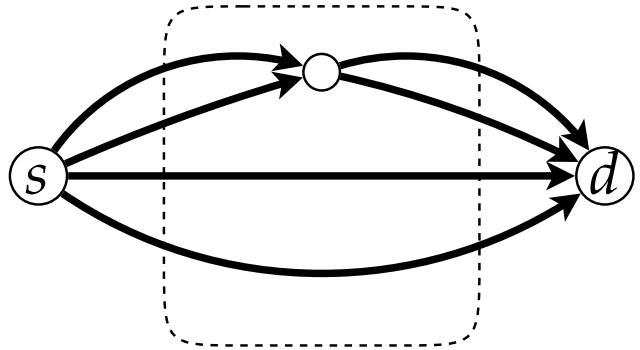
A Coded-Feedback Construction of Locally Minimum-Cost Multicast Network Codes

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

Max-Flow & Multicast Network Codes

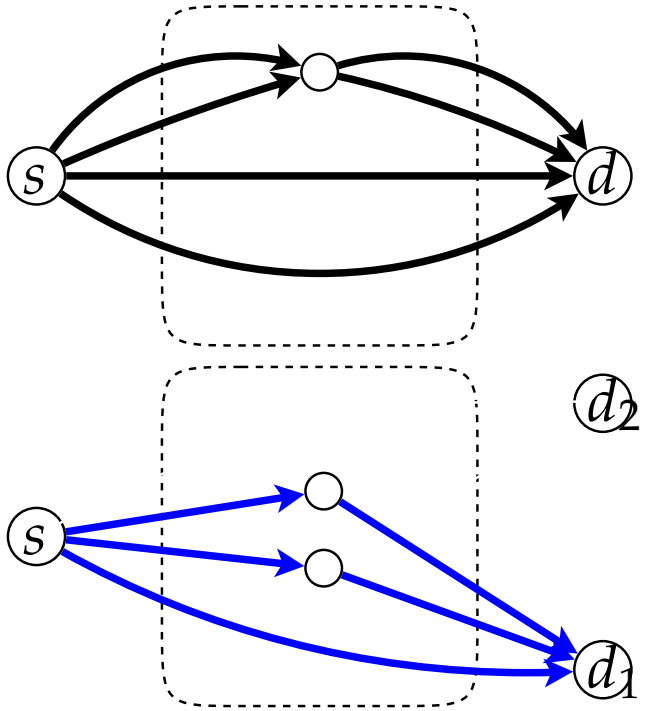
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



Max-Flow & Multicast Network Codes

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

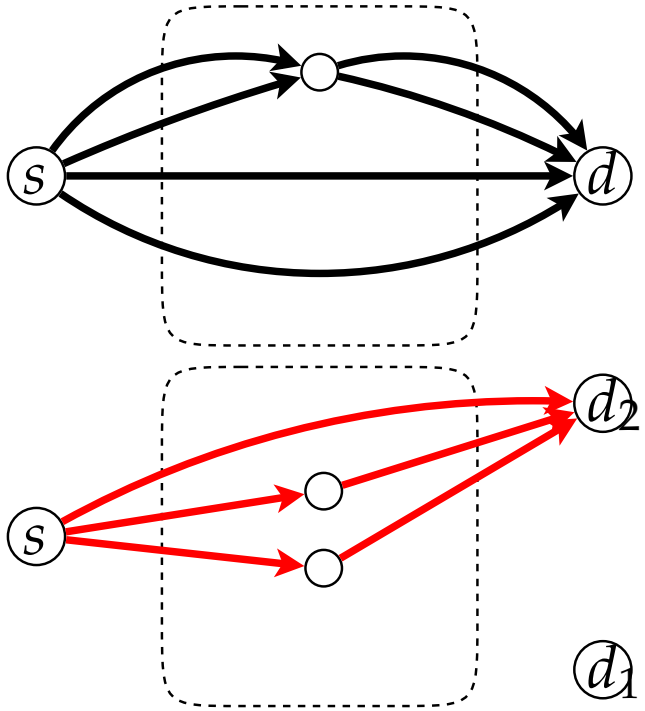
- Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \text{MFV}_i$ for all source-destination pairs (s,d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]



