Information Theory and Networks

Lecture 28: Network Coding

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

October 22, 2013

Part I

Network Coding

> *E pluribus unum* (out of many, one).
>
> *de facto motto of the United States until 1956*

Section 1

Multicast

The two biggest classes of information transportation are

- point-to-point (unicast): from 1 source to 1 sink
- broadcast: 1 one source to all sinks

But there is a third, which sits in between: multicast

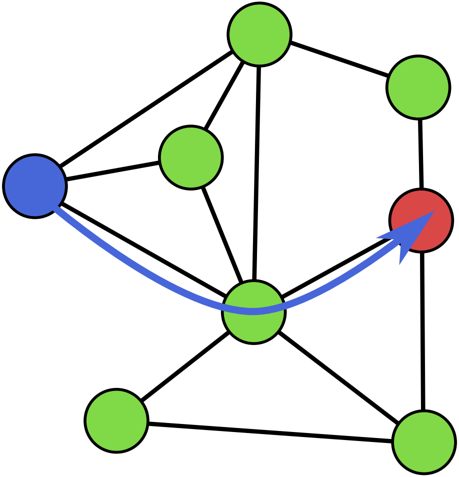
Multicast

- 1 source and multiple destinations (but not everyone)
  - e.g., send video of an event to a group of interested viewers
  - e.g., each person in a video conference multicasts their video to the other participants
- Could be achieved either
  - broadcast to everyone
  - multiple point-to-point links

  but these are inefficient
- Goal to distribute the information as efficiently as possible
  - avoid sending the same information over the same links more than once
  - routers have to not just forward, but also replicate packets
  - ideally: optimise information transport as a whole, not just per connection

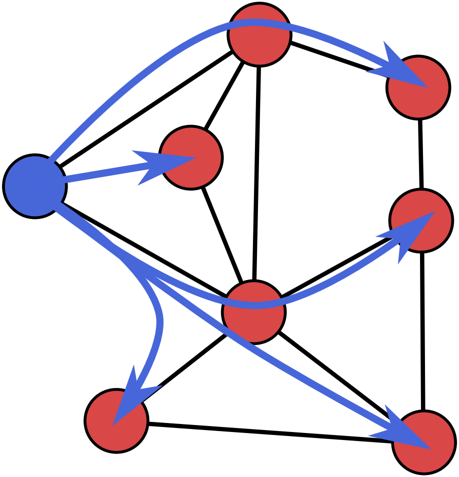
Multicast

Point to point



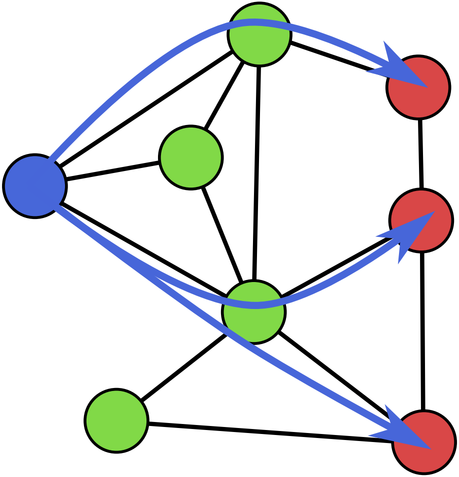
Multicast

Broadcast



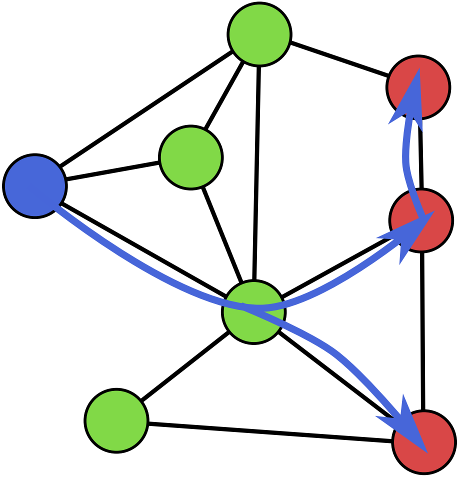
Multicast

Multicast (dumb)



Multicast

Multicast (smart)



Multicast rate

How much can we send over a network using multicast?

