Alan Holt

Network Performance Analysis

Using the J Programming Language

Springer

Alan Holt

Network Performance Analysis

Using the J Programming Language

Alan Holt, PhD
agholt@gmail.com

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2007936013

Printed on acid-free paper

ISBN 978-1-84628-822-7 e-ISBN 978-1-84628-823-4

9 8 7 6 5 4 3 2 1

Springer Science+Business Media

springer.com

1

Introduction

For reasons of tractibility, classical queuing theory assumes that the properties of network traffic (arrival rates, service times) include the Markovian property. While the Markovian assumption is valid for telephony networks, it has been widely reported that, for data networks, traffic is fractal in nature [6, 13, 16, 37]. One way to represent a traffic flow is as a stochastic process. Modelling traffic as Markovian stochastic processes is attractive because flows can be characterised by only a small number of parameters. Models of non-Markovian processes, however, are more complex and analytical results are difficult to derive. Markovian traffic can be clearly distinguished from Internet traffic patterns which exhibit properties of *long-range dependence* and *self-similarity*. That is, traffic is bursty over a wide range of time scales. Backlog and delay predictions are frequently underestimated if the Markovian property is assumed when traffic is actually long-range dependent (and self-similar) [16, 28].

A number of methods for generating fractal-like traffic processes have been put forward. Examples include Fractional ARIMA processes, Fractional Brownian motion, chaotic maps and the superposition of heavy-tailed on/off sources. Network performance can then be investigated using simulation techniques. Some of these fractal traffic generating techniques are described in this book. We use J to simulate network systems and analyse their performance under certain traffic conditions.

An alternative approach to simulation is *network calculus*. Network calculus is a recent development where the exact properties of flows are unknown. The mathematical foundations are based upon *min-plus* algebra, whereby the *addition* and *multiplication* operators of conventional algebra are exchanged for *minimum* and *plus* operators. Instead of representing a traffic flow as stochastic process, a flow is characterised by an *envelope* function. Service curves can also be expressed in this way. Performance bounds can be derived through the application of min-plus algebraic methods on flow arrival curves and network system service curves.

Another approach is to consider *closed-loop* methods of modelling networks. Internet transport protocol such TCP (Transmission Control Protocol), and more recently DCCP (Datagram Congestion Control Protocol [34]), adjust their transmission rates

according to the congestion state of the network [29]. TCP sends data a "window" at a time. TCP initially probes the network for bandwidth by increasing its window size each time it receives an acknowledgment. TCP's transmission rate is regulated by the acknowledgment round-trip time. In the event of congestion, TCP reduces its window size (and thus its transmission rate) and begins probing the network for bandwidth once again. Congestion control algorithms can be modelled as dynamical feedback systems, and flow patterns are governed by transport protocols reacting to network events.

The aim of this book is to present network performance analysis techniques that do not rely on the Markovian assumption. Topics in network performance analysis are presented with the aid of the J programming language. J is rich in mathematical functionality, which makes it an ideal tool for analytical methods. Throughout the book a practical approach is favoured. Functions in J are developed in order to demonstrate mathematical concepts, this enables the reader to explore the principles behind network performance analysis.

The topics covered in this book are motivated by the author's own research and industrial experience. The book outline is as follows. Chapters 2 and 3 provide an introduction to the J programming language. Chapter 2 introduces the basic concepts such as the data types and built-in J functions. Chapter 3 covers more advanced topics, where the focus is programming in J.

Network calculus is introduced in Chapter 4. We demonstrate how arrival and service curves can be expressed as J functions. We show how to derive upper bounds for backlog, delay and output flow. Effective bandwidth and equivalent capacity techniques are introduced as a means of deriving the network resource requirements for deterministic QoS bounds.

Chapter 5 focuses on statistical analysis and stochastic processes. The concepts of short-range and long-range dependence are introduced and we implement models for generating time series with these properties. We introduce Autoregressive (AR) and Moving Average (MA) models for generating short-range dependent time series. Fractional Autoregressive Integrated Moving Average (FARIMA) models are presented for generating time series with long-range dependence and self-similarity.

In Chapters 6 and 7, we show how to simulate traffic with both short-range and long-range dependence properties. In Chapter 6, we simulate traffic with discrete on/off models using various random number generators to generate traffic with the desired correlation properties. In Chapter 7, we show how chaotic maps can be used to generate simulated traffic.

