Analytical Methods for Network Congestion Control

Synthesis Lectures on Communication Networks

Synthesis Lectures on Communication Networks is an ongoing series of 50- to 100-page publications on topics on the design, implementation, and management of communication networks. Each lecture is a self-contained presentation of one topic by a leading expert. The topics range from algorithms to hardware implementations and cover a broad spectrum of issues from security to multiple-access protocols. The series addresses technologies from sensor networks to reconfigurable optical networks.
The series is designed to:

- Provide the best available presentations of important aspects of communication networks.
- Help engineers and advanced students keep up with recent developments in a rapidly evolving technology.
- Facilitate the development of courses in this field

Wireless Network Pricing
Jianwei Huang and Lin Gao
2013

Performance Modeling, Stochastic Networks, and Statistical Multiplexing, Second Edition
Ravi R. Mazumdar
2013

Packets with Deadlines: A Framework for Real-Time Wireless Networks
I-Hong Hou and P.R. Kumar
2013

Energy-Efficient Scheduling under Delay Constraints for Wireless Networks
Randall Berry, Eytan Modiano, and Murtaza Zafer
2012

NS Simulator for Beginners
Eitan Altman and Tania Jiménez
2012

Network Games: Theory, Models, and Dynamics
Ishai Menache and Asuman Ozdaglar
2011

An Introduction to Models of Online Peer-to-Peer Social Networking
George Kesidis
2010

Stochastic Network Optimization with Application to Communication and Queueing Systems
Michael J. Neely
2010

Scheduling and Congestion Control for Wireless and Processing Networks
Libin Jiang and Jean Walrand
2010

Performance Modeling of Communication Networks with Markov Chains
Jeonghoon Mo
2010

Communication Networks: A Concise Introduction
Jean Walrand and Shyam Parekh
2010

Path Problems in Networks
John S. Baras and George Theodorakopoulos
2010

Performance Modeling, Loss Networks, and Statistical Multiplexing
Ravi R. Mazumdar
2009

Network Simulation
Richard M. Fujimoto, Kalyan S. Perumalla, and George F. Riley
2006

Analytical Methods for Network Congestion Control

Steven H. Low

www.morganclaypool.com

ISBN: 9781627057332 paperback
ISBN: 9781627055994 ebook

DOI 10.2200/S00778ED1V01Y201705CNT018

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON COMMUNICATION NETWORKS

Lecture #18
Series Editor: R. Srikant, *University of Illinois at Urbana-Champaign*
Founding Editor Emeritus: Jean Walrand, *University of California, Berkeley*
Series ISSN
Print 1935-4185 Electronic 1935-4193

Analytical Methods for Network Congestion Control

Steven H. Low
California Institute of Technology (Caltech)

SYNTHESIS LECTURES ON COMMUNICATION NETWORKS #18

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

The congestion control mechanism has been responsible for maintaining stability as the Internet scaled up by many orders of magnitude in size, speed, traffic volume, coverage, and complexity over the last three decades. In this book, we develop a coherent theory of congestion control from the ground up to help understand and design these algorithms. We model network traffic as fluids that flow from sources to destinations and model congestion control algorithms as feedback dynamical systems. We show that the model is well defined. We characterize its equilibrium points and prove their stability. We will use several real protocols for illustration but the emphasis will be on various mathematical techniques for algorithm analysis.

Specifically we are interested in four questions:

1. How are congestion control algorithms modelled?
2. Are the models well defined?
3. How are the equilibrium points of a congestion control model characterized?
4. How are the stability of these equilibrium points analyzed?

For each topic, we first present analytical tools, from convex optimization, to control and dynamical systems, Lyapunov and Nyquist stability theorems, and to projection and contraction theorems. We then apply these basic tools to congestion control algorithms and rigorously prove their equilibrium and stability properties. A notable feature of this book is the careful treatment of projected dynamics that introduces discontinuity in our differential equations.

Even though our development is carried out in the context of congestion control, the set of system theoretic tools employed and the process of understanding a physical system, building mathematical models, and analyzing these models for insights have a much wider applicability than to congestion control.

KEYWORDS

communication networks, congestion control, projected dynamics, convex optimization, network utility maximization, Lyapunov stability, passivity theorems, gradient projection algorithm, contraction mapping, Nyquist stability

Contents

Preface

The congestion control mechanism has been responsible for maintaining stability as the Internet scales up by many orders of magnitude in size, speed, traffic volume, coverage, and complexity over the last three decades. Our primary goal is to develop a coherent theory of Internet congestion control from the ground up to help understand and design the equilibrium and stability properties of large-scale networks under end-to-end control.

In addition, we have two broader purposes in mind. First we wish to introduce a set of system theoretic tools and illustrate their application to concrete problems. Second we wish to demonstrate in depth the entire process of understanding a physical system, building mathematical models of the system, analyzing the models, exploring the practical implications of the analysis, and using the insights to improve a design. Even though our development is carried out in the context of congestion control, these basic analytical tools and the research process are much more broadly applicable.

The Internet, called ARPANet at the time, was born in 1969 with four nodes. The Transmission Control Protocol (TCP) was published by Vinton Cert and Robert Kahn in 1974 [14], split into TCP/IP (Transmission Control Protocol/Internet Protocol) in 1978, and deployed as a standard on the ARPANet by 1983. An Internet congestion collapse was detected in October 1986 on a 32-kilobits-per-second (kbps) link between the University of California Berkeley campus and the Lawrence Berkeley National Laboratory that is 400 yards away, during which the throughput dropped by a factor of almost 1,000 to 40 bits-per-second (bps). Two years later Van Jacobson implemented and published the congestion control algorithm in the Tahoe version of TCP [26] based on an idea of Raj Jain, K.K. Ramakrishnan, and Dah-Ming Chiu [27]. Before Tahoe, there were mechanisms in TCP to prevent senders from overwhelming receivers, but no effective mechanism existed to prevent the senders from overwhelming the network. This was not an issue because there were few hosts, until the mid-1980s. By November 1986 the number of hosts was estimated to have grown to 5,089 [1], but most of the backbone links have remained 50–56 bps since the beginning of the ARPANet. Jacobson's scheme adapts sending rates to the congestion level in the network, thus preventing the senders from overwhelming the network.

Jacobson anticipated even in his original paper [26] the network environments in which his algorithm will perform poorly: "... TCP spans a range from 800 Mbps [megabits per second] Cray channels to 1200 bps packet radio links." The algorithm worked very well over a network with relatively low transmission capacity, small delay, and few random packet losses. This was mostly the case in the 1990s, but as the network speed underwent rapid upgrades (see Figure 1), as Internet exploded onto the global scene beyond research and education, and as wireless

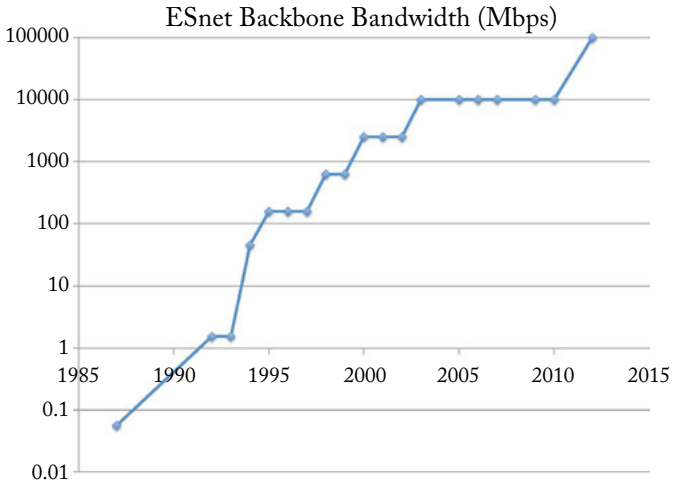


Figure 1: Highest link speed of U.S. Department of Energy's Energy Sciences Network (ESnet) from 1987 (56 kbps) to 2012 (100 Gbps) [2].

infrastructure was integrated with and mobile services proliferated on the Internet, the strain on the original design started to show. This motivated a flurry of research activities on TCP congestion control in the 1990s. A mathematical understanding of Internet congestion control started in the late 1990s with Frank Kelly's work on network utility maximization [28]. An intensive effort ensued and lasted for a decade to develop a theory to reverse engineer existing algorithms and understand structural properties of large-scale networks under end-to-end congestion control, systematically design new algorithms based on analytical insights, and deploy some of these innovations in the field.

This book is a personal account of that effort, focusing on the theory development.

