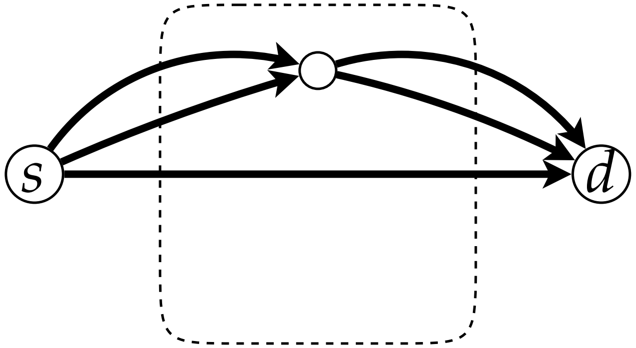
Pruning Network Coding Traffic By Network Coding

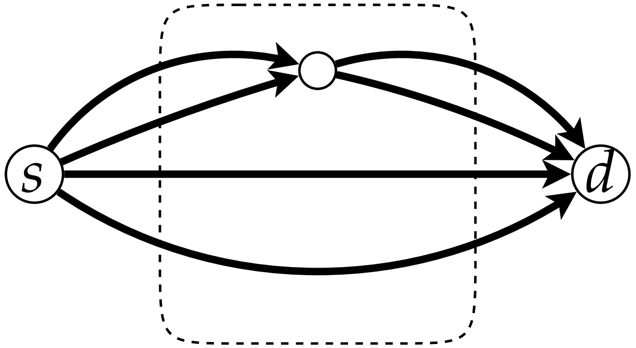
A New Class of Max-Flow Algorithms

Chih-Chun Wang
Center for Wireless Systems and Applications
School of ECE
Purdue University

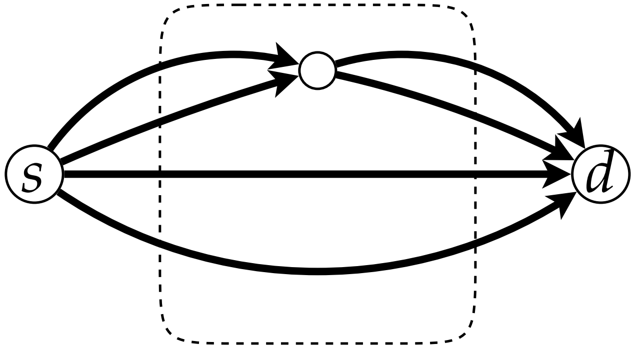
(s, d)-Flow, max (s, d)-flow, and the max-flow value (MFV).



(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

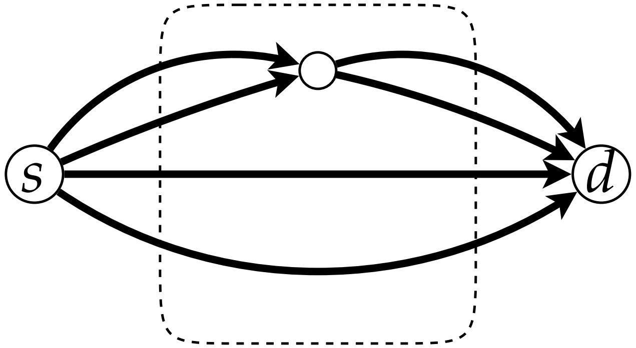


(s, d)-Flow, max (s, d)-flow, and the max-flow value (MFV).



Why study the max flow problem?

(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

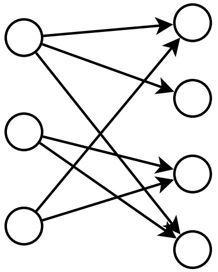


Why study the max flow problem?

- CS: a classic optimization problem:

(s, d)-Flow, max (s, d)-flow, and the max-flow value (MFV).

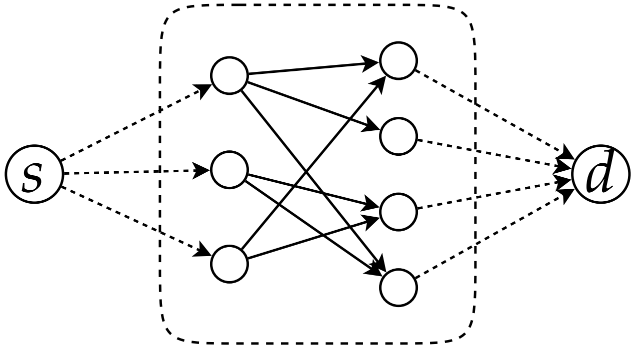
Why study the max flow problem?



- CS: a classic optimization problem:

  ex: finding maximum matching,

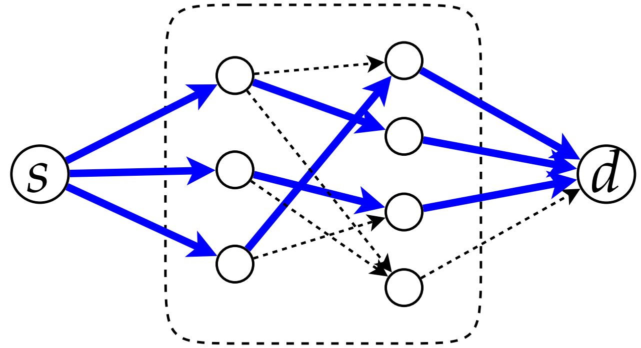
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



Why study the max flow problem?

- CS: a classic optimization problem:
  ex: finding maximum matching,

(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

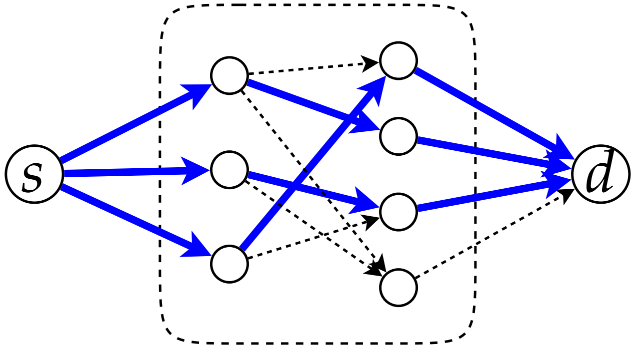


Why study the max flow problem?

- CS: a classic optimization problem:

  ex: finding maximum matching,

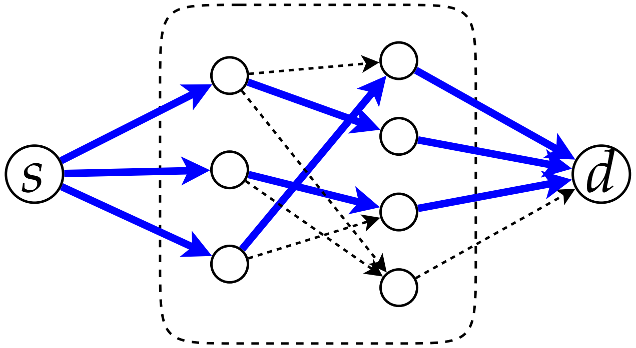
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



Why study the max flow problem?

- CS: a classic optimization problem:
  ex: finding maximum matching,
  finding the minimum separation (min-cut).

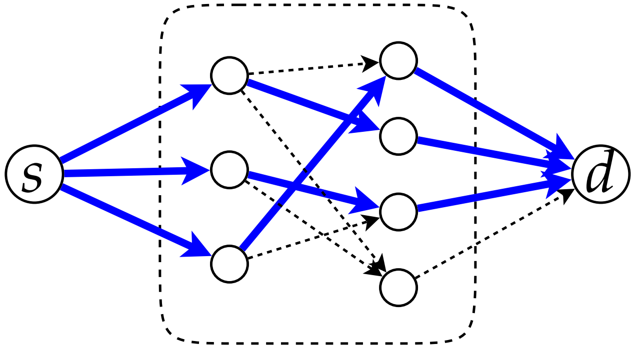
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



Why study the max flow problem?

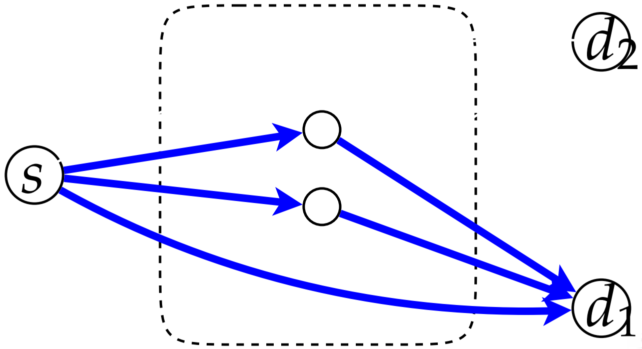
- CS: a classic optimization problem:
  ex: finding maximum matching,
  finding the minimum separation (min-cut).
- EE: Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \mathrm{MFV}_i$ for all source-destination pairs (s,d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]

(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

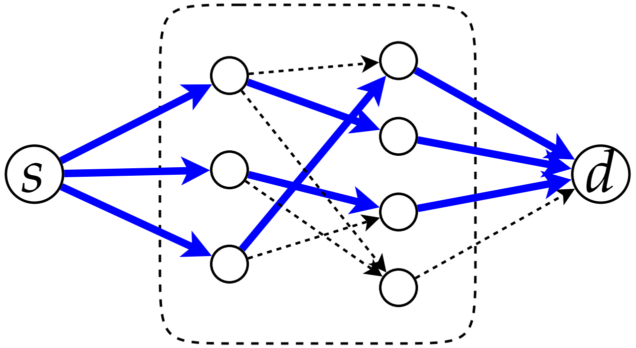


Why study the max flow problem?

- CS: a classic optimization problem:
  ex: finding maximum matching,
  finding the minimum separation (min-cut).
- EE: Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \mathrm{MFV}_i$ for all source-destination pairs (s, d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]

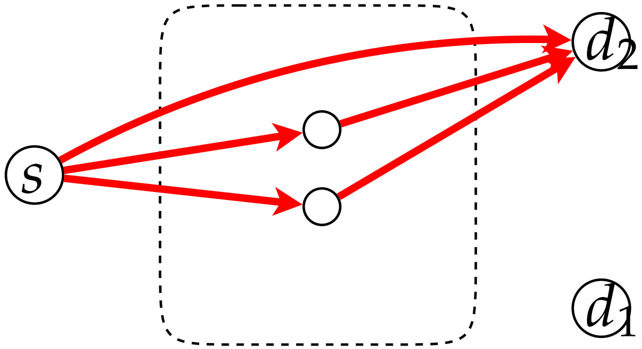


(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

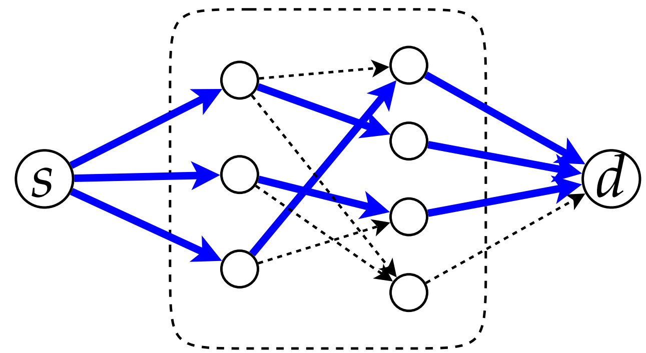


Why study the max flow problem?

- CS: a classic optimization problem:
  ex: finding maximum matching,
  finding the minimum separation (min-cut).
- EE: Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \text{MFV}_i$ for all source-destination pairs (s, d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]

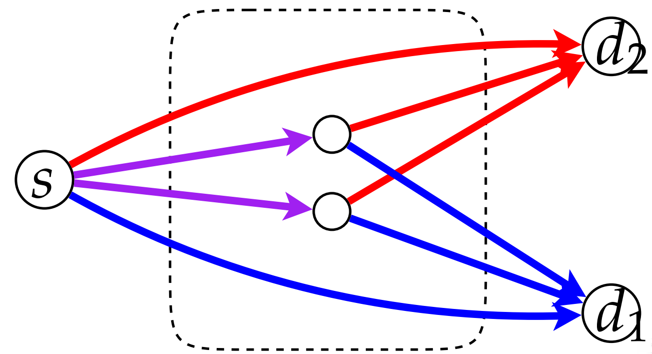


(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



Why study the max flow problem?

- CS: a classic optimization problem:
  ex: finding maximum matching,
  finding the minimum separation (min-cut).
- EE: Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \mathrm{MFV}_i$ for all source-destination pairs (s, d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]



Existing Max-Flow Algorithms

- Linear-programming (LP) based max-flow algorithms
  - $$\max_{f_e \geq 0} \sum_{e \in \mathrm{Out}(s)} f_e$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$

Existing Max-Flow Algorithms

- Linear-programming (LP) based max-flow algorithms
  - $$\max_{f_e \geq 0} \quad \sum_{e \in \mathrm{Out}(s)} f_e$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Suitable for different objective functions, ex: $\min \sum_e c_e$.

Existing Max-Flow Algorithms

- Linear-programming (LP) based max-flow algorithms
  - $$\max_{f_e \geq 0} \sum_{e \in \mathrm{Out}(s)} f_e$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Suitable for different objective functions, ex: $\min \sum_e c_e$.
  - Complexity: queue-length exchange,

Existing Max-Flow Algorithms

- Linear-programming (LP) based max-flow algorithms
  - $$\max_{f_e \geq 0} \sum_{e \in \mathrm{Out}(s)} f_e$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Suitable for different objective functions, ex: $\min \sum_e c_e$.
  - Complexity: queue-length exchange,
  - Convergence speed: small step sizes of the gradient methods,

Existing Max-Flow Algorithms

- Linear-programming (LP) based max-flow algorithms
  - $$\max_{f_e \geq 0} \sum_{e \in \mathrm{Out}(s)} f_e$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Suitable for different objective functions, ex: $\min \sum_e c_e$.
  - Complexity: queue-length exchange,
  - Convergence speed: small step sizes of the gradient methods,
  - Fractional rate vs. packet-by-packet coding operations.
    - Time-averaging? Practical generation size (# of to-be-mixed packets) is 30–100.

Existing Max-Flow Algorithms

- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.

Existing Max-Flow Algorithms

- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:

Existing Max-Flow Algorithms

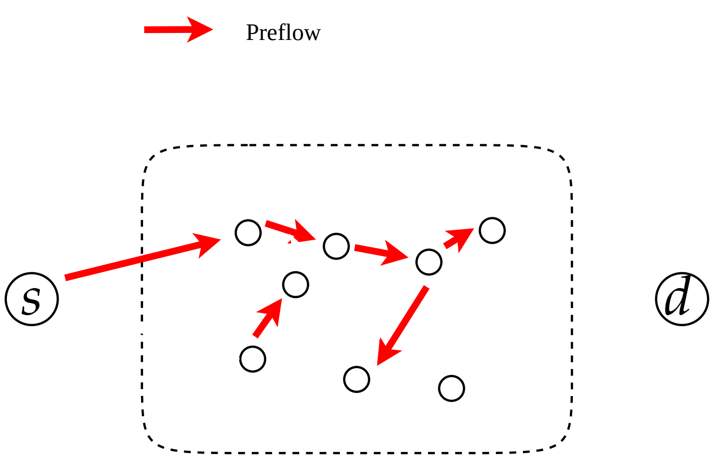
- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.

Existing Max-Flow Algorithms

- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.
    - Based on the non-coded paradigm.
    - “Preflows" are not allowed to be mixed with each other.

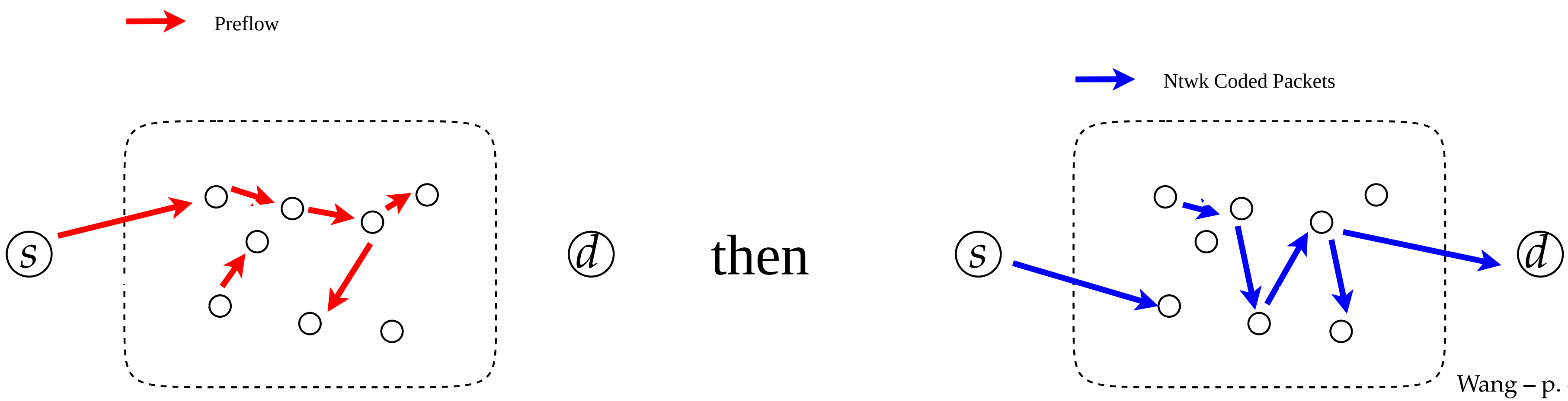
Existing Max-Flow Algorithms

- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.
    - Based on the non-coded paradigm.
    - “Preflows" are not allowed to be mixed with each other.



Existing Max-Flow Algorithms

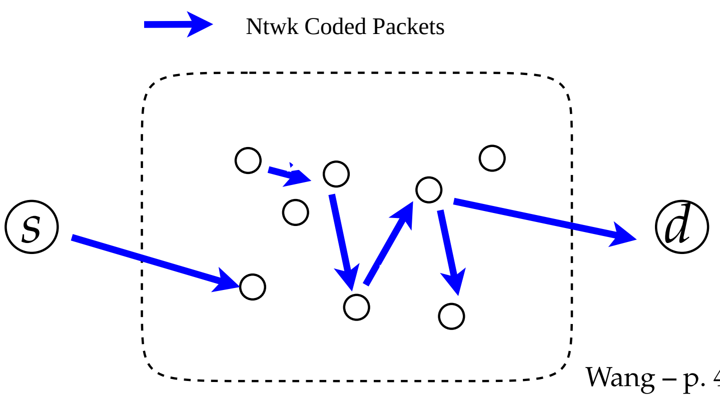
- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.
    - Based on the non-coded paradigm.
    - “Preflows" are not allowed to be mixed with each other.



Existing Max-Flow Algorithms

- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.
    - Based on the non-coded paradigm.
    - “Preflows" are not allowed to be mixed with each other.

