Network Flows for Functions

D. Manjunath
[Joint work with Virag Shah and Bikash Kumar Dey]

IIT Bombay

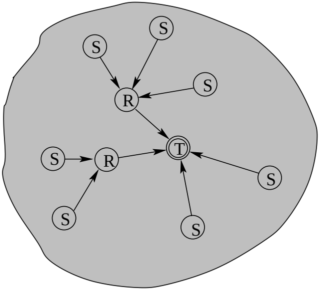
NCC 2011; 29 Jan 2011

Outline

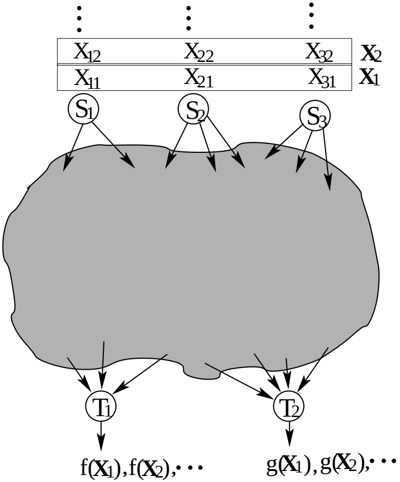
- The problem setup
- The LP formulations and algorithms
- Extensions and open problems
- Other works
- Summary

The setting

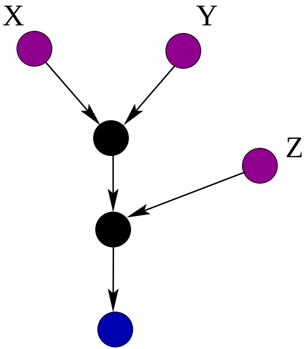
- Terminal (aka sink) nodes want to recover functions of data from distributed sources. Example of functions: $\max, \min, average, etc.$
- Application example: Average temperature sensed by many sensors, average/maximum traffic at different parts of a network.



The setting

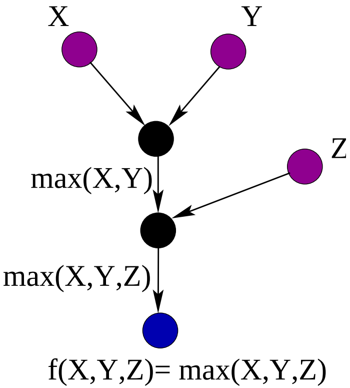


Computing function: a simple example



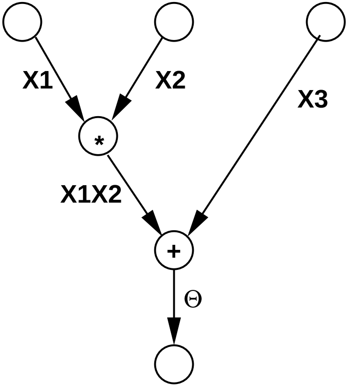
f(X,Y,Z)= max(X,Y,Z)

Computing function: a simple example



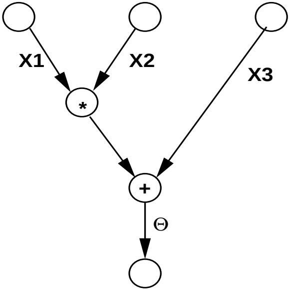
- Computing at internal nodes is a natural choice—distributed computation of distributed data.
- We assume: no mixing of data across different realizations.

Computation tree: $\mathcal{G}$



Computation tree of $\Theta(X_1, X_2, X_3) = X_1X_2 + X_3$

Computation tree: $\mathcal{G}$



- A single computation tree may serve different functions.
- Our techniques depend only on the computation tree and not on the function.
- A single function may allow multiple computationt trees; we start by assuming a single computation tree and generalise to multiple trees.

$$\Theta(X_1, X_2, X_3) = X_1 X_2 + X_3$$

OR

$$(X_1 + X_2) X_3$$

OR

$$\vdots$$

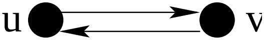
Our setup

- A *single* terminal wants to compute a *single* function of the distributed using a given computation tree. Many generalizations will follow.

