Tạp chí Tin học và Điều khiển học, T. 18, S. 2 (2002), 187–200

TCP ENHANCEMENTS AND PERFORMANCE OVER NETWORKS WITH WIRELESS LINKS

NGUYEN DINH VIET

Abstract. Transmission Control Protocol (TCP) uses end-to-end flow, congestion, and error control mechanisms to provide reliable delivery over the Internet. During many years of extensive use in inter-networks, all weak-points of TCP are discovered and tuned, so that it performs very well in traditional network (wired network). However, inter-networks growth explosively and may consist of networks with wireless links characterized by high and sporadic bit error rate and intermittent connectivity due to hand off. TCP performance is degraded severely in such networks. Many studies have been carrying out to improve the performance of TCP in networks with wireless links. Some solutions are proposed and implemented.

This paper presents four main issues: congestion avoidance and control mechanisms implemented in traditional TCP protocol, step-by-step enhancements in traditional TCP, characteristics of wireless links degrade TCP performance severely, mechanisms to enhance TCP performance over network with wireless links.

Tóm tắt. Giao thức TCP sử dụng cơ chế điều khiển thông lượng, kiểm soát tắc nghẽn và lỗi từ hai đầu mút để vận chuyển thông tin trên Internet một cách tin cậy. Trong suốt nhiều năm được sử dụng rộng rãi, các nhược điểm của TCP thể hiện ra đã được phát hiện và khắc phục. Chính vì thế TCP hoạt động rất tốt trong các mạng kiểu truyền thống (mạng có dây). Tuy nhiên các liên mạng đã có sự phát triển bùng nổ và chúng có thể bao gồm cả các mạng không dây. Đường truyền không dây có đặc trưng là tỉ suất lỗi bit cao, thất thường đồng thời thỉnh thoảng bị đứt đoạn do sự chuyển cuộc gọi. Trong các mạng như vậy, hiệu suất của TCP bị giảm trầm trọng. Người ta đã tiến hành rất nhiều nghiên cứu nhằm nâng cao hiệu suất của TCP trong các mạng có đường truyền không dây, một số giải pháp đã được đưa ra và áp dụng.

Bài báo này trình bày cô đọng bốn vấn đề chính: các cơ chế tránh tắc nghẽn và điều khiển đã được áp dụng trong giao thức TCP truyền thống, các bước cải tiến trong giao thức TCP truyền thống, các đặc trưng của đường truyền không dây làm giảm trầm trọng hiệu suất của giao thức TCP, các cơ chế để nâng cao hiệu suất của giao thức TCP trong các mạng có đường truyền không dây.

1. INTRODUCTION

Reliable transport protocols as TCP have been tuned for traditional networks comprising wired links and stationary hosts. TCP protocols assume congestion in the network to be the primary cause for packet losses and unusual delays. TCP performs well over such networks by adapting to end-to-end delays and congestion losses. The TCP sender uses the cumulative acknowledgments it receives to determine which packets have reached the receiver, and provides reliability by retransmitting lost packets. For this purpose, it maintains a running average of the estimated roundtrip delay and the mean linear deviation from it. The sender identifies the loss of a packet either by the arrival of several duplicate cumulative acknowledgments or the absence of an acknowledgment for the packet within a timeout interval. TCP reacts to packet losses by dropping its transmission (congestion) window size before retransmitting packets, initiating congestion control or avoidance mechanisms, and backing off its retransmission timer. These measures result in a reduction in the load on the intermediate links, thereby controlling the congestion in the network.

Current networks often include wireless links and mobile hosts, in particular, there will be LANs composed of wireless cells of only a few meters in diameter. TCP protocol will thus encounter types of delay and loss that are unrelated to congestion. Some performance degradation due to these delays and losses is unavoidable. These events also trigger congestion control procedures that further degrade performance.

2. CONGESTION AVOIDANCE AND CONTROL IN TRADITIONAL NETWORKS

Computer networks have grown explosively for more than twenty five years and with that growth have come severe congestion problems. In October of '86, the first time, the Internet had suffered from a series of "congestion collapses". During this period, the data throughput from Lawrence Berkeley Laboratory to University of California at Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps. Van Jacobson and his group of scientists started an investigation of why things had gotten so bad [15]. Jacobson has shown that much of the cause lies in transport protocol implementations (*not* in the protocols themselves): The 'obvious' ways to implement a window-based transport protocol can result in exactly the wrong behavior in response to network congestion. He gives examples of 'wrong' behavior and describes some simple algorithms that can be used to solve problems. The algorithms are rooted in the idea of achieving network stability by forcing the transport connection to obey a 'packet conservation' principle; if this principle were obeyed then, congestion collapse would become the exception rather than the rule. *Thus congestion control involves finding places that violate conservation and fixing them.* The concept of 'conservation of packets' means that for a connection running stably with a full window of data in transit, a new packet is not put into the network until an old packet leaves the network. The physics of flow predicts that systems with this property should be robust in the face of congestion.

