CHAPTER 1

Introduction to Computer Network
Introduction

Computers and their applications are now an integral part of daily life. Their widespread usage ranges from entertainment, to processing and storage of data, to control of critical infrastructures such as power grids and industrial plants. In many cases, computer applications need to communicate and share data with each other to realize their objectives. Examples of such applications are web browsing, electronic mail, text messaging, file sharing, audio streaming, video streaming, and electronic commerce.

One of the key enabling technologies for today's computer technology is the Internet that is used by millions of computers to communicate and share data with each other. At a high level, the Internet is composed of many specialized computers called routers that forward data between end systems. The routers are connected to each other by various communication links. End systems are typically the ultimate sources and sinks of data, i.e. end system typically do not forward data that they do not transmit or receive.

Since a computer network is a shared infrastructure, several end systems can be using the network at the same time. Overload or network congestion occurs when end systems simultaneously transmit more data than routers can forward. End systems can ameliorate network congestion by employing congestion control algorithms that detect and react to infrastructure and enjoy good performance. It significantly contributed to the growth and success of the Internet in the nineteen eighties and nineties. However, as the Internet has grown and the number of end systems has increased, concerns have been raised about the scalability and effectiveness of the pure congestion control implemented solely on end systems.
Active queue management (AQM) in routers has been proposed as a measure to preserve and improve Internet performance. AQM algorithms detect incipient congestion by typically monitoring a router's queue size. In its simplest form, when impending congestion is detected, AQM algorithms notify the end systems to reduce their transmission rates by proactively dropping packets before congestion actually occurs. In this manner, AQM algorithms can help end systems' congestion control mechanisms detect and react to congestion faster and more effectively. A large number of AQM algorithms have been proposed in research literature. However, many of them have been evaluated only by simulations under rather simple and unrealistic traffic conditions. Moreover, evaluations are often incomparable as they use different traffic models, network topologies, etc. Thus, there is currently a lack of understanding of the performance of AQM algorithms in complex and realistic network environments. This dissertation is concerned with evaluation of AQM algorithms in a complex network. In the rest of this Chapter, we will give a short introduction to computer networks, congested network, end systems' loss recovery and congestion control, and AQM algorithms. The loss recovery and congestion control algorithms described in this Chapter are used by the Transmission Control Protocol (TCP), a protocol that is currently implemented and used widely by the end systems in the Internet. Finally, we will conclude this Chapter with motivation for my dissertation, my thesis statement, and a summary of major results and contributions.

1.1 Introduction to Computer Networks

When an end system has data to send, it fragments data into units called packets. The end system also packages some control information into a packet header and places it at the beginning of each packet. In an IP network, the packet header contains the
source IP address to specify the sender of the packet. The packet header also contains the destination IP address that tells whom the packet is destined for. The destination address is used by the routers to locate the receiver of the packet and to subsequently forward the packet toward the receiver. In many cases, the sender and the receiver are not directly connected. In these cases, the sender sends the packet to a router that it is directly connected to. When the packet arrives at a router, the router uses the packet’s destination address as a search key and searches its routing information database for a forwarding decision. If the router and the receiver are directly connected, the router forwards the packet to the receiver. If not directly connected, the router forwards the packet to another router that is closer to the receiver. This forwarding process is repeated until the packet finally reaches a router that is directly connected to the receiver. That router can then send the packet directly to the receiver. Figure 1.1 shows the basic architecture of a router. A router usually has multiple network interfaces attached to different links that connect the router to other routers or sub networks. A sub network is usually composed of one or multiple end systems. When a packet arrives at a router, the router uses the destination address and its local routing information database to determine the output link for that packet. In most cases, the output link is not the input link (the link on which the packet arrives at the router) and packet has to be transferred from the input link to the output link via an internal switch. If multiple packets that are destined for the same output link arrive at the router via different input links simultaneously, only one packet can be forwarded onto the link and other packets are stored temporarily in a queue at the output link.
When packets arrive at a router for a given destination link at a rate faster than the link's transmission rate, the packets are buffered in a queue at the destination link. This queue is usually maintained in a "first-in, first-out" (FIFO) manner. With FIFO queue, arriving packets are enqueued at the end of the queue. When the router is done with the transmission of a packet, it dequeues the packet at the head of the queue and transmits that packet onto the link. The transmission time for a packet on a link is equal to the product of the packet size and the link's transmission rate. If the router's forwarding rate is greater than the packets' arrival rate (the average transmission time is smaller than the inter arrival time of packets), the queue will shrink over a time interval until it becomes empty. On the other hand, if the router's forwarding rate is less than the packets' arrival rate (the average transmission time is greater than the inter arrival time of packets), the queue will grow over a time interval until the queue becomes full. When a packet arrives at a full queue, i.e., the router has exhausted its storage resources and cannot enqueue another packet, that packet is dropped. When this occurs, we say that the router's queue overflows. Further, the default behavior of routers that only drop arriving packets when the queue is full is called drop-tail queuing. Drop-tail queuing is also called drop-tail FIFO because packets are forwarded in a FIFO (first-in first-out) manner.
1.2 Congestion Control

The area of Internet congestion control was baptised in 1986-1987 when the then ARPANET suffered 'congestion collapse'. Congestion collapse had been predicted by Nagel. Congestion collapse occurs when mounting levels of traffic result in high packet loss inside the network, such that few or no packets are actually delivered to their destination, yet each link is highly loaded.