ATM QoS is covered in Chapter 8. Leaky bucket and virtual scheduling algorithms are developed in J. We show how cell conformance can be analysed using these algorithms by running them on simulated traffic from continuous on/off models.

Chapter 9 examines Internet congestion control. We use J to build binomial congestion control algorithms and explore the parameters that govern congestion window increase and decrease.

1.1 Quality of Service

The Internet was principally designed to offer a *best-effort* service [12]. While the network makes a sincere attempt to transmit packets, there are no guarantees with regard to reliable or timely delivery. Packets may be delayed, delivered out of order or dropped. The end-to-end protocols (such as TCP) are given the responsibility of recovering from these events. Packets incur delays during transmission due to link speeds, and propagation delays. Delays are also incurred by packets waiting in buffer queues. Memory buffers are used to resolve contention issues when two (or more) packets arrive simultaneously at a communications link. A packet's waiting time is determined by the number of packets that are scheduled ahead of it; thus waiting times increase with traffic volume. Furthermore, if traffic volumes are excessive, contention for buffer memory arises, which is resolved by dropping packets.

Traffic management methods (and associated policies) can help to alleviate the trade-off between QoS and network optimality. Traffic management is a complex technical issue, but it is also an economical, political and ethical one. Like any limited resource, the allocation of network capacity can be somewhat controversial. Traffic management policies are selected either to implement a *neutral* network, or a differentiated one. In a neutral network, there is parity between users and services in terms of how the network treats their packets. A neutral network however, does not necessarily support fairness. Due to the flat-rate pricing policy of the Internet, heavy users do not incur any additional financial cost over light users. Yet the burden of congestion is borne by all.

In a differentiated network, there is discrimination between users and/or services. The network provider controls the allocation of resources, electing to give some users a better than best-effort service (and others a less than best-effort). Better than best-effort services are covered extensively in the literature ([22, 73], for example). Certain classes of user require either priority allocation or an assurance of a minimum allocation of resources during periods of congestion. Resources are allocated according to the particular QoS requirements of the user's application. The aim is either to guarantee or at least optimise QoS metrics, such as delay, jitter, throughput or loss.

Less than best-effort (LBE), services are not so clearly defined. Typically, best-effort is considered the lowest level of service [23]; that is, high-priority users are allocated resources according to their needs, and any remaining resources are allocated to lower-priority users. One may argue, however, that if a particular set of users is receiving a preferential service relative to another, then any "best-effort" commitment to the low-priority users is not being met. The service they are receiving, therefore, is almost certainly *less than* best-effort.

Nevertheless, the literature maintains that a best-effort service can be supported despite the differentiation in flows and preferential resource allocation. Furthermore, low-cost LBE services may be offered alongside best-effort and high-priority services to users of applications that are tolerant to high loss rates, delay and jitter

[19, 25]. During periods of congestion, LBE packets are delayed or dropped in preference to best-effort or high-priority traffic. Applications with LBE delivery can make use of off-peak periods when network resources are underutilised. This enables network providers to use otherwise spare capacity, without affecting best-effort or higher priority traffic.

LBE services however, extend beyond low-cost service offerings by providers as a means of selling available capacity on under-utilised links. There have been cases where network providers have attempted to "discourage" the use of certain services over their network. Voice over IP (VoIP) is a typical example. There have been reports of provider discrimination against VoIP services. At the most basic level, providers simply *block* IP ports such that VoIP traffic cannot pass between the sender and receiver. A more subtle form of discouraging VoIP applications is delay packets such that the QoS is degraded beyond a usable level [50]. VoIP users may be unaware they are being discriminated against, believing that the low voice quality is merely a by-product of a best-effort service, when in actual fact the provider is inflicting a less-than best-effort service on them.

