Index

Y
Send Ack

Decrement Free buffer

Store Packet in pepqueue

Schedule packet to send to receiver

N
Store packet in congestion

A

Return

Continue...
Fig 5.3 Flow chart of Function recv()
5.4 Examining the throughput by Simulation of TCP and PEP

Simulation is carried out to compare throughput of standard TCP and PEP. By doing we are trying to find the optimum value of buffer space and watermark index for which we get the better throughput. The behavior of PEP is also analyzed when there are more than one PEP element is present between the sender and receiver. Different simulation cases have been tried out.

5.4.1 Simulation setup

A linear topology is selected for simulation. Simulation is carried out in ns2 [73-81]. Topology considered for simulation includes two routers between TCP sender and receiver as shown in Fig 5.4 PEP element is attached to router1

![Topology Diagram]

Fig. 5.4 Topology of Simulation Network

5.4.2 Simulation cases

Case I: Comparison of TCP and PEP without congestion

Congestion window (cwnd) and transmitted sequence number (seqno) are the two parameters of the sender side TCP stack which give a good idea of the transmission characteristics of the TCP session. We have cwnd and seqno to compare standard TCP connection with the two separate TCP connections established using PEP. Both cwnd and seqno are measured in terms of packets. In all the figures TCP refers to standard TCP connections without PEP element. When PEP is used, there will be two separate TCP connections. The TCP connection between the sender and the PEP node will be referred to as PEP conn-1 and the TCP connection between the PEP node and the receiver will be referred to as PEP conn-2. Later on we have used TCP throughput (number of segments received from the receiving host per unit time) to measure performance.

The ftp application is attached to the sender. Continuous packets are sent to the receiver. Fig 5.5 shows the graph of congestion window v/s time for standard TCP
connection, PEP conn1 (connection between sender and PEP node) and PEP conn2
(connection between PEP node and the receiver), whereas Fig 5.7 shows the plot of
throughput at receiver v/s time

The Fig 5.5 shows the comparison of TCP and PEP in terms of congestion window
(cwnd). Fig 5.6 is a plot of sequence no against time. This graph shows when each
packet was sent

![Graph showing congestion window comparison of TCP and PEP](image)

**Fig 5.5 Congestion Window Comparison of TCP and PEP**
Fig 5.6 Transmitted Packets Comparison of TCP and PEP

Fig 5.7 Throughput Comparison of TCP and PEP

Remarks: PEP gives about 56% increased throughput
Since the standard throughput remains same, all further throughputs are plotted for variations in PEP parameter.

**Case II: Performance of PEP in presence of delay (congestion)**

Now we introduce the congestion between Router 1 and Router 2. The congestion is in terms of large delays. Fig 5.8 shows the sequence no comparison of PEP with delays 10ms, 100ms, 200ms, 300ms and 400ms. Fig 5.9 shows the throughput comparison in presence of delay.

Fig 5.8 Transmitted Packet Comparison of TCP and PEP in presence of delay
Fig 5.9 Throughput Comparison of TCP and PEP in presence of delay

Remarks: PEP gives 56% increased throughput with low delay (delay = 10 ms). For large delays percentage increase in throughput with PEP is: for 300 ms – 13%, 400 ms – 12%.

Case III: Throughput of PEP with different buffer size

For different buffer size viz. 10, 20, 30, 40, 50, 70, 85, 100, 200, 300 packets throughput at the receiver is measured. Fig 5.10 show the throughput v/s time for these buffer sizes.
Fig 5.10 Transmitted Packets Comparison of PEP with different buffer size

Fig 5.11 Throughput Comparison of PEP with different buffer size

Remarks: For moderate buffer size throughput increases. For buffer size increasing from 20 packets to 100 packets; the percentage increasing in throughput ranges from 13 to 60. For very small buffer size (10 packets) and large buffer size (200 and 300 packets) throughput decreases.
Case IV: Throughput of PEP with variation of watermark index.