We start in Chapter 1 with a summary of classical Internet congestion control protocols. We explain how to model them as dynamical systems using ordinary differential equations:

$$\begin{aligned} \dot{x} &= f(x(t), q(t)), & q(t) &= R^T p(t) \\ \dot{p} &= g(y(t), p(t)), & y(t) &= R\, x(t) \end{aligned}$$

and its variants, where $x(t), q(t) \in \mathbb{R}^N$, $p(t), y(t) \in \mathbb{R}^L$, and $R \in \{0,1\}^{L\times N}$ for a network with N nodes and L links. The graph structure is described by the routing matrix R. The decentralized nature of the system manifests itself in the structure of f and g:

$$\begin{aligned} \dot{x}_i &= f_i(x_i(t), q_i(t)), & q_i(t) &= \sum_l R_{li}\, p_l(t) \\ \dot{p}_l &= g_l(y_l(t), p_l(t)), & y_l(t) &= \sum_i R_{li}\, x_i(t) \end{aligned}$$

i.e., each node i (link l) updates its state $x_i(t)$ ($p_l(t)$) based only on local variables $(x_i(t), q_i(t))$ $(y_l(t), p_l(t))$. We prove the existence and uniqueness of solution trajectories to these equations. This ensures that the models are well defined. This class of network models is more general than congestion control and therefore the techniques developed here may be of wider applicability.

We prove in Chapter 2 that the equilibrium point of an arbitrary network under congestion control is the unique optimal solution of a simple convex optimization problem, called network utility maximization. Hence we can interpret congestion control as a distributed algorithm carried out by traffic sources and network resources to maximize utility over the Internet in real time. We explain several implications of this insight.

We present in Chapters 3–5 three different methods to study the global asymptotic stability of the equilibrium point, assuming there is no feedback delay. These methods are based on Lyapunov stability theorems, passivity theorems, gradient descent and contraction mapping theorems. The Lyapunov method is the basic tool for proving stability of general nonlinear systems. The passivity method allows one to analyze the stability of an interconnection of multiple dynamical systems in terms of the passivity of the component systems in open loop. The last method treats congestion control as a gradient algorithm for solving the dual of the network utility maximization.

Finally we describe in Chapter 6 the Nyquist stability method for analyzing local stability around the equilibrium point in the presence of feedback delay.

There is a large amount of literature on congestion control and we have not attempted to provide a survey. Pointers are provided at the end of each chapter only to some papers that are directly related to or extend materials covered in that chapter. We present proofs for some, but not all, of the classical results to illustrate techniques or concepts that we find particularly useful.

Many applications, including congestion control, can be modeled by a system of nonlinear differential equations of the form:

$$\dot{x} \quad = \quad (f(x(t))^+_{x(t)}$$

where the projection operation $(\cdot)^+_{(\cdot)}$ on the right-hand side ensures that the state variable $x(t)$ remains nonnegative. For example, $x(t)$ may represent the sending rates of traffic sources or the prices of an economy. The projection introduces discontinuity to the vector field, even when f itself is continuous, and complicates analysis. Analytical models often ignore projection even though nonnegative dynamics is prevalent in reality. A notable feature of this book is the careful treatment of the projected dynamics. In particular we include detailed proofs that extend standard results on the existence, uniqueness, equilibrium, and stability properties of smooth unprojected systems to discontinuous projected systems. Some of the stability proofs for congestion control algorithms modeled by projected dynamics in Chapters 3 and 4 are new. As we will see, projection mostly preserves these properties.

Steven H. Low
Pasadena, CA, June 2017

Acknowledgments

The idea of this book started when Jean Walrand of the University of California, Berkeley asked me in early 2004 to write a little book on TCP congestion control for Morgan & Claypool's Synthesis Lectures on Communication Networks, of which he was the inaugural Editor. I agreed but did not start writing until the summer of 2010, when I visited Karl Åström and Anders Rantzer at Lund University in Sweden with my extended family. That was a memorable summer! It was also when my research switched from Internet to power systems, so writing again went onto the backburner after the first draft at Lund. Major revisions were done during the summer of 2015, when I visited Janusz Bialek at Skoltech in Russia to give a short course on analytical methods for Internet and power systems, and during spring 2016, when I visited Jiming Chen, Youxian Sun, and Zaiyue Yang at Zhejiang University in China. I thank the tremendous encouragement and patience of Jean Walrand and the gentle prodding of the publisher Michael Morgan over more than a decade. It's a relief to have paid my debt. I also thank the warm hospitality of my hosts at Lund University, Skoltech, and Zhejiang University.

This book is a product of our FAST project at Caltech from 2000–2007, and I have learned a lot from my collaborators, especially John Doyle, Harvey Newman, and Fernando Paganini, and from the first generation of Netlab members, including Lachlan Andrew, Lijun Chen, Cheng Jin, George Lee, Lun Li, Mortada Mehyar, Christine Ortega, A. Kevin Tang, Jiantao Wang, David Wei, and Bartek Wydrowski. I thank the U.S. National Science Foundation (especially Darleen Fisher), Army Research Office, Air Force Office of Scientific Research, Cisco, and Caltech's Lee Center for Advanced Networking for their generous financial support. Some of us took the effort to deploy our research in the real world through a startup FastSoft. Since 2014, FastTCP has been accelerating more than 1 TB of Internet traffic every second. I experienced first hand the thrill and the challenge in crossing the gap from theory to practice and I thank my colleagues and supporters at FastSoft.

Linqi Guo has worked through the entire draft carefully and corrected numerous errors. I thank him for his meticulous reading and helpful suggestions. Teaching assistants of my networking course (cs/ee 143) at Caltech have contributed some of the exercises, especially Lingwen Gan, Ben Yuan, and Changhong Zhao. Finally, I thank my family, Jenny, Zhi, Zhiyou, my parents, and my sister's family for their unwavering support and trust.

Steven H. Low
Pasadena, CA, June 2017

Notations

We collect some of the notational conventions in this book.

Let $\mathbb{R}^n$, $n \geq 1$, be the set of n-dimensional real vectors, $\mathbb{R}^n_+$ the set of n-dimensional nonnegative real vectors, and $\mathbb{R}^{n\times m}$ the set of $n \times m$ real matrices. If x is a vector or matrix then x^T denotes its transpose. By default a vector x is taken to be a *column* vector and can be specified as either

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{or} \quad x = (x_1, \ldots, x_n) \quad \text{or} \quad x = (x_i, i = 1, \ldots, n).$$

Inequalities are taken componentwise, i.e., $x \geq 0\,(x > 0)$ means $x_i \geq 0\,(x_i > 0)$ for $i = 1, \ldots, n$. If $x_i \in \mathbb{R}^{n_i}$, $i = 1, \ldots, k$, are defined then, unless otherwise specified, x denotes the vector $x := (x_i, i = 1, \ldots, k)$ with dimension $n := \sum_i n_i$. Conversely if a vector x is defined then x_i denotes its ith component in $\mathbb{R}^{n_i}$. Similarly for functions $f_i : \mathbb{R}^{k_i} \to \mathbb{R}^{m_i}$, $i = 1, \ldots, n$, and $f := (f_i, i = 1, \ldots, n) : \mathbb{R}^K \to \mathbb{R}^M$ where $K := \sum_i k_i$ and $M := \sum_i m_i$.

For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, $\frac{\partial f}{\partial x}$ is the row vector and $\nabla f(x)$ is the column vector, both with components $\frac{\partial f}{\partial x_i}$. For a vector function $f : \mathbb{R}^n \to \mathbb{R}^n$, $\frac{\partial f}{\partial x}$ is the $n \times n$ Jacobian matrix defined by

$$\left[\frac{\partial f}{\partial x}\right]_{ij} := \frac{\partial f_i}{\partial x_j}.$$

Given a set of utility functions $U_i(x_i) : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, N$, $U_i'(x_i)$ denote their derivatives. We sometimes use $U : \mathbb{R}^N \to \mathbb{R}$ to denote the sum $U(x) := \sum_i U_i(x_i)$. Since U is separable in x_i we use $U'(x)$ to denote the *vector* $U'(x) := (U_i'(x_i),\ i = 1, \ldots, N)$.

For a scalar $a \in \mathbb{R}$, $(a)^+ := \max\{a, 0\}$; for a vector a, $(a)^+$ is defined componentwise, i.e.,

$$(a)^+ := \left([a_i]^+,\ \forall i\right).$$

For scalars $a, b \in \mathbb{R}$

$$(a)_b^+ := \begin{cases} a & \text{if } a > 0 \text{ or } b > 0 \\ 0 & \text{otherwise.} \end{cases}$$

If $a, b \in \mathbb{R}^n$ are vectors of the same dimension then $(a)_b^+$ is defined componentwise, i.e.,

$$\left[(a)_b^+\right]_i := (a_i)_{b_i}^+ \qquad \forall i.$$

We use $\|\cdot\|$ to denote an arbitrary norm and $\|x\|_2 := \sqrt{\sum_i x_i^2}$ the Euclidean norm. $B_\delta(x^*) := \{x | \|x - x^*\| \leq \delta\}$ is a *closed* ball around x^* in $\mathbb{R}^n$ unless otherwise specified. $A \subseteq B$ means A is a subset of B and $A \subset B$ means A is a strict subset of B. Given $a_i, i = 1, \ldots, n$, $\mathrm{diag}(a_i, i = 1, \ldots, n)$ denotes the diagonal matrix with a_i as its ith diagonal entry.