Less-than best-effort services can also arise from network provider business practices. The Internet is no longer a network formed through the "cooperative anarchy" [65] of government agencies and academic institutions. The infrastructure is divided amongst many competitive Internet Service Providers (ISPs). Tier 2 ISPs "buy" routing tables wholesale from transit providers; thus traffic is exchanged between ISPs via transit networks. Rival ISPs, however, may elect to cooperate and exchange traffic through a mutual *peering* arrangement [48]. The benefits of peering are (potentially) two-fold. Firstly, ISPs in a peering arrangement reduce their transit costs. Secondly, by eliminating the transit hop and taking a more direct route, traffic latency is reduced. A peering arrangement between two ISPs may be attractive to one ISP but not the other. Without the consent of both parties, peering will not occur. This has given rise to a number of dubious business practices, whereby one ISP tries to encourage other to peer with it. For example, if ISP B receives traffic from ISP A through some prior peering arrangement with another ISP, then ISP A could, using policy routing, forward its traffic to ISP B through a transit provider. Furthermore, ISP A may compound ISP B's transit costs by "replaying" traffic using a traffic generator [49]. Such practices do not constitute a sincere attempt to deliver packets, and thus packet delivery is *not* best-effort.

Textbooks describe the Internet Protocol as a best-effort packet delivery mechanism [12] and Internet commentators talk of "retaining" a neutral network [9]. According to Sandvig, however, "the Internet isn't neutral now" [60]. Nor has it been for a long time. As the Internet grew in the 1990s, it suffered from the *tragedy of the commons* [46] and was subject to "overgrazing." Furthermore, the Internet carries content that some find undesirable. Providers adopt differentiated traffic management policies in order to address these issues. Consider the following scenarios:

- According to reports, 75 to 95 percent [18, 74] of electronic mail is unsolicited (SPAM). Providers go to a great deal of trouble trying to distinguish SPAM from legitimate e-mail so that it can be purged from the network.
- Users vary considerably when it comes to their consumption of network resources. In order to prevent a small number of heavy users causing congestion, some ISPs impose *caps* on their subscribers. Subscribers that exceed their caps are either blocked or charged extra.
- The Internet has undergone a revolution in the People's Republic of China, yet it is a prime example of how "the net can be developed and strangled all at once" [67]. Content providers, in accordance with legislation, cooperate with the Chinese government to censor traffic that carries information deemed to be "sensitive."

Discriminatory practices on the Internet are a reality; it is merely a question of which practices one finds acceptable. As Sandvig points out, network neutrality is not the issue, it is "who discriminates and for what purpose" [60]. Furthermore it may not be appropriate to view QoS in terms of the service levels centred around *best-effort*, but rather as varying degrees of the preferential allocation of resources.

As the amount of real-time traffic on the Internet increases, such as voice and video, it is essential that support for QoS is provided. First and foremost, QoS is about capacity planning. Network resources need to meet traffic demands. Users, however, have a sporadic need for resources, and peak demand is very rarely sustained for long periods. If the network capacity level is set to the peak demand, then the network will be idle for long periods. Fortunately, many applications are not entirely intolerant to some delay or loss. The network, therefore, does not have to be excessively overprovisioned. Nevertheless, it is not easy, given the stochastic nature of network traffic, to balance QoS requirements and resource efficiency.

1.2 Network Utilisation

In this section, we discuss the effects of network utilisation on performance. We take the opportunity to introduce J and use it to explore some of the concepts presented. Network utilisation is the ratio of demand over capacity. The load on a network link cannot exceed 100 percent of capacity. It is possible, however, for the *offered* load to exceed this figure, in that case the available buffer memory temporarily stores excess traffic. Excessive backlog, and thus delay, can be avoided by limiting the level of network utilisation. This is achieved by monitoring the traffic volumes and setting the capacity relative to the offered load, so that the demand/capacity ratio (utilisation) is sufficiently low so that the backlog is kept within acceptable bounds. The problem is finding the utilisation level which yields "acceptable bounds" for the backlog.

A commonly cited rule of thumb for network utilisation is 30 percent [61, 68]. There appears to be some confusion in some literature between *achievable utilisation* and

the *level of utilisation* that can support a given QoS. Passmore conveys the performance "limitations" of Ethernet [51]:

> shared-media CSMA/CD Ethernet LANs where thirty percent is the effective utilisation limit.