Our setup

- A *single* terminal wants to compute a *single* function of the distributed using a given computation tree. Many generalizations will follow.

- The network has undirected half-duplex links with a total capacity constraint. Easily applicable to directed networks.

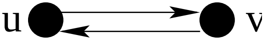


total flow $\leq c(uv)$

Our setup

- A *single* terminal wants to compute a *single* function of the distributed using a given computation tree. Many generalizations will follow.

- The network has undirected half-duplex links with a total capacity constraint. Easily applicable to directed networks.

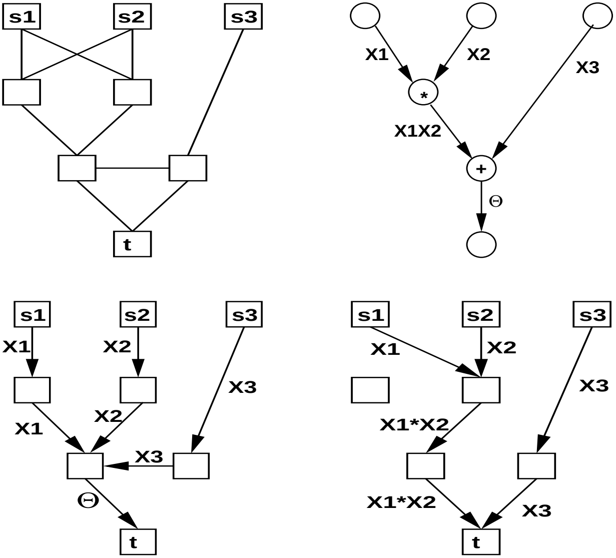


total flow $\leq c(uv)$

- Objective: to find the *maximum computation rate* per use of the network and to find an *optimum computation and communication scheme*.

Embedding: Illustration for $\Theta = X_1 X_2 + X_3$

$\mathcal{N}$ and $\mathcal{G}$ and two possible embeddings.



Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \cdots, v_l;\ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l-1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.

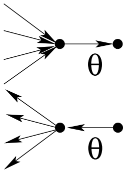
Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \cdots, v_l;\ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l-1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.

$$\text{Define } \Phi_\uparrow(\theta) \triangleq \{\eta \text{ßΓ} | \textit{head}(\eta) = \textit{tail}(\theta)\} \text{ and}$$

$$\Phi_\downarrow(\theta) \triangleq \{\eta \in \Gamma | \textit{tail}(\eta) = \textit{head}(\theta)\}.$$



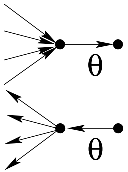
Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \cdots, v_l;\ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l-1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.

$$\text{Define } \Phi_{\uparrow}(\theta) \triangleq \{\eta \text{ß} \Gamma | \textit{head}(\eta) = \textit{tail}(\theta)\} \text{ and}$$

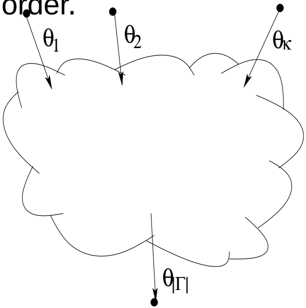
$$\Phi_{\downarrow}(\theta) \triangleq \{\eta \in \Gamma | \textit{tail}(\eta) = \textit{head}(\theta)\}.$$



The edges of $\mathcal{G}$ are ordered in a topological order.

Definition: An *embedding* of $\mathcal{G}$ into $\mathcal{N}$ is a map $B : \Gamma \to \mathcal{P}$ such that

1. $\text{start}(B(\theta_l)) = s_l$ for $l = 1, 2, \ldots, \kappa$
2. $\text{end}(B(\eta)) = \text{start}(B(\theta))$ if $\eta \in \Phi_{\uparrow}(\theta)$
3. $\text{end}(B(\theta_{|\Gamma|})) = t$.



Embedding-Edge LP

- What is the best time-sharing between the different embeddings?