CHAPTER 1

Congestion Control Models

We consider a network under end-to-end congestion control as a deterministic feedback dynamical system described by a set of ordinary differential equations (ODEs). In Section 1.1 we present a simple model that treats data packets in the network as fluids that flow from their sources to their destinations. In Section 1.2 we describe classical Internet congestion control protocols. In Section 1.3 we illustrate how to model these protocols as feedback dynamical systems. In Section 1.4 we present general ODE models of congestion control algorithms and discuss limitations and extensions of these models. In Section 1.5 we discuss conditions that guarantee the existence and uniqueness of the solution to an ODE model. Finally we show that these conditions are satisfied by the ODE models of congestion control, ensuring that these models are well defined.

1.1 NETWORK MODEL

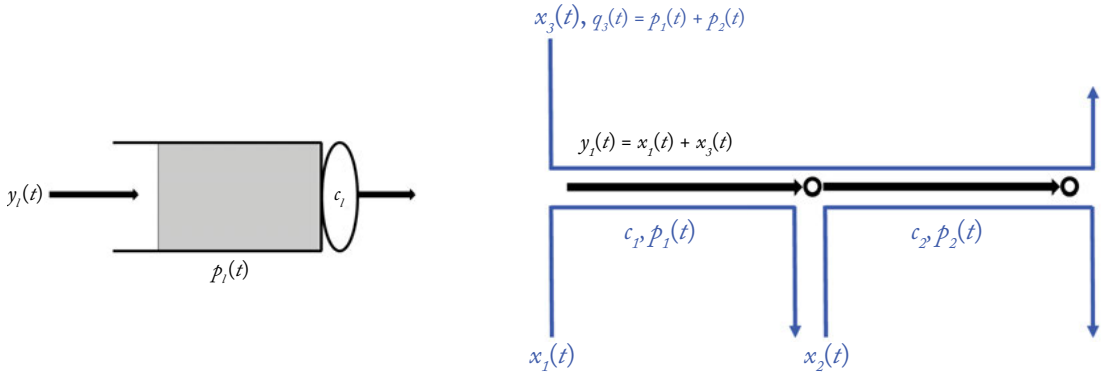
A network is an interconnected set of computing, storage, or communication resources shared by competing users. For our purposes, a user is typically not a human, but a traffic flow from a source to a destination through a subset of these resources. A computing or communication resource is characterized by how fast it can process or transmit information, in units of bits per second or packets per second. A storage resource queues up packets while they wait to be processed or transmitted. We will model each resource abstractly as a "link" that consists of a single server with a buffer (waiting space); see Figure 1.1a. We often assume that the buffer capacity is infinite. We will call the users "sources" or "flows."

Formally, a network is a set of L links with finite capacities $c = (c_l, l \in L)$ in packets per second (pps). They are shared by a set of N sources. We abuse notation and use L and N to both denote sets and their cardinalities. Each source i uses a set $L_i \subseteq L$ of links. The sets L_i define an $L \times N$ routing matrix R with entries:

$$R_{li} = \begin{cases} 1 & \text{if } l \in L_i \\ 0 & \text{otherwise.} \end{cases}$$

We refer to the set L_i of links as source i's path.

Each source i adapts its transmission rate $x_i(t)$ at time t, in pps, according to an algorithm based on some measure of congestion locally observed at source i. This local measure of congestion, denoted by $q_i(t)$ at time t, summarizes the congestion information on the path of source i. Each link l adapts, implicitly or explicitly, a congestion measure $p_l(t)$ at time t in response



(a) Link model (b) Network model

Figure 1.1: (a) A link is modeled as a fluid queue with link capacity c_l, infinite buffer size, input rate $y_l(t)$ at time t, and a congestion price $p_l(t)$ at time t. (b) A network is modeled as a collection of links l shared by a set of sources i with sending rates $x_i(t)$ at time t. The routing matrix in this example is $R = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. Source i observes the sum $q_i(t) = \sum_l R_{li}\, p_l(t)$ of link prices in its path and link l observes the aggregate rate $y_l(t) = \sum_i R_{li}\, x_i(t)$ from sources sharing the link.

to the aggregate input traffic rate $y_l(t)$ locally at link l. Due to its economic interpretation, we call $p_l(t)$ the congestion price at link l or the link price.

We will model below the source algorithms that update the source rates $x_i(t)$ and link algorithms that update link prices $p_l(t)$ by a set of differential and algebraic equations. These local algorithms are interconnected by the routing matrix R that aggregates the link prices $p_l(t)$ on the path of source i into a scalar price $q_i(t)$ observed at source i:

$$q_i(t) \;:=\; \sum_l R_{li}\, p_l(t), \qquad i \in N.$$

The routing matrix R also aggregates the source rates $x_i(t)$ of flows i that traverse link l into an aggregate flow rate $y_l(t)$ at link l:

$$y_l(t) \;:=\; \sum_i R_{li}\, x_i(t), \qquad l \in L.$$

These definitions are illustrated in Figure 1.1b.

1.2 CLASSICAL TCP/AQM PROTOCOLS

To model the algorithms adapting the source rates and link prices, we start by summarizing the basic mechanism of window-based congestion control. We then describe several classical

congestion control protocols. In the next section we will present mathematical models of these algorithms.

1.2.1 WINDOW-BASED CONGESTION CONTROL

Transmission Control Crotocol (TCP) is one of the transport layer protocols on the Internet (the other being User Datagram Protocol (UDP)). It provides a reliable bit stream end-to-end from a source to a destination over an unreliable datagram service provided by the Internet Protocol (IP) layer. It hides bit errors, packet losses, and reordering in the underlying network and delivers to the destination a bit stream that is free of error, loss or duplicate, in the same order it has been sent by the source. A TCP connection operates in three phases. In phase one a virtual circuit is set up between two end points. In phase two they exchange data and acknowledgment packets. Both end points can send data packets, and receive acknowledgments, regardless of which of them initiated the connection. In phase three, the connection is terminated; either end point can initiate the termination. An application that exchanges a large amount of data, e.g., video streaming, usually spends most of its time in phase two. Congestion control is used in phase two to regulate the sending rate of a sender. This is achieved through a *window mechanism*, as we now explain.

Consider a sender that wishes to send a large data file, e.g., a video server transferring a movie to a subscriber. The movie is broken into small chunks and control information is added to each chunk to form a *packet*. Examples of control information include sender and receiver information (IP addresses and TCP port numbers), control bits for error detection, sequence number to detect packet loss and to enable the assembly of the original movie at the destination from received packets. These packets are numbered consecutively and then sent into the network one by one from the source toward the destination.[1] An idealized operation is illustrated in Figure 1.2. When a packet is correctly received at the destination an ACK is sent from the destination toward the source. Figure 1.2 assumes an ACK packet has a negligible size compared with a data packet and incurs zero transmission time.[2] The time between sending of a data packet and the arrival of its ACK at the source is called the round-trip time (RTT). If a packet is lost in the network or incurs an excessive delay, the source will not receive its ACK in time and will retransmit the packet after a timeout. This is the basic mechanism to compensate for bit errors and packet losses.

If the original packet was not lost but only delayed, the receiver will receive duplicate packets. The sequence number in the packet will indicate that the packet is a duplicate and

[1] TCP is actually a "byte stream" where a packet is identified by the byte numbers the packet contains. This works in the face of packet fragmentation, but it is simpler for our purpose to think of packets being numbered by consecutive integers.

[2] If both end points send and receive data packets (two-way transfer) then each end point can piggybak its ACK in the data packet it sends to the other end. The header of a TCP packet contains both the field `sequence number` that indexes the data packet an end point originates and the field `ack number` that acknowledges the data packet it has received from the other end.

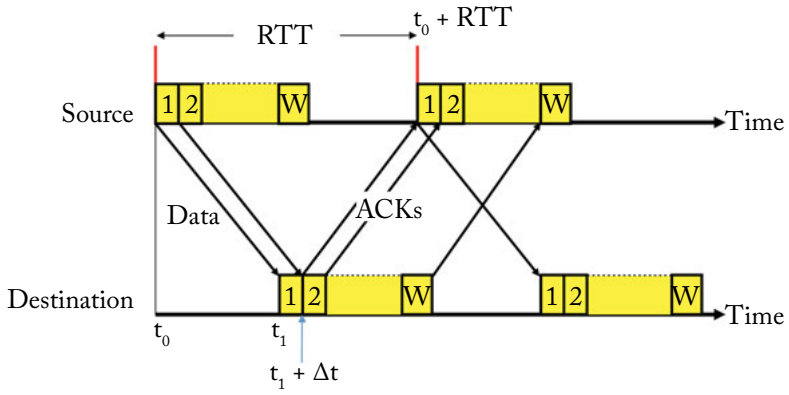


Figure 1.2: Window control mechanism (idealized). Packets are numbered 1, 2, ... W is the *window size*. In the first cycle, the transmission of packet 1 starts at time t_0 and finishes at $t_0 + \Delta t$. The first bit of packet 1 is received at t_1 and the last bit at $t_1 + \Delta t$. Immediately packet 2 starts to be transmitted at time $t_0 + \Delta$ and received at time $t_1 + \Delta$. The ACK for packet 1 is sent at $t_1 + \Delta$ and received at the sender at time t_0+RTT. The cycle then repeats.

will be discarded. The receiver will send an ACK for each duplicate packet in case the duplicate packets were sent because their ACKs were lost. Duplicate ACKs at the sender will be discarded.