Ethernet, however, has no such limitation, it is capable of much higher utilisation levels, as reported in [7]. Furthermore it is fairly straightforward to demonstrate empirically the utilisation capabilities of Ethernet. One of the origins of this "magic number" is possibly the (slotted) Aloha wireless protocol, which, due to the contention mechanism in the system, achieves a maximum throughput of 37 percent. Here, we carry out an analysis of the throughput of Aloha using J. We also demonstrate how throughput can be improved using carrier sensing techniques. The throughput S for slotted Aloha is the function of the offered load G and is given by the expression [66]:

$$S = Ge^{-G} \tag{1.1}$$

The mathematical expression in Equation (1.1) can be implemented by the J command line:

```
   (*^@-) 0 1 2 3
0 0.367879 0.270671 0.149361
```

From the example above, the indented line is the J command that is entered by the user (the J prompt is a three-space indentation). The sequence of characters `*^@-` (enclosed in brackets) form a function; in this case they define the relationship between S and G in Equation (1.1). The list of values that follows is the argument passed by the function (in this case the argument represents various values of G). The function is applied to each value in the argument list and outputs a corresponding value of S on the nonindented line below. The symbols `^`, `*` and `-` are arithmetic operators and represent the exponential function, multiplication and negation respectively. The `@` primitive is called a *conjunction* in J and acts as a sequencing operator. Without going into detail at this point, think of `@` as an *apply* operator. Thus, the exponential function is applied to the negated values of the arguments (G). The result of this operation is then multiplied by the argument values. J composition rules may not seem intuitive at first, but they will be covered in more detail later. It can be seen that the throughput reaches a maximum of approximately $S \approx .0.37$ for $G = 1$. For pure Aloha (nonslotted), the maximum achievable throughput is even lower:

$$S = Ge^{-2G} \tag{1.2}$$

Equation (1.2) can be realised by the J expression below:

```
   (*^@-@+:) 0 0.5 1 2 3
0 0.18394 0.135335 0.0366313 0.00743626
```


image
not
available

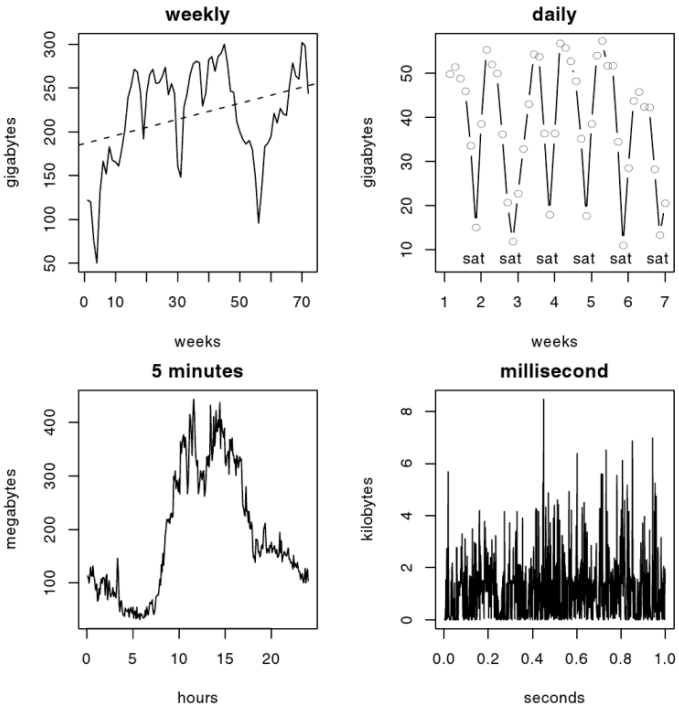

Fig. 1.2. Measured network traffic at various time scales

The top-right graph of Fig 1.2 shows the aggregate daily traffic volumes over a seven-week period. It shows the periodicity of peaks and troughs for weekday and weekend traffic respectively. There appears to be anomalously low traffic on Friday of the second week and Monday of the third week; this corresponds to the Easter bank holiday. The bottom-right graph (Fig 1.2) shows the five-minute traffic volumes over one day. The busy and off-peak periods of the day are evident.

Traffic flows on these time scales do not provide much useful insight into queue dynamics. Rather they are of more interest to the forecaster for the purpose of predicting future resource requirements or identifying peak and off-peak periods. While useful for long/medium-term forecasting, the time intervals over which traffic is aggregated are too long and the degree of "smoothing" does not convey any detailed burst-level behaviour.

The bottom-right graph of Fig 1.2 shows burst-level traffic. The traffic aggregates for each millisecond are shown for a one-second period. At these time scales, traffic is (typically) *stationary*. In traditional telephony, traffic arrival processes smooth exponentially quickly with the increase in aggregation interval. However, traffic patterns in packet data networks have complex structures, even for periods of stationarity,

when packet arrivals can be highly correlated in time, exhibiting local trends and cycles [6, 13, 16, 28].