- We are aiming to solve multicast problem
- But start simple with degenerate case: unicast

Unicast Maximum Flow Problem

- A network is a directed graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$
  - each link $e \in \mathcal{E}$ has a rate limit, denoted by r_e (≥ 0)
  - can think of a mapping $r : \mathcal{E} \to \mathbb{R}^+$
- We have a source s and a sink t
- A flow f_e represents the traffic on a link $e \in \mathcal{E}$
  - its also a mapping $f : \mathcal{E} \to \mathbb{R}^+$
  - it is constrained such that $0 \leq f_e \leq r_e$ for all $e \in \mathcal{E}$
  - we also require flows to be conserved, so flow into a node must flow out

$$\sum_u f_{(u,v)} = \sum_w f_{(v,w)}$$

  for all $v \in \mathcal{N} \backslash \{s, t\}$

- The maximum flow problem is to maximise $\sum_s f_{(s,v)} = \sum_t f_{(v,t)}$, i.e., the total flow between s and t

Cutsets

Definition (Partition)

X, $\bar{X}$ is a partition of a set $\mathcal{N}$, if $\bar{X} = \mathcal{N} \backslash X$, that is

$$\begin{aligned} X \cup \bar{X} &= \mathcal{N} \\ X \cap \bar{X} &= \phi \end{aligned}$$

Definition (Cutset)

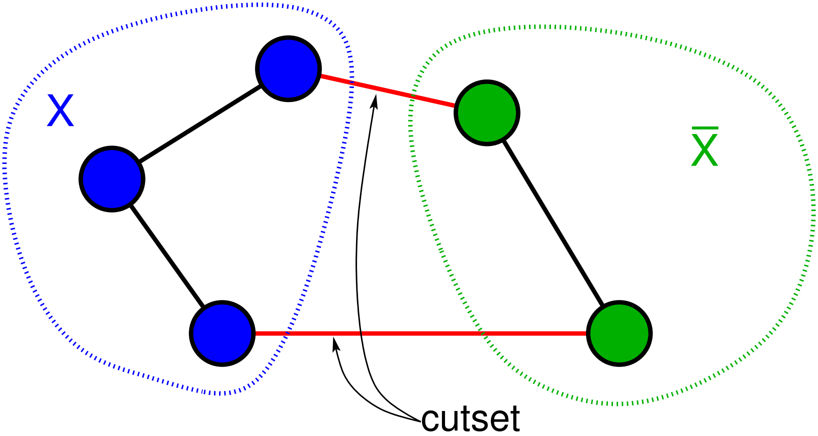
A cutset of a partition $(X, \bar{X})$ of the nodes of $\mathcal{G}(\mathcal{N}, \mathcal{E})$ is the set of links that create the partition

$$\mathcal{C}(X, \bar{X}) = \{(i,j) \in \mathcal{E} \mid i \in X, j \in \bar{X}\}$$

Definition (Cutset capacity)

The capacity $c(X, \bar{X})$ of a cutset is the sum over the capacities of the links in the cutset.

Cutset example



Max-Flow Min-Cut Theorem

Theorem

The maximum flow is equal to the minimum cutset capacity over all cutsets that separate the source s and sink t.

Essentially, we can formulate the max-flow problem as a linear program, with the min-cut problem as its dual.

Generalisations

- The maximum multi-commodity flow problem is a generalisation of max flow problem
  - same basic setup
  - now multiple "commodities" each with source and sink
  - alternative versions of the problem (e.g., satisfy demands with minimum cost, or maximise possible flows)
  - can be solved as a linear program
    - ★ however, often, we don't want to separate flows
    - ★ then it becomes an NP-complete linear integer program
- We can imagine generalising to multicast
  - breaks flow conservation rules
  - we deal with that by introducing new nodes

Multicast max flow

- Assume we have flow we want to maximise from source s to a set of sinks $\mathcal{T}$
- Expand the graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ to $\mathcal{G}'$
  - install a new node t'
  - every node t in $\mathcal{T}$ is connected to t' by an arc (t, t')
  - rate constraint of edges (t, t') is $r_{(t,t')} = \infty$
- Now we find max flow by finding the min cutset between s and t'
- Intuitively, t' acts as a single sink node that collects all of the flows into $\mathcal{T}$
  - thus by maximising the flow to t', we maximise the total flow to $\mathcal{T}$ [Yeu10, p.423]
- note that this isn't really how multicast works, but neither was the unicast, flow maximisation problem
  - they are idealisations that ignore the realities of routing in current networks
  - but they provide an upper bound on what is achievable

Multicast

It has some problems (that don't really concern us here):

- you need support for it in routers, and it can cause them performance issues
- multicast routing is harder than p-2-p routing
- congestion control is hard

result is management complexity, and easier attacks for hackers.