As shown in Figure 1.2, a source keeps a variable called *window size* W that determines the maximum number of outstanding packets that are allowed to be transmitted but not yet acknowledged. When the window size is exhausted (i.e., the number of outstanding packets reaches W), the source must wait for an ACK before sending a new packet. The new packet can be sent only after the next ACK arrives correctly. In reality packets may incur bit errors, they can be dropped in the network, the RTTs can be random and packets may arrive at the destination out of order. Figure 1.2 ignores these complications and depicts an idealized scenario where all packets are of the same size and arrive at the destination correctly with constant RTT. In this idealized scenario the data transfer process is deterministic and periodic where exactly W packets are sent and acknowledged in each RTT. Moreover, in each RTT, the sender sends packets $1, 2, \ldots, W$ back-to-back, waits for their ACKs to return, and these ACKs trigger the next batch of W packets in the next RTT, and the cycle repeats.

Two features are important for our purpose. The first is the "self-clocking" feature that automatically slows down the source when a network becomes congested and ACKs are delayed. The second is that the window size controls the source sending rate: roughly W packets are sent every RTT (see Figure 1.2). Before Jacobson's proposal in 1988 the window size W was fixed by each TCP connection and the self-clocking feature was the only congestion control mechanism on the Internet. Jacobson's idea is to *dynamically* adapt W to network congestion.

By a *congestion control algorithm* we mean an algorithm that infers congestion and adjusts W during phase two of a TCP connection. Since data transfer can be in both directions such an algorithm can be executed in both directions. We will however fix the direction of a TCP connection in our mathematical model so we do not take into account the interaction of data transfers in both directions between the same pair of end points. Even though TCP congestion control is source-based, a congestion control algorithm actually involves two components: a source algorithm that dynamically adjusts the sending rate (or window size) in response to congestion in its path, and a link algorithm that updates, implicitly or explicitly, a congestion measure and sends it back, implicitly or explicitly, to sources that use that link.

On the current Internet, the source algorithm is carried out by TCP, and the link algorithm is carried out by (active) queue management (AQM) schemes such as DropTail or RED in, e.g., routers. Different protocols use different metrics to measure congestion, e.g., as we will see below, TCP Reno and its variants use loss probability as a congestion measure, and TCP Vegas and FAST use queueing delay as a congestion measure. Both are implicitly updated at the links and implicitly fed back through end-to-end loss or delay measurements at the sources. The equilibrium and dynamics of the network depend on the TCP/AQM protocol pair.

We now summarize some of the classical TCP protocols (Section 1.2.2) and AQM protocols (Section 1.2.3). We then derive mathematical models of these protocols in Section 1.3. These models will be used in the rest of the book to illustrate various methods to analyze congestion control algorithms.

1.2.2 TCP ALGORITHMS

TCP Tahoe and Reno

The first congestion control algorithm on the Internet was implemented by Jacobson in Tahoe (1988) and Reno (1990) versions of TCP. The window adjustment algorithm is based on an idea of Jain, Ramakrishnan and Chiu that a source should gently probe the network for spare capacity by linearly increasing its window and exponentially reduce its window when congestion is detected. This is called Additive Increase Multiplicative Decrease (AIMD). Congestion is detected when the source detects a packet loss.

Specifically the Tahoe protocol works as follows. A connection starts cautiously with a small window size of one packet and the source increments its window size by one every time it receives an ACK. This roughly doubles the window size every round-trip time and is called the `slow-start` phase. When the window size reaches a threshold the source enters the `congestion avoidance` phase where it increases its window size by the reciprocal of the current window size every time it receives an acknowledgment. This increases the window size by one packet in each RTT and is referred to as additive increase. The threshold, called the slow-start threshold `ssthreshold`, that determines the transition from `slow-start` to `congestion avoidance` is meant to indicate the available capacity in the network and is adjusted each time a packet loss is detected. On detecting a loss the source sets `ssthreshold` to half of the current

window size, retransmits the lost packet, and re-enters `slow-start` by resetting its window size to one packet.

A packet is deemed lost in one of two ways by TCP Tahoe. The first is if the sender does not receive its ACK within a pre-specified time called a timeout period. The second is if the sender receives three duplicate ACKs. Suppose the sender sends packets 0, 1, 2, 3, 4 back to back. Packet 0 is received correctly, packet 1 is lost, and packets 2, 3, 4 are received correctly. The receiver sends an ACK for packet 0. Each packet 2, 3, 4 that arrives at the receiver triggers a duplicate ACK for packet 0, indicating that the receiver is expecting packet 1. When these three duplicate ACKs arrive at the sender, packet 1 is deemed lost. Often, especially when window size is large, a packet loss is detected through duplicate ACKs much sooner than a timeout.

Two refinements were subsequently implemented in TCP Reno to recover from packet losses more efficiently. Call the time from detecting a loss (through three duplicate ACKs) to receiving the ACK for the retransmitted packet the `fast retransmit/fast recover (fr/fr)` phase. In TCP Tahoe the window size is frozen in the `fr/fr` phase. This means that a new packet can be transmitted only a round-trip time later. Moreover the "pipe" from the source to the destination is cleared when the retransmitted packet reaches the receiver and some of the routers in the path may become idle during this period, resulting in a loss of efficiency. The first refinement allows a Reno source to temporarily increment its window size by one on receiving each duplicate ACK while it is in the `fr/fr` phase. The rationale is that each duplicate ACK signals that a packet has left the network. When the window size becomes larger than the number of outstanding packets, a *new* packet can be transmitted in the `fr/fr` phase while the source is waiting for a (nonduplicate) ACK for the retransmitted packet. The second refinement essentially sets the window size at the end of the `fr/fr` phase to half of the window size when `fr/fr` starts and then enters `congestion avoidance` directly. Hence `slow-start` is entered only rarely in TCP Reno when the connection first starts and when a loss is detected by a timeout rather than three duplicate ACKs.

The pseudocode for the `congestion avoidance` phase of Reno is shown in Figure 1.3a and the resulting window trajectory in Figure 1.3b.

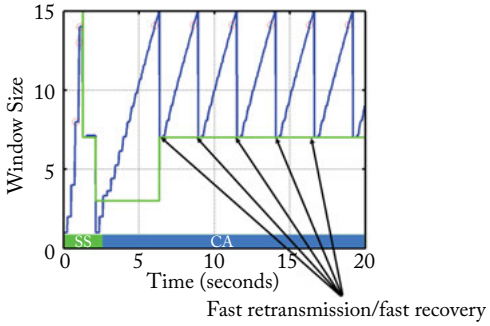
TCP Vegas

TCP Vegas improves upon TCP Reno through three main techniques. The first is a modified retransmission mechanism where timeout is checked on receiving the first duplicate ACK, rather than waiting for the third duplicate ACK (as Reno would), and results in a more timely detection of loss when the clock resolution was low back in the early 1990s (e.g., 500 ms). The second technique is a more prudent way to grow the window size during the initial use of `slow-start` when a connection starts up and it results in fewer losses.

The third technique is a new congestion avoidance algorithm that corrects the oscillatory behavior of Reno. The pseudocode is shown in Figure 1.4a and the resulting window trajectory in Figure 1.4b. Here `W` is the current window size, `RTT` is the current round-trip time, $\texttt{RTT}_{\texttt{min}}$ is the

```
for every ACK {
    W += 1/W          (AI)
}
for every loss {
    W = W/2           (MD)
}
```

(a) Pseudocode

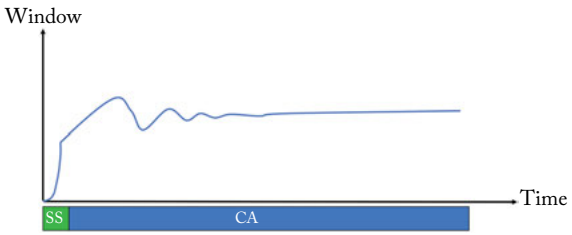


(b) Window trajectory

Figure 1.3: Congestion avoidance behavior of TCP Reno.

```
for every ACK
{
    if W/RTT_min - W/RTT < α then W += 1/W
    if W/RTT_min - W/RTT > β then W -= 1/W
}
for every loss {
    W = W/2
}
```

(a) Pseudocode



(b) Window trajectory

Figure 1.4: Congestion avoidance behavior of TCP Vegas.

minimum round-trip time observed so far, and $\alpha < \beta$ are protocol parameters in packets/second.