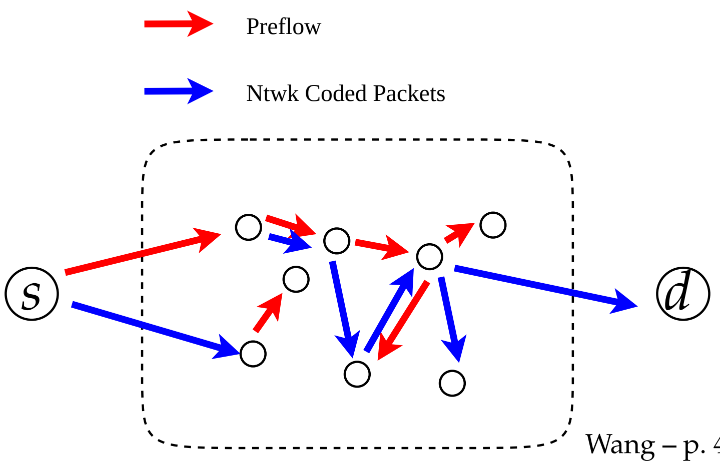
or



Existing Max-Flow Algorithms

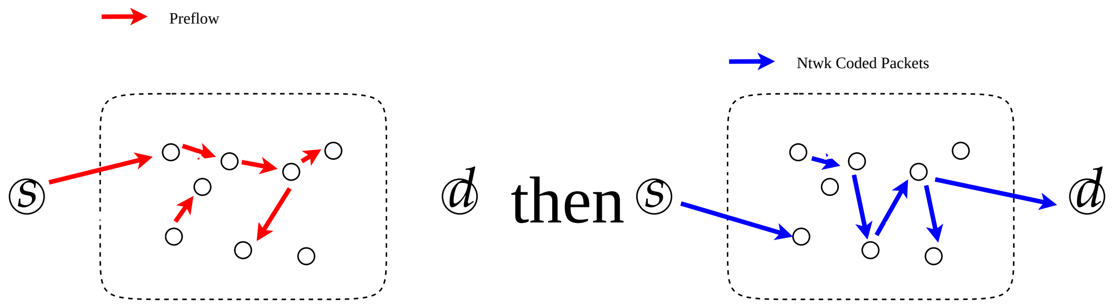
- Graph-theoretic max-flow algorithms
  - Ford-Fulkerson 1956: Residue graph vs. augmenting path
  - Edmonds-Karp 1972: Breadth-first search + FF
  - Dinitz blocking flow algorithm 1970.
  - Push & relabel algorithm [Goldberg, Tarjan 1988]:
    - Fully distributed implementation.
    - Based on the non-coded paradigm.
    - “Preflows" are not allowed to be mixed with each other.

or



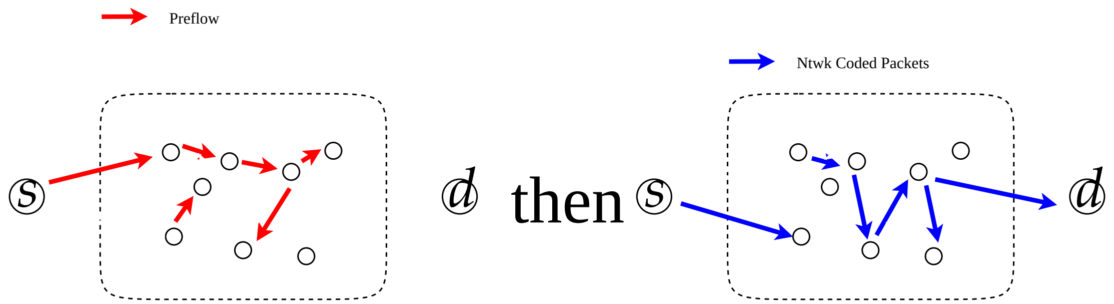
Delay Minimality of NC

- The sequential approach:



Delay Minimality of NC

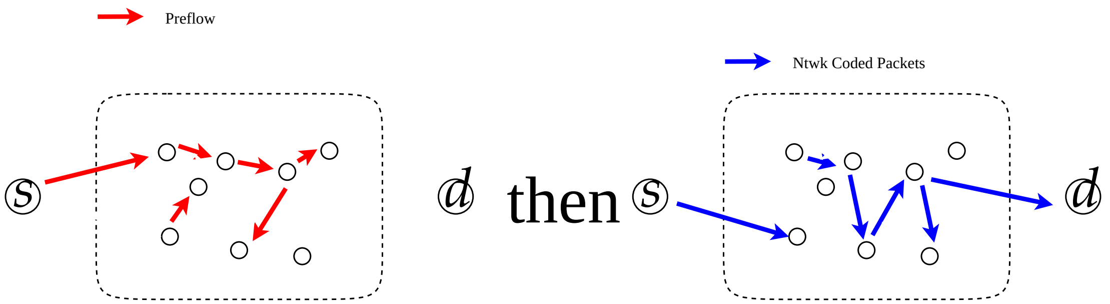
- The sequential approach:



  - Induces delay

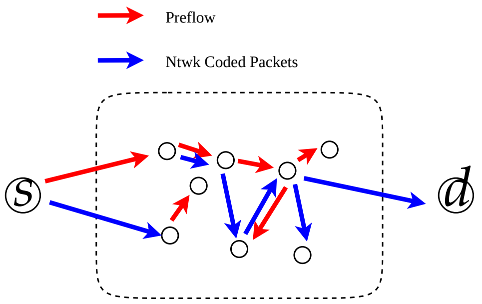
Delay Minimality of NC

- The sequential approach:



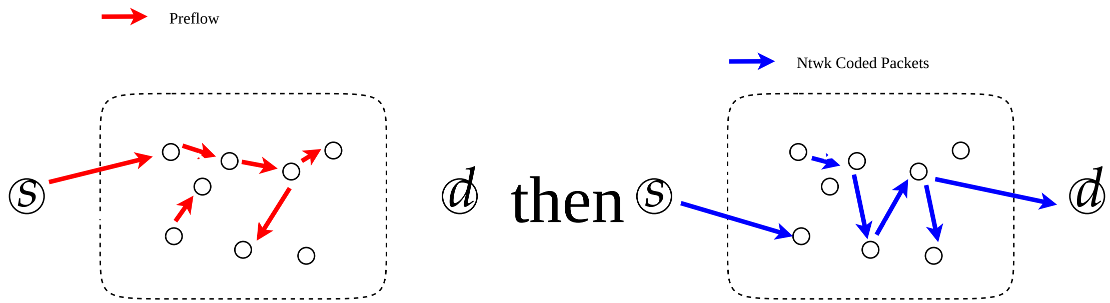
  - Induces delay

- The parallel approach reduces the delay:

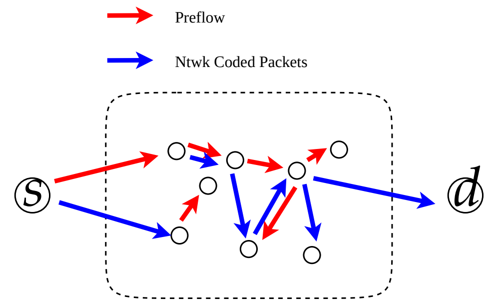


Delay Minimality of NC

- The sequential approach:
  - Induces delay

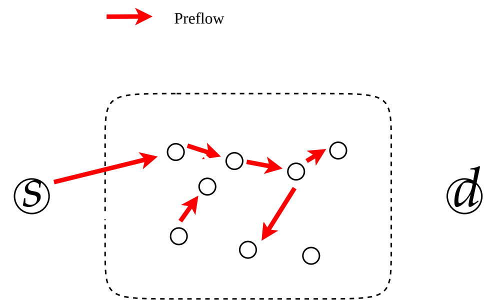


- The parallel approach reduces the delay:
  - NC achieves the min-cut max-flow rate without knowing the max flow.
  - One simply performs random mixing + broadcasting.
  - Network coding is delay-optimal.

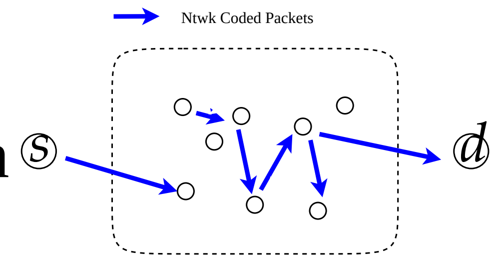


Delay Minimality of NC

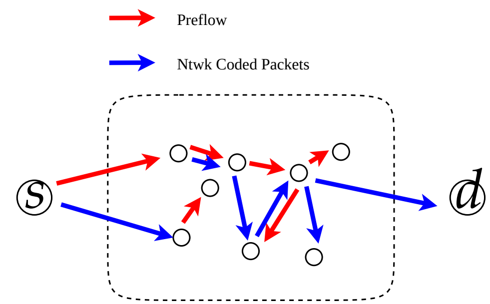
- The sequential approach:



then



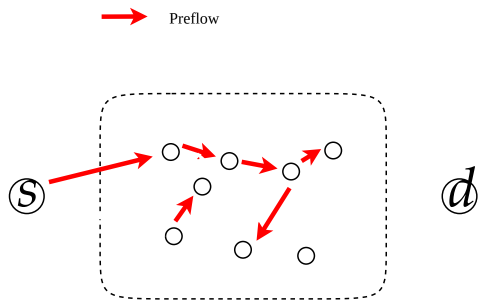
  - Induces delay
- The parallel approach reduces the delay:



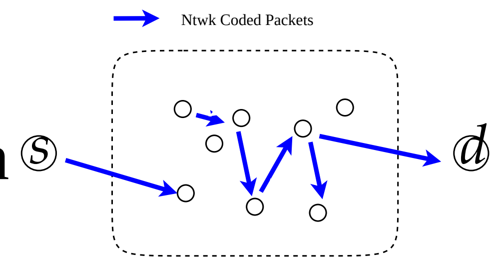
  - NC achieves the min-cut max-flow rate without knowing the max flow.
  - One simply performs random mixing + broadcasting.
  - Network coding is delay-optimal.
  - Coding eliminates the need to decide which edge to send.

Delay Minimality of NC

- The sequential approach:

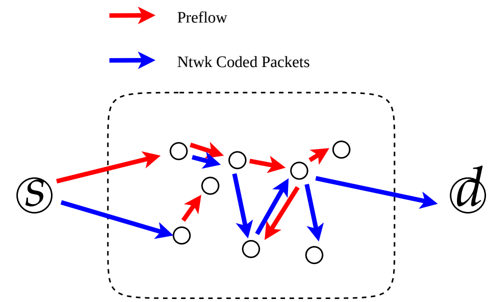


then



  - Induces delay

- The parallel approach reduces the delay:



  - NC achieves the min-cut max-flow rate without knowing the max flow.
  - One simply performs random mixing + broadcasting.
  - Network coding is delay-optimal.
  - Coding eliminates the need to decide which edge to send.
  - Significant control and communication overhead.

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach:

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach:
  Run network coding

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach:
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges

  Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality

- A new coding-theoretic approach:
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach: Delay optimal.
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

  Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality

- A new coding-theoretic approach: Delay optimal.
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

  Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.

- Comparison to Ford-Fulkerson:
  Start from an empty subgraph

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach: Delay optimal.
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.
- Comparison to Ford-Fulkerson:
  Start from an empty subgraph $\longrightarrow$ Repeatedly add augmenting paths

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach: Delay optimal.
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

  Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.
- Comparison to Ford-Fulkerson:
  Start from an empty subgraph $\longrightarrow$ Repeatedly add augmenting paths $\longrightarrow$ A max flow.

- Classic sequential graph-theoretic approach:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality

- A new coding-theoretic approach: Delay optimal.
  Run network coding $\longrightarrow$ Repeatedly stop the traffic on redundant edges $\longrightarrow$ Bandwidth optimality

Redundant edges are the edges such that the removal of which will not interrupt the network coded traffic.

- ...ing

The key question: How to find distributedly the redundant edges?

The Integer-Rate Ntwk Model

- Finite directed acyclic graph $G = (V, E)$.

The Integer-Rate Ntwk Model

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.

The Integer-Rate Ntwk Model

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.
- A single unicast session (s, d): Intrasession network coding

The Integer-Rate Ntwk Model

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.
- A single unicast session (s, d): Intrasession network coding
- Coding vector $m = (c_1, c_2, c_3) \Longleftrightarrow X = c_1 X_1 + c_2 X_2 + c_3 X_3$.

The Integer-Rate Ntwk Model

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.
- A single unicast session (s, d): Intrasession network coding
- Coding vector $m = (c_1, c_2, c_3) \Longleftrightarrow X = c_1X_1 + c_2X_2 + c_3X_3$.
- Arbitrary GF(q), ex: $q = 2^1, 2^8, 2^{16}$ or $q = 3$.

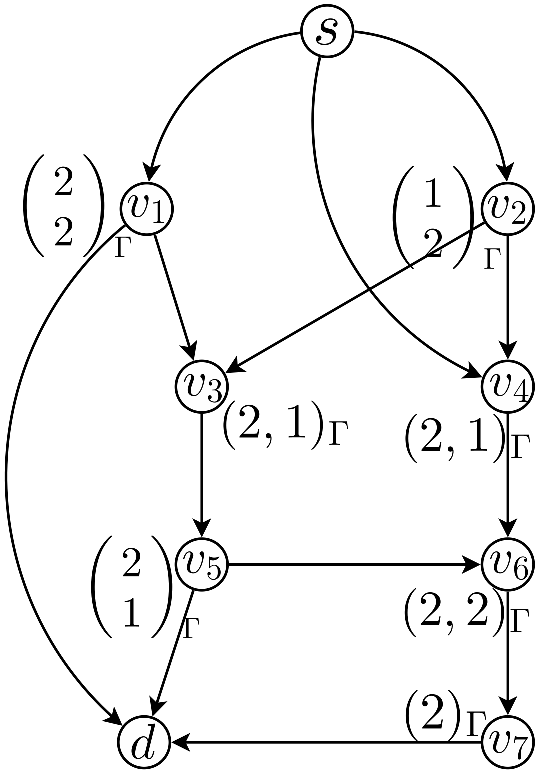
The Coded Feedback Approach

Network coding on $\mathrm{GF}(3)$

The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Network coding on $\mathsf{GF}(3)$

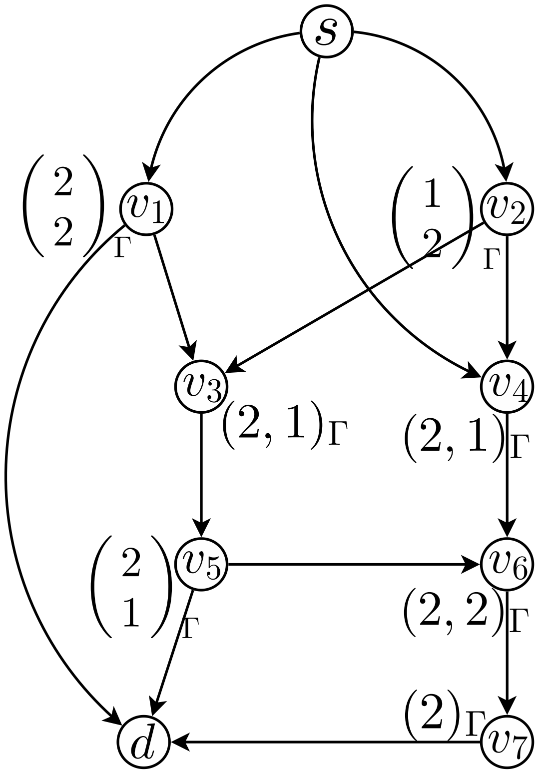


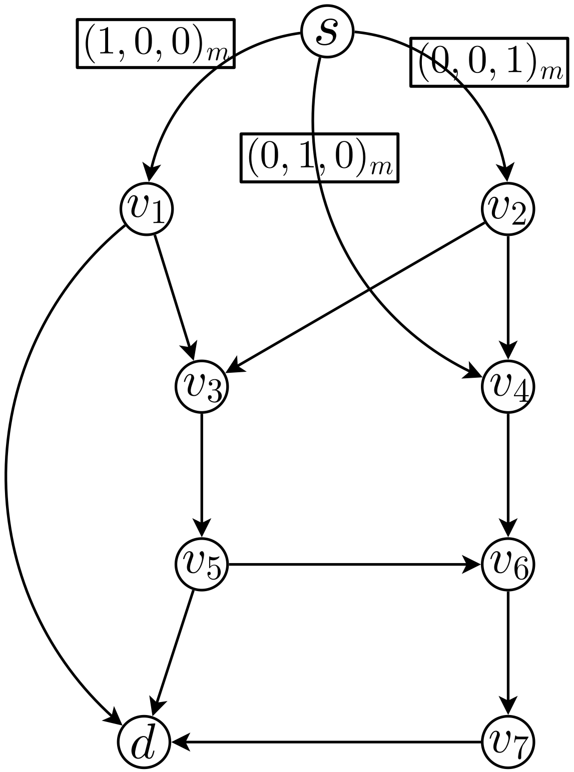
The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on $\mathsf{GF}(3)$



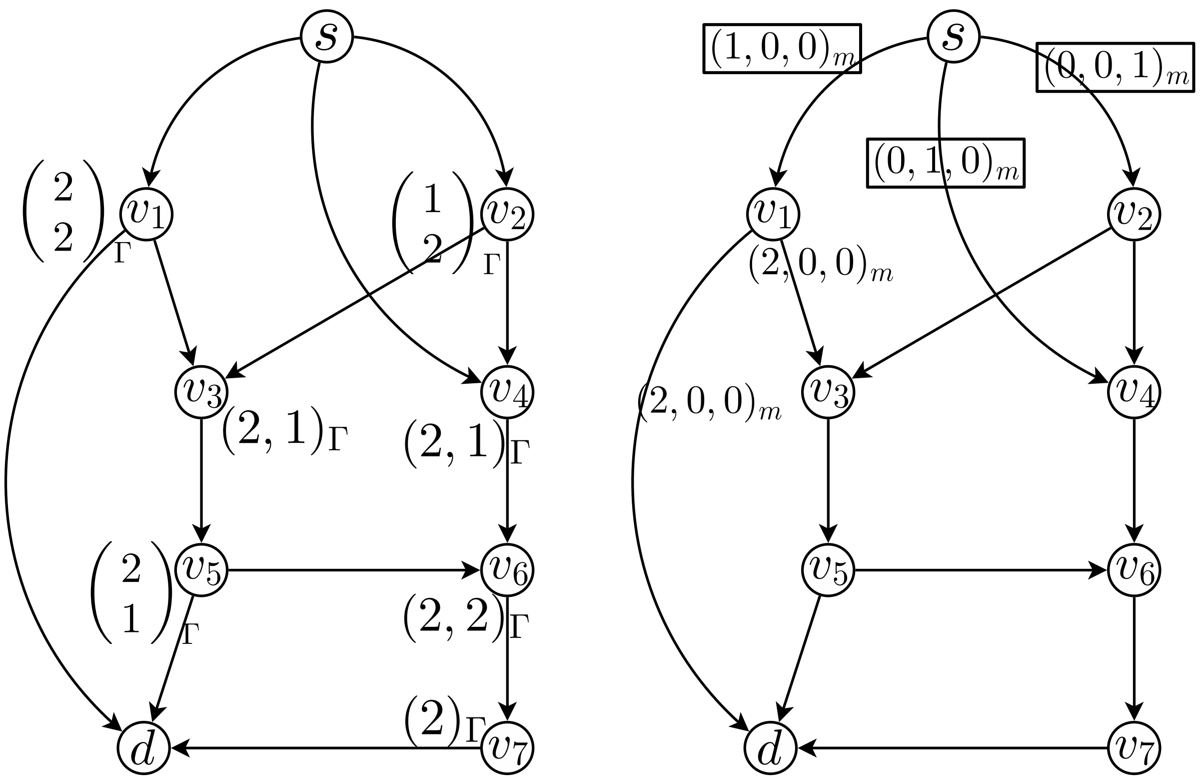


The Coded Feedback Approach

Step 1: Choose the $|\text{Out}(v)| \times |\text{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on GF(3)