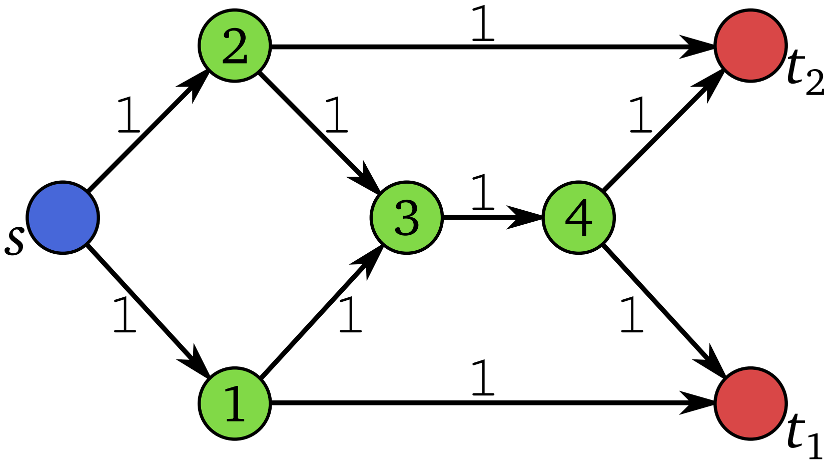
It also has problems that **do** concern us:

- reliability through error detection is hard
  - if a packet is corrupted early on, all the receivers see an error, and all request a resend
  - all the resends come back to the one source
- is copying packets really an efficient way to do this?
  - can we actually achieve the type of max flow described above?

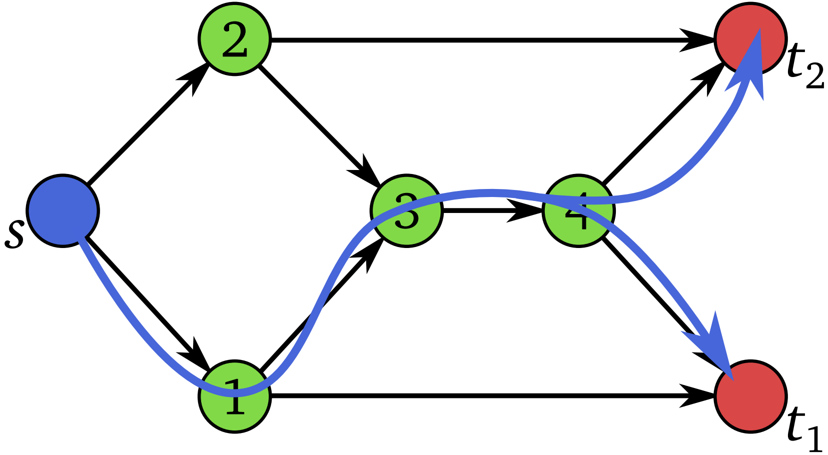
Section 2

Network Coding

Butterfly Network Example [Yeu10, Section 17.1, p.413]