Embedding-Edge LP: Maximize $\lambda = \sum_{B \in \mathcal{B}} x(B)$ subject to

1. Capacity constraints

$$\sum_{B \in \mathcal{B}} r_B(e) x(B) \leq c(e), \ \forall e \in E \tag{1}$$

2. Non-negativity constraints

$$x(B) \geq 0, \ \forall B \tag{2}$$

$r_B(e) = \#$ of times that network edge e of $\mathcal{N}$ is used in embedding B.

Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of λ.

Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of λ.
- $|\mathcal{B}|$ is exponential in $|V|$. So this LP has exponential complexity.

Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of λ.
- $|\mathcal{B}|$ is exponential in $|V|$. So this LP has exponential complexity.
- We need to seek simpler solutions
  - We can identify each edge of $\mathcal{G}$ to be a flow.

Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of λ.
- $|\mathcal{B}|$ is exponential in $|V|$. So this LP has exponential complexity.
- We need to seek simpler solutions
  - We can identify each edge of $\mathcal{G}$ to be a flow.
  - We can thus explore an efficient *Node-Arc LP* based on "flow conservation."

Flow-conservation

- A complication: Flow can be destroyed or generated at internal nodes.

Flow-conservation

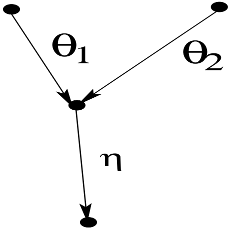
- A complication: Flow can be destroyed or generated at internal nodes.

- A destroyed flow of type θ_1 or θ_2 generates a flow of type η of the same volume.

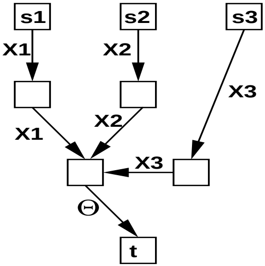
Flow-conservation

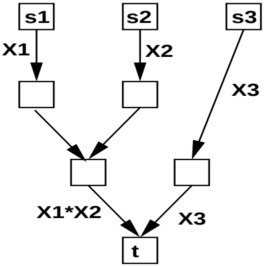
- A complication: Flow can be destroyed or generated at internal nodes.

- A destroyed flow of type θ_1 or θ_2 generates a flow of type η of the same volume.
- A flow generated at a node is assumed to flow on a virtual self-loop at that node.
- The flow in the self-loop contributes to the incoming flow, but not to the outgoing flow.

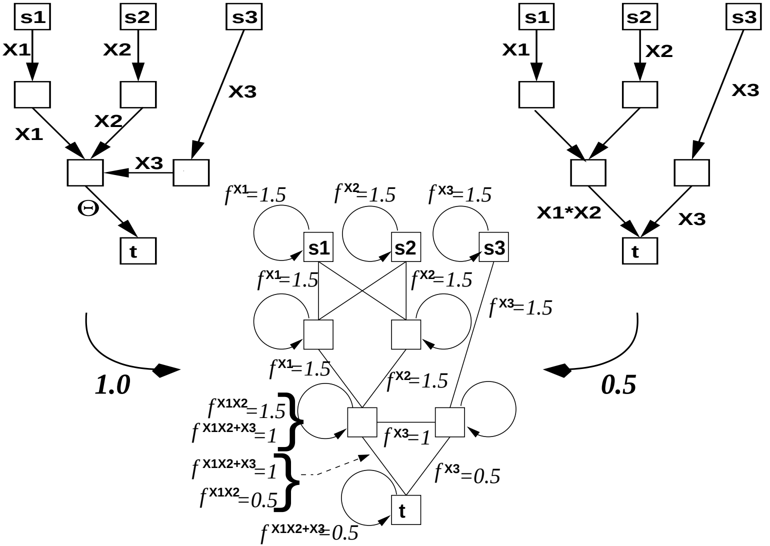


Flow-conservation





Flow-conservation



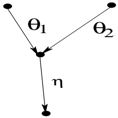
Node-Arc LP

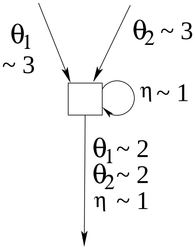
Node-Arc LP: Maximize λ subject to following constraints. For any node $v \in V$

