Modelling data networks

Richard G. Clegg (richard@richardclegg.org)

1 Introduction

There are many difficulties to modelling the internet, for a well-known and excellent summary see [6]

- The internet is big (and growing).
- The internet is heterogeneous to a large degree.
- No central maps exist of the internet.
- The internet is not always easy to measure.
- The internet is rapidly changing.
- It is extremely important to be able to model the internet.

The internet cannot possibly be modelled, yet we must model the internet. How can this be resolved?

The intention of this lecture is to teach you three things:

1. A general approach to such modelling problems.
2. Some specific mathematical techniques necessary for certain modelling problems (Markov chains, queuing theory, graph theory).
3. An approach to "sanity checking" your modelling.

- How you model the network depends critically on the problem you are solving.
- What are you trying to show with your model?
- Metrics: what are we trying to measure?
    1. Throughput?
    2. Goodput?
    3. System efficiency?
- Validation: what real data can be used to check the model?
- Sensitivity: what happens if your assumptions change?
    1. What if the demand on the system is slightly different?
    2. What happens if delays and bandwidths are changed?

3. What happens if users stay longer or download more?

Important questions for modelling.

1. How much of the network do we model?
   - Whole internet (then we can't even model every computer – every AS?)
   - A few typical nodes?
   - A sub net?
   - A single queue and buffer?
2. What level of modelling is appropriate?
   - Mathematical – solution "instant" (or quick) but which mathematical techniques are useful?
   - Detailed simulation
   - Combined approach (equations abstract away some details with approximations)
3. How far down the network stack need we go?

1.1 Example model – peer-to-peer network

Modelling Task: Test the possible improvements expected if we try a locality aware peer selection policy on a global bittorrent network.

What must our model include?

1. The distribution of nodes (peers) on the overlay network (not the whole network).
2. The delay and throughput between these peers (must depend on distance to some extent).
3. How users arrive and depart.
4. What users choose to download.

Note that this might already be a vast modelling task with hundreds of thousands or even millions of nodes.

- Research existing P2P models, do any fit? Don't reinvent the wheel.
- Real data: What real-life measurements exist to validate against?
- If we are modelling a new peer selection we must be sure our model covers existing peer selection well.
- Metrics: what must we measure in our model?
  1. Overall throughput/goodput?
  2. Distribution of time taken for peers to make their download?
  3. Total resources used in system?

- Validation: Instrumented P2P clients exist – how do they compare to our simulation.
- Sensitivity: Different distribution of users? Different delays and throughputs?

1.2 Example model – Buffer provisioning model

Modelling task: Given a router with a buffer, how does the buffer size in packets affect the probability of packet loss?

What must our model include?

1. A model of the incoming packets to the buffer.
2. The rate at which packets leave the buffer.
3. Possibly distribution of packet lengths in bytes.
4. Possibly the feedback (TCP) between packet loss and arrival rate.

- Research: what is known about the statistics of internet traffic?
- What is the distribution of inter-arrival times and packet lengths?
- Metrics:
  1. Packet loss.
  2. Packet delay.
- Sensitivity: What if we change the following parameters:
  1. The total arrival rate.
  2. The bandwidth of the outgoing link.
- Validation: Real traffic traces (CAIDA has a collection).

1.3 Example model – TCP throughput

Modelling Task: Test a possible improvement to the TCP model which aims to improve fairness and throughput when flows share a link.

What must our model include?

1. Individual packet model with existing TCP protocol as accurately as possible.
2. A reasonable estimate of how long each connection lasts and the rate at which new connections.
3. A model of the probability of round trip time for the parts of the connection not on the link being modelled.
4. A model of the probability of packet loss on the link (due to buffer overflow?)

- Can existing network models help (ns-2 could be an obvious choice)?

- What if the existing protocol shares a link with flows using the old protocol.
- Metrics:
  1. Throughput and goodput.
  2. Fairness between flows.
- Sensitivity, what if we change these parameters:
  1. Number of flows using existing and new protocol.
  2. Bandwidth of link.
  3. Round trip time of flows.
  4. Probability of packet loss.
- Validation: Does our model agree with real measurements?

2 Modelling Overview

2.1 Modelling areas

Now let us focus on several specific areas of interest to modellers.

1. Topology modelling — how are the nodes in the internet connected to each other?
   - See the internet as nodes and edges (graph theory).
   - Consider numbers of hops between nodes.
   - How do we find shortest paths in a network (Bellman–Ford or Dijkstra algorithm).
2. User/flow arrival modelling — how does traffic arrive on the internet?
   - See arrivals as a stochastic process (probability/statistics)
   - How long do connections last?
3. Application level protocols — what traffic do applications place on the internet?
   - For example peer-to-peer networks use an overlay (graph theory again?)
   - A web page might make connections to many different places.
4. Traffic statistics — what does the traffic along a link look like in statistical terms?
   - See internet traffic as a stochastic process (queuing theory).
   - How does TCP congestion control alter this?
5. Transport/network protocols — how do TCP/IP protocols affect the traffic?
   - See internet traffic as a feedback process (control theory).