The throughput for the watermark index 5, 10, 5, 20, 25, 30, 50, 70, 90 is computed and plotted. The Fig 5.12 shows throughput of PEP for these watermark index.

![Graph showing throughput comparison of PEP for different watermark index]

Fig 5.12 Throughput Comparison of PEP for different watermark index

**Remarks:** It is observed that for watermark index value 10, increased throughput is obtained.

Case V: Throughput of PEP with different bandwidth.

The bandwidth of the link between router 1 and 2 is varied for PEP throughput. Measurement is done for the link capacities 1, 5, 40, 50, 100, 500, 1000 mbps. Fig 5.13 shows the graph of throughput at the receiver v/s time for above link capacities.
Fig 5.13 Throughput comparison of PEP with different bandwidth

Remarks: Observation shows that with the increase of media capacity, throughput increases for higher bandwidth. For the link capacities from 10 to 1000 mbps the increase in throughput ranges from 50 to 64.

Case VI: Effect of multiple nodes on throughput

Performance is analyzed with more than one PEP node present between the sender and receiver. For this analysis, the linear topology with 15 nodes (Fig. 5.14) is used.

Fig 5.14 Simulation Topology for Multiple PEP Nodes Comparison

Experiments were carried out by attaching PEP node at various positions. One, two, three and four PEP nodes are attached at a time. Throughput at the receiver is measured for all possible combinations. Table 5.2 shows summary of the results.
**Single PEP node:** In between sender and receiver one of the nodes is made to function as PEP node. The PEP agent is attached to nodes 2, 4, 7, 10 and 11. Throughput is calculated at the receiver and the results are tabulated in Table 5.1.

**Table 5.1:** Throughput for single PEP Node

<table>
<thead>
<tr>
<th>Nodes to which PEP is attached</th>
<th>Throughput (Mb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.7</td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
</tr>
<tr>
<td>7</td>
<td>4.4</td>
</tr>
<tr>
<td>10</td>
<td>3.0</td>
</tr>
<tr>
<td>11</td>
<td>2.65</td>
</tr>
</tbody>
</table>

**Fig 5.15** Throughput Comparison of PEP when no of PEP = 1

For two, three and four PEP nodes attached at time and throughput is measured at the receiver for all possible combinations. The results are tabulated in appendix B
Fig 5.16 Effect of multiple PEP nodes on TCP throughput

Table 5.2: Observations of effect of multiple PEP nodes for maximum throughput.

<table>
<thead>
<tr>
<th>No. of PEP nodes</th>
<th>Combinations of nodes to which PEP is attached for maximum Throughput</th>
<th>Throughput in Mb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>4.4</td>
</tr>
<tr>
<td>2</td>
<td>(2,8), (3,9), (4,10), (5,10), (5,11), (6,12)</td>
<td>5.0</td>
</tr>
<tr>
<td>3</td>
<td>(5,8,9), (5,11,13), (5,9,11), (5,9,12), (6,10,11), (6,10,13), (6,8,12), (6,8,13), (6,9,11), (6,9,12), (6,9,13), (3,7,11), (3,8,10)</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>(5,7,10,13), (6,8,11,12), (6,11,12,13)</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Remarks: For a chosen topology, it is observed that with 1, 2 or 3 nodes with PEP attached gives better throughput. Although for 3 and 4 nodes with PEP gives maximum throughput, the overheads are involved in setting up multiple PEP node session. With one PEP node maximum 66% increase in throughput is obtained.
5.5 Conclusion

PEP is being used in practice, particularly used in wireless environment. In this chapter we have implemented PEP by extending ns2 simulator in wired environment. Influence of parameters like buffer size, watermark index is examined and results indicate that there is an improvement in throughput.

The increase in throughput by PEP over TCP for various optimized parameters is as under

I. without congestion: 56.6%
II. In presence of higher delay: 13%
III. With buffer size of 100 packets: 60%
IV. For bandwidth of 1000mbps: 64%
V. with one PEP node at center location: 66%

For the considered network more number of PEP nodes, except one or two does not influence the performance, on the contrary it degrades because of overhead.