1. Functional conservation of flows:

$$f_{vv}^{\eta} + \sum_{u \in N(v)} f_{vu}^{\theta} - \sum_{u \in N'(v)} f_{uv}^{\theta} = 0,$$

$$\forall \theta \in \Gamma \setminus \{\theta_{|\Gamma|}\} \text{ and } \forall \eta \in \Phi_{\downarrow}(\theta).$$





2. Conservation and termination of $\theta_{|\Gamma|}$:

$$\sum_{u \in N(v)} f_{vu}^{\theta_{|\Gamma|}} - \sum_{u \in N'(v)} f_{uv}^{\theta_{|\Gamma|}} = \begin{cases} -\lambda & v = t \\ 0. & \text{otherwise} \end{cases}$$

Node-Arc LP

3. Generation of θ_l $\forall l \in \{1, 2, \ldots, \kappa\}$:

$$f_{vv}^{\theta_l} = \begin{cases} \lambda & v = s_l \\ 0. & \text{otherwise} \end{cases}$$

4. Capacity constraints

$$\sum_{\theta \in \Gamma} \left(f_{uv}^{\theta} + f_{vu}^{\theta} \right) \leq c(uv), \ \forall uv \in E.$$

5. Non-negativity constraints

$$f_{uv}^{\theta} \geq 0, \forall uv \in E \text{ and } \forall \theta \in \Gamma$$
$$f_{uu}^{\theta} \geq 0, \forall u \in V \text{ and } \forall \theta \in \Gamma$$
$$\lambda \geq 0.$$

Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.

Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.
- An algorithm *Extract-Embedding* converts a solution of *Node-Arc LP* to a solution of *Embedding-Edge LP*.

Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.
- An algorithm *Extract-Embedding* converts a solution of *Node-Arc LP* to a solution of *Embedding-Edge LP*.
- In each iteration, *Extract-Embedding* extracts' a non-zero flow on an embedding or a nonzero redundant flow on a cycle.

Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.
- An algorithm *Extract-Embedding* converts a solution of *Node-Arc LP* to a solution of *Embedding-Edge LP*.
- In each iteration, *Extract-Embedding* extracts' a non-zero flow on an embedding or a nonzero redundant flow on a cycle.
- Atleast one flow f_{uv}^{θ} is completely removed in each iteration of *Extract-Embedding.*

Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.
- An algorithm *Extract-Embedding* converts a solution of *Node-Arc LP* to a solution of *Embedding-Edge LP*.
- In each iteration, *Extract-Embedding* extracts' a non-zero flow on an embedding or a nonzero redundant flow on a cycle.
- Atleast one flow f_{uv}^{θ} is completely removed in each iteration of *Extract-Embedding.*
- The algorithm has the overall complexity $O(\kappa^2|E|^2)$.
- The LP has $O(\kappa|E|)$ number of variables, $O(\kappa|E|)$ number of non-negativity constraints, and $O(\kappa|V| + |E|)$ number of other constraints.

Toward an efficient ϵ-approximate solution

- For multi-commodity flow, and more general packing LPs, Garg and Konemann [1998] gave a primal-dual algorithm to compute a solution which achieves at least $(1 - \epsilon)$ fraction of the optimal rate.
- Our *Embedding-Edge LP* is also such a ‘packing LP’.
- So, Garg-Konemann algorithm can be used for our problem.
- The algorithm uses an oracle subroutine that solves a ‘dual’ problem.

Toward an efficient ϵ-approximate solution

Let $l(e)$ be the weight of edge e. Define the weight of an embedding B as

$$w_L(B) = \sum_{e \in B} r_B(e) l(e).$$

- The oracle subroutine *OptimalEmbedding(L)* finds an embedding with minimum weight for a given set L of edge weights.

Toward an efficient ϵ-approximate solution

Let $l(e)$ be the weight of edge e. Define the weight of an embedding B as

$$w_L(B) = \sum_{e \in B} r_B(e) l(e).$$

- The oracle subroutine *OptimalEmbedding(L)* finds an embedding with minimum weight for a given set L of edge weights.
- We later give an efficient algorithm for doing this.