- How do these protocols interact with the rest of the network?

6. Other things to model:
   - Reliability modelling — what happens when links or nodes fail?
   - Overlay networks — P2P increasingly important.

2.2 Topology modelling

- Two levels of topology are usually considered "router level" and "autonomous system" (AS) level.
- Router level topology is still the least well-known — often ISPs take trouble to protect this information for security reasons.
- Topology metrics — these quantities are all rigorously defined and can be found in the literature:
  1. Graph diameter (longest possible "shortest path" between nodes).
  2. Node degree distribution (what proportion of nodes have k neighbours).
  3. Assortivity/disassortivity (do well-connected nodes connect with each other?) – sometimes called "rich club".
  4. Clustering (triangle count) – are the neighbours of a node also neighbours of each other.
  5. Clique size – largest group where everyone is everyone's neighbour (a clique in graph theory).

The node-degree distribution in AS networks is particularly well-studied. Let $P(k)$ be the proportion of nodes with degree k (having k neigbours). To a good approximation

$$P(k) \sim k^{-\alpha},$$

where α is a constant.

- Power law topology of the AS graph shown by Faloutsos et al [5].
- This graph has some interesting properties — some extremely highly connected nodes, what happens if they fail?
- Same type of graph as:
  1. Links on websites, wikipedia and many other similar online systems.
  2. Academic citations in papers.
  3. Human sexual contacts.

Albert–Barabasi [2] "Preferential attachment" model

Constructive model start with a small "core" network. When a new node arrives, attach it to an old node with the following probability

$$\mathbb{P}\left[\text{Attaching to node } i\right] = \frac{d(i)}{\sum_{j \in \text{all nodes}} d(j)},$$

where $d(i)$ is the degree of node i.

- This model "grows" a network with a powerlaw.
- Many similar models have been created which are more general.
- Current best model may be Positive Feedback Preference [8]. which adds a small "faster than exactly proportional" term.

Why not save work by using existing models to generate your network?

2.3 User/flow arrival modelling

- As a first approximation the arrival of users can be modelled as a Poisson process.
- You might want to consider periodic effects:
  1. Daily – with people's sleep cycles.
  2. Weekly – weekends different.
  3. Yearly – year-on-year growth in traffic.
- Perhaps simpler just to simulate some peak hour and some estimate of growth?

2.4 Application level protocols

- If you are modelling a specific application there will be details associated with this.
- Common applications (www, ftp, p2p) will have existing research — read what is done before setting out on your own.
- If no studies are done what could you compare your application to?
- Could your application be viewed as:
  1. A series of ftp-like transfers of data.
  2. UDP bursts at a given rate for given periods of time
  3. A p2p application which might use existing p2p research methods.
- An important thing to simulate is the length of transfers and for many applications this is heavy-tailed [1].

A variable X has a heavy-tailed distribution if

$$\mathbb{P}[X > x] \sim x^{-\beta},$$

where $\beta \in (0, 2)$ and $\sim$ again means asymptotically proportional to as $x \to \infty$.

- Obviously an example of a power law.
- A distribution where *extreme values* are still quite common.
- Examples: Heights of trees, frequency of words, populations of towns.
- Best known example, Pareto distribution $\mathbb{P}[X > x] = (x/x_m)^{-\beta}$ where $x_m > 0$ is the smallest value X can have.
- The following internet distributions have heavy tails:
  1. Files on any particular computer.
  2. Files transferred via ftp.
  3. Bytes transferred by single TCP connections.
  4. Files downloaded by the WWW.
- This is more than just a statistical curiousity.
- Consider what this distribution would do to queuing performance (no longer Poisson).
- Non mathematicians are starting to take an interest in heavy tails (reference to "the long tail".

2.5 Traffic statistics

Long-Range Dependence (LRD) is considered to be an important characteristic of internet traffic.

- In 1993 LRD was found in a time series of bytes/unit time measured on an Ethernet LAN [Leland et al '93].
- This finding has been repeated a number of times by a large number of authors (however recent evidence suggests this may not happen in the core).
- A higher Hurst parameter often increases delays in a network. Packet loss also suffers.
- If buffer provisioning is done using the assumption of Poisson traffic then the network will probably be under-specified.
- The Hurst parameter is "a dominant characteristic for a number of packet traffic engineering problems".