To understand the pseudocode, consider a source i. The round-trip time $T_i(t)$ a packet of source i experiences is the sum of a fixed component d_i and a variable component $q_i(t)$, i.e., $T_i(t) = d_i + q_i(t)$. The fixed component d_i, we call round-trip propagation delay, accounts for signal propagation time and fixed processing time along its path. The variable component $q_i(t)$ accounts for queueing delay at routers in its path. The quantity $\texttt{RTT}_{\texttt{min}}$ $(\min_t T_i(t))$ is an estimate of a source's round-trip propagation delay d_i. Let $w_i(t)$ be source i's window size, $x_i(t)$ be its sending rate, and d_i be its round-trip propagation delay. Then the pseudocode in Figure 1.4a increments or decrements the window size by 1 packet according as (multiplying both sides of the conditionals by d_i)

$$w_i(t) - x_i(t)\, d_i < \alpha_i\, d_i \quad \text{or} \quad w_i(t) - x_i(t)\, d_i > \beta_i\, d_i.$$

The quantity $x_i(t)d_i$ represents the number of source i's packets propagating in the communication channel (e.g., fiber optic cables) from source to destination. Hence the quantity $w_i(t) - x_i(t)d_i$ represents the number of i's packets queued at some routers in i's path when there are $w_i(t)$ outstanding packets. A Vegas source estimates the number of its own packets buffered in the path and tries to keep this number between $\alpha := \alpha_i d_i$ (originally 1 packet) and $\beta := \beta_i d_i$ (originally 3 packets) by adjusting its window size. The window size is incremented or decremented by approximately 1 packet in each round-trip time according as the current estimate is less than α or greater than β. Otherwise the window size is unchanged. The rationale is that each source should maintain a small number of its own packets in the pipe to take advantage of extra capacity when it becomes available.

Another interpretation of Vegas observes that

$$w_i(t) - x_i(t)d_i \quad = \quad x_i(t)(T_i(t) - d_i) \quad = \quad x_i(t)q_i(t)$$

where $q_i(t)$ is the round-trip *queueing* delay. Then the conditional in the pseudocode of Vegas becomes

$$x_i(t) \;<\; \alpha_i \frac{d_i}{q_i(t)} \quad \text{or} \quad x_i(t) \;>\; \beta_i \frac{d_i}{q_i(t)},$$

i.e., a Vegas source sets its rate to be proportional to the ratio of its round-trip propagation delay to queueing delay, the proportionality constant being between α_i and β_i. The more congested its path is, the higher the queueing delay and hence the lower the rate.

TCP FAST

TCP FAST can be thought of as a high-speed version of Vegas. The pseudocode for the congestion avoidance phase is shown in Figure 1.5. Both use queueing delay as the measure of con-

```
periodically
{
  W = γ( (RTT_min / RTT) W + α ) + (1−γ)W
}
```

Figure 1.5: The pseudocode of TCP FAST congestion avoidance agorithm.

gestion (price) and both have the same equilibrium point. Vegas works well when the network is slow or small, but its response is too sluggish in high-speed long-distance networks because, regardless of how far the network state is from its equilibrium, Vegas adjusts the window by the same amount (one packet per RTT). In contrast the size of the window adjustment in FAST is proportional to the distance of the network state from its equilibrium. It converges rapidly toward the equilibrium when it is far away and smooths into the equilibrium when it is close. See Section 1.3 for more details.

1.2.3 AQM ALGORITHMS

DropTail

Congestion control on the Internet is still predominantly source-based in that the link algorithm is implicit. A link (router) simply drops a packet that arrives at a full buffer. This is called DropTail (or Tail Drop) and the implicit link algorithm is carried out by the queue process. The congestion measure it updates depends on the TCP algorithm.

For TCP Reno and its variants, the congestion measure is packet loss probability. The end-to-end loss probability is observed at the source and is a measure of congestion on the end-to-end path. For TCP Vegas and FAST the congestion measure turns out to be link queueing delay when first-in-first-out service discipline is used. The congestion measure of a path is the sum of queueing delays at all constituent links.

RED

Random Early Detection (RED) is an alternative way to generate packet loss as a congestion measure. Instead of dropping only at a full buffer, RED maintains an exponentially weighted queue length and drops (or marks) packets with a probability that increases with the weighted queue length. When the weighted queue length is less than a minimum threshold no arrival packets are dropped. When it exceeds a maximum threshold all packets are dropped. When it is in between, a packet is dropped with a probability that is a piecewise linear and increasing function of the weighted queue length. See Figure 1.6a.

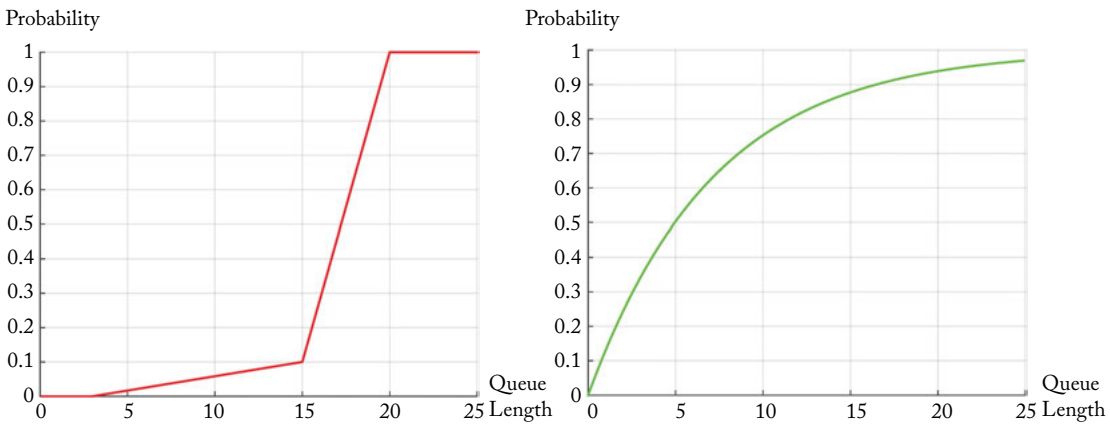


Figure 1.6: Loss probabilities of RED and REM as functions of (weighted) queue length.

REM

Random Exponential Marking (REM) consists of two simple ideas. First, in RED, the link price (loss probability) $p_l(t)$ depends on the queue length $b_l(t)$. As the number of flows sharing the link increases the steady-state price and hence queue length must also increase, leading to large delay. REM in contrast maintains a steady-state queue length around a target regardless of the number of flows sharing the link, thus decoupling the steady-state link delay from its congestion price. It achieves this by adjusting the link price $p_l(t)$ according to

$$\begin{aligned} \dot{p}_l &= \left(\alpha_l \left(y_l(t) - c_l\right) + \beta_l \left(b_l(t) - b_l^0\right)\right)^+_{p_l(t)} \\ \dot{b}_l &= \left(y_l(t) - c_l\right)^+_{b_l(t)} \end{aligned}$$

where

$$(a)^+_b := \begin{cases} a & \text{if } a > 0 \text{ or } b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $y_l(t)$ and c_l are respectively the input rate and link capacity at link l, and $b_l(t)$ and $b_l^0 \geq 0$ are respectively the queue length and its target at link l. Hence $y_l(t) - c_l$ is a rate mismatch and $b_l(t) - b_l^0$ is a queue mismatch. REM increases $p_l(t)$ if the weighted sum of the rate mismatch and queue mismatch is positive, and decreases it otherwise. In equilibrium where $\dot{p} = 0$, REM matches rate and queue length to their target values:

$$y_l^* = c_l \quad \text{and} \quad b_l^* = b_l^0$$

unless l is not a bottleneck link in which case $p_l = 0$, $y_l^* \leq c_l$ and $b_l^* = 0$.

The second idea of REM is a novel way to convey the link prices $p_l(t)$ to the sources through loss probability. Specifically REM explicitly embeds the *sum* of link prices along a path into the end-to-end loss probability that can be observed at the source, as follows. Like RED, REM drops a packet that arrives at a link with a probability independently of all other packets. Unlike RED whose loss probability is a piecewise linear function of the weighted queue length, the loss probability $\tilde{p}_l(t)$ of REM however is exponential in the link price (see Figure 1.6b):

$$\tilde{p}_l(t) := 1 - \phi^{-p_l(t)}.$$

Then the end-to-end probability $\hat{q}_i(t)$ that a packet of source i is lost in its path is exponential in the end-to-end price $q_i(t) = \sum_l R_{li} p_l(t)$:

$$\hat{q}_i(t) := 1 - \prod_l \left(1 - \tilde{p}_l(t)\right)^{R_{li}} = 1 - \phi^{-q_i(t)}.$$

Hence if source i estimates its end-to-end loss probability $\hat{q}_i(t)$ it can compute the end-to-end price $q_i(t)$ as

$$q_i(t) = -\log_\phi \left(1 - \hat{q}_i(t)\right).$$

As we will see below, the end-to-end price $q_i(t)$ can be useful for general TCP algorithms.