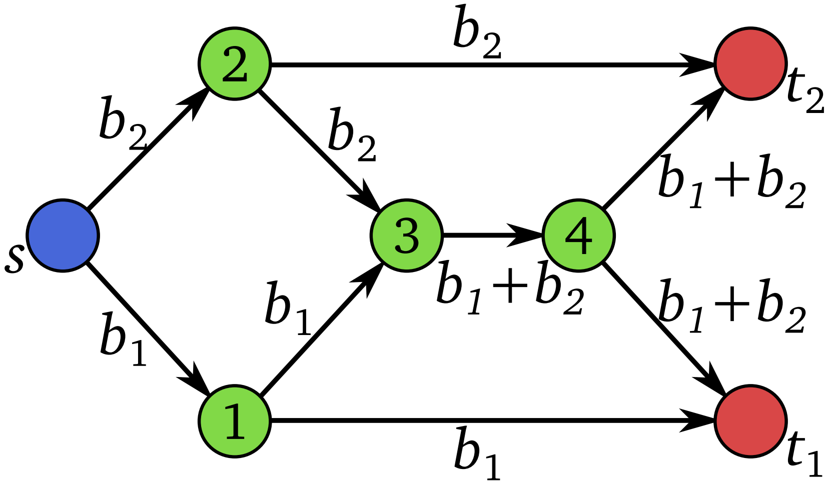


Multicast Solution



- replicate traffic at relevant places to avoid resending

Network Coding Solution

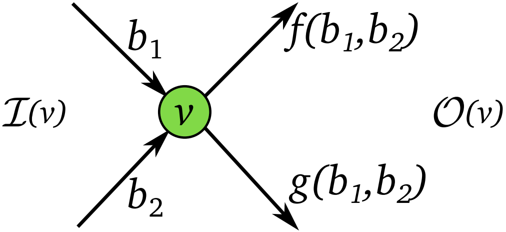


- exploit the full information capacity of network

Setup

- network represented by directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$
  - arcs are lossless with rate 1 symbol per unit time
  - but we allow multiple arcs between two nodes (e.g., its a multigraph)
  - an arc ℓ has destination $d(\ell)$ and origin $o(\ell)$
  - a node v has a set $\mathcal{I}(v)$ of input arcs
  - a node v has a set $\mathcal{O}(v)$ of output arcs
  - the network is acyclic
- Source node $s \in \mathcal{N}$
  - originates r source processes $X_1, \ldots, X_r$
  - each source is a stream of independent random bits, 1 per unit time
- There is a set $\mathcal{T}$ of sink nodes $t_i \in \mathcal{N} \backslash s$
  - all the source processes must be communicated to all $\mathcal{T}$

Basic Operation



- b_1 and b_2 are "packets" of bits, both the same length
- we will look at noiseless networks
- treat it as if everything happens at the same time
- limit $f(\cdot)$ and $g(\cdot)$ to be scalar, linear functions

More notation

- r source processes $X_1, \ldots, X_R$
- refer to bitstream on arc ℓ as arc process and denote it by Y_ℓ
  - the arc process on the output arcs of v will be functions of one or more of the inputs of v
- each sink $t \in \mathcal{T}$ forms output processes $\{Z_{t,i}\}_{i=1}^{r}$
  - a solution to such a problem defines the coding operations at the nodes, and decoding operations at the sinks, such that each sink reconstructs the source processes perfectly, i.e.,

$$Z_{t,i} = X_i, \text{ for all } i = 1, 2, \ldots r, \text{ and } t \in \mathcal{T}$$

Communications Processes

- r source processes $X_1, \ldots, X_R$
  - in groups (packets) of m bits
  - $X_i \in F_q$, where $q = 2^m$
- refer to bitstream on arc ℓ as arc process and denote it by Y_ℓ
  - the arc process on the output arcs of v will be functions of one or more of the inputs of v
- each sink $t \in \mathcal{T}$ forms output processes $\{Z_{t,i}\}_{i=1}^r$
  - a solution to such a problem defines the coding operations at the nodes, and decoding operations at the sinks, such that each sink reconstructs the source processes perfectly, i.e.,

$$Z_{t,i} = X_i, \text{ for all } i = 1, 2, \ldots r, \text{ and } t \in \mathcal{T}$$

- assume no delays, or memory, so we only need to consider one symbol at a time

Scalar Linear Functions

$$
\begin{aligned}
Y_\ell &= \sum_{k \in \mathcal{O}(\ell)} f_{k,\ell} Y_k + \begin{cases} \sum_i a_{i,\ell} X_i, & \text{if } o(\ell) = s \\ 0, & \text{otherwise} \end{cases} \\
Z_{t,i} &= \sum_{k \in \mathcal{O}(t)} b_{t,i,k} Y_k
\end{aligned}
$$