Let $\{X_1, X_2, X_3, \dots\}$ be a weakly stationary time series.

The Autocorrelation Function (ACF) is defined as

$$\rho(k) = \frac{\mathrm{E}\left[(X_t - \mu)(X_{t+k} - \mu)\right]}{\sigma^2},$$

where μ is the mean and σ^2 is the variance.

The ACF measures the correlation between X_t and X_{t+k} and is normalised so $\rho(k) \in [-1, 1]$. Note symmetry $\rho(k) = \rho(-k)$.

A process exhibits LRD if $\sum_{k=0}^{\infty} \rho(k)$ diverges (is not finite).

Definition of Hurst Parameter

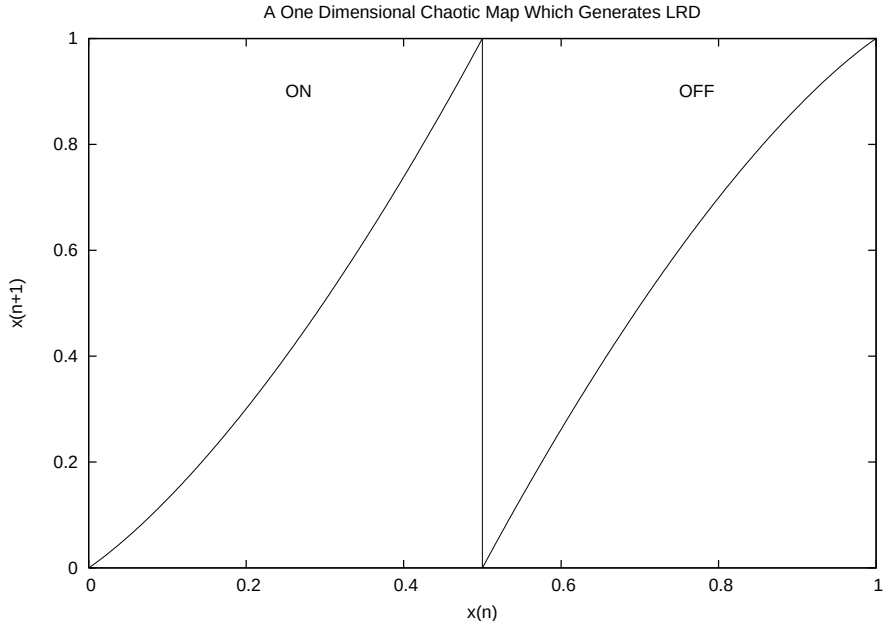
The following functional form for the ACF is often assumed

$$\rho(k) \sim |k|^{-2(1-H)},$$

where $\sim$ means asymptotically proportional to and $H \in (1/2, 1)$ is the Hurst Parameter.

- Think of LRD as meaning that data from the distant past continue to effect the present.
- LRD was first spotted by a hydrologist (Hurst) looking at the flooding of the Nile river.
- For this reason Mandelbrot called it "the Joseph effect".
- Stock prices (once normalised) also show LRD.
- LRD can also be seen in the temperature of the earth (once the trend is removed).
- Models include Markov chains, Fractional Brownian Motion (variant on Brownian motion), Chaotic maps and many others [4].

Iterated map model for LRD.



$$x_{n+1} = \begin{cases} x_n + \frac{1-d}{d^{m_1}} x_n^{m_1} & 0 < x_n < d, \\ x_n - \frac{d}{(1-d)^{m_2}} (1 - x_n)^{m_2} & d < x_n < 1, \end{cases}$$

where $x_n, d \in (0, 1)$, $m_1, m_2 \in (3/2, 2)$. Produces ON and OFF series — packets and not packets with Hurst $H = min(m_1, m_2) - 1$.

2.6 Transport and network level protocols

- It might be important if we are considering a packet level model to model specific details of the TCP/IP protocols.
- Usually this will involve simulating the window size (additive increase multiplicative decrease) of the TCP protocol.
- Remember that a detailed simulation to this level will extremely limit the number of nodes which can be simulated.
- A mathematical model will be demonstrated in the next section.
- In addition, the ns-2 model will be shown which is a packet level simulation of TCP/IP.

2.7 Other things to model

- Of course depending on the nature of your modelling, there may well be other aspects of the network to be modelled.
- Some examples might be:
  1. Reliability of nodes and links.
  2. An overlay network.

3. Possible hostile attacks to the network.

- In all cases, an important starting point is to find out what research already exists in the area.
- Are any real-life data sets available which could inform your modelling? Could you gather such data?