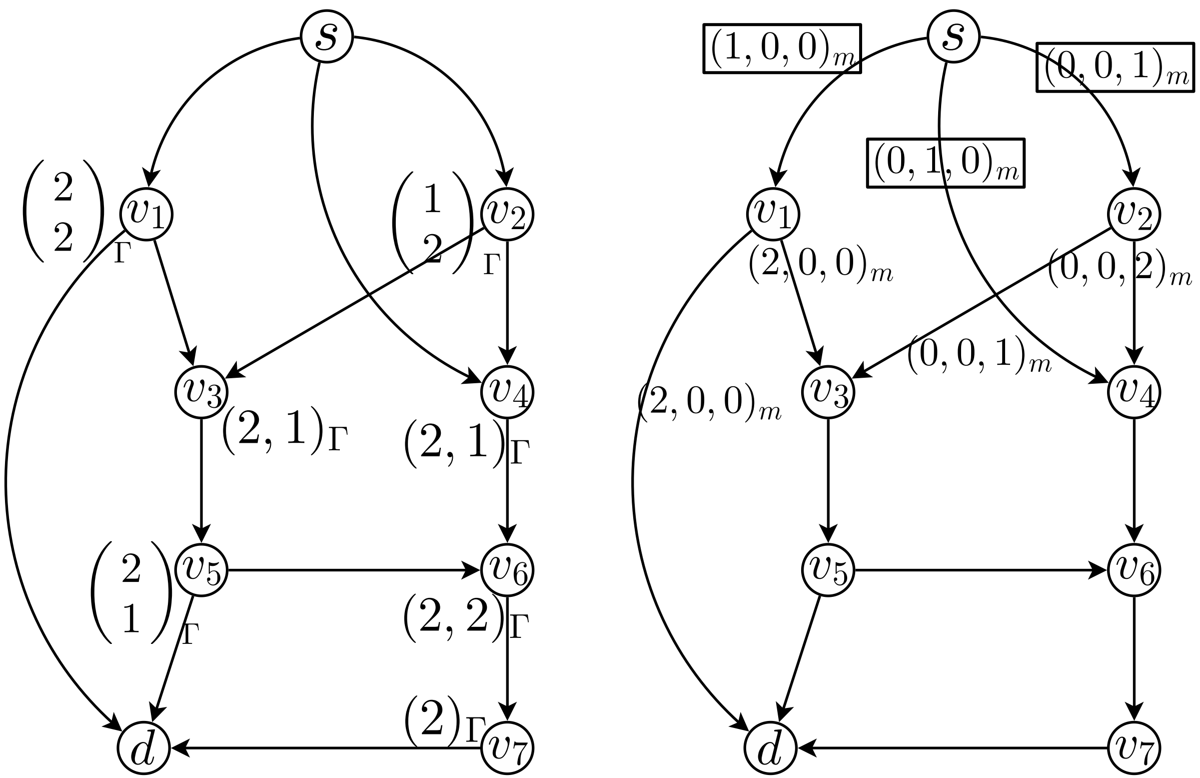


The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on GF(3)

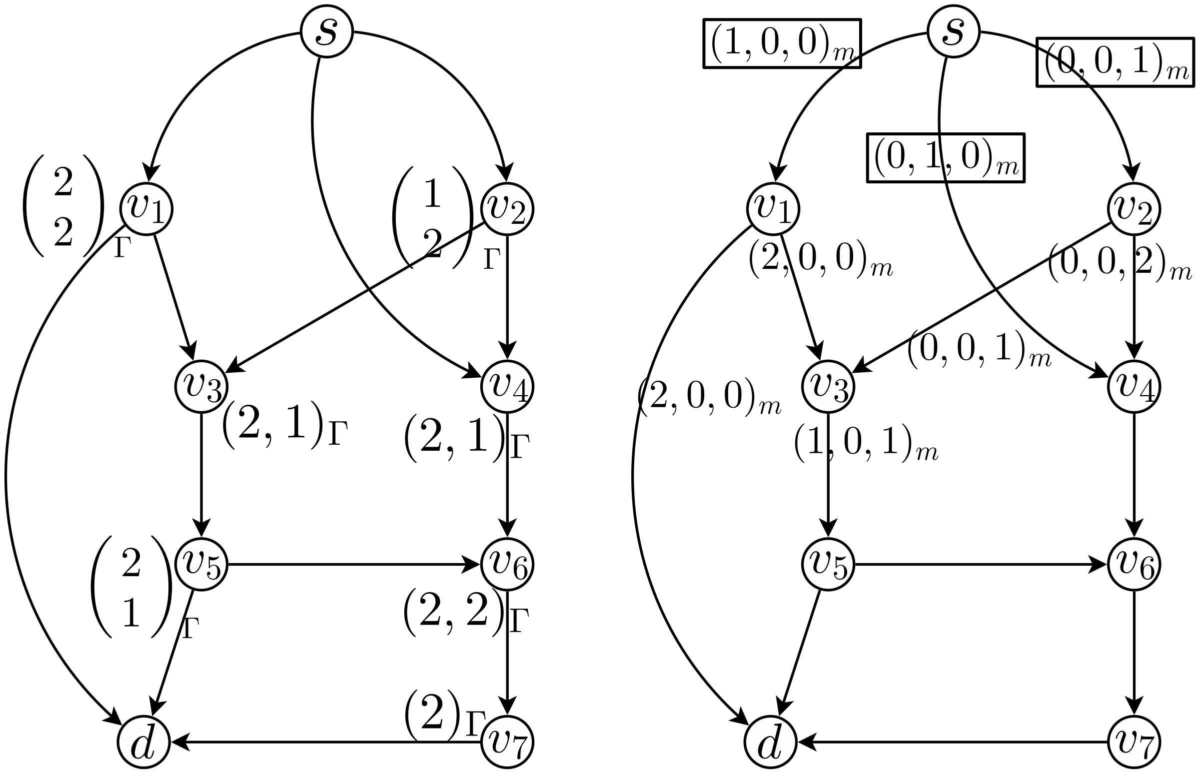


The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on $\mathsf{GF}(3)$

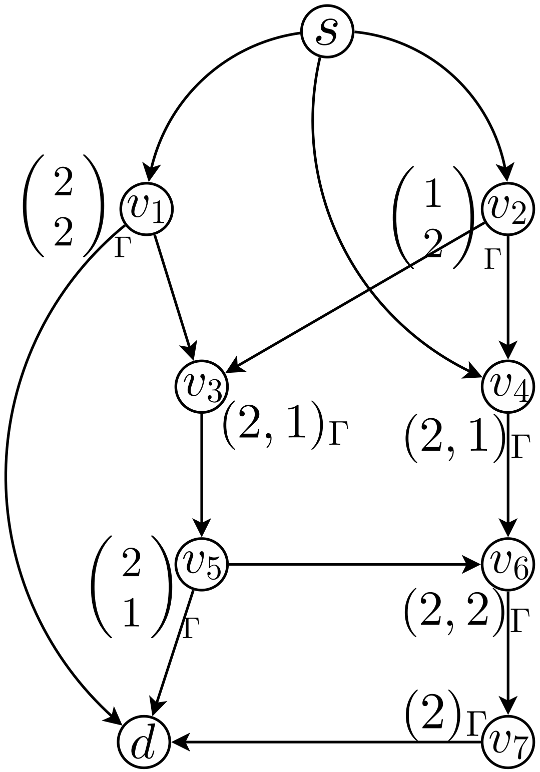


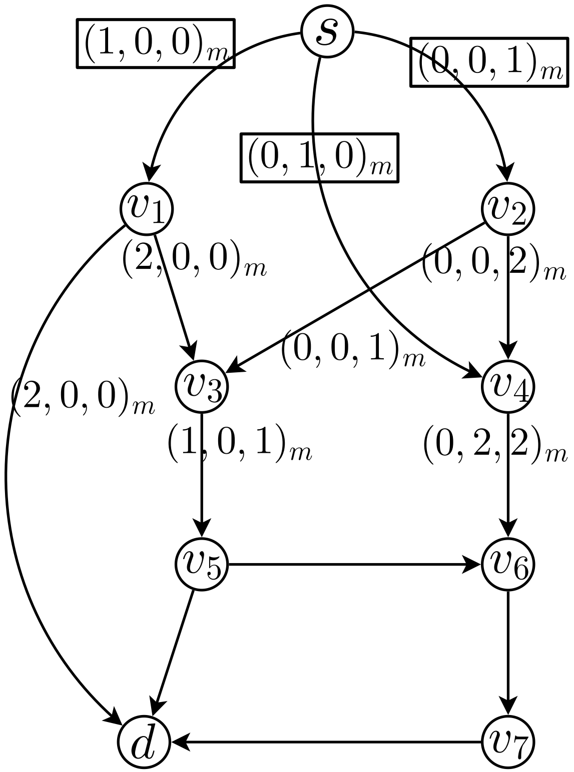
The Coded Feedback Approach

Step 1: Choose the $|\text{Out}(v)| \times |\text{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on GF(3)



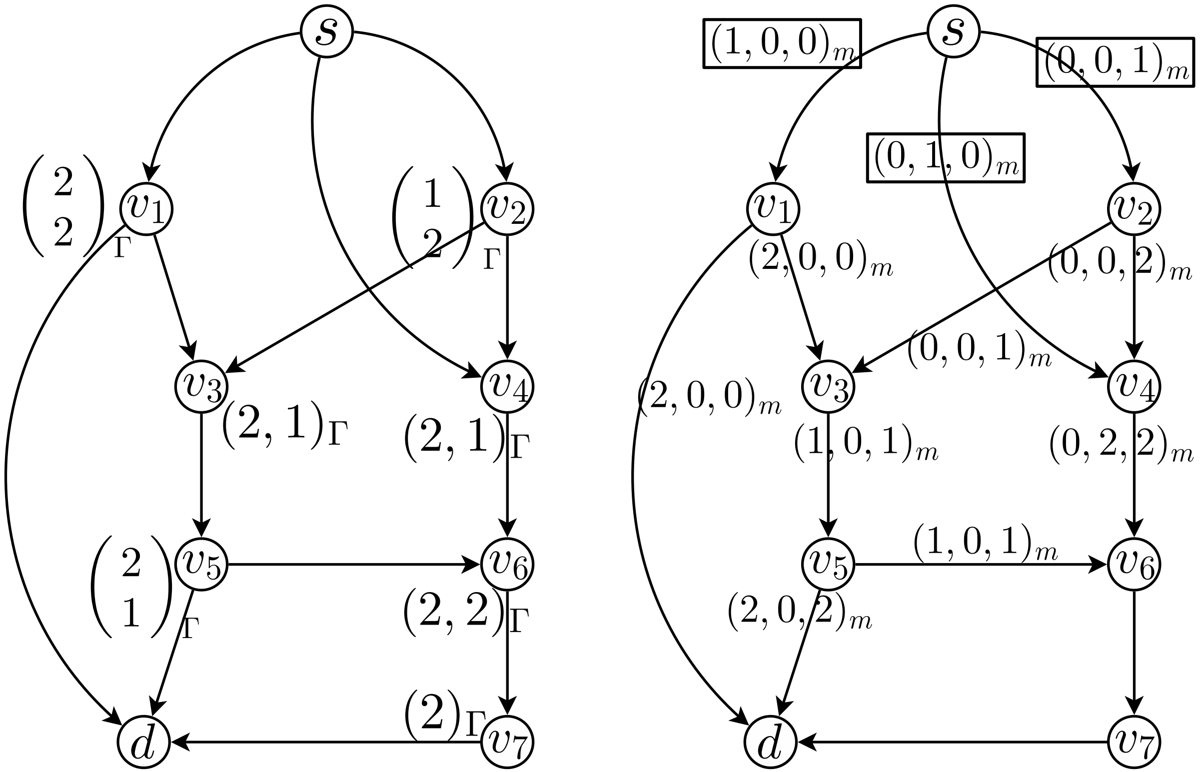


The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on $\mathsf{GF}(3)$

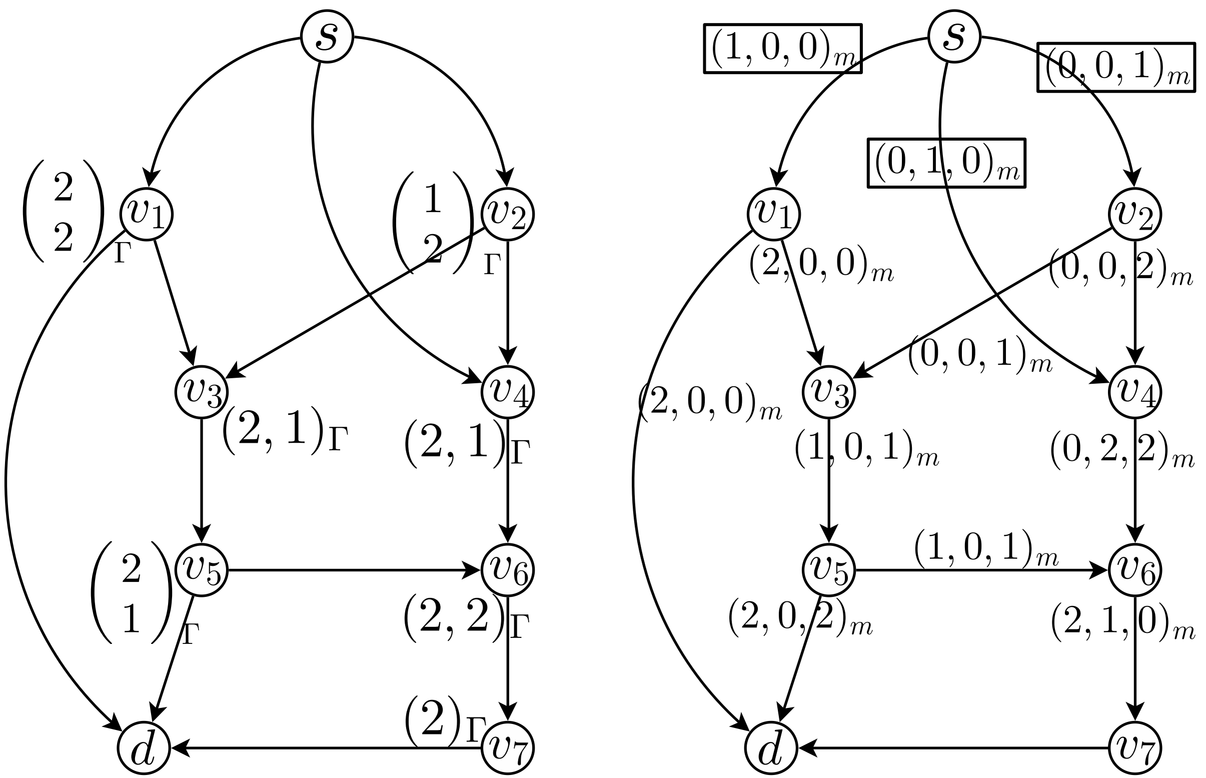


The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on $\mathsf{GF}(3)$

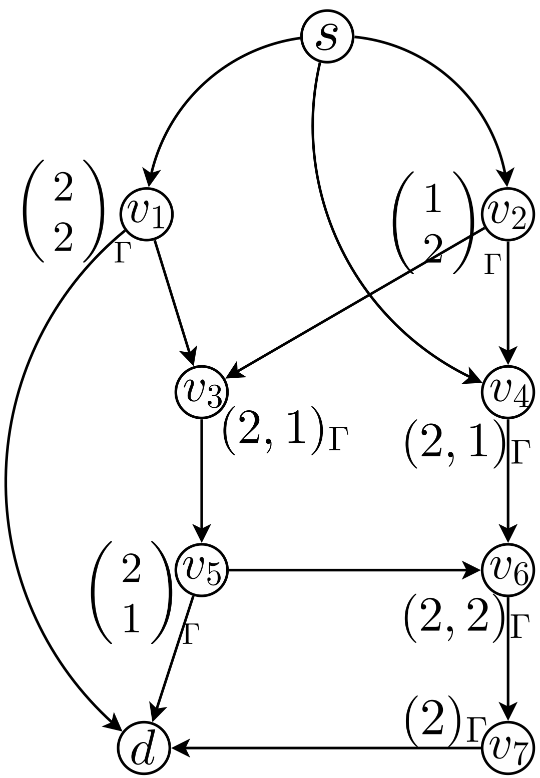


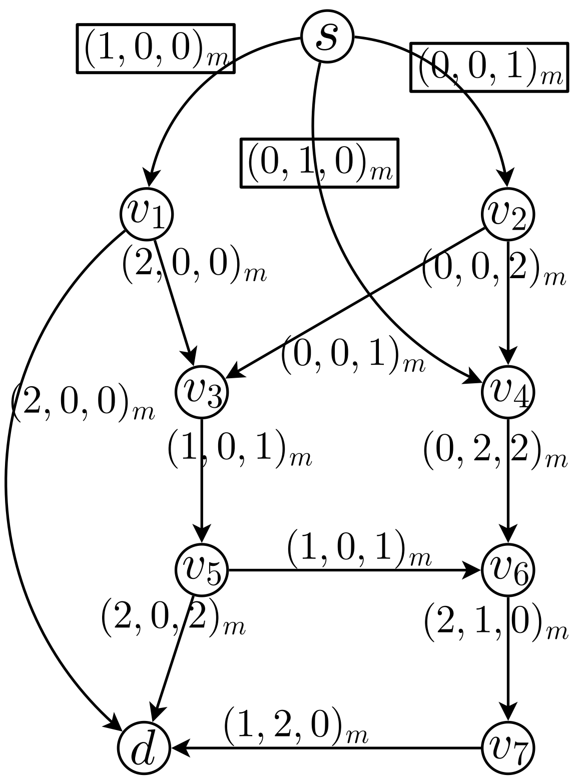
The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on $\mathsf{GF}(3)$