Observation of the Internet suggests that it was not particularly robust. Why the discrepancy? There are only three ways for packet conservation to fail [15]:

Failure 1: The connection does not get to equilibrium, or

Failure 2: A sender injects a new packet before an old packet has exited, or

Failure 3: The equilibrium can not be reached because of resource limits along the path.

Jacobson [15] also proposed algorithms to solve these failures; they are summarized in the following.

2.1. Slow start

Reasons for failure (1): The network announces, via a dropped packets, when demand is excessive but says nothing if a connection is using less than its fair share (since the network is stateless, it cannot know this). Thus a connection has to increase its bandwidth utilization to find out the current limit, and it can only find out the limit after an excessive use of bandwidth.

Algorithm: *slow-start:* to gradually increase the amount of data in-transit to get to equilibrium

- Add a congestion window, cwnd, to the per-connection state.
- When starting or restarting after a loss, set cwnd to one packet.
- On each ack for new data, increase cwnd by one packet.
- When sending, send the minimum of the receiver' s advertised window and cwnd.

In fact, the slow-start window increase exponentially, it takes time $R \log_2 W$ to reach the size of W, where R is the round-trip-time and W is the window size in packets. This means the window opens quickly enough to have a negligible effect on performance, even on links with a large bandwidth-delay product. With slow-start algorithm, a *sender will transmit data at a rate at most twice the maximum possible on the path.* This means that packet loss is inevitable.

2.2. Intelligent round-trip timing

Reasons for failure (2): In the network, packet loss is inevitable, so the problem is how soon a sender retransmits lost packets, and how are the spaces between retransmitted packets, while remaining conservation of packets. This problem relates to sender' s retransmit timer. A good round trip time (RTT) estimator, the core of the retransmit timer, is the single most important feature of any protocol implementation that expects to survive heavy load. Unfortunately, it is frequently botched, because network topology and the number of competing connections are unknown, unknowable and constantly changed [8, 17]. The first mistake is that the sender cannot estimate correctly the variation,

σ_R, of the round trip time, R. From queuing theory we know that R and the variation in R increase quickly with load. The second mistake is the backoff after a retransmit: If a packet has to be retransmitted more than once, how should the retransmits is spaced?

Solution for the first mistake

The TCP protocol specification [4] suggests estimating mean round trip time via the low pass filter:

$$R \leftarrow \alpha R + (1 - \alpha)M.$$

Where R is the average RTT estimate, M is a round trip time measurement from the most recently acked data packet, and α is a filter gain constant with a suggested value of 0.9. Once the R estimate is updated, the retransmit timeout interval, rto, for the next packet sent is set to βR. The parameter β accounts for RTT variation, the suggested $\beta = 2$.

Solution for the second mistake

Only one scheme has any hope of working that is exponential backoff. TCP resets the transmission timer to a backoff interval that doubles with each consecutive timeout. This scheme is explained in [15, 12].

2.3. Congestion avoidance

Reasons for failure (3): If the timers are in good shape, it is possible to state with some confidence that a timeout indicates packet loss. Packets get lost for two reasons: are damaged in transit, or dropped at some node on the path in the network, because of the insufficient buffer capacity of that node. On wired networks, where bit error rate (BER) is very low[1], it is possible to say that packet loss is for network congestion. The case of networks with wireless links is different and will be discussed in Sec. 4.

Solutions: A 'congestion avoidance' strategy, such as the one proposed in [9], will have two components: *The network must be able to signal the transport endpoints that congestion is occurring or about to occur, and the endpoints must have a policy that decreases utilization if this signal is received and increases utilization if the signal is not received.*

Endnode action: Congestion avoidance: to adapt to the path

It is additive increase/multiplicative decrease policy, as was implemented in BSD [15]. It can be expressed as follows:

- On any timeout, set cwnd to half the current window size (this is the multiplicative decrease).
- On each ack for new data, increase cwnd by 1/cwnd (this is the additive increase)[2].
- When sending, send the minimum of the receiver's advertised window and cwnd.

In practice, slow start and congestion avoidance algorithms are implemented together as explained in more detail in Section "3.2 Tahoe TCP".