3 Mathematical modelling

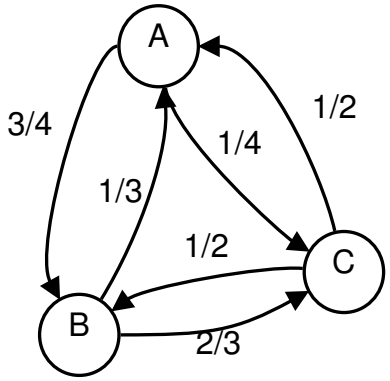
- To create a simulation model we need to be able to write down equations for the system.
- The more work we can do "on paper" the easier the computational burden.
- This will be illustrated with two mathematical techniques related to networks.
- Firstly the concept of the Markov Chain is introduced and used in queue modelling.
- Secondly basic concepts from Graph Theory are used and illustrate path finding in networks.
- These models can be used as a basis for computer simulation.

3.1 A Markov model for the leaky-bucket

- A "leaky bucket" is a mechanism for managing buffers and to smooth downstream flow.
- What is described here is sometimes known as a "token bucket".
- A queue holds a stock of "permit" generated at a rate r (one permit every $1/r$ seconds) up to a maximum of W.
- A packet cannot leave the queue without a permit – each packet takes one permit.
- The idea is that a short burst of traffic can be accommodated but a longer burst is smoothed to ensure that downstream can cope.
- Assume that packets arrive as a Poisson process at rate λ.
- A Markov model will be used [3, page 515].

3.1.1 What is a Markov Chain?

Now consider the idea of things moving about randomly in space. For example consider a hitch-hiking hippy who, for some reason, has no short term memory. He hitch hikes at random between A-town, B-town and C-town. At A-town he has a 3/4 chance to get to B-town and a 1/4 chance to get to C-town the next day. He moves town every day according to the diagram below.



Now, the next step is to ask questions about where the hippy is at a given day. If he starts in A-town on day one then on day 2 he has a 1/4 chance of being in C town and a 3/4 chance of being in B town. On day 3 he has a 3/8 chance of being in A town ($1/3 \times 3/4 = 1/4$ via $A \to B \to A$ plus $1/2 \times 1/4 = 1/8$ via $A \to C \to A$). We could carry on making calculations like this for day 4 day 5 and so on but it would get boring. We need a smarter way.

Define the transition probabilities. Let us number instead of name our towns, 0, 1 and 2 instead of A, B and C. Let p_{ij} be the probability that if the hippy began at i he moves to j in one day. So, for example $p_{11} = 0$ (there is zero probability the hippy begins in B and ends in B). $p_{12} = 2/3$ (the probability that the hippy at B moves to C is 2/3). We can now define the *transition matrix* P. The transition matrix is the matrix of these probabilities:

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix}.$$

Assume we know some initial vector of probabilities of his location, for example, we know he is definitely in town zero (A town) and call this $\boldsymbol{\lambda}_0$.

$$\boldsymbol{\lambda}_0 = [100].$$

In general define $\lambda_{i,j}$ as the probability the hippy is in state j on day i and $\boldsymbol{\lambda}_i = [\lambda_{i,0}\lambda_{i,1}\lambda_{i,2}]$ as the vector of all states on day i. We can then use the following equation

$$\boldsymbol{\lambda}_1^T = \boldsymbol{\lambda}_0^T \mathbf{P},$$

where T means transpose (make the row vector a column vector). This does not seem to help much but we can also say

$$\boldsymbol{\lambda}_i^T = \boldsymbol{\lambda}_{i-1}^T \mathbf{P},$$

This tells us how to work out the probabilities of where the hippy is on a given day when we have the starting probabilities and the transition matrix.

We might be interested in something more final such as where the hippy "ends up". Define $\boldsymbol{\pi}$ as the vector $[\pi_0, \pi_1, \pi_2]$ where $\pi_j = \lim_{i\to\infty} \lambda_{i,j}$ that is π_j is the probability at the "final" day that the hippy is in town j. Two questions arise: Does this exist and does it depend on where the hippy was on the first day. The answer for most Markov chains usually encountered (those which are

connected, finite and not periodic) is that these probabilities do exist and that they do not depend on the initial conditions.