- operations are over the finite field F_q
- $f_{k,\ell}$ and $a_{i,\ell} \in F_q$ are called local coding coefficients
- $b_{t,i,k} \in F_q$ are the decoding coefficients
- represent each set of coefficients by corresponding vector: $\mathbf{f}$, $\mathbf{a}$, $\mathbf{b}$

Scalar Linear Functions

Since all coding operations are scalar linear operations, and the network is acyclic, we can recursively define the arc processes in terms of their ancestors, and ultimately in terms of the sources:

$$Y_\ell = \sum_{i=1}^{r} c_{i,\ell} X_i$$

- the **c** are called the global coding coefficients
- the **c** are a function of **f**, **a**
    - we could recursively calculate these with the formula on previous slide

Scalar Linear Functions

Likewise we can write the output processes as scalar linear operations on the source

$$\begin{aligned} Z_{t,i} &= \sum_{i=1}^{r} m_{t,i,k} X_k \\ \mathbf{z}_t &= \mathbf{x} M_t \end{aligned}$$

- $\mathbf{z} = [Z_{t,1}, \ldots, Z_{t,r}]$
- $M_t = [m_{t,i,k}]$ is a function of $\mathbf{f}$, $\mathbf{a}$, and $\mathbf{b}$, which can be calculated using

$$M_t = A(I - F)^{-1} B_t^T$$

Scalar Linear Functions

Likewise we can write the output processes as scalar linear operations on the source

$$\begin{aligned} \mathbf{z}_t &= \mathbf{x} M_t \\ M_t &= A(I-F)^{-1} B_t^T \end{aligned}$$

- $A = [a_{i,\ell}]$ is a $r \times |\mathcal{A}|$ transfer matrix of coding coefficients that transfer sources to source output arcs.
- $F = [f_{k,\ell}]$ is a $|\mathcal{A} \times |\mathcal{A}|$ transfer matrix of coding coefficients that $d(k)$ transfers its inputs into outputs (with zeros where arcs don't exists). The matrix F^n gives the transfer function along n-hop paths. Thus
  $$(1-F)^{-1} = I + F + F^2 + \cdots$$
  gives the transfer function along all possible paths.
- $B = [b_{i,\ell}]$ is a $r \times |\mathcal{A}|$ transfer matrix in which the sinks transform their inputs into the output processes.

Scalar Linear Functions

Likewise we can write the output processes as scalar linear operations on the source

$$\begin{aligned} \mathbf{z}_t &= \mathbf{x} M_t \\ M_t &= A(I-F)^{-1} B_t^T \end{aligned}$$

- The matrices (or originally vectors) A, F and B specify the scalar linear network code
- We are looking for codes such that for all $t \in \mathcal{T}$

$$M_t = A(I-F)^{-1} B_t^T = I$$

Solvability and Throughput

- How do we determine the maximum possible rate, or decide if a given problem is solvable?
- How would we construct a solution?
- What is the throughput advantage over conventional packets/multicast?

Solvability and Throughput

Generalisation of the max-flow/min-cut theorem

Theorem

The following three statements are equivalent:

1. *There exists a flow rate r between s and t.*
2. *The capacity of the minimum cutset between s and t is at least r.*
3. *The determinant of the transfer matrix M_t is non-zero over the ring of polynomials in $\mathbb{F}_2[\mathbf{a}, \mathbf{f}, \mathbf{b}]$.*

- So this essentially says that we can achieve the maximum flow using the type of coding described above.
- We know we can't achieve it (at least in the butterfly network example) with simple packet-based multicast, so there is some improvement here.
- There are generalisations to multicast, with the same basic implication.