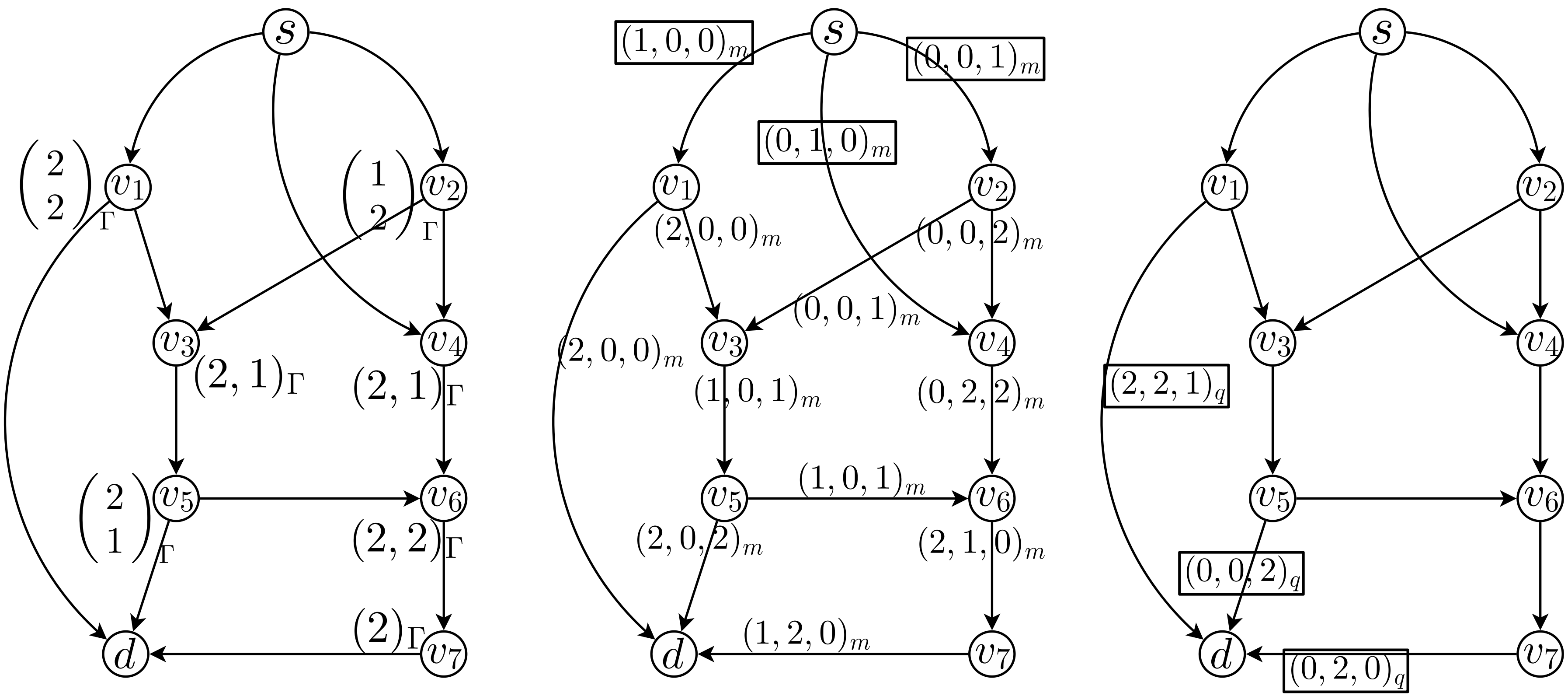
The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Network coding on GF(3)

Step 3: Compute the coded feedback q_e



The Coded Feedback Approach

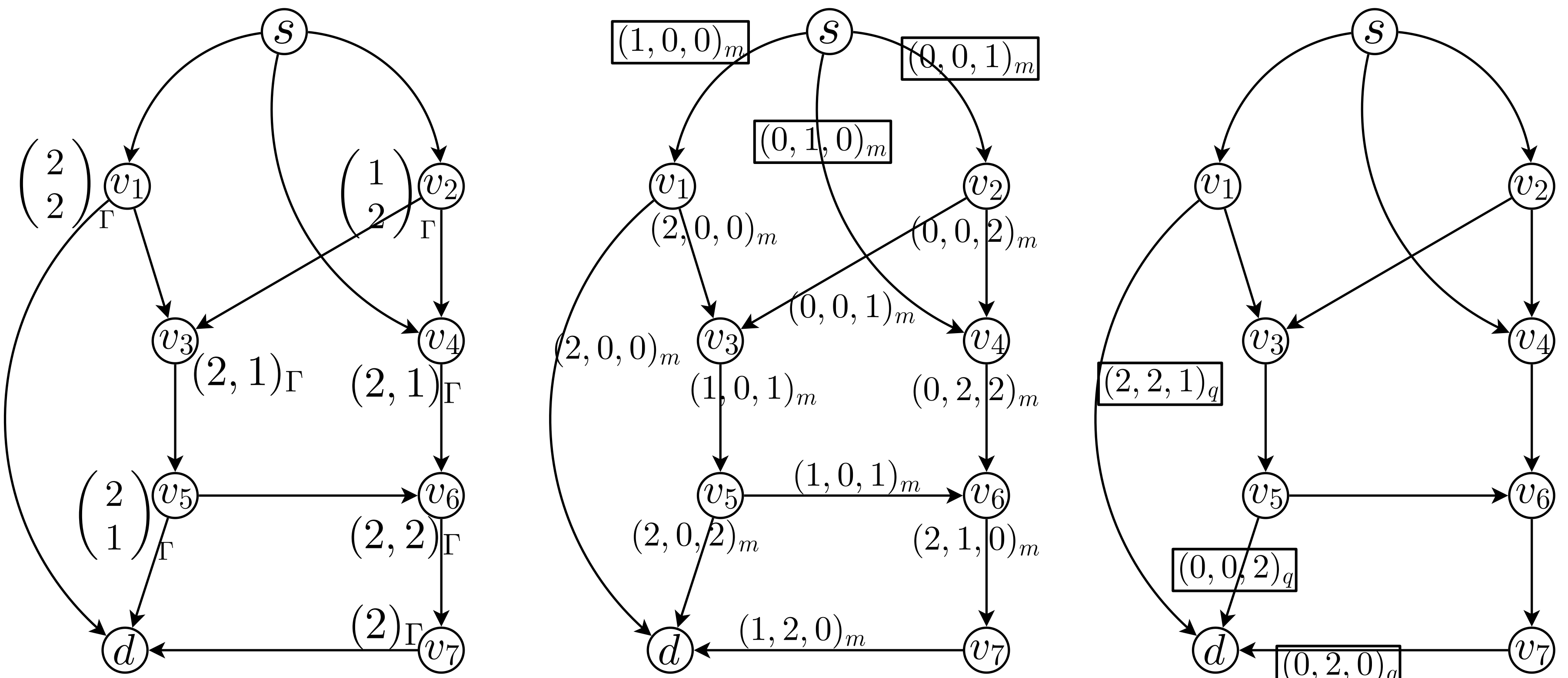
Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

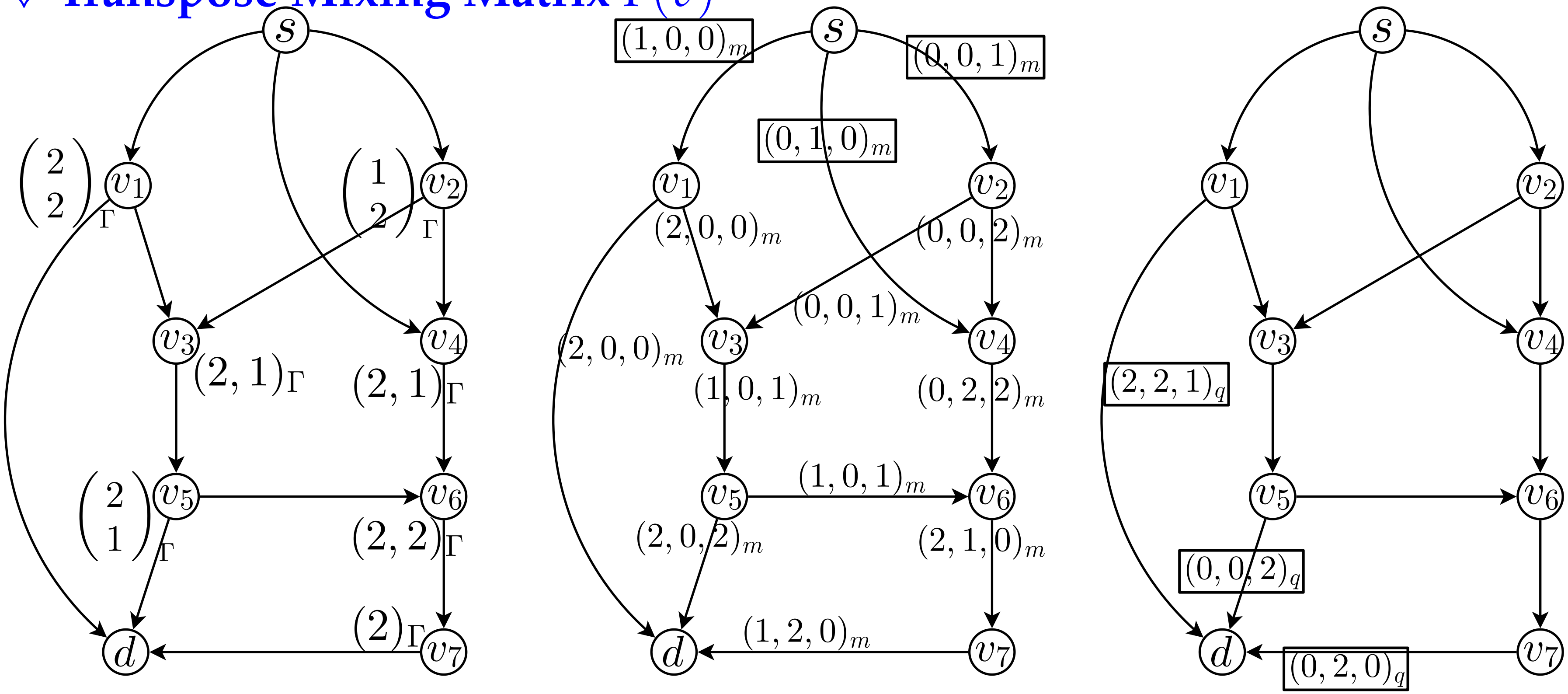
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^T$**



The Coded Feedback Approach

Step 1: Choose the $|\text{Out}(v)| \times |\text{In}(v)|$ mixing matrix $\Gamma(v)$

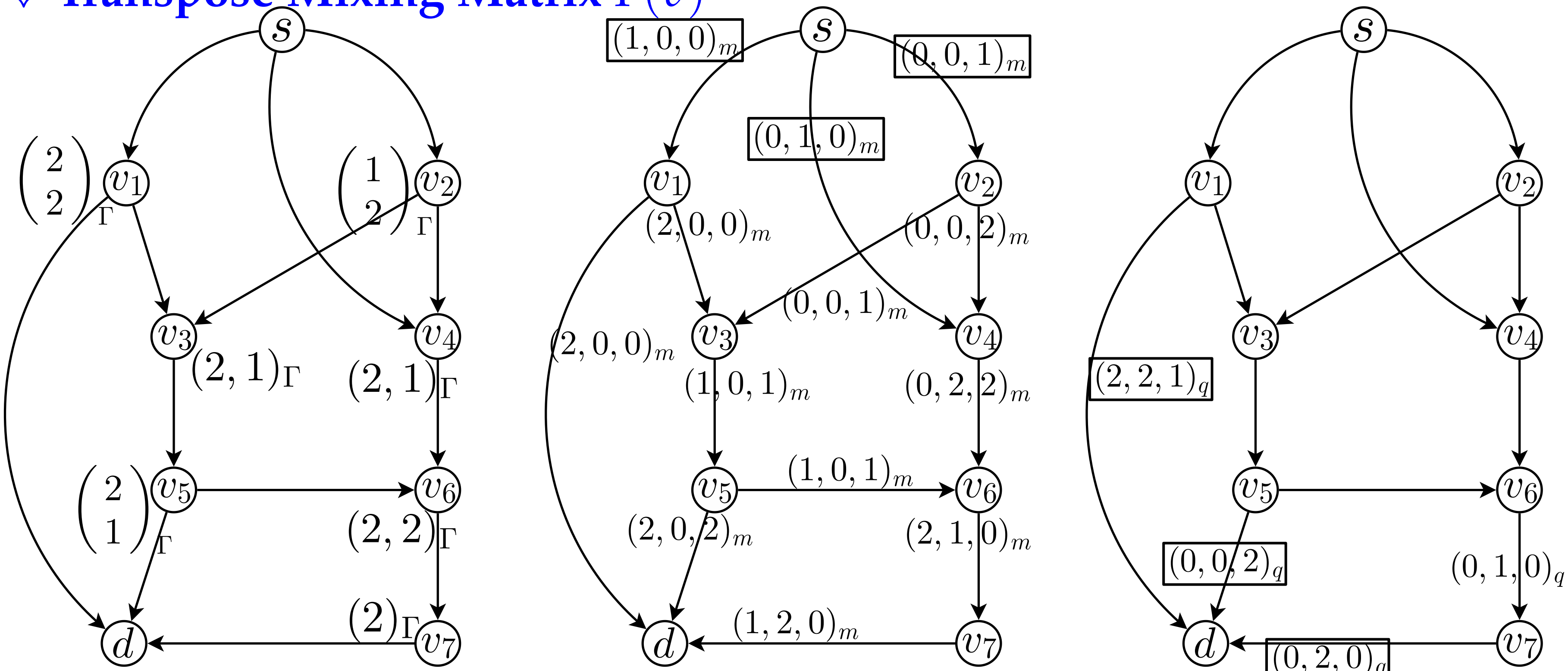
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix** $\Gamma(v)^{\mathrm{T}}$



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

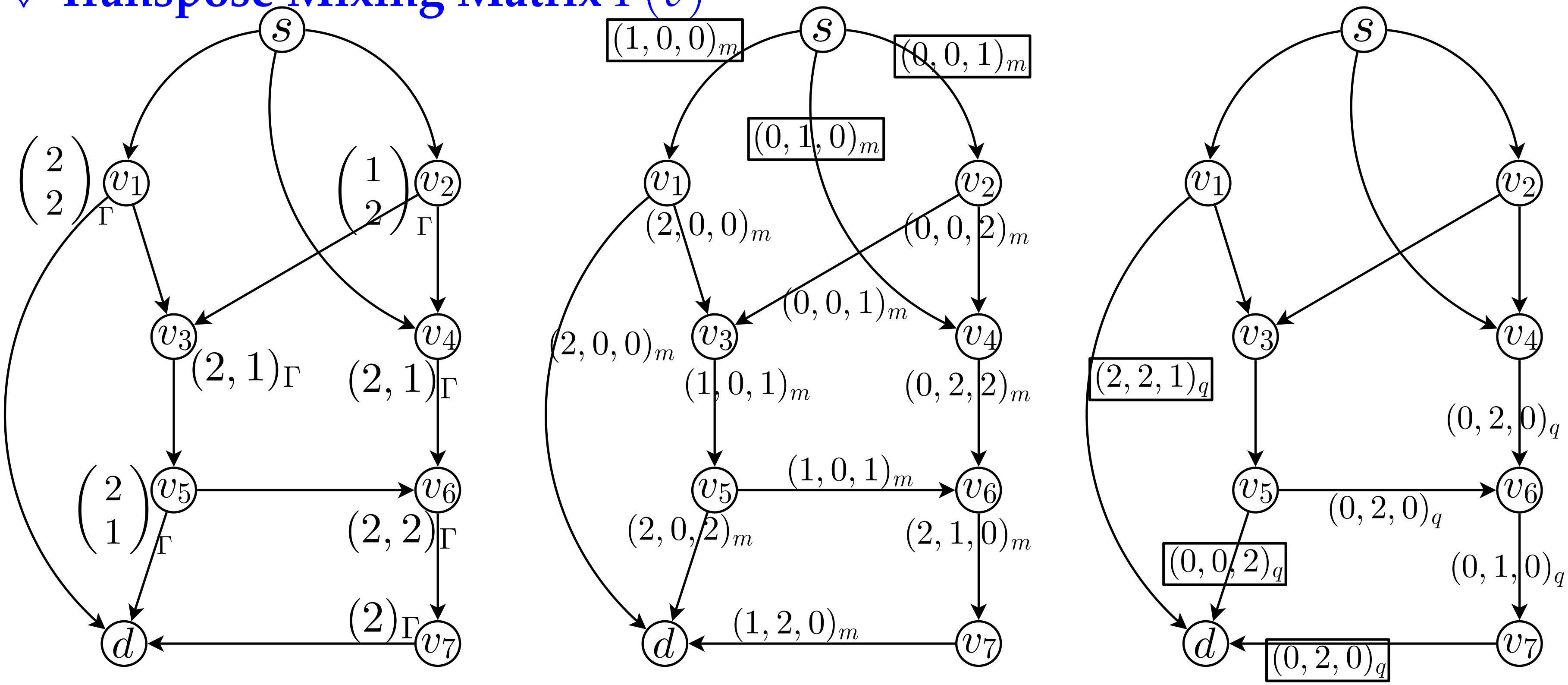
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^T$**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

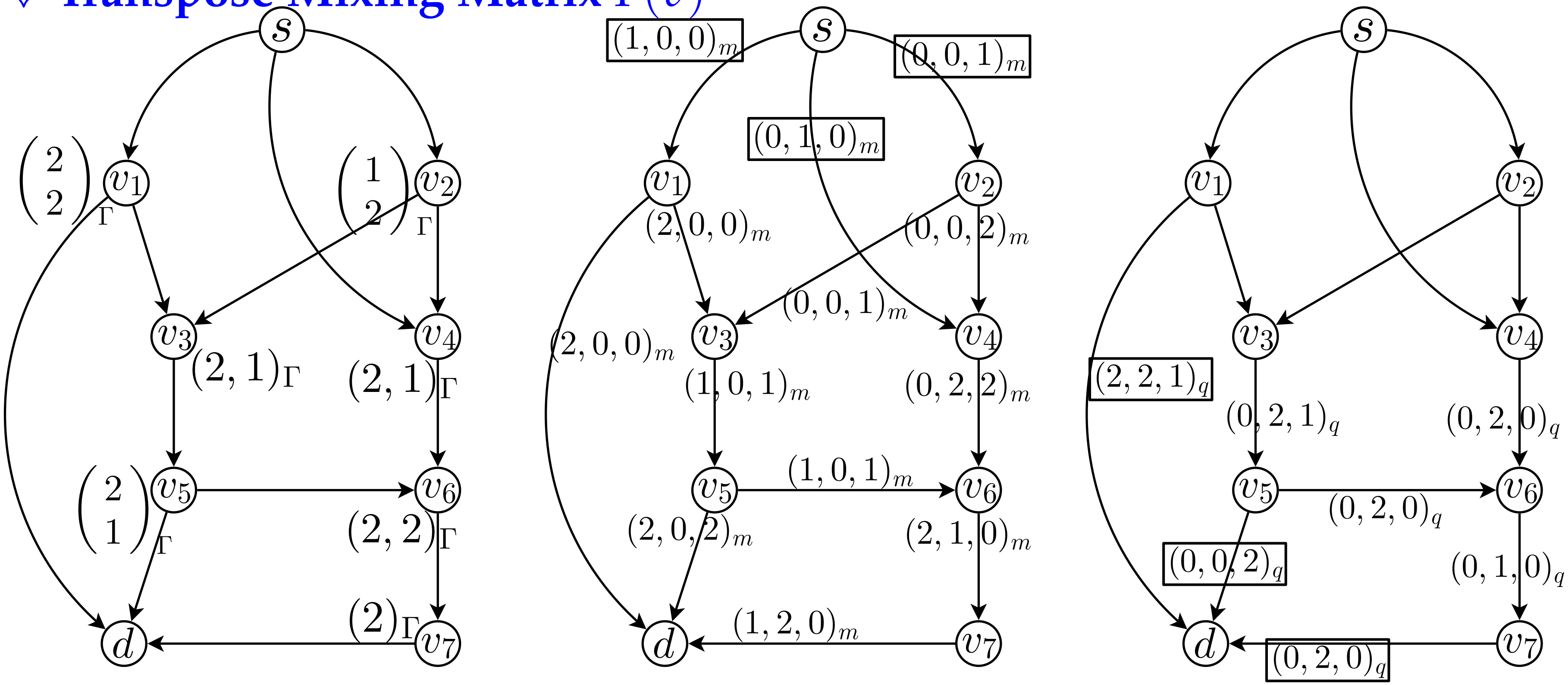
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^{\mathrm{T}}$**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