These probabilities are known as equilibrium probabilities and can be simply calculated in a lot of cases using what are know as the equilibrium conditions – that is $\sum_i \pi_i = 1$ (the probabilities sum to one) and for all i then the flow into a state is equal to the probability of the state $\sum_j \pi_i p_{i,j} = \pi_i$. In matrix terms this is

$$\boldsymbol{\pi}^T = \mathbf{P}\boldsymbol{\pi}.$$

For our hippy example this gives us these equations (remembering that $p_{00}, p_{11}, p_{22} = 0$ then

$$\begin{aligned}
\pi_0 + \pi_1 + \pi_2 &= 1 && \text{probabilities sum to one} \\
\pi_1 p_{10} + \pi_2 p_{20} &= \pi_0 && \text{balance for city 0} \\
\pi_1 p_{01} + \pi_2 p_{21} &= \pi_1 && \text{balance for city 1} \\
\pi_1 p_{02} + \pi_2 p_{12} &= \pi_2 && \text{balance for city 2}
\end{aligned}$$

Note that we do need the first equation because the balance equations contain one dependent equation. These could be solved using standard techniques for simultaneous equations to give: $\pi_0 = 16/55$, $\pi_1 = 21/55$ and $\pi_2 = 18/55$.

3.1.2 The birth-death process example

What does all this have to do with queues? Well, the concept does not apply just to hippies and towns but is much more general. If we consider a system in some "state" and the "state" at the next time depends only on the current "state" then we can model this with a Markov chain. The "state" in the hippy example was the town and the town on day $i + 1$ depended only on the town on day i. For a more useful example, think of the "state" number as being the number of packets in a queue.

Then let us consider the following scenario. Imagine a queue of packets in a buffer. The buffer operates over certain time periods. With probability p every time period a new packet arrives. With probability q a packet in the queue is sent out by the server. The queue can have K packets in it and packets over this get dropped. We can formulate this queue as a Markov chain. We can write down transition probabilities, for example, $p_{01} = p$ (with probability p a new packet arrives, since there are no packets in the queue then no packets can leave). Similarly $p_{11} = (1 - p - q) + pq$ – there remains one packet in the queue if there is already one packet in the queue and either (with probability pq) a packet arrives but one is served by the queue or (with probability $1 - p - q$) no packets arrive or leave. This sort of process where packets (or customers) arrive or leave a queue is known as a "birth-death" process and they are integral to queuing theory.

The problem could be completed by writing down the complete transition

matrix

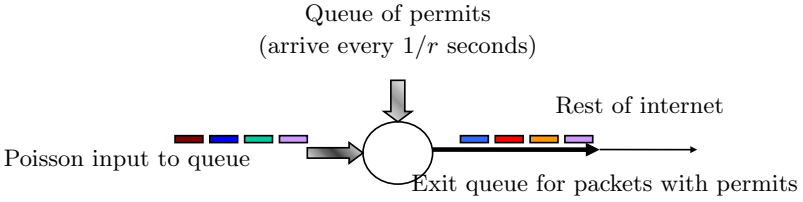
$$\mathbf{P} = \begin{bmatrix} 1-p & p & 0 & \cdots & 0 \\ q(1-p) & 1-p-q+pq & p(1-q) & \cdots & 0 \\ 0 & q(1-p) & 1-p-q+pq & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 1-q+pq \end{bmatrix}$$

and then proceeding to solve as in the previous case.

3.1.3 Back to the leaky bucket example

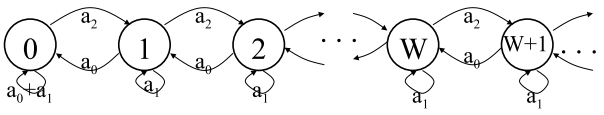
Use a discrete time Markov chain where we stay in each state for time $1/r$ seconds (the time taken to generate one permit). Let a_k be the probability that k packets arrive in one time period. Since arrivals are Poisson,

$$a_k = \frac{e^{-\lambda/r}(\lambda/r)^k}{k!}.$$



- In one time period (length $1/r$ secs) one token is generated (unless W exist) and some may be used sending packets.
- States $i \in \{0, 1, \ldots, W\}$ represent no packets waiting and $W - i$ permits available. States $i \in \{W+1, W+2, \ldots\}$ represent 0 tokens and $i - W$ packets waiting.
- If k packets arrive we move from state i to state $i + k - 1$ (except from state 0).
- Transition probabilities from i to j, $p_{i,j}$ given by

$$p_{i,j} = \begin{cases} a_0 + a_1 & i = j = 0 \\ a_{j-i+1} & j \geq i - 1 \\ 0 & \text{otherwise} \end{cases}$$



Let π_i be the equilibrium probability of state i. Now, we can calculate the probability flows in and out of each state.

For state one

$$\pi_0 = a_0\pi_1 + (a_0 + a_1)\pi_0$$
$$\pi_1 = (1 - a_0 - a_1)\pi_0/a_0.$$

For state $i > 0$ then $\pi_i = \sum_{j=0}^{i+1} a_{i-j+1}\pi_j$. Therefore,

$$\pi_1 = a_2\pi_0 + a_1\pi_1 + a_0\pi_2$$
$$\pi_2 = \frac{\pi_0}{a_0}\left(\frac{(1 - a_0 - a_1)(1 - a_1)}{a_0} - a_2\right).$$

In a similar way, we can get π_i in terms of $\pi_0, \pi_1, \ldots, \pi_{i-1}$.