Max-Flow & Multicast Network Codes

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

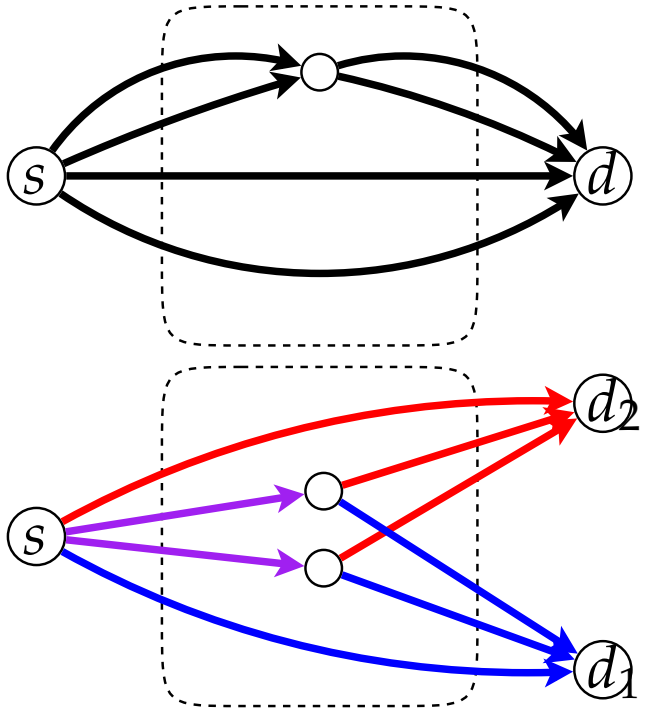
- Bandwidth-efficient network coding solutions.
  - A multicast rate r is supportable iff $r \leq \text{MFV}_i$ for all source-destination pairs (s,d_i). [Ahlswede *et al.* 00], [Li *et al.* 03]



Max-Flow & Multicast Network Codes

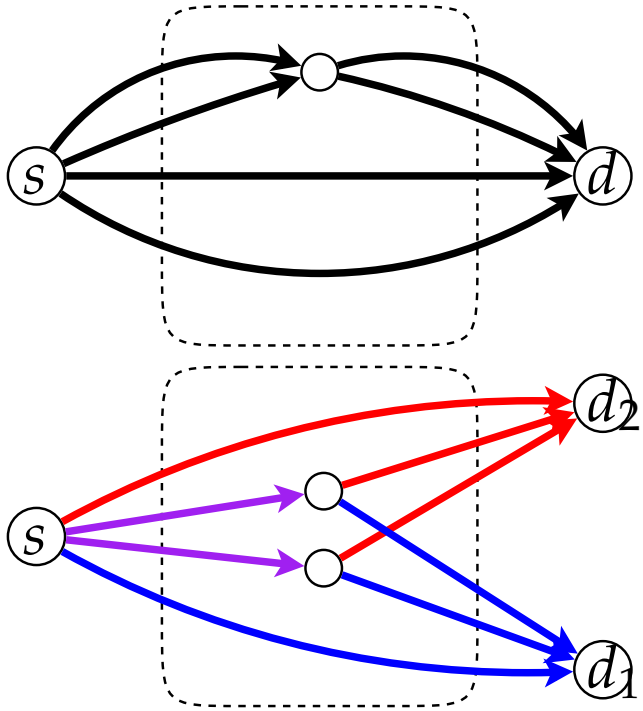
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).