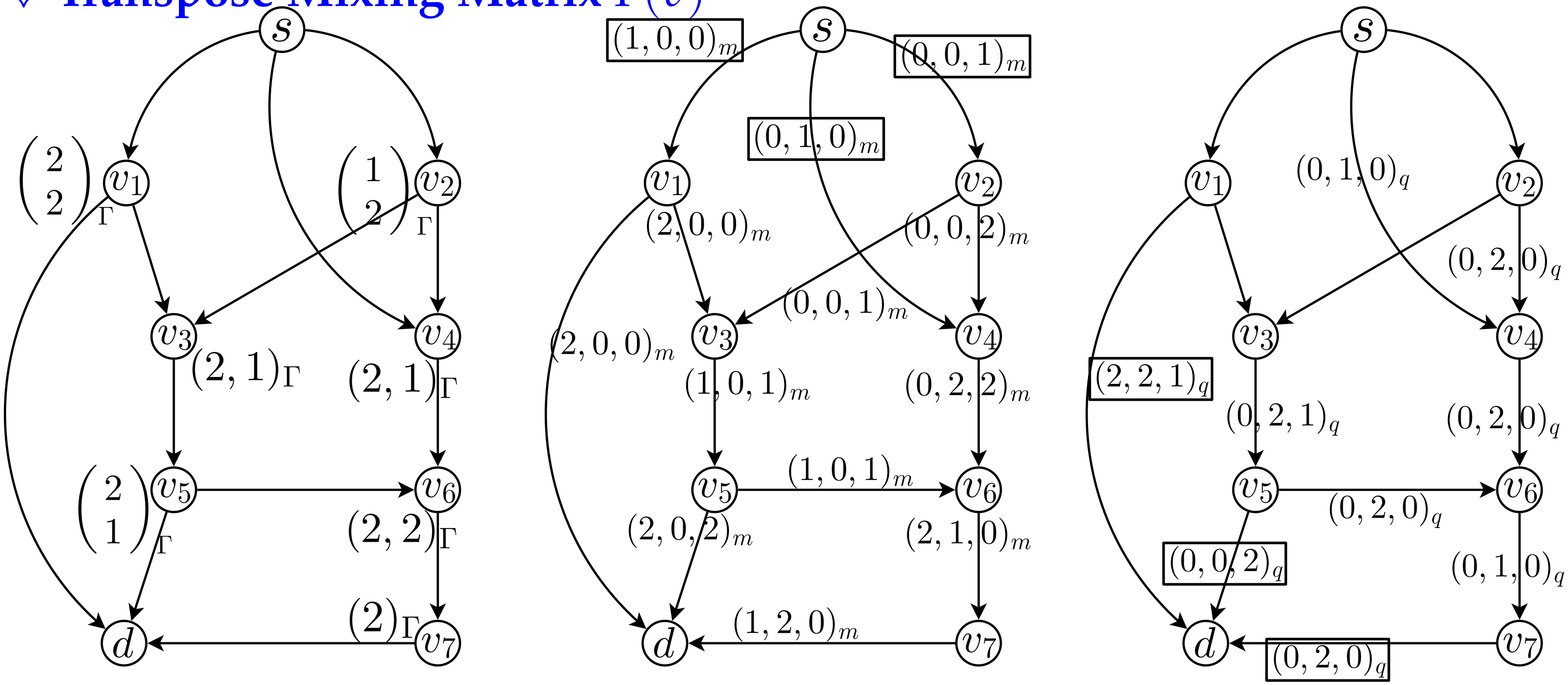
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^{\mathrm{T}}$**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

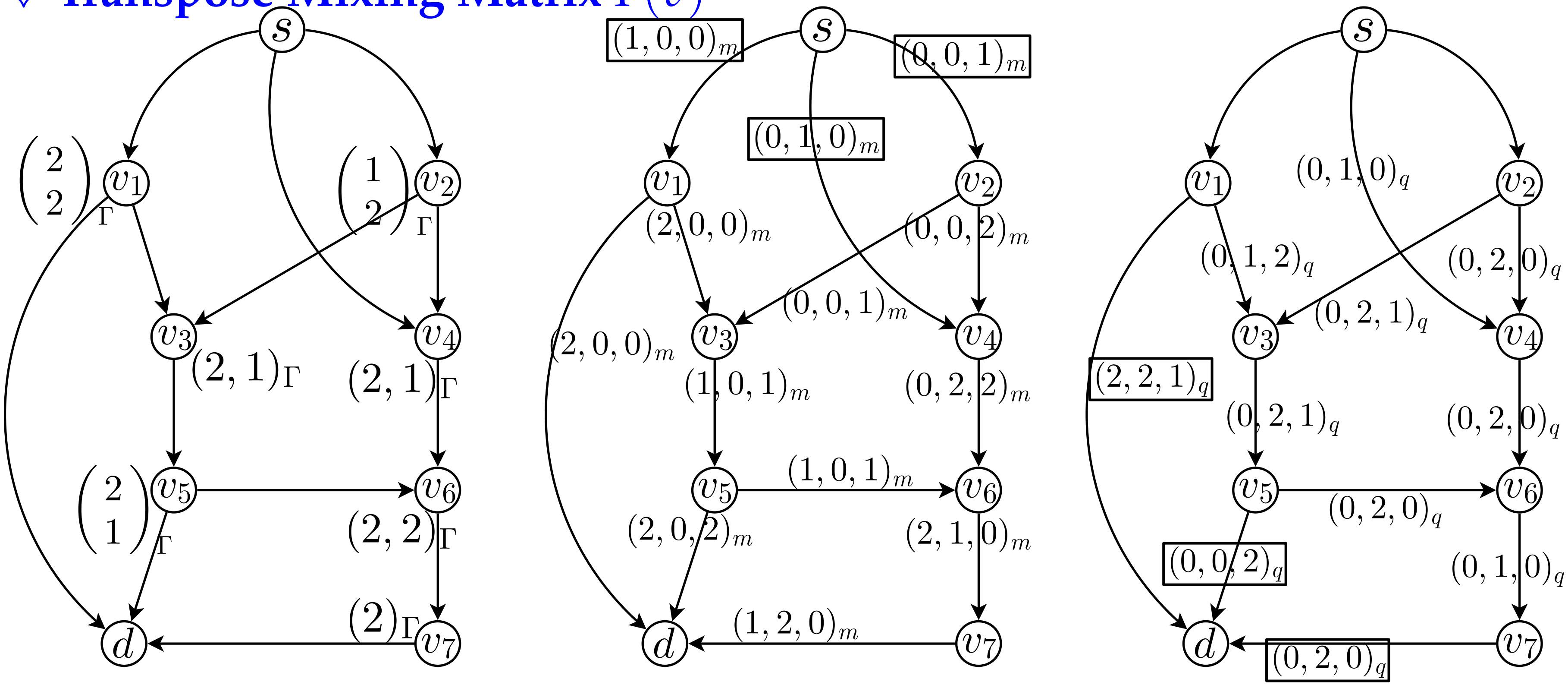
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^T$**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

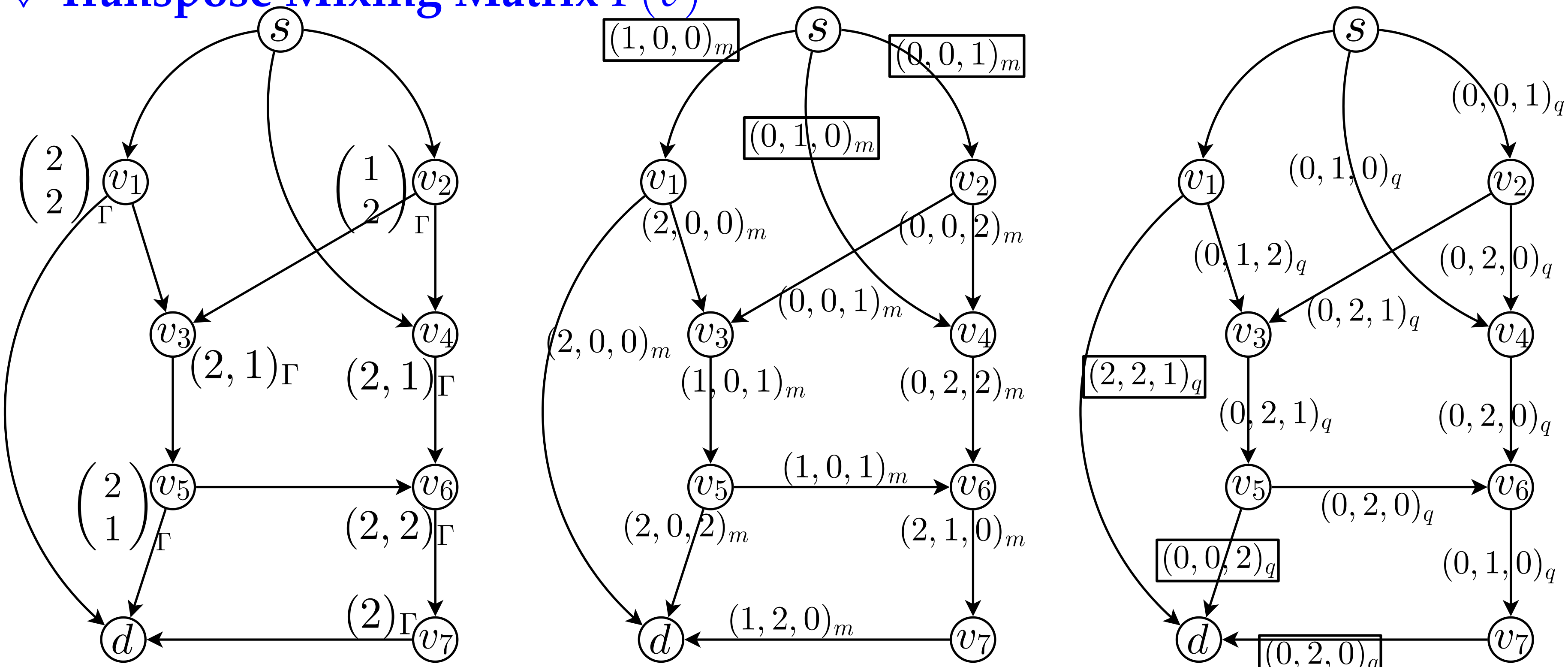
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on GF(3)

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix $\Gamma(v)^{\mathrm{T}}$**



The Coded Feedback Approach

Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

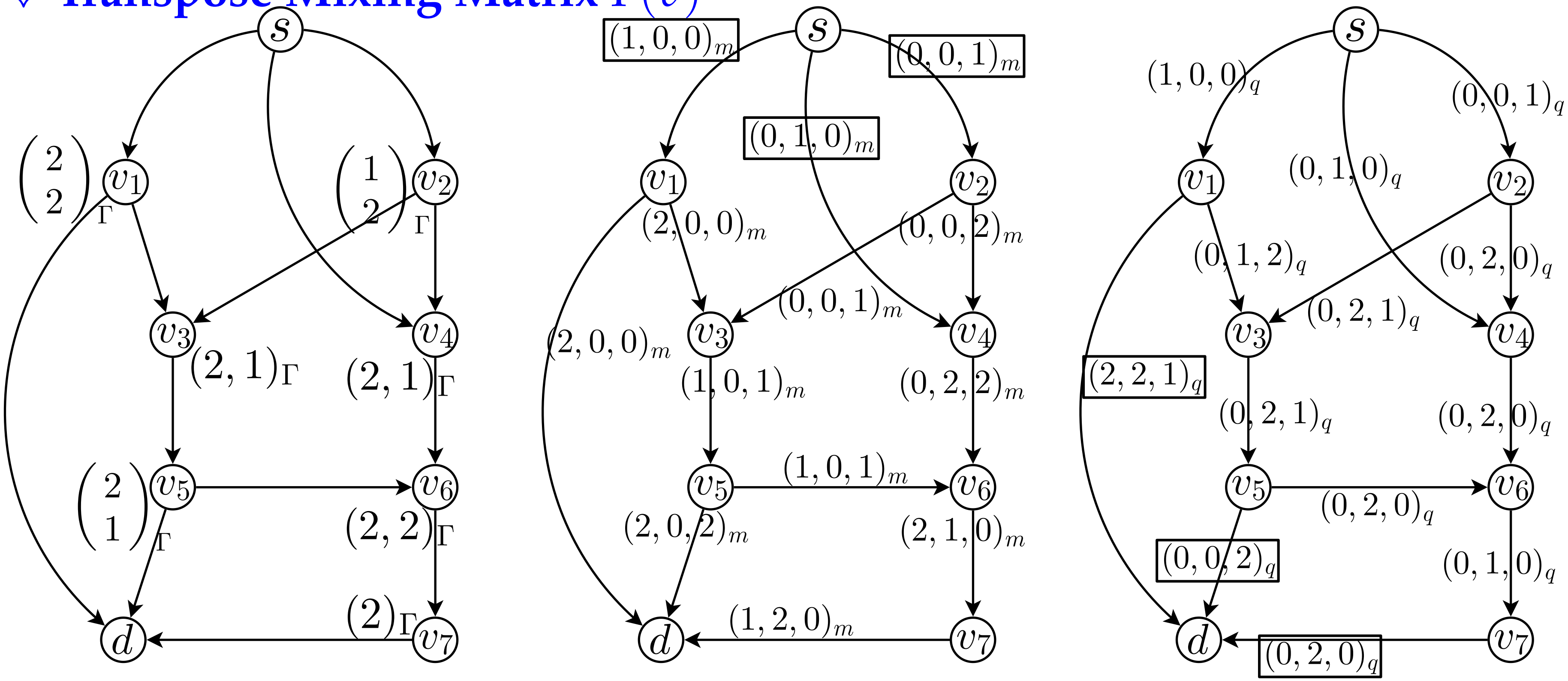
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on $\mathsf{GF}(3)$

♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix** $\Gamma(v)^{\mathrm{T}}$



The Coded Feedback Approach

Step 1: Choose the $|\text{Out}(v)| \times |\text{In}(v)|$ mixing matrix $\Gamma(v)$

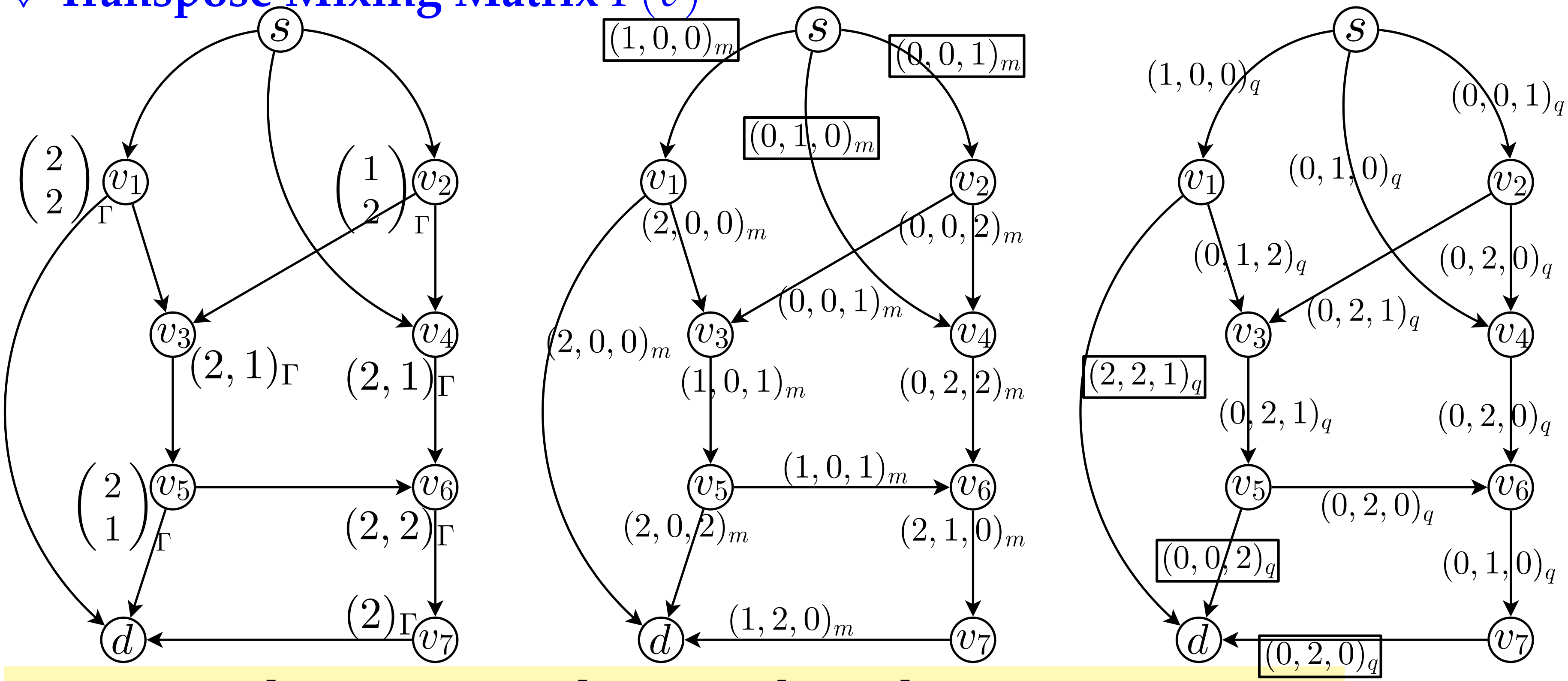
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

Network coding on $\mathsf{GF}(3)$

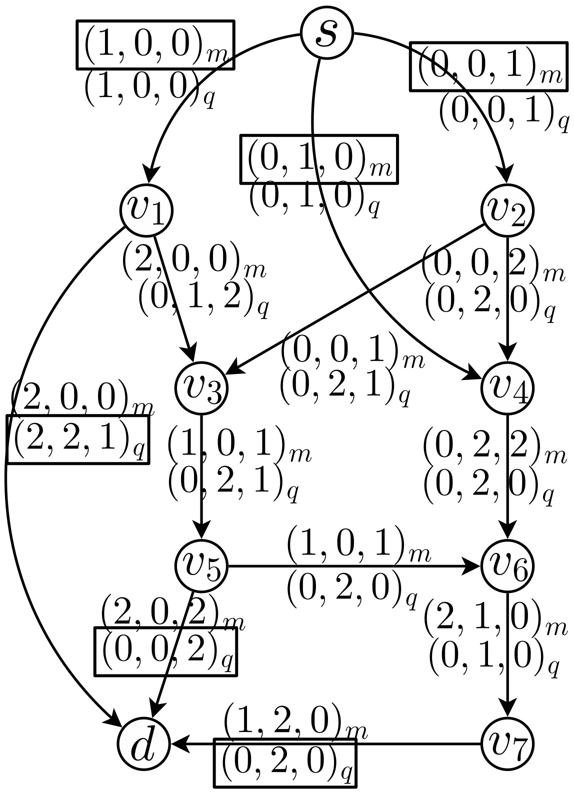
♡ **Orthogonal Coded Feedback**

♡ **Transpose Mixing Matrix** $\Gamma(v)^{\mathrm{T}}$



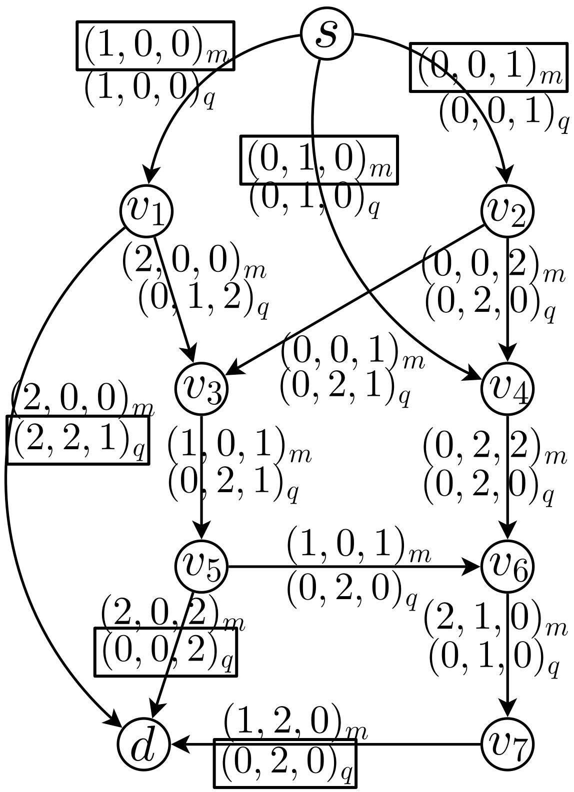
Steps 1 and 2 are Normal Network Coding. Step 3 is new.