The initial response to ARPANET’s congestion collapse problem was to increase the capacity of the network. This helped temporarily, but the ARAPNET continued to suffer congestion collapses until a strategy to control the load of packets entering the network was developed. In 1988 Van Jackson enhanced the famous Transport control protocol (TCP) so that the transmission rate was responsive to the level of network congestion. TCP was made to reduce the rate of transmission of hosts when it sensed the network load was nearing congestion collapse. Since the introduction of this enhanced TCP, congestion collapse did not reoccur.

This history of the Internet reflects the two fundamental approaches to the problem of controlling congestion in networks 1) Capacity Provisioning and 2) Load control. Since Congestion collapse occurs when the load of packets placed onto the network exceeds the network's capacity to carry the packets, the capacity provisioning approach is to ensure that there is enough capacity to meet the load. The load control approach is to ensure that the load of packets placed onto the network is within the capacity of the network. Capacity provisioning is achieved either by accurate performance analysis and traffic modeling, or the brute force approach of over provisioning. There is a range of load control strategies for networks, from connection admission control schemes through to best-effort flow control as on the Internet.
As is evident from the ARPANET experience, economic and technical reasons limit the capacity provisioning approach as a sufficient solution to the congestion control problem. Provisioning for peak loads is an expensive proposition, and in any case, it may not always be possible to anticipate what the peak load will be. Therefore, load control has a permanent role in the congestion control in networks. In this thesis, we will develop the areas of Internet congestion control by developing load control as well as provisioning techniques.

1.3 Load Control Mechanisms

When the capacity available is less than the demand for capacity, load control is the critical element which determines how many packets are allowed onto each link of the network, who gets to send them and when. This controls the quality of service (QoS) metrics such as bandwidth, latency and jitter experienced by users. There are a number of load control mechanisms, and these can be classified by the type of QoS guarantees they can deliver.

At one end of the spectrum are the connection admission control (CAC) schemes, such as the Resource Reservation Protocol (RSVP). Such schemes require the network to maintain information about each connection and arbitrate whether connections are admitted or rejected so that the connections that are admitted can be absolutely guaranteed their required bandwidth for the duration of the connection. When the load of requested connections is increased beyond the capacity of the network then some new users will be rejected in order to maintain the bandwidth guarantees made to already admitted users. CAC is good for honoring bandwidth supply contracts that specify minimum rates. The Internet Service proposal uses RSVP signaling protocol to extend Internet functionality to include CAC. However,
CAC schemes on the Internet are not widely deployed and it is believed they cannot scale to widespread use due to the amount of per-connection information required inside Internet routers and switches.

At the opposite end of the load control spectrum is the best-effort network. Unlike a CAC based network, the best-effort network does not need information about each connection to be stored in the routers and switches on the data path, because there is no resource reservation for a connection. The best-effort network sees each packet transmitted as independent of any other packet. The sources themselves decide how much they should send and on the Internet the sources are predominantly TCP/IP. All requested connections are admitted, and the available capacity is shared between the connections. As a result, no explicit guarantees can be given about the bandwidth available to each connection. However, the simplicity of the network infrastructure is compelling, since there is no concept of connection within the network, only the ability to forward packets is required of the network. This simplicity is a key reason for the success of IP networks.

Somewhere between CAC and best-effort networks, are flow aggregate schemes such as DiffServ. With DiffServ, although individual connections are given no guarantees of minimum bandwidth, classes of connections are given minimum bandwidth guarantees. The bandwidth allocated to each connection within a class is essentially made by a best-effort mechanism, and each connection with a class competes for bandwidth. However, the aggregate of connections within the class is guaranteed a minimum bandwidth in the presence of other classes in the network. Such aggregate schemes require only a small amount of state information in the network, such as the relative bandwidth assignment between the classes, and this state information is proportional to the number of classes. Although not offering the per-
connection guarantees of CAC systems, they allow the network operator to give important applications some protection from less important flows.

1.4 TCP/IP

By far, the most dominant load control paradigm on the Internet is source flow control. The dominant source flow control protocol on today’s Internet is TCP. In measurements of traffic at core routers indicate that about 95% percent of traffic volume on the Internet is generated by the Transport Control Protocol TCP source algorithm. The remaining traffic is UDP (about 4%) and ICMP. There are various versions of TCP, including Reno, SACK and Vegas. The most widely deployed version is TCP Reno. This modern version of TCP stems from the first congestion controlled TCP, TCP-Tahoe, as described by Jackobson and Karels.