- We could use $\sum_{i=0}^{\infty} \pi_i = 1$ to get result but this is difficult.
- Note that permits are generated every step except in state 0 when no packets arrived (W permits exist and none used up).
- This means permits arrive at rate $(1 - \pi_0 a_0)r$.
- Rate of tokens arriving must equal λ unless the queue grows forever (each packet gets a permit).
- Therefore $\pi_0 = (r - \lambda)/(ra_0)$.
- Given this we can then get π_1, π_2 and so on.

To complete the model we want to calculate T average delay of a packet.

- If we are in states $\{0, 1, \ldots, W\}$ packet exits immediately with no delay.
- If we are in states $i \in \{W + 1, W + 2, \ldots\}$ then we must wait for $i - W$ tokens $(i - W)/r$ seconds to get a token.
- The proportion of the time spent in state i is π_i.
- The final expression for the delay is

$$T = \frac{1}{r} \sum_{j=W+1}^{\infty} \pi_j(j - W).$$

- For more analysis of this model see [3, page 515].

3.2 Shortest Paths in networks

Definition 3.1. A weighted graph $G = (\mathcal{N}, \mathcal{A})$ is one where each arc $(i, j) \in \mathcal{A}$ has associated with it a weight w_{ij}.

The concept of a weighted graph is extremely useful. The weights can be thought of, for example, as the cost of sending a message down a particular arc. (Not necessarily a monetary cost but some combination of time and distance for example). Weighted graphs can be used to formulate the shortest path problem for routing packets.

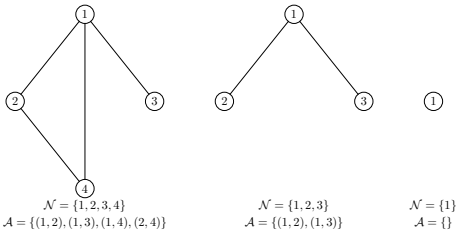


Figure 1: Example graphs

Definition 3.2. A graph $G = (\mathcal{N}, \mathcal{A})$ is a finite set of $\mathcal{N}$ nodes and a set $\mathcal{A}$ of unordered pairs (i, j) where $i, j \in \mathcal{N} : i \neq j$ (known as arcs).

Definition 3.3. If n_1 and n_2 are nodes and (n_1, n_2) (where $n_1 \neq n_2$) is an arc then this arc is said to be *incident* on n_1 and n_2.

Definition 3.4. A *walk* in a graph G is a sequence of nodes in a graph $(n_1, n_2, \ldots, n_l)$ such that each adjacent pair $(n_1, n_2), (n_2, n_3), \ldots (n_{l-1}, n_l)$ are arcs in G.

Definition 3.5. A *path* is a walk with no repeated nodes.

Definition 3.6. A weighted graph $G = (\mathcal{N}, \mathcal{A})$ is one where each arc $(i, j) \in \mathcal{A}$ has associated with it a weight w_{ij}.

The concept of a weighted graph is extremely useful. The weights can be thought of, for example, as the cost of sending a message down a particular arc. (Not necessarily a monetary cost but some combination of time and distance for example). Weighted graphs can be used to formulate the shortest path problem for routing packets.

The problem of finding a shortest path from O to D in a directed graph $G = (\mathcal{N}, \mathcal{A})$ (with arc weights w_{ij} associated with each arc $(i, j) \in \mathcal{A}$) is the problem of finding the directed path $(n_1, \ldots n_l)$ where $n_1 = O$ and $n_l = D$ which minimises the sum $\sum_{i=1}^{l-1} w_{n_i n_{i+1}}$. We will also find it useful to use the convention that $w_{ij} = \infty$ if $(i, j) \notin \mathcal{A}$. Two methods in common use are Bellman-Ford and Dijkstra's Algorithm.

Dijkstra's Algorithm

Dijkstra's Algorithm was discovered by the pioneering mathematician and programmer E.W.Dijkstra (1930 – 2002). The algorithm finds the shortest path to all nodes from an origin node, on a graph $G = (\mathcal{N}, \mathcal{A})$. It requires that all arc weights are non-negative $\forall (i, j) \in \mathcal{A} : w_{ij} \geq 0$.