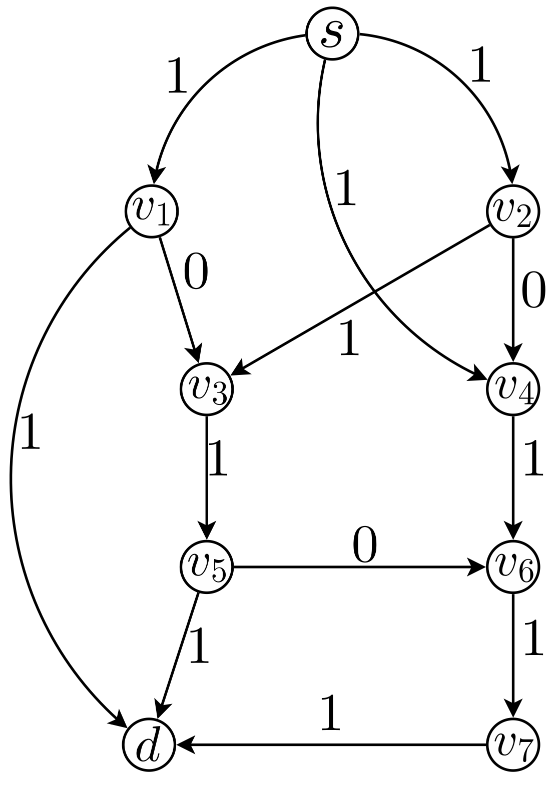
Cont'd


s
$(1,0,0)_m$
$(1,0,0)_q$
$(0,0,1)_m$
$(0,0,1)_q$
$(0,1,0)_m$
$(0,1,0)_q$
v_1
v_2
$(2,0,0)_m$
$(0,1,2)_q$
$(0,0,2)_m$
$(0,2,0)_q$
$(0,0,1)_m$
$(0,2,1)_q$
v_3
v_4
$(2,0,0)_m$
$(2,2,1)_q$
$(1,0,1)_m$
$(0,2,1)_q$
$(0,2,2)_m$
$(0,2,0)_q$
v_5
v_6
$(1,0,1)_m$
$(0,2,0)_q$
$(2,0,2)_m$
$(0,0,2)_q$
$(2,1,0)_m$
$(0,1,0)_q$
d
v_7
$(1,2,0)_m$
$(0,2,0)_q$

Cont'd

Step 4: Compare the inner products

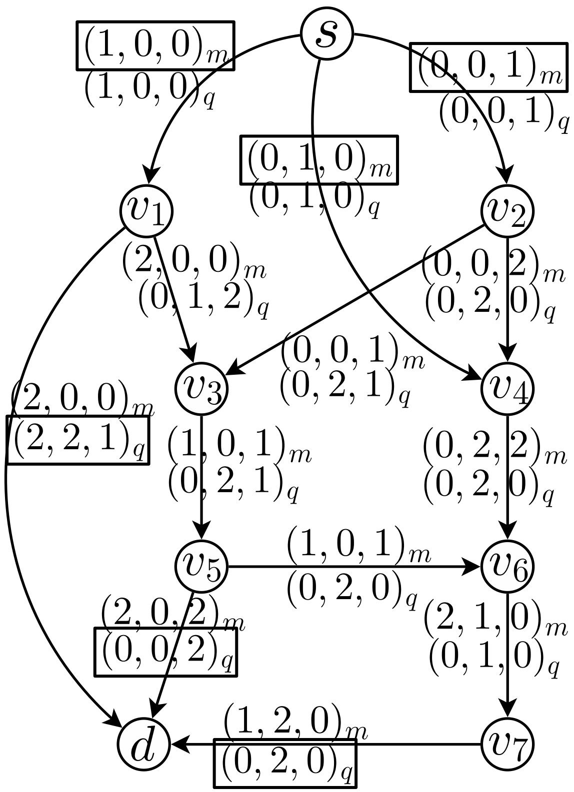


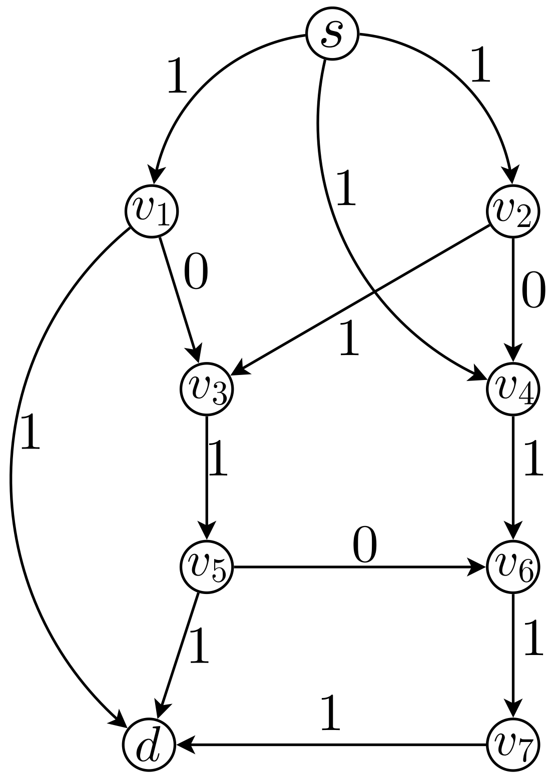


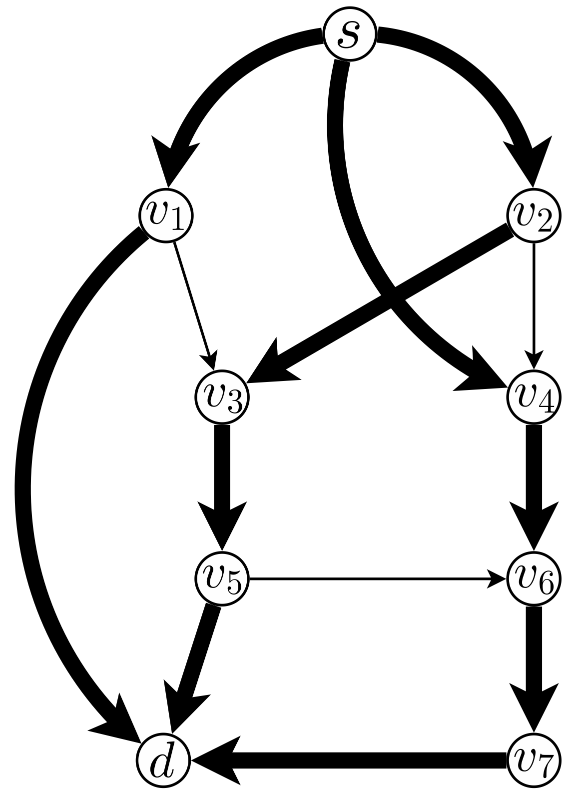
Cont'd

Step 4: Compare the inner products

Comparison to the true max flow found offline



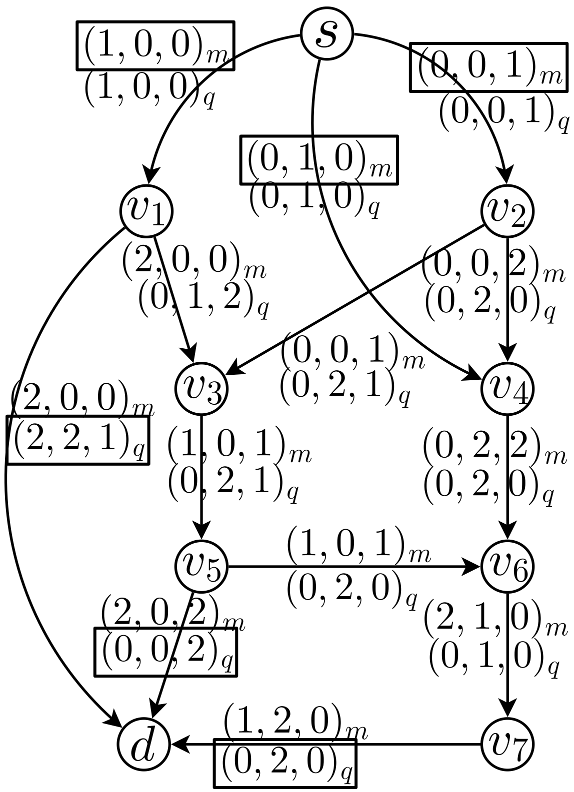


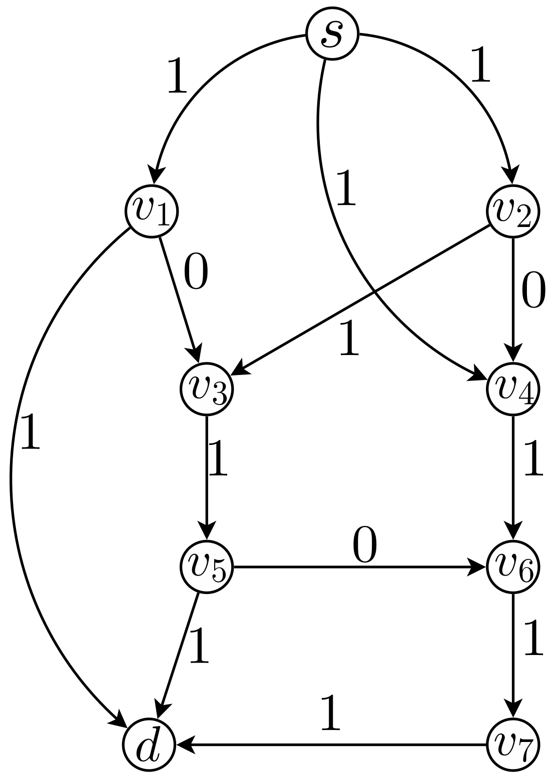


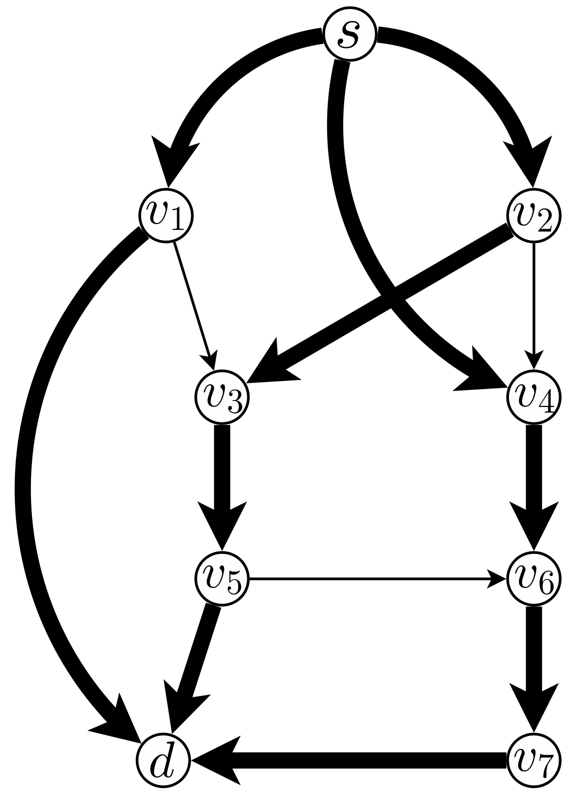
Cont'd

Step 4: Compare the inner products

Comparison to the true max flow found offline





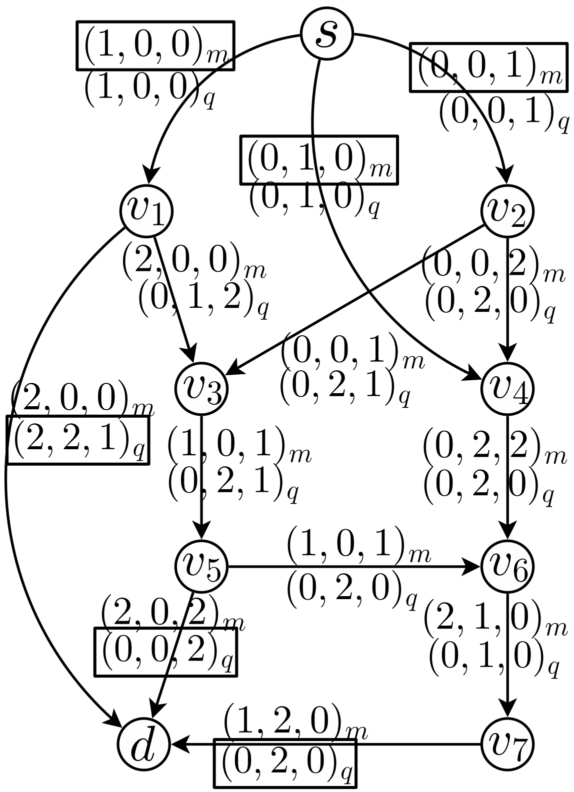


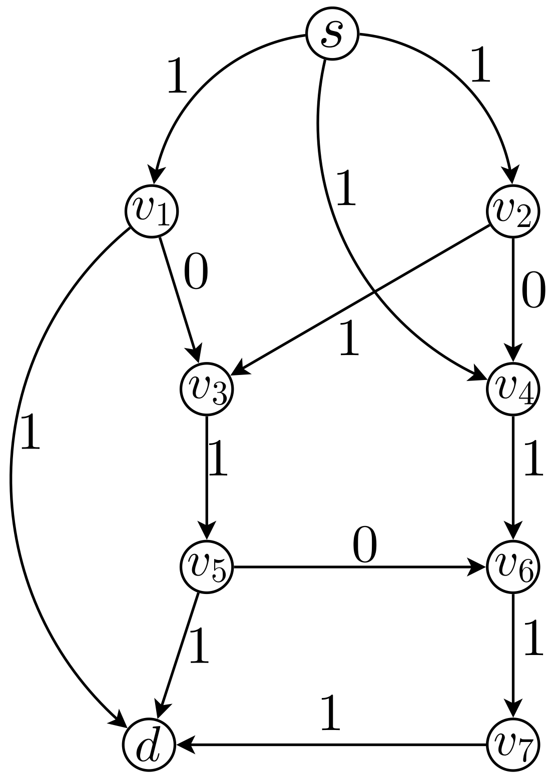
Voila!

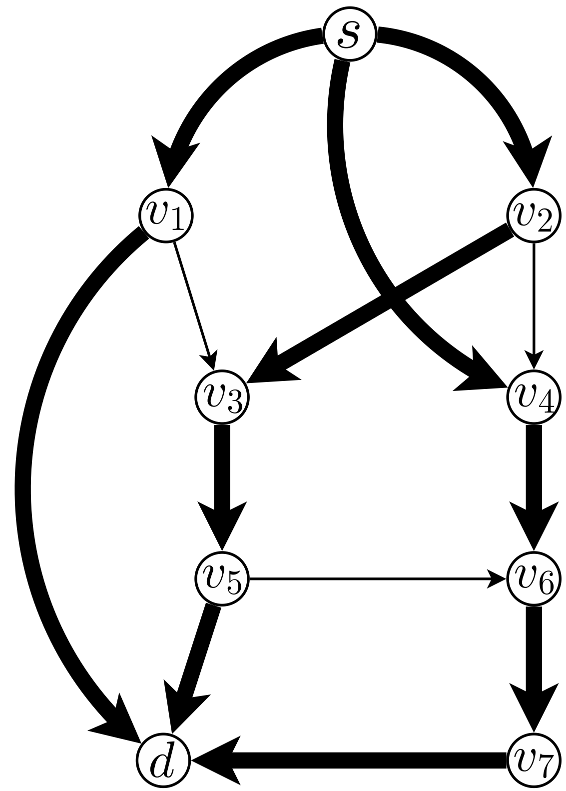
Cont'd

Step 4: Compare the inner products

Comparison to the true max flow found offline







Voila!

Not so fast! For more complicated networks, some unexpected scenario may arise. We need a provably correct algorithm.