The congestion control behavior of TCP protocol is important to the techniques developed in this thesis and we will detail the congestion control mechanism of TCP in this section. The key function of TCP is to provide a reliable connection for the application layer across an unreliable best-effort network. TCP provides a number of guarantees to the application layer: (1) delivery of data (2) in order delivery of data and (3) error free delivery of data. These guarantees are achieved by error detection, buffering and retransmission. However, in the case of congestion collapse, none of these guarantees can be met. The congestion control mechanism in TCP is in place to protect the integrity of the network and prevents congestion collapse.

To ensure the integrity of the network, TCP is designed with the “conservation of packet principle”. This principle demands that, in equilibrium, a packet must be removed from the network, for a new packet to be placed onto the network. The intuitive argument to explain this principle is that a successful delivery of a packet...
frees the network resources for the delivery of another packet. To support this conservation principle, TCP has a window based flow control.

Window based flow control limits how many packets TCP is able to transmit onto the network which have not been acknowledged as having been received. As shown in Fig. 1.2, only the amount of packets which fit into the window are allowed onto the network, and packets generated by the application which do not fit, wait at the source. TCP sends acknowledgement packets from the destination for packets received successfully. Once the acknowledgement for a packet is received at the source, the acknowledged packet and the acknowledgement packet are removed from the transmission window, leaving room for the transmission of the new packets awaiting transmission.

![Figure 1.2 Window Based Flow Control](image)

The window size $w$ limits how many packets the TCP connection can have in flight in the network; including acknowledgement and data packets. This effectively controls the TCP throughput. The packets, by convention known as segments of data, have a maximum segment size $MSS$. If we consider that there are $w$ packets in transit at time $t$, then after one round-trip-time (RTT) at time $(t + RTT)$, these packets would be delivered to the destination and acknowledged at the source and $w$ new packets would be in the network. Therefore the transmission rate of the TCP source, in bits per second, can be described by:
The window size is controlled by TCP so that the transmission rate is matched to the most severe bottleneck, either the network or the receiving host's ability to consume the information. The actual window size is the minimum of the network's capacity to transmit, $cwnd$, and the receiver's capacity to receive information, $awnd$:

$$w = \min (cwnd, awnd)$$

In practice, $awnd$ is seldom the limiting factor, and the network is typically the bottleneck. Therefore we can assume $w = cwnd$. However, the question remains how TCP knows what the network capacity is. For a source to fully utilise the available capacity, $w$ should be set to the bandwidth delay product of the bottleneck capacity of the end-to-end connection. In this way, TCP fills the 'pipe' between the source and destination with packets. However, the best-effort Internet does not provide any explicit information about the available capacity between hosts.

TCP is able to gauge the available capacity by loading the network with traffic. TCP increases its transmission rate above the capacity, which causes backlog and packet loss to occur. The TCP source detects loss by sensing that the lost packets have not been acknowledged. When packet loss is detected, TCP knows it has exceeded the capacity, and reduces its rate. This increase/decrease cycle continues throughout the whole connection time to ensure TCP is fully utilizing available capacity, and not exceeding network capacity.

TCP has two key modes of operation with different mechanisms for increasing and decreasing the window; 1) slow-start and 2) congestion avoidance. TCP begins the connection with window size of 1, and uses the slow start algorithm to increase the windows size until the first packet loss is detected. After the first loss, TCP uses...
the congestion avoidance mode to increase and decrease the window size about the operating point.

![TCP Congestion Control By Control of Window](image)

**Figure 1.3: TCP Congestion Control By Control of Window**

In the counter-intuitively named slow-start mode, TCP increases the window size by 1 for each acknowledged received, which results in a very fast exponential growth of the window size. The slow-start mode is intended to bring the window size, and therefore the transmission rate of TCP, quickly to roughly the correct value. Once the window size grows over the bandwidth delay product of the transmission path, packet loss will inevitably occur in the network, and then TCP switches to the congestion avoidance mode when the first loss is detected.