An efficient ϵ-approximate solution

Algorithm: Algorithm for finding approximately optimal x and λ

Input: Network graph $\mathcal{N} = (V, E)$, capacities $c(e)$, set of source nodes S, terminal node t, computation tree $\mathcal{G} = (\Omega, \Gamma)$, the desired accuracy ϵ

Output: Primal solution $\{x(B), B \in \mathcal{B}\}$

Initialize $l(e) := \delta / c(e), \forall e \in E, x(B) := 0, \forall B \in \mathcal{B}$;

while $D(l) < 1$ **do** % $D(l) = \sum c(e) l(e)$

B^* := OptimalEmbedding(L);

e^* := edge in B^* with smallest $c(e)/r_{B^*}(e)$;

$x(B^*) := x(B^*) + c(e^*)/r_{B^*}(e^*)$;

$l(e) := l(e)(1 + \epsilon \frac{c(e^*)/r_{B^*}(e^*)}{c(e)/r_{B^*}(e)}), \forall e \in B^*$;

end

$x(B) := x(B) / \log_{1+\epsilon} \frac{1+\epsilon}{\delta}, \forall B$;

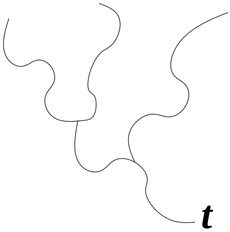
OptimalEmbedding(L): overview

- For each edge θ_i, starting from θ_1, it finds a way to compute θ_i at each network node at the minimum cost possible.
- It keeps track of that minimum cost and also the ‘predecessor’ node from where it receives θ_i.
- If θ_i is computed at that node itself then the predecessor node is itself.

OptimalEmbedding(L)

- Computing θ_i for $i \in \{1, 2, \ldots, \kappa\}$ at the minimum cost at a node u is equivalent to finding the shortest path to u from s_i. We do this by using Dijkstra's algorithm.
- For any other i, the node u can either compute θ_i from $\Phi_\uparrow(\theta_i)$ or receive it from one of its neighbors.

- To take this into account, unlike Dijkstra's algorithm, we initialize the cost of computing θ_i with the cost of computing $\Phi_\uparrow(\theta_i)$ at the same node. The rest is similar to Dijkstra's algorithm.
- Finally the predecessors are backtracked from t to find the optimal embedding.



Complexity

- Overall complexity of *OptimalEmbedding(L)*: $O(\kappa(|E| + |V| \log |V|))$
- The number of iterations in the primal-dual algorithm is of the order $O(\epsilon^{-1}|E| \log_{1+\epsilon} |E|)$.
- Thus the overall complexity of the primal-dual algorithm is $O\left(\epsilon^{-1}\kappa|E|(|E| + |V| \log |V|) \log_{1+\epsilon} |E|\right)$.

Extensions and Open Problems

Extensions

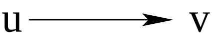
- Multiple trees for the same function.
- Multiple terminals and functions of distinct sources.
- Computing with a specified precision.
- Consider energy limited sensors.

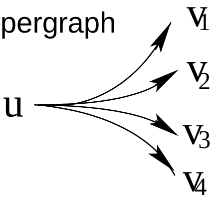
Open problems

- An immediate open problem: The computation graph $\mathcal{G}$ is a DAG and not a tree.

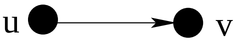
In perspective: Other setups for function networks

- Wired/wireless networks: Graph or hypergraph

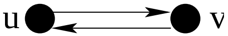




- Directed or undirected links



flow $\leq c(uv)$



total flow $\leq c(uv)$

- block computation/coding vs. bit-wise computation
- zero-error recovery vs. small-error recovery
- correlated vs. independent sources
- single terminal vs. multiple terminals
- same vs. different functions at different terminals
- fixed vs. random networks