It is unlikely that there is some panacea utilisation figure that meets all capacity planning requirements and delivers QoS. It is not that the "30 percent" rule is wrong; in the absence of a better figure, it is as good as any. However, the QoS requirements will vary according to the users and the applications they run.

1.3 Traffic Management

The network processes packets according to three categories of traffic management policy:

- forwarding policy
- scheduling policy
- drop policy

If network providers do not apply any active traffic management, the respective forwarding, scheduling and drop policies are (typically): shortest-path/lowest-cost to destination, first-come-first-served (FCFS) and drop-from-tail. In a neutral network environment, all packets, regardless of user and service, are treated the same. Users compete with each other for resources. The (neutral) network provider does not protect users from one another, thus greedy users may get a larger proportion of resources, while the pain of congestion is shared equally. Network providers may elect to employ traffic management policies, such as Fair-Queuing [68] and Random Early Detect (RED), [20] in order to distribute congestion costs amongst users in proportion to their contribution to it.

Assuming dynamic routing protocols are being used (rather than static routing), the traditional IP forwarding policy is the shortest-path (or least-cost) to destination. Upon receiving the packet, each router examines the destination address in the IP header and looks up the next hop of the corresponding router. The "shortest-path" depends on the routing protocol. For example, RIP [12] uses number-of-hops as a routing metric, whereas OSPF [69] uses a cost metric based upon the speed of the link. Shortest-path routing algorithms can lead to traffic hotspots where links on the shortest-path are congested, while those that are not, are underutilised.

Load balancing can be achieved by manually manipulating the routing metrics. This, however, is not an ideal method, as a slight change in operating conditions can significantly degrade the stability of the network.[2]

A better method is to use *policy routing* [44], whereby packets are routed based on some attributes of the packet header other than the destination address (source

[2] Speaking from bitter experience.

address or port numbers for example). Packet routes are determined through network management functions and traffic engineering policies. Policy routing does present a number of problems. First of all, keeping track of routing policies currently in effect is administratively difficult. Secondly, each packet incurs extra processing overheads in determining the flow to which it belongs.

Protocols, such as Multiprotocol Label Switching (MPLS), [2] provide an infrastructure for the management of predefined paths through the network. MPLS routers attach *tags* to packets at the edge of the network according to the flow to which they belong. Flows can be identified by routers in the core with minimal overhead and packets can be forwarded/scheduled/dropped accordingly.

There are two QoS frameworks for IP networks, namely, Differentiated services (DiffServ) and Integrated services (IntServ), Differentiated services involve classifying flows at the edge of the DiffServ "cloud" by setting the Differentiated Services Code-Point (DSCP) in the IP header. Each code-point is associated with some class of service. Thus, flows *mapped* to a particular class are forwarded, scheduled and dropped according to a set of policies that implement the service class. DiffServ specifies a *coarse-grained* QoS. Given that the DSCP is only one byte, there is a significant restriction on the number of flows that can be represented. QoS, therefore, is applied to *aggregates* of flows.

In contrast, IntServ is a *fine-grained* architecture. Resources are reserved in the network for individual flows using a signalling protocol, namely, the Resource reSerVation Protocol (RSVP) [73]. The Traffic SPECification (TSPEC) specifies the parameters for a leaky-bucket algorithm, such as the token arrival rate and bucket depth. A flow $A = \{A(t), t = 0, 1, \ldots\}$, where $A(t)$ is the amount of traffic in the interval $[0, t]$, and conforms to the traffic specification function α if:

$$A(t) - A(s) \leq \alpha(t - s) \quad \forall t \geq s \tag{1.4}$$

The TSPEC parameters $\{M, p, r, b\}$ describe a dual leaky-bucket, where p is the peak rate, M is the maximum packet size, r is the sustainable rate and b is the burst tolerance. Thus $\alpha(t) = \min[pt + M, rt + b]$.