In congestion avoidance, TCP increases the window size by a factor of $1/cwnd$ for every packet acknowledged. However, for every packet loss, the window is decreased by a half. This additive-increase/multiplicative-decrease (AIMD) algorithm was first introduced and added to TCP by Van Jacobson. An important property of the AIMD mechanism is that it results in equal capacity sharing at links between TCP sessions which traverse the same end-to-end path.
A further mechanism which occurs when there is catastrophic transmission failure is called Retransmission Timeout (RTO). Whenever TCP sends out a packet, it starts a timer. If the acknowledgement of the packet sent is not received with a certain time, a timeout occurs. TCP adjusts the time allowed for an acknowledgement in view of the RTT of the network. When a RTO occurs, it is a sign of very heavy congestion or link failure. RTO timeout should be rare, and when it happens, TCP reduces the congestion window to 1 and restarts in slow start

1.4.1 TCP Slow Start

The TCP slow start algorithm is used by an end system to seek and enter an equilibrium state of the network. TCP slow start allows a sender to increase its transmission rate rather fast by growing its cwnd exponentially as follows. The cwnd is initialized to a small number of packets (typically one or two) by the sender after a connection has been established. TCP also reinitializes cwnd to this value and reenters TCP slow start after a long idle period between the sender and the receiver or after the sender has experienced severe losses such that it has to seek for a new equilibrium state of the network. While in slow start phase, the sender increases its congestion window by one packet for each ACK packet that it receives. If the receiver acknowledges each data packet with an ACK packet, the sender can double its congestion window within a round-trip time (RTT). Thus, when there is no packet loss, the sender can grow its congestion window at an exponential rate. Obviously, slow start is not slow at all despite its name.

Since the sequence number in ACK packets are cumulative, a receiver can increase efficiency in network usage and reduce the number of ACK packets by acknowledging every other data packet that it receives with an ACK packet. In this
Introduction

case, the sender’s congestion window grows more slowly than in the previous case but the growth rate is still exponential.

Slow start allows a TCP sender to probe for available bandwidth very fast because the sender can grow its congestion window exponentially in this phase. As the sender approaches the limit of capacity in the network, it exits slow start and enters congestion avoidance. The TCP sender infers that it has reached the limit of the network’s capacity after it experiences the first packet loss. When this happens, TCP exits its slow start phase and enters the congestion avoidance phase. Alternatively, a TCP sender also exits the slow start phase and enters the congestion avoidance phase after its congestion window becomes larger than a variable called slow start threshold or ssthresh. This variable is a conservative estimate of available bandwidth in the network and is dynamically updated as explain in section 1.4.2.

1.4.2 TCP Congestion Avoidance

Unlike in the slow start phase, a TCP sender increases its cwnd more moderately in the congestion avoidance phase. The rationale for this is that the TCP sender has already reached the equilibrium state of the network and should avoid any drastic change in its transmission rate that may destroy the equilibrium state. In the congestion avoidance phase, the TCP sender increases its cwnd by 1/cwnd for each ACK packet that it receives. This means that the TCP sender can increase its cwnd by 1 packet after successfully sending cwnd packets. Since the TCP sender can have at most cwnd outstanding packets, i.e., at most cwnd packets that have not been acknowledged by the receiver, and it takes a roundtrip time for an ACK of a data packet to arrive at the sender, the sender can increase its cwnd by 1 packet per roundtrip time. This gives the TCP sender an additive increase of its cwnd in the congestion
Introduction

avoidance phase because cwnd grows linearly as a function of time (as opposed to cwnd growing exponentially in the slow start phase).

When a TCP sender detects loss of packets, it infers that packets are lost due to congestion in the network. More specifically, a TCP sender assumes that loss is an indication that the current transmission rates of the end systems have exceeded the available capacity of the link. In this case, the TCP sender, together with other TCP senders sharing a bottleneck link, is sending packets at a faster rate than this link can forward the packets. Depending on how packet loss is detected, the TCP sender can take one of the two following alternate approaches.

If packet loss is detected via triple duplicate ACKs, the sender knows that at least some of the packets arrived at the receiver. In this case, the sender remains in the congestion avoidance phase but reduces its cwnd by half. The rationale for this is that the TCP sender was previously able to transmit packets without loss at that transmission rate and hence it is safe to start probing for available bandwidth again from that transmission rate. This gives the TCP sender a multiplicative decrease of its cwnd.

If packet loss is detected via a timer's expiration, the sender infers that a large fraction of its packets were lost (because the sender has not received any ACKs) and that there is severe congestion in the network. In this case, the TCP sender reduces its cwnd to one packet and switches to the slow start phase to rediscover the equilibrium state of the network.

After a loss event, a TCP sender infers that the available share of bandwidth is less than its current cwnd. Hence, the TCP sender sets its thresh to half of its current cwnd to prevent cwnd from overshooting the available bandwidth when it enters the slow start phase next time.
1.4.3 Round-trip Time Estimation

Estimation of round-trip time (RTT) plays an important role in the TCP recovery algorithm. Round-trip times experienced by a connection usually consist of a propagation delay and queuing delay. While propagation delays usually are constant, queuing delays can vary because queues at routers along the path of a connection can grow or shrink over time.

Due to the variable queuing delays, it is not easy for the end systems to estimate the round-trip time accurately. However, an accurate estimation of round-trip times allows TCP to set its timers efficiently. An underestimation of round-trip times can lead to timer intervals that are too short. Since timers are used to detect loss of data packets at the sender upon their expiration, short timers will cause unnecessary retransmissions of data packets and waste of bandwidth. On the other hand, an overestimation of round-trip times can cause the sender to wait too long to detect a packet loss and increase experienced delay for applications running on top of TCP.