- 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]



Max-Flow & Multicast Network Codes

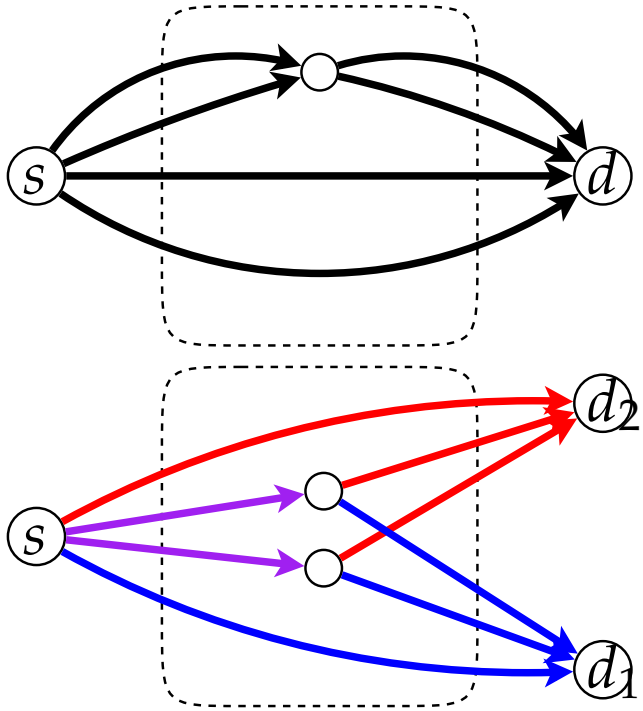
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



- 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]
- Max flows are not unique.

Max-Flow & Multicast Network Codes

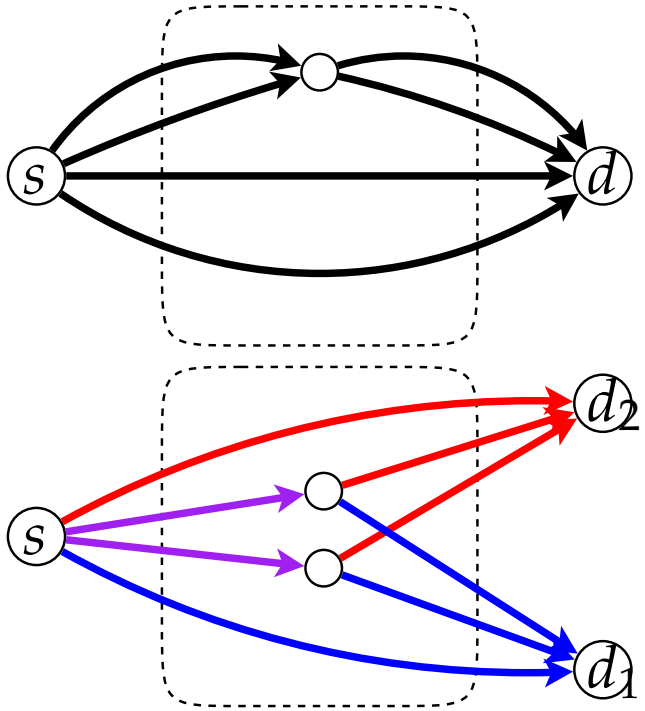
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



- 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]
- Max flows are not unique.
- Unicast: A cost-minimizing max flow.