1.3 MODELS OF CLASSICAL ALGORITHMS

Even though TCP is a source-based mechanism, we must consider congestion control as a feedback system where source rates $x(t)$ interact with link prices $p(t)$. Different protocols choose different algorithms to adapt $(x(t), p(t))$. In this section we present mathematical models of the protocol pairs Reno/RED, Vegas/DropTail, and FAST/DropTail. The model captures the `congestion avoidance` phase of these protocols.

1.3.1 RENO/RED

We only model the average behavior of the additive increase multiplicative decrease (AIMD) algorithm used to control the window size and does not differentiate between TCP Reno and its variants such as NewReno, SACK, etc. All these protocols (henceforth referred to as "Reno") increase the window by one packet every round-trip time if there is no loss in the round-trip time, and halve the window otherwise.

Let $w_i(t)$ be the window size of source i. Let T_i be the round-trip time (propagation plus queueing delay), which we assume to be constant. Let $x_i(t) := w_i(t)/T_i$ be the source rate at time t. The time unit is on the order of several round-trip times and source rate $x_i(t)$ should be interpreted as the average rate over this timescale. Dynamics smaller than the timescale of a round-trip time is not captured by the fluid model. Let $p_l(t)$ be the loss (or marking) probability at link l at time t. We make the key assumption that the end-to-end loss (or marking) probability $q_i(t)$ to which source algorithm reacts is the *sum* of link loss probabilities: $q_i(t) = \sum_l R_{li}\, p_l(t)$. This is reasonable when $p_l(t)$ are small, in which case

$$q_i(t) \;=\; 1 - \prod_l (1 - p_l(t))^{R_{li}} \;\simeq\; \sum_l R_{li}\, p_l(t).$$

Consider the pseudocode of AIMD in Figure 1.3a. In period t it transmits at rate $x_i(t)$ packets per unit time and receives (positive and negative) ACKs at approximately the same rate, assuming every packet is acknowledged. Hence on average source i receives $x_i(t)(1 - q_i(t))$ number of positive ACKs per unit time and each positive ACK increases the window $w_i(t)$ by $1/w_i(t)$. It receives on average $x_i(t)q_i(t)$ negative ACKs (losses) per unit time and each halves the window. Hence the net change to the window in period t is roughly

$$\dot{w}_i \;=\; x_i(t)(1 - q_i(t))\,\frac{1}{w_i(t)} \;-\; x_i(t)q_i(t)\,\frac{w_i(t)}{2}.$$

Since T_i are assumed to be constant the sending rate $x_i(t) := w_i(t)/T_i$ has the same dynamics as the window $w_i(t)$ except for a constant scaling. Hence we will model the dynamics of the sending rate directly, as:

$$\dot{x}_i \;=\; \left(\frac{1 - q_i(t)}{T_i^2} \;-\; \frac{1}{2}\, q_i(t)\, x_i^2(t)\right)^+_{x_i(t)} \tag{1.1a}$$

where

$$(a)_b^+ \quad := \quad \begin{cases} a & \text{if } a > 0 \text{ or } b > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The quadratic term in (1.1a) captures the property that, if the rate doubles, the multiplicative decrease occurs at twice the frequency with twice the amplitude. The operation $(\cdot)^+_{x_i(t)}$ ensures that $x_i(t)$ stays nonnegative, i.e., $\dot{x}_i$ stays 0 when $x_i(t) = 0$ and the quantity in the bracket is negative.

There are variants of this model. One is that the window increases deterministically by 1 every round-trip time. This modifies the additive increase term in (1.1a) into:

$$\dot{x}_i \quad = \quad \left(\frac{1}{T_i^2} - \frac{1}{2} q_i(t)\, x_i^2(t) \right)^+_{x_i(t)} . \tag{1.1b}$$

It is approximately the same as (1.1a) when the loss probabilities $p_l(t)$ are small.

Another variant is that, instead of halving the window on each negative acknowledgment, the window is halved once in each round-trip time that contains one or more negative acknowledgments. This modifies the multiplicative decrease term in (1.1a) into:

$$\dot{x}_i \quad = \quad \left(\frac{1 - q_i(t)}{T_i^2} - \frac{1}{2T_i} q_i(t)\, x_i(t) \right)^+_{x_i(t)} .$$

RED updates the queue length $b_l(t)$ according to:

$$\dot{b}_l \quad = \quad (y_l(t) - c_l)^+_{b_l(t)} . \tag{1.1c}$$

The queue length $b_l(t)$ is an internal variable and (a "gentle" version of) RED drops (or marks) a packet with a probability $p_l(t)$ that is a piecewise linear increasing function of $b_l(t)$ (see Figure 1.6a):

$$p_l(t) \quad = \quad \begin{cases} 0, & b_l(t) \le b_1 \\ \rho_1(b_l(t) - b_1), & b_1 \le b_l(t) \le b_2 \\ \rho_2(b_l(t) - b_2) + m, & b_2 \le b_l(t) \le b_3 \\ 1, & b_l(t) \ge b_3 \end{cases} \tag{1.1d}$$

where $m \in (0, 1)$ is the nonzero loss probability at the break point,

$$\rho_1 \quad = \quad \frac{m}{b_2 - b_1} \qquad \text{and} \qquad \rho_2 \quad = \quad \frac{1 - m}{b_3 - b_2}.$$

In summary the dynamical system that models Reno/RED consists of equations (1.1).[3]

[3] In practice, the loss probability in RED depends not directly on the instantaneous queue length $b_l(t)$, but on an exponentially weighted average $r_l(t)$, modeled by

$$\dot{r}_l \quad = \quad -\alpha_l (r_l(t) - b_l(t))$$

where $\alpha_l \in (0, 1)$ is an exponential weight. Then the loss probability $p_l(t)$ is given by (1.1d) with $b_l(t)$ replaced by $r_l(t)$.

The AQM model (1.1c)–(1.1d) contains internal variables $b_l(t)$ that are not observed at TCP sources. We can eliminate $b_l(t)$ and derive the dynamics of $p_l(t)$ from (1.1c) (see Figure 1.6a):

$$\dot{p}_l(t) \;=\; \begin{cases} \rho_1(y_l(t)-c_l)^{+}_{p_l(t)}, & 0 \le p_l(t) \le m \\ \rho_2(y_l(t)-c_l)^{p_l(t)-1}_{-}, & m < p_l(t) \le 1 \end{cases} \tag{1.2}$$

where

$$(a)^{+}_{b} \;:=\; \begin{cases} a & \text{if } a>0 \text{ or } b>0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad (a)^{b}_{-} \;:=\; \begin{cases} a & \text{if } a<0 \text{ or } b<0 \\ 0 & \text{otherwise.} \end{cases}$$

Then Reno/RED can be modeled by ordinary differential equations (1.1a) (1.2) without internal variables and algebraic equations. Most of our analysis will be on models of this type.

1.3.2 VEGAS/DROPTAIL

Consider the pseudocode in Figure 1.4a of the congestion avoidance mechanism of TCP Vegas. Let d_i denote the round-trip propagation delay of source i, $q_i(t)$ the round-trip queueing delay, $T_i(t) := d_i + q_i(t)$ the round-trip time, $w_i(t)$ the window size and $x_i(t) := w_i(t)/T_i(t)$ the sending rate of flow i. For simplicity assume $\alpha = \beta$. As discussed earlier the conditional in the pseudocode is

$$w_i(t) - x_i(t)\,d_i \;<\; \alpha_i\,d_i \quad \text{or} \quad w_i(t) - x_i(t)\,d_i \;>\; \alpha_i\,d_i.$$

The quantity $x_i(t)d_i$ represents the number of source i's packets propagating in the communication channel (e.g., fiber optic cables) from source to destination, and the quantity $w_i(t) - x_i(t)d_i$ represents the number of i's packets buffered at the routers in its path. A Vegas source estimates the number of its own packets buffered in the path and tries to keep this number at $\alpha := \alpha_i d_i$ packets by incrementing or decrementing its window size $w_i(t)$ by 1 packet per round-trip time. Since

$$w_i(t) - x_i(t)\,d_i \;=\; x_i(t)(T_i(t)-d_i) \;=\; x_i(t)\,q_i(t),$$

i.e., $x_i(t)q_i(t)$ is the number of i's own packets buffered in the queues in its path, we model the window dynamics as:

$$\dot{w}_i \;=\; \frac{1}{d_i + q_i(t)}\,\text{sign}\,(\alpha_i d_i \;-\; x_i(t)q_i(t))^{+}_{w_i(t)} \tag{1.3a}$$

$$x_i(t) \;=\; \frac{w_i(t)}{d_i + q_i(t)} \tag{1.3b}$$

where $\text{sign}(z)$ is -1 if $z<0$, 0 if $z=0$, and 1 if $z>0$. The operation $(\cdot)^{+}_{w_i(t)}$ ensures that $w_i(t) \ge 0$. Hence (1.3a) says that the window is adjusted by 1 packet per round-trip time by comparing

the number $x_i(t)q_i(t)$ of packets buffered in its path with the target $\alpha_i d_i$. In equilibrium each source i maintains $\alpha_i d_i$ packets in its path.