The Detailed Description

High-level description:

Choose $\Gamma(v)$
loop
  Compute Forward Messages m_e
  Compute Coded Feedback q_e
  Find redundant edge set $E_R(v)$
  if $E_R(v) \neq \emptyset$ **then**
    Remove $E_R(v)$.
  else
    return the remaining graph G
  end if
end loop

The Detailed Description

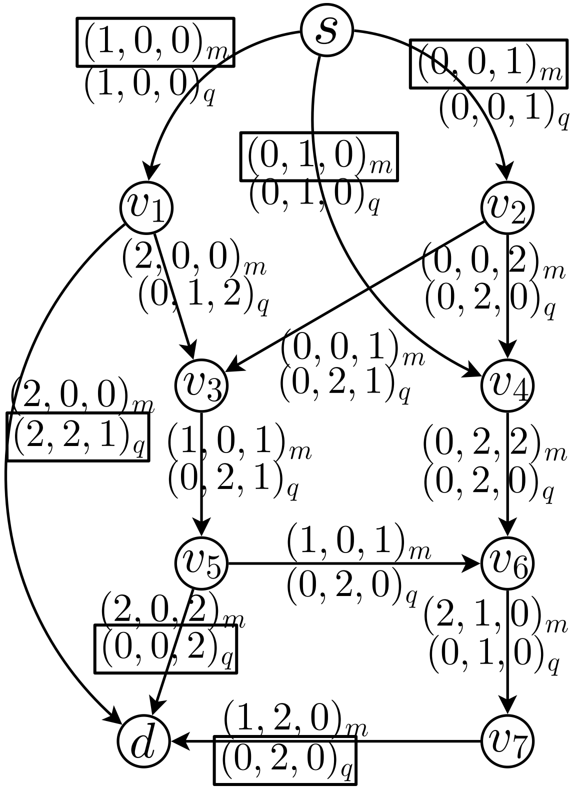
High-level description:

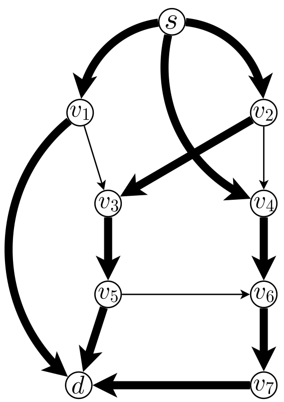
Choose $\Gamma(v)$
loop
  Compute Forward Messages m_e
  Compute Coded Feedback q_e
  Find redundant edge set $E_R(v)$
  if $E_R(v) \neq \emptyset$ **then**
    Remove $E_R(v)$.
  else
    return the remaining graph G
  end if
end loop

Find redundant edge set $E_R(v)$:

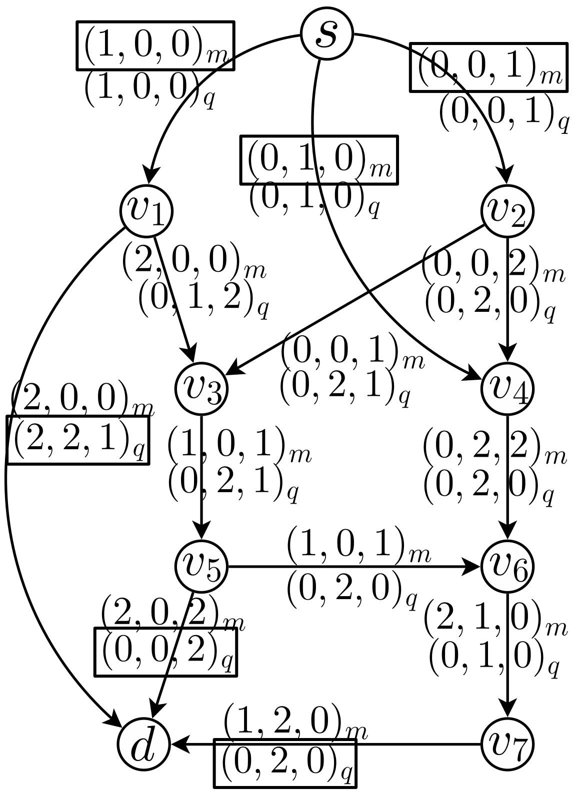
while searching v from downstream d back to upstream s **do**
  Generate the inner product matrix Φ between $[q_e : e \in \mathrm{Out}(v)]$ and $[m_e : e \in \mathrm{In}(v)]$.
  Choose an $E_v \subseteq \mathrm{In}(v)$ such that the corresponding submatrix of Φ being of a full rank square matrix.
  if $E_v \neq \mathrm{In}(v)$ **then**
    return $E_R(v) \leftarrow \mathrm{In}(v) \backslash E_v$.
  end if
end while

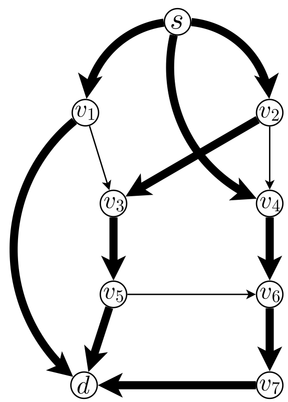
An Illustrative Example





An Illustrative Example



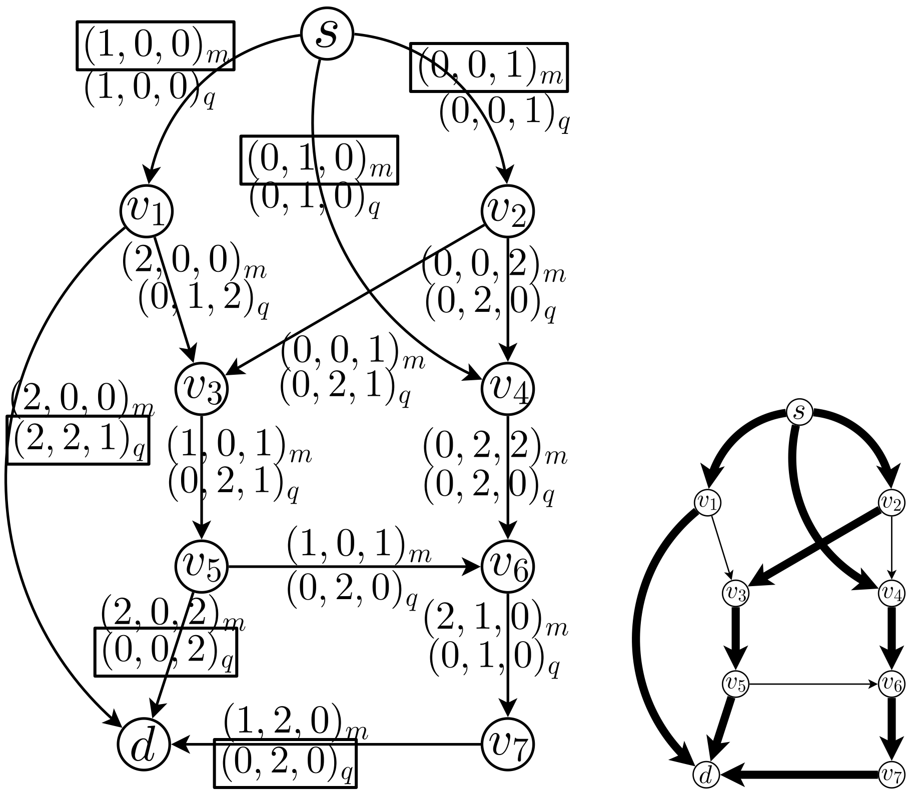


Search v_7:

Out(v_7): $q_{v_7,d} = (0, 2, 0)$
In(v_7): $m_{v_6,v_7} = (2, 1, 0)$

Φ: 2 $\Rightarrow$ Full rank

An Illustrative Example

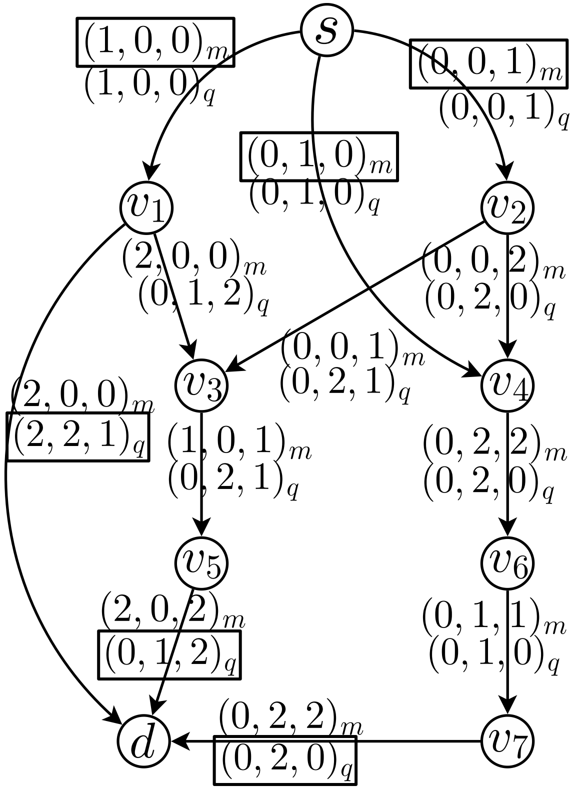


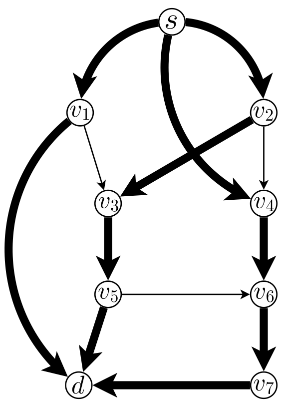
Search v_6:

Out(v_6): $q_{v_6,v_7} = (0,1,0)$
In(v_6): $m_{\cdot,v_6} = (1,0,1), (0,2,2)$

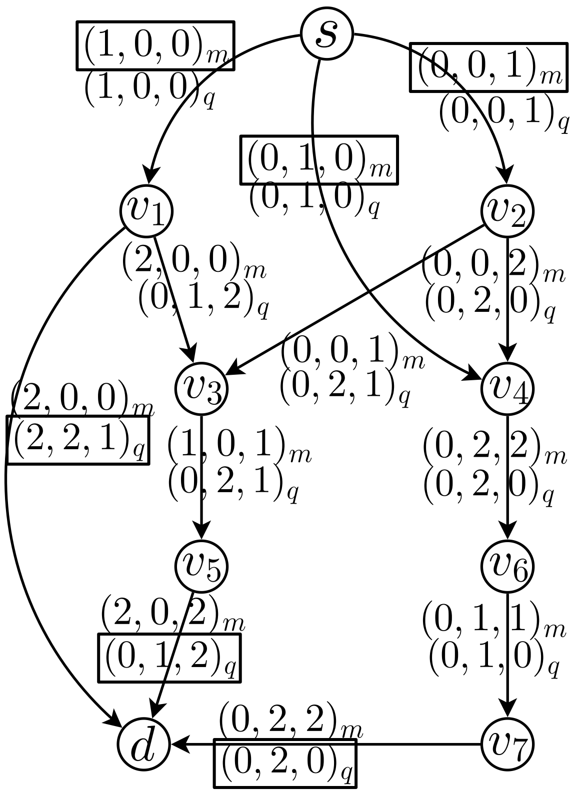
Φ: $[0, 2] \Rightarrow E_v = \{(v_4, v_6)\}$
$E_R(v) = \{(v_5, v_6)\}$

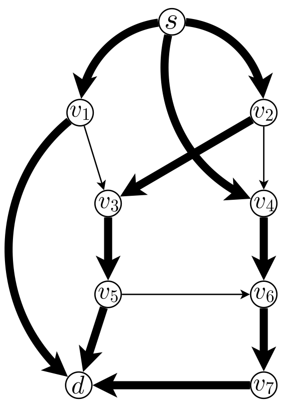
An Illustrative Example





An Illustrative Example



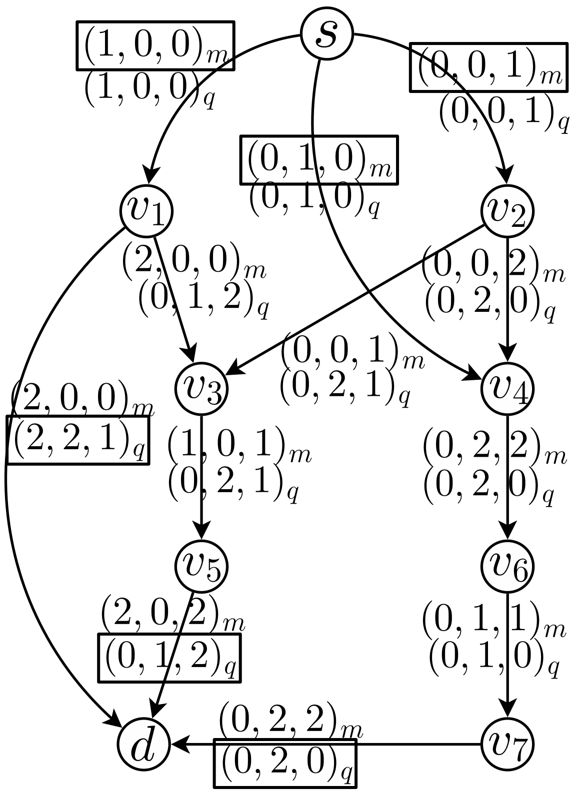


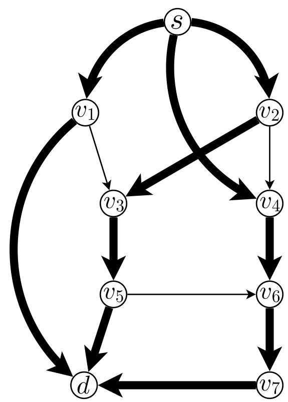
Search v_5:

Out(v_5): $q_{v_5,d} = (0,1,2)$
In(v_5): $m_{v_3,v_5} = (1,0,1)$

Φ: 2 $\Rightarrow$ Full rank

An Illustrative Example



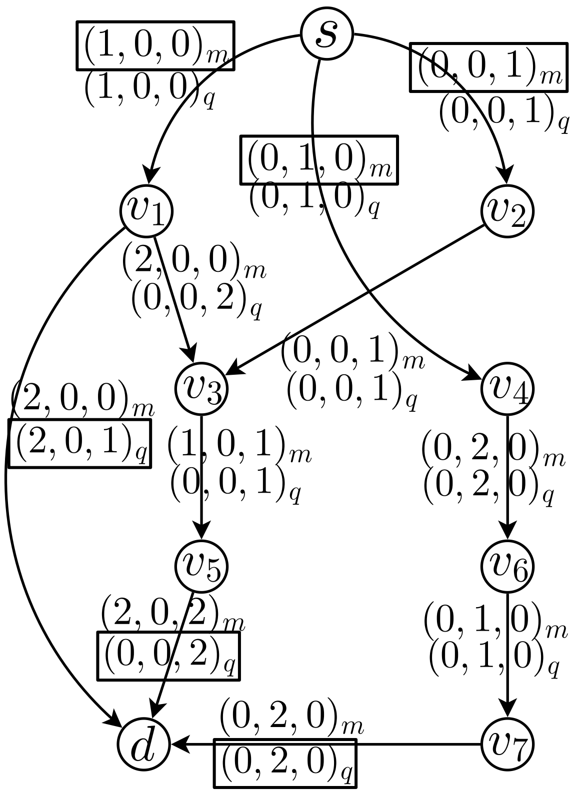


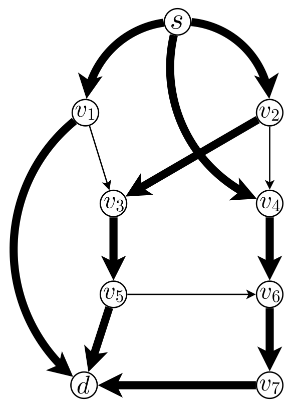
Search v_4:

Out(v_4): $q_{v_4,v_6} = (0,2,0)$
In(v_4): $m_{\cdot,v_4} = (0,1,0), (0,0,2)$

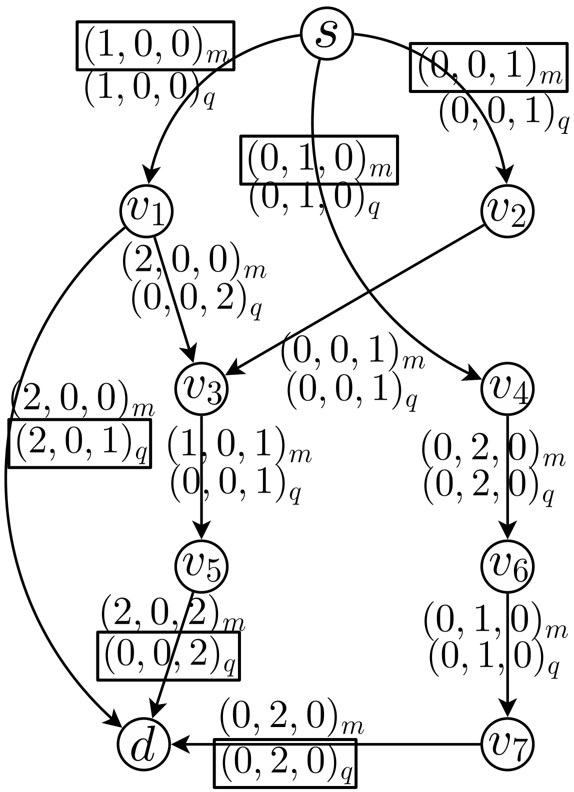
Φ: $[\,2, 0] \Rightarrow E_v = \{(s, v_4)\}$
$E_R(v) = \{(v_2, v_4)\}$

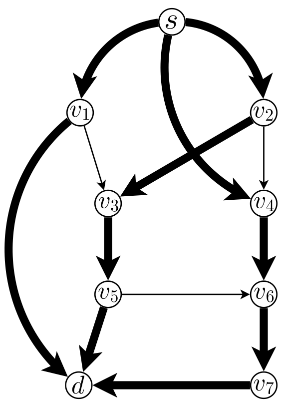
An Illustrative Example





An Illustrative Example



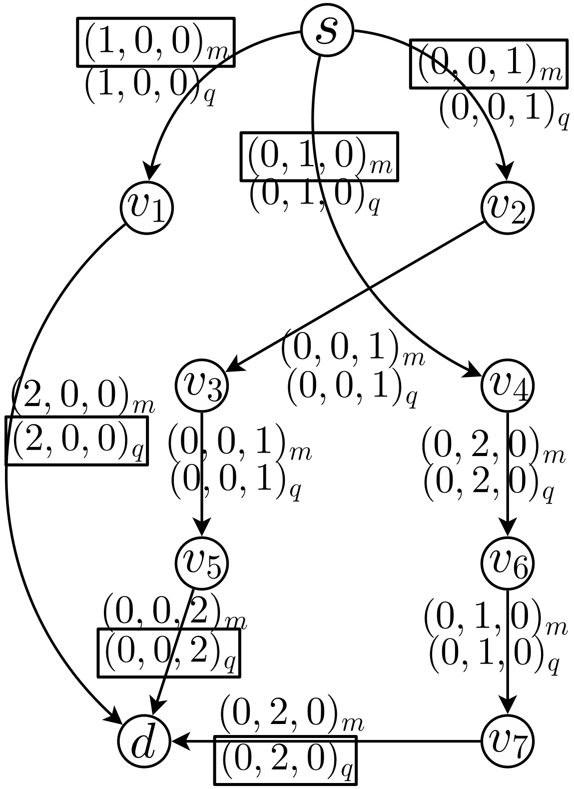


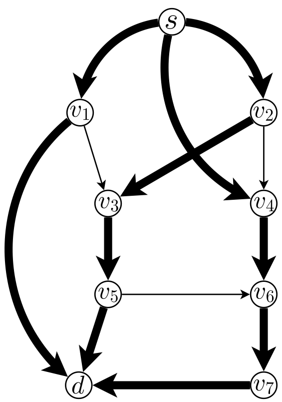
Search v_3:

Out(v_3): $q_{v_3,v_5} = (0,0,1)$
In(v_3): $m_{\cdot,v_3} = (2,0,0), (0,0,1)$

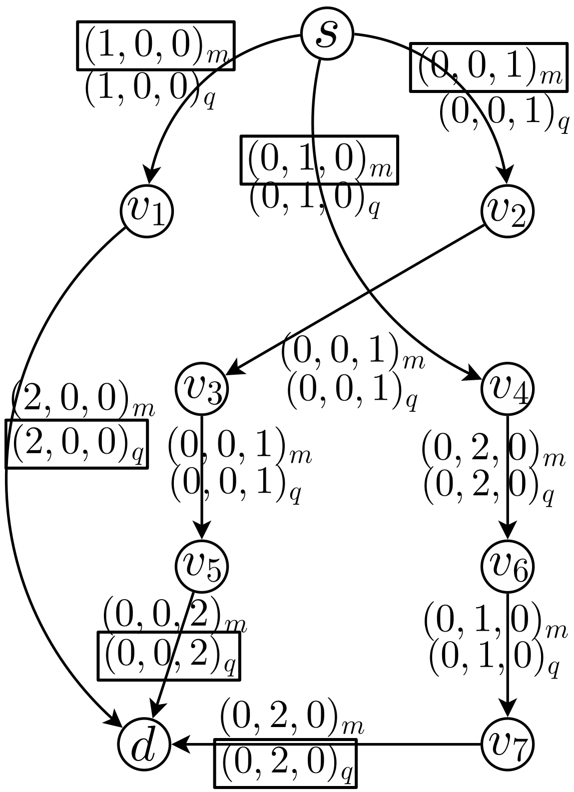
Φ: $[0, 1] \Rightarrow E_v = \{(v_2, v_3)\}$
$E_R(v) = \{(v_1, v_3)\}$