Max-Flow & Multicast Network Codes

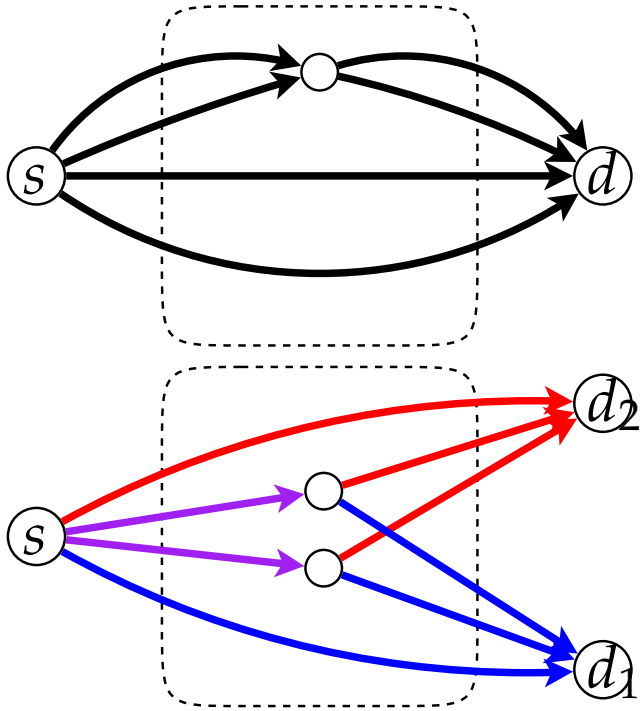
(s,d)-Flow, max (s,d)-flow, and the max-flow value (MFV).



- 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]
- Max flows are not unique.
- Unicast: A cost-minimizing max flow.
- Multicast: The union of cost-minimizing max flows?

Max-Flow & Multicast Network Codes

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



- 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]
- Max flows are not unique.
- Unicast: A cost-minimizing max flow.
- Multicast: Find a min-cost $G' \subseteq G$ s.t. $\text{MFV}_i(G) = \text{MFV}_i(G')$.

Content

- Existing solutions: graph-theoretic versus LP-based ones.

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].
- Finding low-cost multicast network codes.
  - ♡ A strengthened coded-feedback approach.

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].
- Finding low-cost multicast network codes.
  - ♡ A strengthened coded-feedback approach.
  - Locally minimum-cost network codes.

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].
- Finding low-cost multicast network codes.
  - ♡ A strengthened coded-feedback approach.
  - Locally minimum-cost network codes.
  - Correctness & complexity analysis

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].
- Finding low-cost multicast network codes.
  - ♡ A strengthened coded-feedback approach.
  - Locally minimum-cost network codes.
  - Correctness & complexity analysis
  - Numerical experiments.

Content

- Existing solutions: graph-theoretic versus LP-based ones.
- A new coded feedback framework [ISIT08].
- Finding low-cost multicast network codes.
  - ♡ A strengthened coded-feedback approach.
  - Locally minimum-cost network codes.
  - Correctness & complexity analysis
  - Numerical experiments.
- Conclusion

Existing Results

- Graph-theoretic max-flow algorithms:
  - Ford-Fulkerson 1956, Edmonds-Karp 1972, Dinitz 1970,
  - Push & relabel algorithm [Goldberg, Tarjan 1988]

Existing Results

- Graph-theoretic max-flow algorithms:
  - Ford-Fulkerson 1956, Edmonds-Karp 1972, Dinitz 1970,
  - Push & relabel algorithm [Goldberg, Tarjan 1988]
  - Fast convergence (although not designed for network communications).

Existing Results

- Graph-theoretic max-flow algorithms:
  - Ford-Fulkerson 1956, Edmonds-Karp 1972, Dinitz 1970,
  - Push & relabel algorithm [Goldberg, Tarjan 1988]
  - Fast convergence (although not designed for network communications).
  - Output one max-flow. No built-in cost minimization.

Existing Results

- Graph-theoretic max-flow algorithms:
  - Ford-Fulkerson 1956, Edmonds-Karp 1972, Dinitz 1970,
  - Push & relabel algorithm [Goldberg, Tarjan 1988]
  - Fast convergence (although not designed for network communications).
  - Output one max-flow. No built-in cost minimization.
  - Multicast?