Code Construction

- The obvious next question is how to construct such a coding scheme
- Variants
  - centralised vs decentralised/distributed
  - generic vs packet
  - deterministic vs random
  - acyclic vs cyclic
  - ...
- We'll do unicast but generalise

Code Construction: centralised

- Solvable multicast on known acyclic network (as above)
  - r source processes (as above)
  - $d = |\mathcal{T}|$ sink nodes
  - code on the finite field $\mathbb{F}_q$
- We have a deterministic algorithm that
  - runs in $O\big(|\mathcal{E}|dr(d+r)\big)$ time
  - achieves the bound for $q = 2^m \geq d$, i.e., 2 to the block length is at least the same as the number of sinks
    - ★ e.g., for the butterfly network $d = 2$, and so we only need size 1 blocks (which we saw)

Code Construction: centralised algorithm

Deterministic algorithm sketch:

1. Find r arc-disjoint paths $P_{t,1}, \ldots, P_{t,r}$ from s to each sink $t \in \mathcal{T}$
    - Take $\cup_{t \in \mathcal{T}, i=1\ldots r} P_{t,i} = \mathcal{A} \subset \mathcal{E}$, i.e., the set of arcs on these paths, then we don't need the others to support the flow, so only look at $\mathcal{G}'(\mathcal{N}, \mathcal{A})$, and all other coding coefficients are zero.
2. Set coding coefficients of $\mathcal{G}'(\mathcal{N}, \mathcal{A})$ in topological order
    - set $S_t =$ the set of arcs from each path $P_{t,1}, \ldots, P_{t,r}$ whose coding coefficients were most recently set
    - must have: for each sink t, the coding vectors $\mathbf{c}_\ell$ of the arcs ℓ in the set S_t form a basis for $\mathbb{F}_q^r$
    - Various tricks (see [HL08, pp.21-23]) are used to allocate the coefficients $\mathbf{c}$ in order such that this condition is satisfied

Code Construction: random codes

- Randomly choose $(\mathbf{a}, \mathbf{f})$ from sufficiently large finite field
  - $(\mathbf{a}, \mathbf{f})$ determines $\mathbf{c}_\ell$ for each arc ℓ by taking ℓth column of $C = A(I-F)^{-1}$
  - $\mathbf{b}_t$ given from B_t given such that $M_t = A(I-F)^{-1}B_t^T = I$
- Finds a solution with high probability

$$Pr(\text{good code}) \geq \left(1 - \frac{d}{q}\right)^{\eta}$$

where η is the number of arcs ℓ with associated random coding coefficients.

  - e.g., in butterfly example $d = 2$, so take take blocks of 2 symbols, and $q = 2^2 = 4$, and put random coefficients on all arcs so $\eta = 9$, and then

$$Pr(\text{good code}) \geq (1/2)^9 \simeq 0.002$$

Code Construction: packet networks

- Problem: packet networks don't have a good way to distribute the coding information
  - e.g., most traditional routing protocols are distributed
- With fixed sized packets, treat each as *m* bit message block
  - *r* exogenous source packets (per unit time)
- Use random coding, coefficients decided locally
  - sinks need to know how to decode
  - use a "pilot" tone
    - for a batch of *r* source packets
    - add to the header of the *i*th header $\mathbf{e}_i$
    - when it gets to the sink, each of the headers will have been converted into the corresponding coding vectors
  - decode by computing inverse of coding transfer matrix
- Introduces overhead
  - *mr* extra bits per sink
  - need to amortise this over multiple packets, or have packets consisting of multiple message blocks

Extensions:

There are many extensions:

- models of temporal dependency
- noisy links
- security extensions
- multiple sources

we just touched on the surface of the topic.
But its all pretty recent, and I don't know any practical implementations that are actually used (yet).

Source Separation

Imagine we want to transmit two sources X and Y (noiselessly)

- If we compress sources separately we need $H(X) + H(Y)$ bits
  - call this source separation
- Joint compression bits =

$$H(X, Y) \leq H(X) + H(Y)$$

  with equality iff they are independent
  - independence means there is no advantage to joint compression (asymptotically)
- But for network coding, we coding two sources together is helpful
  - we don't have the same source/channel separation we had before
  - compressing the signal, and then sending it over the channel isn't optimal

Assignment

- Take the butterfly example above, and simulate for $q = 2$ and $q = 3$ all of the possible codings for this network, and from this enumeration calculate the exact probability that the code is workable.
- Compare this to the estimated probability of getting a valid random code.

Further reading I

- Tracey Ho and Desmond S. Lun, *Network coding: An introduction*, Cambridge University Press, 2008.

- Raymond W. Yeung, *Information theory and network coding*, Springer, 2010.