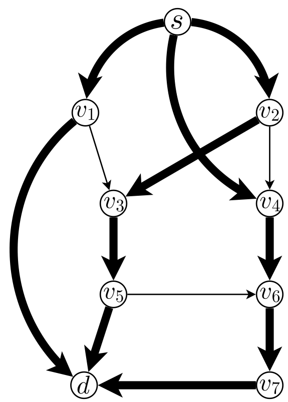
An Illustrative Example





An Illustrative Example



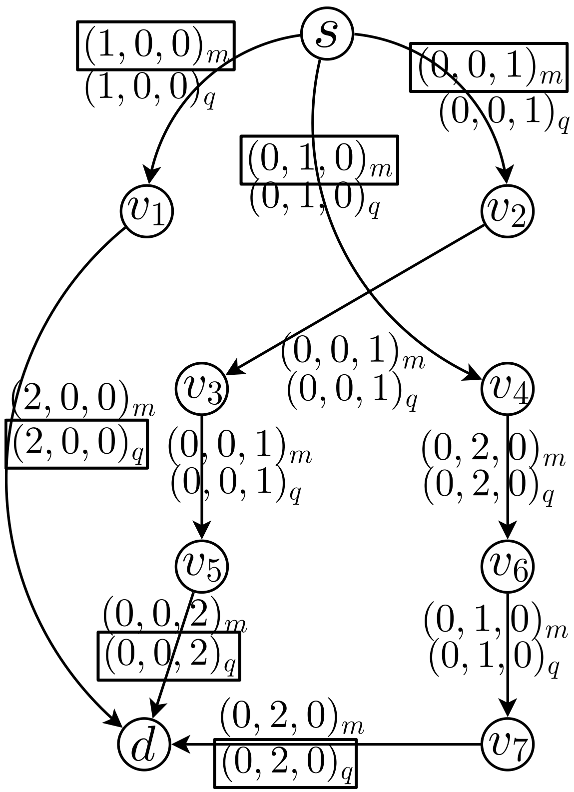


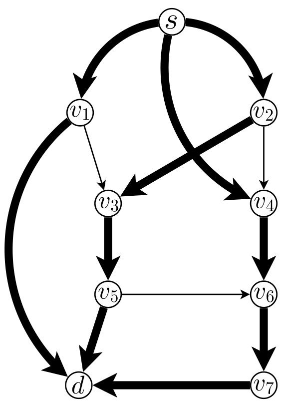
Search v_2:

Out(v_2): $q_{v_2,v_3} = (0,0,1)$
In(v_2): $m_{s,v_2} = (0,0,1)$

Φ: 1 $\Rightarrow$ Full rank

An Illustrative Example



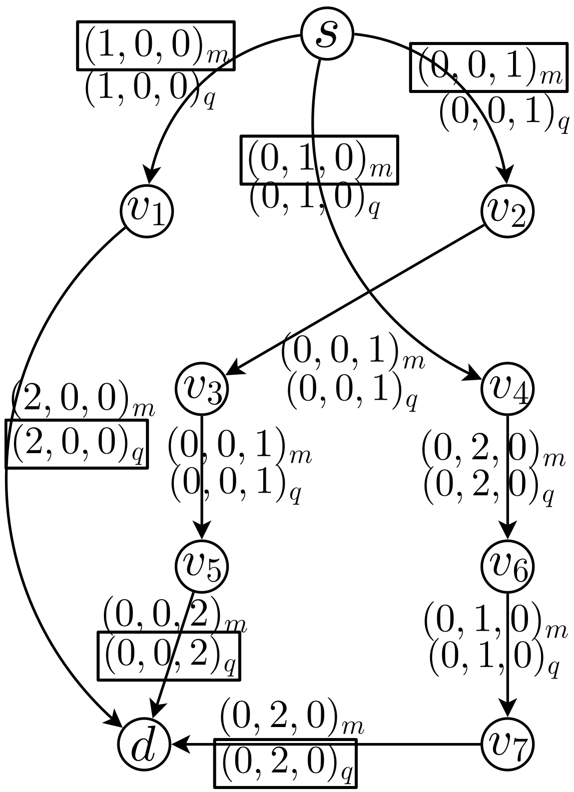


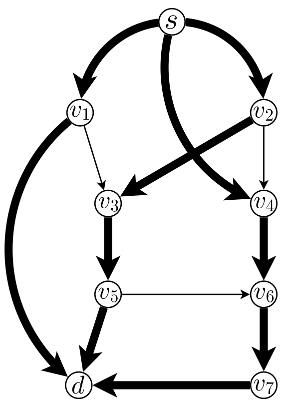
Search v_1:

Out(v_1): $q_{v_1,d} = (2,0,0)$
In(v_1): $m_{s,v_1} = (1,0,0)$

Φ: 2 $\Rightarrow$ Full rank

An Illustrative Example





Search v_1:

Out(v_1): $q_{v_1,d} = (2,0,0)$
In(v_1): $m_{s,v_1} = (1,0,0)$

Φ: 2 $\Rightarrow$ Full rank

The feedback q_e tells node v what is critical for d. Choose m_e that indeed carries the critical info.

Provable Properties

- **Convergence:** The algorithm stops in $\mathcal{O}(|V|^2)$ seconds.
  - The distributed push-&-relabel algorithm converges in $\mathcal{O}(|V|^2)$.

Provable Properties

- **Convergence:** The algorithm stops in $\mathcal{O}(|V|^2)$ seconds.
  - The distributed push-&-relabel algorithm converges in $\mathcal{O}(|V|^2)$.
- **No interruption to the forward traffic:** Throughout iterations, the dimension of the space received by destination d remains identical.

Provable Properties

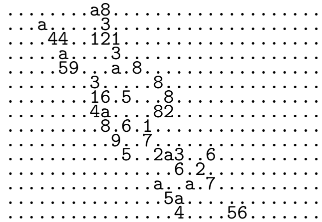
- **Convergence:** The algorithm stops in $\mathcal{O}(|V|^2)$ seconds.
  - The distributed push-&-relabel algorithm converges in $\mathcal{O}(|V|^2)$.
- **No interruption to the forward traffic:** Throughout iterations, the dimension of the space received by destination d remains identical.
- **Correctness:** The remaining graph is a flow.

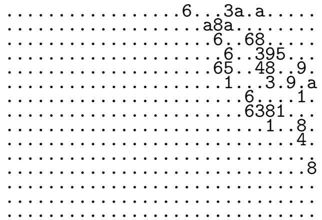
Provable Properties

- **Convergence:** The algorithm stops in $\mathcal{O}(|V|^2)$ seconds.
  - The distributed push-&-relabel algorithm converges in $\mathcal{O}(|V|^2)$.
- **No interruption to the forward traffic:** Throughout iterations, the dimension of the space received by destination d remains identical.
- **Correctness:** The remaining graph is a flow.
- **Correctness with random network coding:** With close-to-1 probability, the output is a max flow.

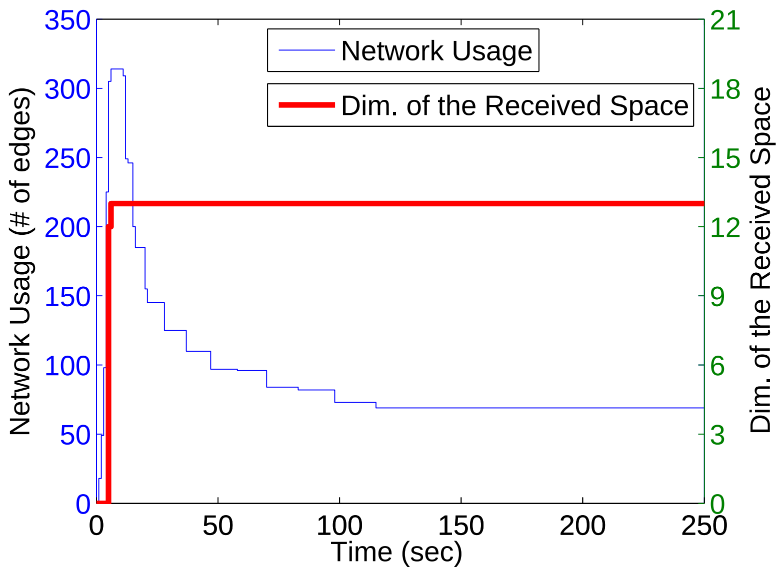
Simulation Results

A 30-node network with

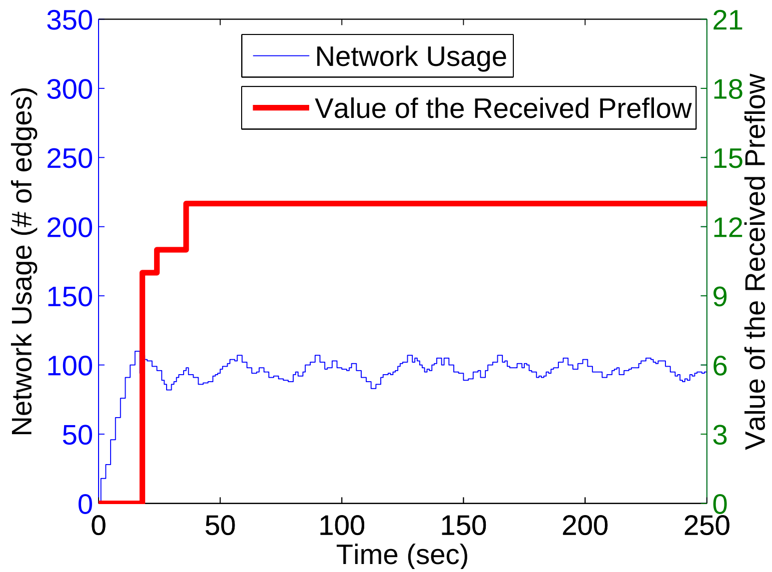




The coding-theoretic approach

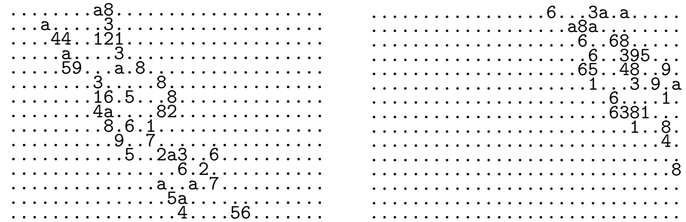


The push-&-relabel algorithm

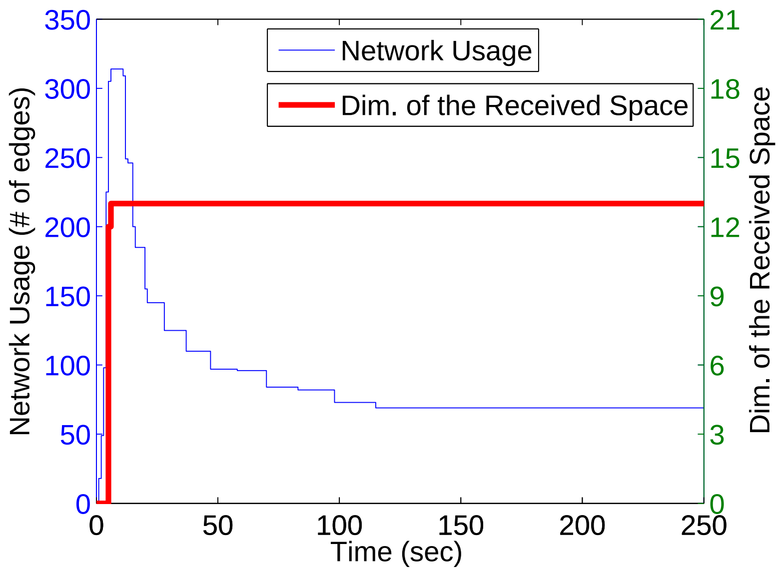


Simulation Results

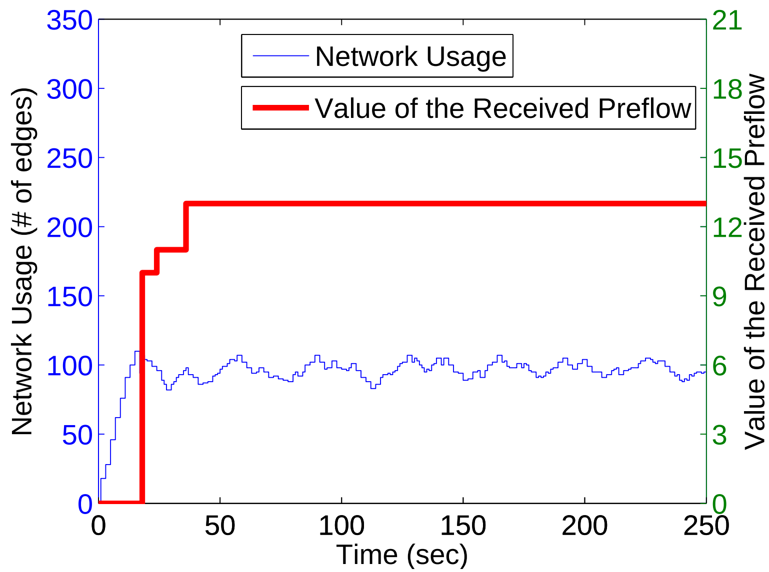
A 30-node network with



The coding-theoretic approach

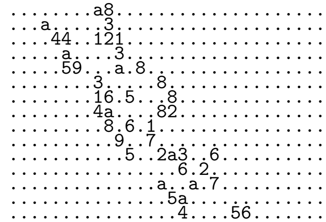
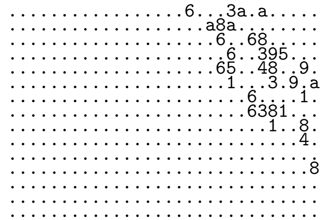


The push-&-relabel algorithm



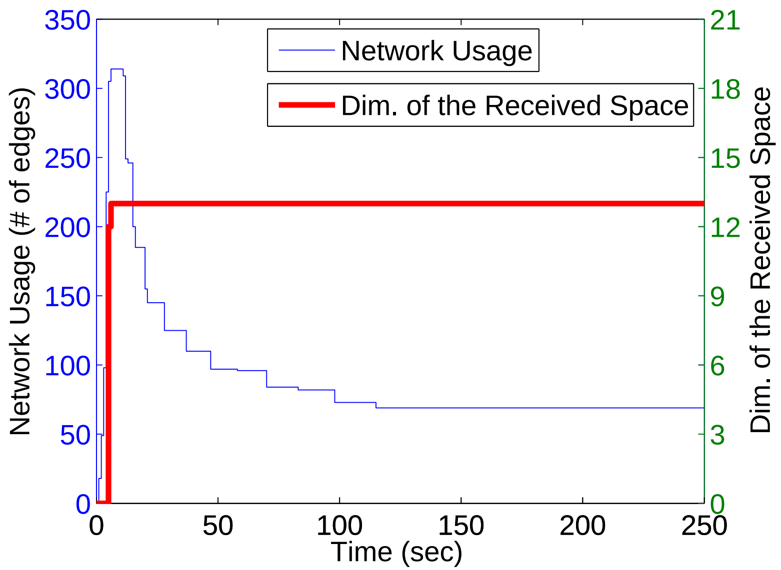
- Achieve the max-flow rate even before convergence.

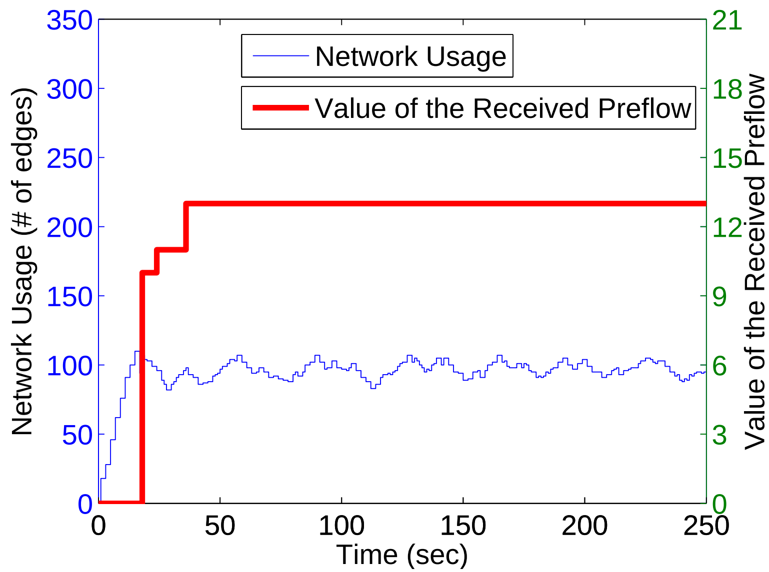
Simulation Results

A 30-node network with  

The coding-theoretic approach

The push-&-relabel algorithm





- Achieve the max-flow rate even before convergence.
- Monotonic traffic reduction vs. oscillating redirction of preflows.

Practical Advantages

- No interruption to forward coded traffic.
- Monotonic traffic reduction.

Practical Advantages

- No interruption to forward coded traffic.
- Monotonic traffic reduction.
- Limited exchange of control packets q_e enables straightforward distributed implementation.

Practical Advantages

- No interruption to forward coded traffic.
- Monotonic traffic reduction.
- Limited exchange of control packets q_e enables straightforward distributed implementation.
- Opposite direction enables easy piggyback. Use q_e to encode reverse data traffic, ex: video conferencing.

Practical Advantages

- No interruption to forward coded traffic.
- Monotonic traffic reduction.
- Limited exchange of control packets q_e enables straightforward distributed implementation.
- Opposite direction enables easy piggyback. Use q_e to encode reverse data traffic, ex: video conferencing.
- No extra hardware requirement. Only linear operations.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small $\mathsf{GF}(q)$ — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.
- Fully distributed implementation
  - Coded feedback admits distributed implementation.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.
- Fully distributed implementation
  - Coded feedback admits distributed implementation.
  - It is easy to identify many $E_R(v)$ simultaneously.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.
- Fully distributed implementation
  - Coded feedback admits distributed implementation.
  - It is easy to identify many $E_R(v)$ simultaneously.
  - Correctness $\Longrightarrow$ Remove one $E_R(v)$ at a time.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.
- Fully distributed implementation
  - Coded feedback admits distributed implementation.
  - It is easy to identify many $E_R(v)$ simultaneously.
  - Correctness $\Longrightarrow$ Remove one $E_R(v)$ at a time.
  - A token-based approach.

Practical Advantages (Cont'd)

- **Flexibility**
  - Delay-sensitive traffic: Controlled broadcast only over paths with $\leq h$ hops
  - Small GF(q) — suboptimal sometimes.
  - Achieve the best possible under suboptimal scenarios.
- Fully distributed implementation
  - Coded feedback admits distributed implementation.
  - It is easy to identify many $E_R(v)$ simultaneously.
  - Correctness $\Longrightarrow$ Remove one $E_R(v)$ at a time.
  - A token-based approach.
  - Random waiting. (With only small probability that the rank will decrease.)

Conclusion

- The first coding-theoretic max-flow algorithm
- Provably good properties and fast convergence speed
- Maintains the delay minimality of network coding
- Many practical advantages as only coded feedback is used.

Conclusion

- The first coding-theoretic max-flow algorithm
- Provably good properties and fast convergence speed
- Maintains the delay minimality of network coding
- Many practical advantages as only coded feedback is used.
- ♡ Generalizable to searching for the max-flow with minimal cost.

Conclusion

- The first coding-theoretic max-flow algorithm
- Provably good properties and fast convergence speed
- Maintains the delay minimality of network coding
- Many practical advantages as only coded feedback is used.

- ♡ Generalizable to searching for the max-flow with minimal cost.
- ♡ Generalizable to multicast traffic (submitted to Allerton 08).