Since the RTT of a connection can change over time, TCP needs to dynamically adjust its RTT estimate. TCP adjusts its RTT estimate to the trend of RTT measurements but uses a low pass filter to smooth out fluctuations in RTT measurements. For a RTT measurement $m_i$ at time $i$, the RTT estimate $a_i$ at time $I$ is updated as follows:

$$a_i = (1 - g)a_{i-1} + gm_i$$

where $g$ is a constant ($0 < g < 1$). TCP also estimates and updates the variation $v_i$ of round-trip times for measurement at time $i$ as follows

$$v_i = v_{i-1} + g(|m_i - a_i|)$$
The retransmission timeout (RTO) has to be on the order of the round-trip time of a connection to allow for data packets to arrive at the receiver and for their corresponding ACK packets to come back to the sender. Van Jacobson proposed that the retransmission timeout has to be computed from both the mean and variance of RTT measurements to reflect wide fluctuations in RTTs. The retransmission timeout RTO\(i\) at time \(i\) is computed as follows:

\[
RTO_i = a_i + 4v_i
\]

1.5 Congested Network

The most common type of computer network is a packet-switched network, where nodes send data in the form of packets to each other. The most common strategy used to transfer data is store-and-forward. Each node waits till it has received a whole packet before forwarding it at a later time to the appropriate output link. The Internet is an example of a network that is prominently packet-switched. The data route from a source to destination is computed by different methods by routers. When we talk about congestion control we essentially talk about control of data packets in these routers. Congestion occurs in a router when the aggregate bandwidth of incoming packets, destined for a particular output link, exceeds the link's bandwidth. An example of congestion occurring at a router is given in Fig 1.4.

![Small congested network](image-url)

**Figure 1.4: Small congested network**
There are two types of congestion: transient and persistent. Transient congestion can be managed with a queue of buffer at the router. During to the congestion period the queue will grow and contain the excess packets. When the congestion period ends, the buffered data is forwarded to the appropriate output link. On the other hand, persistent congestion is said to occur when the data overflows the buffer. While transient congestion only introduces a delay in data transmission, persistent congestion results in data loss. These problems are tackled in two ways. Either the router detects the queue build up and informs the sources to decrease their transmission rate. Such a strategy is called Congestion avoidance. The other method is to use end-to-end strategies where the routers do not get directly involved but the hosts use indirect methods to detect congestion. Such a mechanism is known as Congestion Control. In TCP/IP, the most commonly used protocol in the Internet, both methods are used to tackle problem related to congestion.

A Control Theoretic approach of flow control by Shrinivasen Keshav[24] is pointed out that the very detection of occurrence of congestion in a network may not be very easy. The immediate side effects of congestion are data loss or unacceptable delay in transmission. However, these may be results of faulty routers or corrupt packets. Hence these do not necessarily form good indices for congestion detection. Also, delay or loss of performance is very much a qualitative measure. Delay may be acceptable for one type of user and not to another. However, proper congestion control mechanisms do result in better network performance under heavy load conditions.
1.6 TCP Loss Recovery

Since packets can be lost in transit between the sender and the receiver due to queue overflows in a computer network, a network service for reliable data exchange between applications requires end systems to implement a mechanism to detect the loss of packets and retransmit the lost packets. TCP detects the loss of packets by associating each data byte with a uniquely identified sequence number.

The receiver acknowledges the receipt of a data packet by sending the sender an acknowledgment packet (ACK packet). The ACK packet carries a sequence number indicating the sequence number of the next in-order byte expected by the receiver (i.e., the sequence number of the last byte in the last in-order data packet received at the receiver plus one). Thus, an ACK packet acknowledges to the sender that all data bytes that have a sequence number smaller than that of the ACK packet have been received by the receiver. For each data packet that arrives at the receiver out of order, the receiver retransmits an ACK packet that contains the sequence number of the last data byte that has been received in order. The ACK packets triggered by the arrival of out-of-order data packets are called duplicate ACKs because they carry the same sequence number of the last in-order data byte received by the receiver.

In the case that the sender can send multiple packets in a sequence, a fast way to detect loss of data packets is via a direct use of duplicate ACKs. As mentioned above, the receiver sends duplicate ACKs to inform the sender that data packets arrived at the receiver out of order. Duplicate ACKs can be caused by different network problems. First, as discussed above, duplicate ACKs can be caused by the loss of some data packets. In this case, the receiver generates a duplicate ACK for each data packet arriving at the receiver after the lost packets. Second, duplicate
ACKs can be caused by replication of data or ACK packets in the network. Third, duplicate ACKs can be triggered by re-ordering of data packets in the network because each data packet arriving at the receiver out of order causes the receiver to generate a duplicate ACK. Of these three causes, loss is by far the most common.