Hence Vegas uses as its congestion price the queueing delay at a link l

$$p_l(t) \;:=\; \frac{b_l(t)}{c_l}$$

where $b_l(t)$ is the queue length and evolves according to (1.1c). The price dynamics is therefore (dividing both sides of (1.1c) by c_l):

$$\dot{p}_l \;=\; \frac{1}{c_l}\,(y_l(t)-c_l)^+_{p_l(t)} \tag{1.3c}$$

where $y_l(t) := \sum_i R_{li}\,x_i(t)$. In summary the dynamical system that models Vegas/DropTail consists of equations (1.3).

The TCP model (1.3a)–(1.3b) contains algebraic equations. To reduce the model to ordinary differential equations, we can eliminate $x(t)$ by substituting (1.3b) into (1.3a)(1.3c) to obtain:

$$\begin{aligned}\dot{w}_i &= \frac{1}{d_i+q_i(t)}\,\mathrm{sign}\left(\alpha_i d_i - \frac{w_i(t)q_i(t)}{d_i+q_i(t)}\right)^+_{w_i(t)}\\ \dot{p}_l &= \frac{1}{c_l}\left(\sum_i \frac{R_{li}w_i(t)}{d_i+q_i(t)} - c_l\right)^+_{p_l(t)}.\end{aligned}$$

1.3.3 FAST/DROPTAIL

TCP Reno and Vegas were designed in the late 1980s and early 1990s when network capacities are much smaller and a large majority of network traffics are relatively local. The window adjustment by one packet per round-trip time was adequate then, but much too slow as networks scale up in capacity and geographical coverage. This has motivated a large effort in the 2000s to develop an understanding of the mathematical structure of congestion control and a theory-aided design approach where performance analysis was done before, as opposed to after, implementation and deployment. TCP FAST came out of that effort.

Even though the implementation of FAST is very different from that of Vegas (compare their pseudocodes in Figure 1.5 and Figure 1.4a), the FAST design was inspired by the underlying mathematical structure of Vegas. In particular FAST attains the same equilibrium of Vegas and therefore achieves proportional fairness (see Chapter 2). It speeds up the dynamics of Vegas by adjusting the window size by an amount proportional to the deviation of the queue length in the path from its target value, instead of by one packet per round-trip time as Vegas does in (1.3a).

Specifically consider the FAST pseudocode in Figure 1.5. It can be modeled by

$$\dot{w}_i = \gamma\,(\alpha_i - x_i(t)q_i(t))^+_{w_i(t)} \tag{1.4a}$$

$$x_i(t) = \frac{w_i(t)}{d_i + q_i(t)} \tag{1.4b}$$

where $q_i(t) := \sum_l R_{li}\, p_l(t)$. Like Vegas, FAST also uses as its congestion price the queueing delay at a link l whose dynamics is described by:

$$\dot{p}_l = \frac{1}{c_l}\,(y_l(t) - c_l)^+_{p_l(t)} \tag{1.4c}$$

where $y_l(t) := \sum_i R_{li}\, x_i(t)$. In summary the dynamical system that models FAST/DropTail consists of equations (1.4).

FAST can be interpreted as a high-speed version of Vegas in the following sense. Compare (1.3a) for Vegas and (1.4a) for FAST (replacing $\alpha_i d_i$ in (1.3a) by α_i). While Vegas increments/decrements the window by 1 packet depending on the sign of $\alpha_i - x_i(t)q_i(t)$, FAST updates the window by an amount proportional to $\alpha_i - x_i(t)q_i(t)$. For high-speed long-distance networks the magnitude of $\alpha_i - x_i(t)q_i(t)$ can be big during transient, in which case FAST closes the gap faster than Vegas.

The FAST model (1.4a)–(1.4b) contains algebraic equations. As for Vegas, we can eliminate $x(t)$ by substituting (1.4b) into (1.4a)(1.4c) to obtain a model consisting of ordinary differential equations:

$$\dot{w}_i = \gamma\left(\alpha_i - \frac{w_i(t)q_i(t)}{d_i + q_i(t)}\right)^+_{w_i(t)} \tag{1.5a}$$

$$\dot{p}_l = \frac{1}{c_l}\,(y_l(t) - c_l)^+_{p_l(t)}\,. \tag{1.5b}$$

An alternative model is to replace the dynamic link model (1.5b) by a static model where the link queueing delay vector $p(t)$ is determined implicitly by the window sizes $w(t)$:

$$\dot{w}_i = \gamma\left(\alpha_i - \frac{w_i(t)q_i(t)}{d_i + q_i(t)}\right)^+_{w_i(t)} \tag{1.6a}$$

$$\sum_i R_{li}\,\frac{w_i(t)}{d_i + q_i(t)} \begin{cases} = & c_l \quad \text{if } p_l(t) > 0 \\ \le & c_l \quad \text{if } p_l(t) = 0. \end{cases} \tag{1.6b}$$

It can be proved that the model (1.6) is well defined: given $w(t)$, there is a unique queueing delay vector $p(t)$ that satisfies (1.6b) provided R has full row rank.

Another alternative model is to replace the dynamic source model (1.5a) by a static model (obtained by setting $\dot{w} = 0$ in (1.4a) and using $x_i(t) = w_i(t)/(d_i + q_i(t))$):

$$\dot{p}_l = \frac{1}{c_l}(y_l(t) - c_l)^+_{p_l(t)} \tag{1.7a}$$

$$x_i(t) = \frac{\alpha_i}{q_i(t)}. \tag{1.7b}$$

1.4 A GENERAL SETUP

1.4.1 THE BASIC MODELS

We have seen in Section 1.3 that one can model a physical system in multiple ways, each making different assumptions. All the models there take the form of differential algebraic equations:

$$\begin{aligned}
\dot{\tilde{x}}_i &= \tilde{f}_i(x_i(t), \tilde{x}_i(t), q_i(t)), & i \in N \\
x_i(t) &= f(x_i(t), \tilde{x}_i(t), q_i(t)), & i \in N \\
\dot{\tilde{p}}_l &= \tilde{g}_l(y_l(t), p_l(t), \tilde{p}_l(t)), & l \in L \\
p_l(t) &= g_l(y_l(t), p_l(t), \tilde{p}_l(t)), & l \in L
\end{aligned}$$

where $(\tilde{x}(t), \tilde{p}(t))$ are (generally vector) internal variables and $(x(t), p(t))$ are source rates and link prices. The most important feature of this model is its decentralized nature: user i adapts its rate $x_i(t)$ based only on local information $(x_i(t), \tilde{x}_i(t))$ and the locally observed congestion measure $q_i(t)$, and link l adapts its price $p_l(t)$ based only on local information $(y_l(t), p_l(t), \tilde{p}_l(t))$. We represent this set of differential algebraic equations in vector form as

$$\begin{aligned}
\dot{\tilde{x}} &= \tilde{f}(x(t), \tilde{x}(t), q(t)) \\
x &= f(x(t), \tilde{x}(t), q(t)) \\
\dot{\tilde{p}} &= \tilde{g}(y(t), p(t), \tilde{p}(t)) \\
p &= g(y(t), p(t), \tilde{p}(t))
\end{aligned}$$

where $q(t) = R^T p(t)$, $y(t) = R\,x(t)$. As we have seen in Section 1.3 above, while including internal variables $(\tilde{x}(t), \tilde{p}(t))$ is convenient for modeling, it is usually easy for practical protocols to eliminate these internal variables to obtain a model involving only $(x(t), p(t))$. Such a model is usually more convenient for analysis.

Our analysis will therefore focus on the following three special cases without the internal variables.

1. *Primal algorithms.* This is the class of algorithms that have dynamics (memory) in the source rates (or window sizes) but not in the link prices:

$$\dot{x} = f(x(t), q(t)) \tag{1.8a}$$

$$p = g(y(t), p(t)). \tag{1.8b}$$

2. *Dual algorithms.* This is the class of algorithms that have dynamics in the link prices but not in the source rates (or window sizes):

$$\dot{p} = g(y(t), p(t)) \tag{1.9a}$$
$$x = f(x(t), q(t)). \tag{1.9b}$$

3. *Primal-dual algorithms.* This is the class of algorithms with dynamics in both source rates and link prices:

$$\dot{x} = f(x(t), q(t)) \tag{1.10a}$$
$$\dot{p} = g(y(t), p(t)). \tag{1.10b}$$

We will refer to any of (1.8), (1.9), (1.10) together with

$$q(t) = R^T p(t) \quad \text{and} \quad y(t) = R\,x(t) \tag{1.11}$$

as the *basic model*. The structure of these models is illustrated in Figure 1.7. For example, FAST/DropTail can be modeled as either a primal, a dual, or a primal-dual algorithm; see Section 1.3.3.