Dijkstra's Algorithm involves the labelling of a set of permanent nodes P and node distances D_j to each node $j \in \mathcal{N}$. Assume that we wish to find the shortest paths from node 1 to all nodes. Then we begin with: $P = \{1\}$ and $D_1 = 0$ and $D_j = w_1 j$ where $j \neq 1$. Dijkstra's algorithm then consists of following the procedure:

1. Find the next closest node. Find $i \notin P$ such that:

$$D_i = \min_{j \notin P} D_j$$

2. Update our set of permanently labelled nodes and our nodes distances: $P := P \cup \{i\}$.

3. If all nodes in $\mathcal{N}$ are also in P then we have finished so stop here.

4. Update the temporary (distance) labels for the new node i. For all $j \notin P$

$$D_j := \min[D_j, w_{ij} + D_i]$$

5. Go to the beginning of the algorithm.

Note that the version of this algorithm is subtly different from that in Bertsekas & Gallager which finds the paths from all nodes to a single destination. (This is not an important detail and it should be obvious how to reverse the algorithm). To understand the algorithm we make the following claims:

Proposition 3.1. *At the beginning of each iteration of Dijkstra's algorithm then:*

1. *$D_i \leq D_j$ for all $i \in P$ and $j \notin P$.*

2. *D_j is, for all j, the shortest distance from 1 to j using paths with all nodes (except, possibly j) in P.*

If this proposition can be proved then we can see that, when P contains every node in $\mathcal{N}$ then all the D_j are shortest paths by the second part of this proposition. Therefore proving the above proposition is equivalent to proving that Dijkstra's algorithm finds shortest paths.

Proof. The proposition is trivially true at the first step since P consists only of the origin point (node 1) and D_j is 0 for $j = 1$, is $w_{ij} \geq 0$ for nodes reachable directly from node 1 and ∞ otherwise.

The first condition is simply shown to be satisfied since it is preserved by the formula:

$$D_j := \min[D_j, w_{ij} + D_i]$$

which is applied to all $j \notin P$ when node i is added to the set P.

We show the second condition by induction. We have established already that it is true at the very start of the algorithm. Let us assume it is true for the beginning of some iteration of the algorithm and show that it must then be true at the beginning of the next iteration.

Let node i be the node we are adding to our set P and let D_k be the label of each node k at the beginning of the step. The second condition must, therefore, hold for node $j = i$ (the new node we have added) by our induction hypothesis. It must also hold for all nodes $j \in P$ by part one of the proposition which is already proven. It remains to prove that the second condition of the proposition is met for $j \notin P \cup \{i\}$.

Consider a path from 1 to j which is shortest amongst those with all nodes except j in $P \cup \{i\}$ and let D'_j be the corresponding shortest distance. Such a path must contain a path from 1 to some node $r \in P \cup \{i\}$ and an arc(r, j). We have already established that the length of the path from 1 to r must be D_r and therefore we have:

$$D'_j = \min_{r \in P \cup \{i\}} [D_r + w_{rj}] = \min[\min_{r \in P}[D_r + w_{rj}], D_i + w_{ij}]$$

However, by our hypothesis $D_j = \min_{r \in P}[D_r + w_{rj}]$ therefore, $D'_j = min[D_j, D_i + w_{ij}]$ which is exactly what is set by the fourth step of the algorithm. Thus, after any iteration of the algorithm, the second part of the proposition is true if it was true at the beginning of the iteration. Thus the proof by induction is complete. □

The observant will have noticed that this provides the shortest distance to each node from the origin. To find the shortest path, simply work backwards from the destination, asking which node in step four the new node was added from.

Bellman-Ford Algorithm

Notation. *D_i^h is the distance of the shortest walk from node 1 to node i of h steps or less.*

Set initially $D_i^0 = \infty$ for $i \neq 1$ and $D_1^h = 0$ for all h.
The Bellman-Ford Algorithm is then simply, for all $i \neq 1$,

$$D_i^{h+1} = \min_j [D_j^h + w_{ji}]$$

The algorithm terminates after h iterations if

$$D_i^h = D_i^{h-1} \quad \forall i$$

Proposition 3.2. *The scalars D_i^h generate by the algorithm from the starting values given for D_i^0 are equal to the shortest walk of length $\leq h$ from node 1 to node i.*

Proof. The first iteration will clearly give us $D_i^1 = w_{1i}$ for all i apart from $i = 1$ — which is, indeed, correct for the walk lengths of ≤ 1 from 1 to i. Let us suppose that D_i^k is the shortest walk of length $\leq k$ (from 1 to i) then complete the proof by induction by showing that D_i^{k+1} is the shortest walk of length $\leq k + 1$.