From the sender's perspective, the last two network problems that cause duplicate ACKs are undesirable but they are rather "harmless" because they do not cause loss of data. However, loss of data packets that causes duplicate ACKs requires the sender to retransmit the lost data packets. Since packet replication and packet re-ordering are rare events, when the TCP sender receives three duplicate ACKs in a row, it assumes that the data packet with the sequence number carried by the duplicate ACKs has been lost and retransmits that packet.

Duplicate ACKs help sender detect and retransmit lost data packets. However, duplicate ACKs are only triggered when a lost data packet is followed by other data packets that arrive at the receiver. If the sender transmits a sequence of data packets and the last data packet in the sequence is lost, the receiver cannot detect the loss of the data packet. In this case, the receiver does not generate duplicate ACKs to inform the sender about the loss of the data packet. Furthermore, duplicate ACKs transmitted by the receiver can also be lost on their way back to the sender. Because of these problems, the sender also needs to rely on other local mechanisms to detect the loss of its data packets. The sender initializes a timer for each data packet that it transmits (more details about the timer's interval will be provided in section 1.4.3). When a timer for a data packet expires and the acknowledgment for that data packet has not been received, the sender assumes that the data packet has been lost and retransmits it.
1.7 Objectives of Congestion Control

The main objective of congestion control is to prevent the saturation of network resources, while ensuring some degree of fairness in the allocation of resources. In simple terms, a suitable feedback control mechanism requires that routers are able to sense the state of congestion and react accordingly by sending congestion signals to sources. This way, sources can decrease the flow rates submitted to the network and remove the congestion state (at packet level) of the routers.

Network feedback can be exercised over two types of traffic sources, so called window-based and rate-based sources. Although in terms of objectives the control over either type of sources is the same (the control of flow rates), there are differences in terms of implementation. The control over window based sources has been considered almost exclusively in the Internet, where as the control over rate-based sources has flourished within the ATM ABR proposals. The reason for this is that, for the case of the Internet, the window-based TCP sources dominate the scene and the control over these sources is based exclusively on binary feedback (implicit feedback). The only alternative for having rate-based feedback in the Internet would be to consider the conversion of rates into window sizes at TCP sources and to apply such feedback to non-TCP sources as well (i.e. UDP). The idea of explicit-window feedback has been considered for a short period within the ATM Forum community (known as credit-based control), but was abandoned shortly afterwards in favor of explicit rate feedback. The implementation of window-based schemes is intrinsically simpler than that of rate-based mechanisms, because in rate-based schemes it is necessary to have scheduling routines for packet generation at the sources with fine granularity clocks. However, the advantages of having routers with algorithms...
performing explicit-rate feedback have been demonstrated by a huge amount of proposals and performance studies.

In the context of the Internet, congestion control mechanisms have an impact on the way bandwidth is shared among IP flows, significantly affecting the performance perceived by the end user. Therefore, the design of congestion control mechanisms necessarily takes into account traffic characteristics and associated QoS requirements, while aiming to use network resources as efficiently as possible.

1.8 Traffic Classification

The Internet traffic can be broadly classified into two categories: streaming traffic and elastic traffic. Streaming traffic corresponds to traffic generated typically by audio and video applications, while elastic traffic is generated by file transfer applications such as point-to-point web browsing or ftp.

The Quality-of-Service (QoS) requirements are fundamentally different for each case. Streaming applications require controlled end-to-end packet delays, delay jitter and packet loss. Elastic applications require reliable data transmission with an "acceptable" transfer rate. They are called "elastic" because they are delay tolerant (at packet level at least) and continuously adapt their transfer rate to available bandwidth.

Elastic flows are subject to closed loop control. In contrast, the transmission rate of streaming flows (e.g. a video sequence) at the source is generally intrinsic (e.g. codec rate) in the sense that it does not depend on network conditions. Streaming flows generally rely on open loop control.

In order to preserve the semantic integrity of elastic traffic, sources use TCP, which guarantees the reliable transmission of data. TCP integrates also the congestion control algorithms (e.g. slow start, congestion avoidance) that are responsible for
adapting the transfer rate to available network resources. The TCP sources react to network indications mainly through loss events.

Sources hosting streaming applications use UDP which is more appropriate for preserving their time integrity. For enhanced quality of service, UDP is generally complemented with control protocols such as RTCP providing packet loss and delay indications that can be exploited by applications.

1.9 Traffic Entity to which Congestion Control Applies

Congestion control actions can be performed at different traffic levels: packet, flow or session, depending on the underlying objectives. A flow consists of a succession of erratically spaced packets pertaining to a particular instance of application, such as a document transfer or an audio stream. Sessions are composed of flows separated by think times.