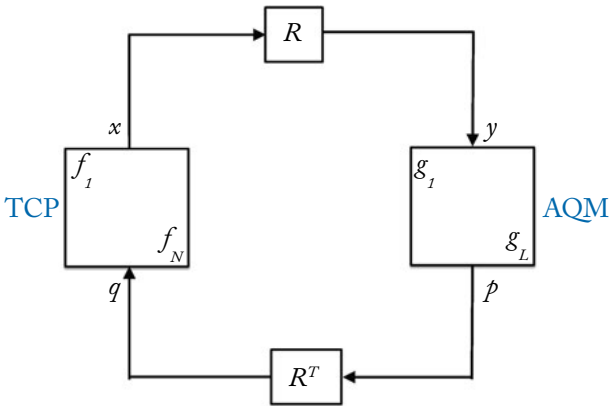


Figure 1.7: The basic model: a multi-source multi-link network.

For primal algorithms, we can often express $p(t)$ in terms of $x(t)$ using (1.8b) because the Jacobian $\frac{\partial g}{\partial p} - I$ is often nonsingular for TCP/AQM models. Substituting $p(t) = p(x(t))$ as a function of $x(t)$ into (1.8a)(1.11) then reduces the primal algorithm into a set of ODEs in $x(t)$. Similarly for dual algorithms, we can often express $x(t)$ in terms of $p(t)$ using (1.9b) and reduce the dual algorithm into a set of ODEs in $p(t)$ (substitute $x(p(t))$ into (1.9a)(1.11)). Therefore

all three basic models can be described by ODEs. Hence, we often cast our discussion in terms of primal-dual algorithms (1.10) as the prototypical model, knowing that it applies to all three models.

Since the source rates $x(t)$ and prices $p(t)$ are nonnegative, hidden in the notation of the functions (f, g) is the projection to the nonnegative quadrant. We sometimes make this explicit by writing, instead of (1.10):

$$\begin{aligned} \dot{x} &= (f(x(t), q(t)))^+_{x(t)} \\ \dot{p} &= (g(y(t), p(t)))^+_{p(t)} \end{aligned}$$

where for any vectors $a, b \in \mathbb{R}^n$, the vector $(a)^+_b$ is defined componentwise, i.e., for all $i = 1, \ldots, n$,

$$\left[(a)^+_b\right]_i := (a_i)^+_{b_i} := \begin{cases} a_i & \text{if } a_i > 0 \text{ or } b_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1.12}$$

As we will see later, the projection on the right-hand side of the differential equations mostly preserves the existence, uniqueness, equilibrium and stability properties of the unprojected system, but the analysis is considerably more complicated because of the discontinuity introduced by the projection. A notable feature of this book is the careful treatment of the projected dynamics.

We are interested in three questions:

1. Under what condition does the basic model (1.10) have a unique solution $(x(t), p(t), t \geq 0)$, given any initial point $(x(0), p(0))$?

2. Does equilibrium of (1.10) exist? Is it unique? How do we characterize it?

3. Is an equilibrium of (1.10) stable?

We answer question 1 in this Chapter, question 2 in Chapter 2, and question 3 in Chapters 3 through 6.

1.4.2 LIMITATIONS AND EXTENSIONS

We now discuss some limitations of our fluid models and various extensions.

1. **No feedback delay.** The basic models ignore feedback delay, i.e., they assume that a change in a link price $p_l(t)$ is instantly observed at all the sources that use that link, and a change in a source rate $x_i(t)$ affects the aggregate flow rates at all the links in the path of the source. In reality, of course, there is feedback delay due to signal propagation between links and sources, and due to packet queueing and processing in the network. Feedback delay can

be modeled by replacing (1.11) with

$$q_i(t) = \sum_l R_{li}\, p_l(t - \tau^b_{li}), \qquad i \in N$$
$$y_l(t) = \sum_i R_{li}\, x_i(t - \tau^f_{li}), \qquad l \in L.$$

Here τ^b_{li} models the backward delay from link l to source i and τ^f_{li} models the forward delays from source i to link l, both assumed to be constant. It is the time it takes for a change in the price at link l to affect the rate at source i, and the time it takes for a change in the rate at source i to affect the price at link l, respectively.

We will use this delayed model in Chapter 6 to study linear stability in the presence of feedback delay.

2. **Window control.** A window-based control does not adapt its sending rate $x_i(t)$ directly, but rather its window size $w_i(t)$. This is modeled using the relationship between $x_i(t)$ and $w_i(t)$

$$x_i(t) \quad = \quad \frac{w_i(t)}{T_i(t)} \quad = \quad \frac{w_i(t)}{d_i + q_i(t)}$$

as in (1.3) for Vegas and in (1.4) for FAST. For convenience however we often assume the round-trip time T_i to be constant and model the dynamics of $x_i(t)$ directly as in (1.1) for Reno.

3. **Round-trip timescale.** The basic models ignore flow arrivals and departures and fix set of flows and their routing matrix. They focus on the network dynamics at the round-trip timescale. This amounts to ignoring short-duration TCP flows.

4. **Heterogeneous protocols.** The basic models assume that, while different sources i can adapt their rates using different algorithms f_i, they all react to the same type of congestion prices $p_l(t)$. For instance all sources react to packet loss probabilities in their paths as in Reno, or all sources react to packet delay in their paths as in Vegas or FAST. If sources that use different types of congestion prices (e.g., Reno and Vegas) share the same network, the network behavior can be much more complicated. For instance while a network of homogeneous sources typically has a unique equilibrium point, a network of heterogeneous sources may not.

5. **Deterministic fluid.** We model traffic as a deterministic fluid and ignore randomness in, e.g., packet processing and queueing times, packet arrival process, or flow arrivals and departures.

6. **Queue output process.** The basic models assume a flow maintains its rate along its path so that every link in its path sees the *source* rate. In reality however as a flow goes through a queue, its output rate depends on the buffering process and is generally different from its input rate, except when the network is in a steady state where all queues are stabilized.

1.5 SOLUTION OF THE BASIC MODELS

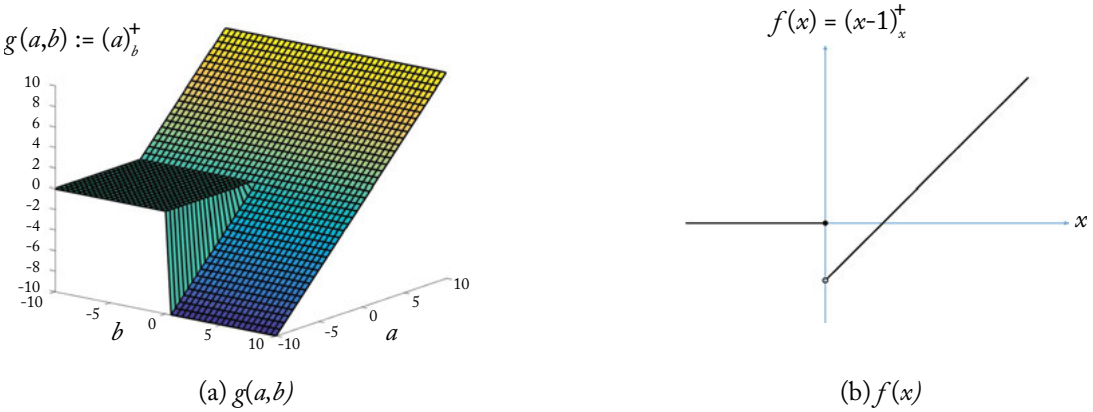
The ODE model (1.10) describes the protocol action in the congestion avoidance phase. In this section we discuss conditions on the TCP/AQM models (f, g) that guarantee the existence of a unique solution trajectory $(x(t), p(t), t \geq 0)$ given any initial point $(x(0), p(0))$. We first summarize sufficient conditions for the existence and uniqueness of the solution to general ODE systems. We then illustrate how to use these results to prove that the TCP/AQM models of Section 1.3 are well defined.

Consider a system of projected ordinary differential equations:

$$\dot{x} = (f(x(t)))^+_{x(t)}, \qquad t \geq 0 \tag{1.13a}$$

$$x(0) = x_0 \geq 0 \quad \text{given} \tag{1.13b}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$, and the projection $(f(x))^+_x$ is defined in (1.12).[4] Even though the projection function $g(a, b) := (a)^+_b$, a, b in $\mathbb{R}$, is continuous in a given any b, it is discontinuous in b for $a < 0$. To visualize the effect of projection see Figure 1.8. Therefore the right-hand side of (1.13a) is discontinuous in $x(t)$ even if f is continuous. It turns out that this is a simple type



(a) $g(a,b)$ (b) $f(x)$

Figure 1.8: (a) The function $g(a, b) = (a)^+_b$ is discontinuous along the line $b = 0$ for $a < 0$. (b) The function $f(x) := (x - 1)^+_x$ is discontinuous at $x = 0$.

[4]We abuse notation to use f to denote either a generic function or a TCP algorithm and x to denote either a generic variable or source rates, depending on the context.