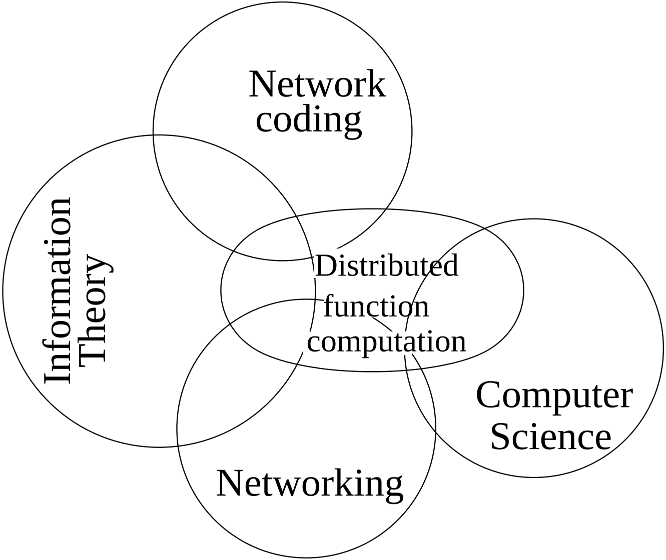
Typical objectives

- For any given link capacities, what is the maximum rate (or rate-region for multiple terminals) that is achievable? More generally, rate-distortion trade-off?
- Assymptotic scaling laws for required communication complexity per node
- Efficient encoding/decoding

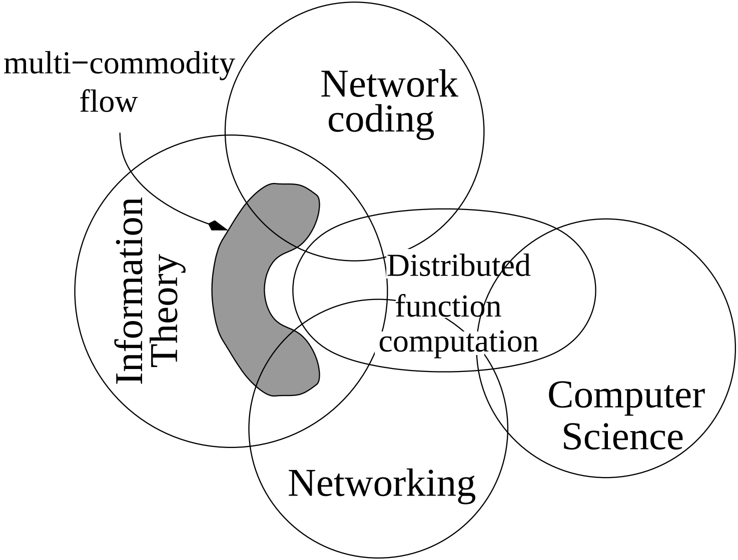
Summary of other views

- In information theory: Small networks, correlated sources with the objective of finding achievable rate-region and rate-distortion.
- Scaling laws for randomly deployed networks: Does not consider a fixed network.
- Network coding: internal nodes are allowed to *mix* received data to construct outgoing data even for communication.

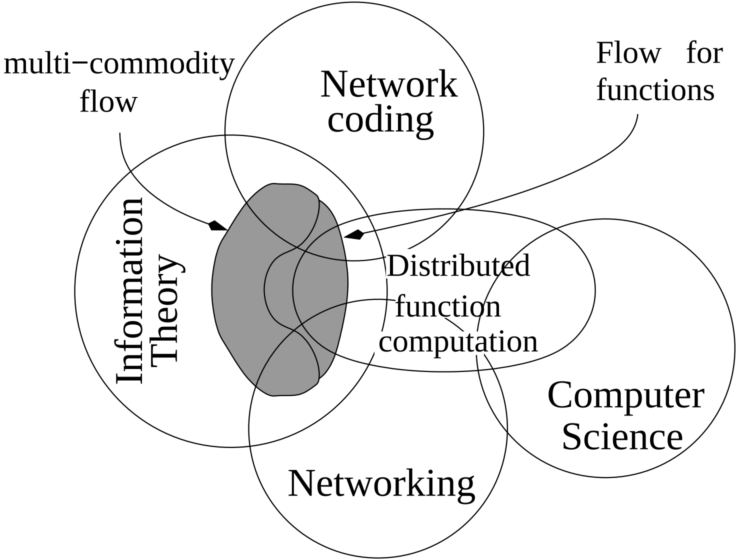
The big picture



The big picture



The big picture



The End

The End

Thank you

The End

Thank you

Questions?