On a given network link congestion occurs whenever traffic demand exceeds the link capacity. Traffic demand corresponds to the product: arrival rate x average size, of packets, flows or sessions. Therefore, it is important to identify the level at which congestion is experienced to take the appropriate congestion control actions. Congestion control actions in the Internet are generally performed at packet level (e.g. packet dropping and packet marking). These actions though performed at packet level affect the transfer rate of e.g. a TCP connection, and have therefore an impact on flow level performance. Conversely, congestion control actions taken at flow level, such as e.g. flow blocking, affect the performance perceived at packet level, ensuring negligible delay and loss for streaming applications, for example.
1.10 Congestion Control Objectives for Elastic Traffic

Fairness and efficiency with respect to the way bandwidth is shared between contending elastic flows are generally considered as the main design objectives of congestion control mechanisms in the Internet. Fairness qualifies a bandwidth allocation where no user is penalized compared to others that share the same bottleneck links. Efficiency means that no bandwidth is wasted in the sense that the throughput of any flow is only constrained by the capacity of some network link along its path. Both concepts are however tightly related as discussed in the sequel.

1.10.1 Fairness

*Fairness objectives and performance* Different concepts of fairness exist. Max min is often the stated bandwidth sharing objective, where the rate of individual flows is made as equal as possible. In a network with a single congested link, max-min fairness implies an equal share of the bandwidth for each competing connection. The congestion control algorithms currently used in data networks generally aim to realize max-min fair sharing, although precision in realizing this objective is often sacrificed in the interest of simplicity. The appropriateness of the max-min fairness concept has been questioned in , where a different concept of fairness, called proportional fairness, was proposed. A proportional fair allocation maximizes an objective function representing the overall utility of flows in progress, assuming that the utility of a flow increases with allocated bandwidth , in proportion to log . In a more general context, proportional fairness is a special case of the well known Nash Bargaining concept which has been used in the economic context for around 50 years and for which some networking applications are specified in. In the optimal property of proportional fairness is derived in a static regime where a fixed set of source-destination pairs
share the network resources for the transfer of infinite sized documents. In reality, the number of flows in progress is dynamic, increasing at the epochs of some arrival process and decreasing once the document transfer is completed.

In the authors evaluate the impact of different fairness concepts on the performance of elastic flows, under random traffic conditions, assuming Poisson arrivals of flows of finite size. A comparative study considering a linear network assuming Poisson flow arrivals shows that average response time is smaller for max-min than proportional fairness under random traffic, while proportional fairness is preferable under the assumption of static configuration of persistent flows.

The evaluation of different forms of fairness in a dynamic context indicates that introducing priority in favor of one type of flows can modify the usual stability condition (load of every link less than 1) and may lead to suboptimal utilization of network resources. However, for a broad class of "sufficiently" fair bandwidth allocation, the throughput of the flows is mainly constrained by their access rate. Therefore, an absolute degree of fairness is not necessary and performance can be approximated by that of an insensitive bandwidth sharing allocation known as balanced fairness.

Accomplishing the fairness objectives in the particular case of the most widely deployed version of TCP, known as TCP New Reno, the objectives of fairness are imperfectly accomplished. This version of TCP has the well known property of sharing bottleneck bandwidth in inverse proportion to the round trip time between source and destination [8]. Investigating the kind of fairness realized by different flavors of TCP has given rise to a lot of work. It appears from that classical AIMD (Additive Increase Multiplicative Decrease) algorithms tend to produce allocations which are proportionally fair rather than max-min fair (sharing the property of biasing
against long RTTs). The fairness properties of end-to-end window-based congestion control mechanisms are also investigated in.

The issue of fairness also arises in environments where TCP New Reno flows share network resources with more aggressive TCP versions, notably versions adapted to high-speed transfers and those with more aggressive starting behavior. Several AQM mechanisms have been proposed to enforce fairness, allowing different versions of TCP to co-exist and penalizing unresponsive flows.

For example, the RED variant CHOKE, and more recently AFD, provides approximate fairness through differential dropping. Also, the use of per flow fair queuing on network links leads naturally to max-min fairness. There have also been proposals specifying the way to perform rate control in TCP, notably RFC 3448 on TCP-friendly Rate Control.

1.10.2 Network Efficiency

Network efficiency is strongly related to the bandwidth sharing objectives and can for instance be measured through overall network good throughput. The slow start algorithm in TCP is a cause of inefficiency for many TCP connections. Modifications to TCP have been proposed to allow more aggressive starting behavior or more intelligent ways of inferring available bandwidth, like TCP Vegas.

To preserve network efficiency during overload periods, i.e. when traffic demand exceeds the network capacity, per-flow implicit admission control is advocated in. Situations of overload may occur due to traffic forecasting errors or network equipment failures. The authors argue that in sustained overload proactively blocking flows at the early onset of congestion is preferable to allowing throughput deterioration. Ultimately the same amount of traffic is transmitted since the link is
saturated. Admission control ensures that this throughput is experienced as good throughput. An Experimentation of admission control for TCP connections was actually conducted on an Internet access link of a campus. In other proposals, efficiency is enforced by the network through the scheduling mechanism. Size-based differentiation, for example, allows in overload to serve a maximum number of users.