The Resource SPECification (RSPEC) specifies the resources requirements of the flow. For a given flow A, the resource function β should yield an output B, such that:

$$B(t) - A(s) \geq \beta(t - s) \quad \forall t \geq s \tag{1.5}$$

In order to support any degree of QoS, something must be known about the nature of the flows entering the network so that resources can be allocated accordingly. Consider a single server where packets arrivals are distributed according to a Poisson distribution with mean rate λ. Furthermore, the number of packets serviced per unit time is also Poisson distributed with mean rate μ. For a flow of traffic intensity $\rho = \lambda/\mu$, the probability that the backlog equals or exceeds n is ρ^n. For a traffic intensity of $\rho = 0.3$, the J expression shows that the backlog diminishes exponentially with n:

```
   0.3^(1 2 3 4 5)
0.3 0.09 0.027 0.0081 0.00243
```

The average delay for a single $M/M/1$ queue (where the M means Markovian) is given by the expression:

$$E[D] = \frac{1/\mu}{1-\rho} \tag{1.6}$$

If $\mu = 5$, then the delay increases exponentially with traffic intensity ρ:

```
   rho =: 0 0.2 0.4 0.6 0.8 0.9
   0.2 % 1-rho
0.2 0.25 0.333333 0.5 1 2
```

Traffic flows in data networks, however, are not necessarily Poisson. Representing traffic as a wide-sense increasing envelope curve lends itself to analytical treatment using network calculus methods. For example, in Equations (1.4) and (1.5), α and β represent arrival and service curves respectively. Given these two (wide-sense increasing) curves, the upper bound for the backlog can be derived by the network calculus result: $\max[\alpha(s) - \beta(s)]$, $\forall s \geq 0$. Bounds for delay and output traffic can also be derived using network calculus, as will be discussed in detail in this book.

1.4 Queue Analysis

We complete this chapter by carrying out an analysis of the queue dynamics of a working conserving link. We show how J can be used to build models of network systems for the purpose of analysis.

Consider a network traffic source transmitting packets across a communications link of capacity c. It is assumed that the scheduling policy is FCFS and the buffer queue is sufficiently large that packets are never dropped. Packets arriving at the interface of the communications link, form a *discrete-time* arrival process $a = \{a(t), t = 0, 1, 2, \ldots\}$. Packets depart from the link at each time interval at a maximum rate c. If $a(t) > c$, then packets in excess of c are buffered. In the next time interval $t + 1$, the link attempts to clear the backlog $q(t)$ and then forwards any new arrivals $a(t+1)$ up to the bound c. The backlog is given by the Lindley equation [39] below:

$$q(t+1) = (q(t) + a(t+1) - c)^+ \tag{1.7}$$

where the $(x)^+$ is $\max(0, x)$ and $q(0) = 0$. Here, we show how to implement the Lindley equation in J, such that we can examine the queuing dynamics of a work conserving link. The function requires two parameters; the initial queue size $q(0)$, and the arrival process over a (finite) sequence of discrete time intervals $t = 1, \ldots, N$. The J function *qnext* returns the backlog $q(t + 1)$, and is defined thus:

```
   qnext =: 0: >. qprev + anext - c
```

Note that the terms: `0:`, `>.`, *anext*, +, *qprev*, − and c in the J expression above are all functions. J supports a *tacit* programming style; that is, there is no explicit reference to arguments. The built-in functions + and − are the addition and subtraction operators respectively. The *larger-of* function `>.`, is also a built-in funtions, and returns the argument with the greater value:

```
   0 >. 1  NB. max
1
```

Note that the `NB.` construct denotes a comment. In order to make the *qnext* more recognisable, we can define the *max* function, and redefine *qnext*, thus:

```
   max =: >.  NB. define max function
   qnext =: 0: max qprev + anext - c
```

The function `0:`[3] is one of J's *constant* functions, it returns a value zero, irrespective of its arguments:

```
   0: 1
0
```

The function *anext* ($a(t+1)$) returns the number of arrivals, *qprev* returns the backlog ($q(t)$) and c is the capacity of the link. These functions will be defined shortly, but first we must introduce a few more built-in J functions. We cannot make explicit reference to arguments when programming tacitly, although we can access arguments through the the functions *left* `[` and *right* `]` which return the left and right arguments, respectively:

```
   0 1 2 [ 3 4 5  NB. return left argument
0 1 2
    0 1 2 ] 3 4 5 NB. return right argument
3 4 5
```

We use `[` and `]` to extract a and q, respectively. However, we need the values of $a(t+1)$ and $q(t)$ from these vectors. The term $q(t)$ is relatively straight forward, as it is the last element of vector q. The *tail* function `{:` returns the last element of a list:

```
   {: 0 1 2 3 4 5
5
```

[3] In addition to `0:` there are a number of corresponding integer *constant* verbs that range from -9 to 9.