Network action (the gateway side) of congestion control

The goal of this algorithm is to send a signal to the endnodes as early as possible, but not so early that the gateway becomes starved for traffic. Gateway simply drops packets sent by endnodes to tell them of using more than its fair share. Thus, the gateway algorithm should reduce congestion even if no endnode is modified to do congestion avoidance. And nodes that do implement congestion avoidance will get their fair share of bandwidth and a minimum number of packet drops.

Since congestion grows exponentially, detecting it early is important. If detected early, small adjustments to the senders' windows will fix it. Otherwise massive adjustments are necessary to

[1] BER = $10^{-9}...10^{-12}$, given packet size 1000 bits, then packet loss probability is $10^{-6}...10^{-9}$.

[2] In TCP, windows and packet sizes are in bytes so the increment translates to *maxseg*maxseg/cwnd* where *maxseg* is the maximum segment size and *cwnd* is expressed in bytes, not packets.

give the net enough spare capacity to pump out the backlog. But the bursty nature of traffic makes reliable detection a difficult problem.

3. ENHANCEMENT TO TCP PROTOCOLS (FOR WIRED NETWORKS)

Early TCP implementations followed a go-back-n model using cumulative positive acknowledgment and requiring a retransmit timer expiration to re-send data lost during transport. These TCPs did little to minimize network congestion. Modern TCP implementations contain a number of algorithms aimed at controlling network congestion while maintaining good user throughput [15, 12].

3.1. Initial TCP

TCP based on concepts first described by Cerf and Kahn [5]. It is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols just above a basic Internet Protocol, which support multi-network applications. TCP is able to send and receive variable-length successive units of data, called segments, enclosed in internet datagram "envelopes". The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

higher–level
TCP
internet protocol
communication network

Fiuure 1. Protocol Layering

Multiplexing: To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a *socket.* A socket may be simultaneously used in multiple connections.

Connections: The reliability and flow control mechanisms of TCP require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a *connection.* Each connection is uniquely specified by a pair of sockets identifying its two sides.

Reliability: The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each byte transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates.

Window Flow Control: TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of bytes that the sender may transmit before receiving further permission. There is an assumption that this is related to the receiver' s data buffer space, which currently available for this connection. Indicating a large window encourages transmissions, but if more data arrives than can be accepted, it will be discarded, and results in excessive retransmissions. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The sending TCP must be prepared to accept from the user and send at least one byte of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval

when the window is zero [4]. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other. When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

3.2. Tahoe TCP

The Tahoe TCP implementation added a number of new algorithms and refinements to earlier implementations. The new algorithms include *Slow-Start, Congestion Avoidance, and Fast Retransmit.* The refinements include a modification to the round-trip time estimator used to set retransmission timeout values [10, 16]. Some modifications have been summarized in Section 2.1, 2.2, and 2.3 above. Figure 2 illustrate the operations of Tahoe TCP, in the case of one dropped packet. Some explanations related to the figure are presented in Sec. 3.6.

Figure 2. Tahoe TCP with one dropped packet

Slow–Start phase: Starting from one packet, the window is increased exponentially by one packet for every nonduplicate ACK until the source estimate of network capacity ("pipe size") is reached. That is the maximum number of packets that can be fit on the path. This is Slow–Start (SS) phase, and the capacity estimate is called the SS threshold (*ssthresh*). SS aims to alleviate the burstiness of TCP while quickly filling the pipe.

Congestion Avoidance phase: Once the *ssthresh* is reached, the source switches to a slower increase in the window by one packet for every window' s worth of ACKs. This phase, called Congestion Avoidance (CA), aims to slowly probe the network for any extra bandwidth. The window increase is interrupted when a loss is detected. Two mechanisms are available for the detection of losses: the expiration of a retransmission timer (timeout) or the receipt of three duplicate ACKs.

Fast Retransmit (FRXT): This algorithm is of special interest because it is modified in subsequent versions of TCP. With FRXT, after receiving a small number of duplicate acknowledgments for the same TCP segment (dup ACKs), the data sender infers that a packet has been lost and retransmits the packet without waiting for a retransmission timer to expire (timeout), then returns to slow start (window $= 1$). This leads to higher channel utilization and connection throughput.

3.3. Reno TCP

The Reno TCP implementation retained the enhancements incorporated into Tahoe, but modified the Fast Retransmit operation to include *Fast Recovery* (FRCV) [10, 16]. The new algorithm prevents the communication path ("pipe") from going empty after FRXT, thereby avoiding the need to SS to re-fill it after a single packet loss. FRCV operates by assuming each dup ACK received represents a