One possibility is that the shortest walk of length $\leq (k+1)$ will be a walk of length k or less — in this case, $D_i^{k+1} = D_i^k$. Otherwise, the walk will be a walk of length $k + 1$ with the final arc being (j, i) added on to some walk of length k to node j. Thus, we can conclude that our shortest walk of length $k + 1$ is given by:

$$\text{Shortest walk to } i \text{ of length} \leq k+1 = \min \left[D_i^k, \min_j D_j^k + w_{ji} \right]$$

However, since a walk of length $\leq k + 1$ must always be shorter or equal to a walk of length $\leq k$ (since the former contains the latter) then this reduces to

$$\text{Shortest walk to } i \text{ of length} \leq k+1 = \min_j D_j^k + w_{ji}$$

which is our original expression for D_j^{k+1} and completes our proof by induction. □

Proposition 3.3. *The algorithm terminates after a finite number of iterations if, and only if, all cycles not containing node i have non negative length. Furthermore, if the algorithm terminates, it does so after at most $h \leq N$ iterations and, at termination, D_i^h is the shortest path length from 1 to i.*

Proof. If negative length cycles exist then such cycles could be repeated as many times as desired to reduce the shortest walk length and thus the algorithm could never converge. Conversely, if no negative length cycles exist then any walks containing cycles could be made shorter or kept the same length by deleting such a cycle. Thus, our shortest walks contain no cycles. The maximum length of a walk with no cycles is $N - 1$ (since such a walk will have covered every node). Thus, it trivially follows that $D_i^N = D_i^{N-1}$ and the algorithm terminates after, at most, N iterations. □

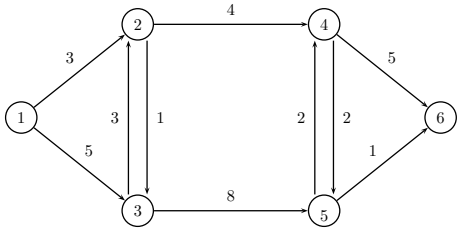


Figure 2: Weighted graph for Dijkstra's algorithm and Bellman-Ford

The values for BellMan-Ford on graph 2 are as shown in this table:

i	D_1^i	D_2^i	D_3^i	D_4^i	D_5^i	D_6^i
1	0	3	5	∞	∞	∞
2	0	3	4	7	13	∞
3	0	3	4	7	9	12
4	0	3	4	7	9	10
5+	0	3	4	7	9	10

The nodes selected for Dijkstra for the same graph are shown on this table

Permanent nodes	Temporary Nodes
1 (0)	2 (3), 3 (5)
1(0), 2(3)	3(4), 4(7)
1(0), 2(3), 3(4)	4(7), 5(12)
1(0), 2(3), 3(4), 4(7)	5(9), 6(12)
1(0), 2(3), 3(4), 4(7), 5(9)	6(10)
1(0), 2(3), 3(4), 4(7), 5(9), 6(10)	

4 The ns-2 simulation

- ns-2 is a freely available event-driven simulator which simulates packet-level traffic.
- It is available from `http://www.isi.edu/nsnam/ns/`
- The simulator is written in C++ but uses tcl for simulations.
- The scripts used for the rest of this lecture are available at `http://www.richardclegg.org/lectures`

5 Final thoughts

- Select an appropriate level of modelling — if you need to model the whole internet you cannot do packet level modelling. If you need to model intricate protocol details for packets you cannot model the whole internet.
- Check against real data where possible that your modelling assumptions are justified.
- Is your experiment repeatable? Do you get similar results if you try slightly different starting scenarios?
- Remember sensitivity analysis: What happens if the bandwidth is a little less? What if the demand is a little more?
- Can statistical analysis of your results help?
- Remember that what you model today is out of date in a year and hopelessly obsolete in ten years.

6 Bibliography

References

[1] R. J. Adler, R. E. Feldman, and M. S. Taqqu, editors. *A Practical Guide to Heavy Tails.* Birkhäuser, 1998.

[2] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[3] D. P. Bertsekas, and R. G. Gallagher Data Networks *Longman Higher Education* 1986

[4] R. G. Clegg. Simulating internet traffic with markov-modulated processes. *Proceedings of UK Performance Engineering Workshop*, 2007. Available online at: `http://www.richardclegg.org/pubs/rgc_ukpew2007.pdf`

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power–law relationships of the Internet topology. *Comput. Commun. Rev.*, 29:251–262, 1999.

[6] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. on Networking*, 9(4):392–403, 2001. `http://www.icir.org/floyd/papers/simulate_2001.pdf`

[7] J. Padhye, V. Firoiu, D. Towsley and J. Kurose Modelling TCP throughput: a simple model and its empirical validation *ACM SIGCOMM Computer Communication Review* 28(4), 1998.

[8] S. Zhou and R. J. Mondragón. Accurately modelling the Internet topology. *Phys. Rev. E*, 70(066108), 2004.