### 1.11 Congestion Control Objectives for Streaming Traffic

As mentioned above, streaming traffic generally relies on open loop control. The quality of service of streaming traffic is generally ensured through admission control along with appropriate scheduling disciplines. Additional traffic control mechanisms are generally necessary to make sure streaming flows to not exceed their contracted or assumed rate (policing and marking). In a best-effort network, however, the main design objective of congestion control schemes for media streaming applications is to ensure their responsiveness to network congestion indications. Otherwise, TCP connections and other responsive flows experience severe throughput degradation. This concern has given rise to several TCP-friendly congestion control protocol proposals.

These protocols try to infer the network congestion level based on the receiver's feedback, and adjust their sending behavior in order to achieve fairness and efficiency. According to the most widely accepted definition, "a stream is said to be TCP friendly if it receives the same share of bandwidth as a TCP connection would receive under the same circumstances". According to a less conservative and less restrictive definition, "a flow is TCP-friendly if other TCP flows connections sharing the same bottleneck link as it do not receive less bandwidth in the long term than they would receive if the flow in concern a TCP connection possibly with a short Round
Trip Time”. This definition allows for better link utilization and takes into account unfairness among the TCP connections themselves.

The sending rate regulated by TCP-friendly protocols can be window-based, rate-based (e.g. RAP, TFRC), or can combine both approaches (e.g. SCP [1]). Window-based protocols control the amount of data flowing in the network, while the rate-based protocols control the sending rate. Rate-based protocols can be further divided into AIMD-based protocols (e.g. RAP) and equation-driven protocols (e.g. TFRC).

The latter class of protocols abstract TCP responses to data events into a model with input parameters like the loss rate and RTT estimation. Most protocols perform better with some network support for enforcing fairness. Indeed drop-tail routers may lead misleads sources by dropping consecutive packets of the same flow. Because the network cannot solely rely on the cooperation of users, it needs to be able to detect unresponsive flows and reduce their rates, using appropriate AQM mechanisms like differential dropping, in order to enforce a certain form of fairness.

1.12 Stability of Congestion Control Schemes

The objective of congestion control mechanisms is to share network resources among contending users in a fair and resource-efficient way. In the current Internet, this resource sharing is performed distributed by TCP and AQM protocols that dynamically adjust source rates and packet markings to achieve these goals in equilibrium. To ensure that the protocols will be able to reach and maintain a favorable equilibrium, it is important to assess the dynamical properties, such as stability and convergence, of the schemes. Instability means that the protocol is unable to sustain the equilibrium, and manifests itself as severe oscillations in
aggregate traffic quantities, such as queue lengths. There are two sources of instability [25]: stochastic instabilities, due to the random nature of packet loss or packet marking, and delay instabilities, due to high gains and time delays. It has been observed that for typical protocol combinations, such as Reno/RED[1], delay instabilities are dominating. An elegant framework for analysis and design of congestion control schemes has been developed.

Stability of the basic schemes, which allow dynamic rate control and static marking, or dynamic queue management schemes and static source rate control, have been established in idealized settings. When both source rate and link price updates are dynamic, stability has been proven using timescale separation in, and for the single bottleneck case in. A unifying framework for establishing global stability of congestion control laws based on passivity has been proposed in. The above results have ignored the effect of network delay, and assumed that price information is available instantaneously at the source, that the sources take immediate action, and that the new rates affect the link prices instantaneously.

However, stability of the protocols in equilibrium depends critically on the feedback delay, and it is crucial to understand this influence in order to design TCP/AQM protocols that are stable for general network topologies, delays and link capacities. Conditions for local stability of a single-user, single bottleneck scenario was derived in, and it was conjectured that the same condition guarantees stability also in the case of heterogeneous round-trip delays.

A weaker version of the conjecture was proven in and the original conjecture was proven in. Local stability of Reno/RED with feedback delays has been studied in. The stability analysis reveals that these protocols tend to become unstable when the delay increases and, more surprisingly, when the capacity increases. This has spurred
an intensive research in protocols that maintains local stability also for networks with high bandwidth-delay product,

1.13 Thesis Statement

The rest of this dissertation is organized as follows. Chapter 2 provides a survey of Active Queue Management Techniques and reviews relevant related work. Chapter 3 explains about the Modified Random Early Detection (ModRED) effects on the performance of TCP applications. Chapter 4 explains about Performance Comparison of ModRED with Load Based Adaptive Virtual queue Management Technique. Chapter 5 and 6 explains about Performance Comparison of ModRED with Load & Queue Based REM queue Management Technique and BLUE queue Management Technique. Chapter 7 summarizes and concludes the dissertation.