Existing Results (Cont'd)

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

Existing Results (Cont'd)

- Linear-programming (LP) based algorithms
  - $$\max_{f_e \geq 0} \quad \sum_{e \in \mathrm{Out}(s)} f_e - \sum_{e} c(f_e)$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Globally optimal. Applicable to multicast.
  - Complexity: queue-length exchange,

Existing Results (Cont'd)

- Linear-programming (LP) based algorithms
  - $$\max_{f_e \geq 0} \quad \sum_{e \in \mathrm{Out}(s)} f_e - \sum_{e} c(f_e)$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Globally optimal. Applicable to multicast.
  - Complexity: queue-length exchange,
  - Convergence speed: small step sizes of the gradient methods,

Existing Results (Cont'd)

- Linear-programming (LP) based algorithms
  - $$\max_{f_e \geq 0} \quad \sum_{e \in \mathrm{Out}(s)} f_e - \sum_e c(f_e)$$
    $$\text{subject to} \quad \forall v, \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e' \in \mathrm{Out}(v)} f_{e'}$$
  - Globally optimal. Applicable to multicast.
  - Complexity: queue-length exchange,
  - Convergence speed: small step sizes of the gradient methods,
  - Separate rate assignments and coding operations.
    - Fractional rate vs. packet-by-packet coding operations.
    - Time-averaging? Practical generation size (# of to-be-mixed packets) is 32–100.

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]:

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]:
  Run network coding

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]:
  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.

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]:
  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.

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]: 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.

Consider a unicast session. If all max-flows are equal ...

- Classic way of minimizing the bandwidth:
  Run the max-flow algorithm until convergence $\longrightarrow$ Run network coding $\longrightarrow$ Bandwidth optimality
- A new coding-theoretic approach [ISIT08]: 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.

The key innovation is to use coding to find distributedly the redundant edges.

A Coding-Theoretic Approach

The Integer-Rate Network Model:

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

A Coding-Theoretic Approach

The Integer-Rate Network Model:

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

A Coding-Theoretic Approach

The Integer-Rate Network Model:

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

A Coding-Theoretic Approach

The Integer-Rate Network Model:

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.
- A single session (s, d) first ((s, d_i) later): Intrasession network coding
- Coding vector $m = (c_1, c_2, c_3) \Longleftrightarrow X = c_1X_1 + c_2X_2 + c_3X_3$.

A Coding-Theoretic Approach

The Integer-Rate Network Model:

- Finite directed acyclic graph $G = (V, E)$.
- Unit-capacity edge. High-rate link $\Longrightarrow$ parallel edges.
- A single session (s, d) first ((s, d_i) later): 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 $q = 3$.

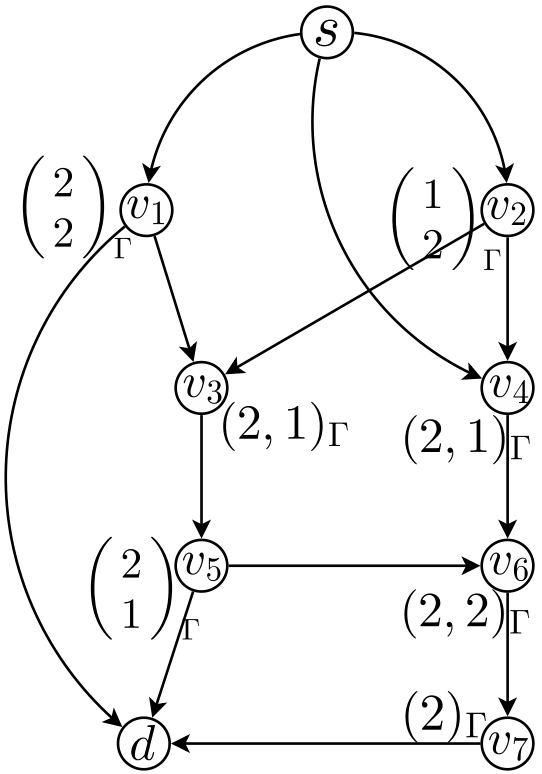
The Coded Feedback Approach

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

The Coded Feedback Approach

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

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

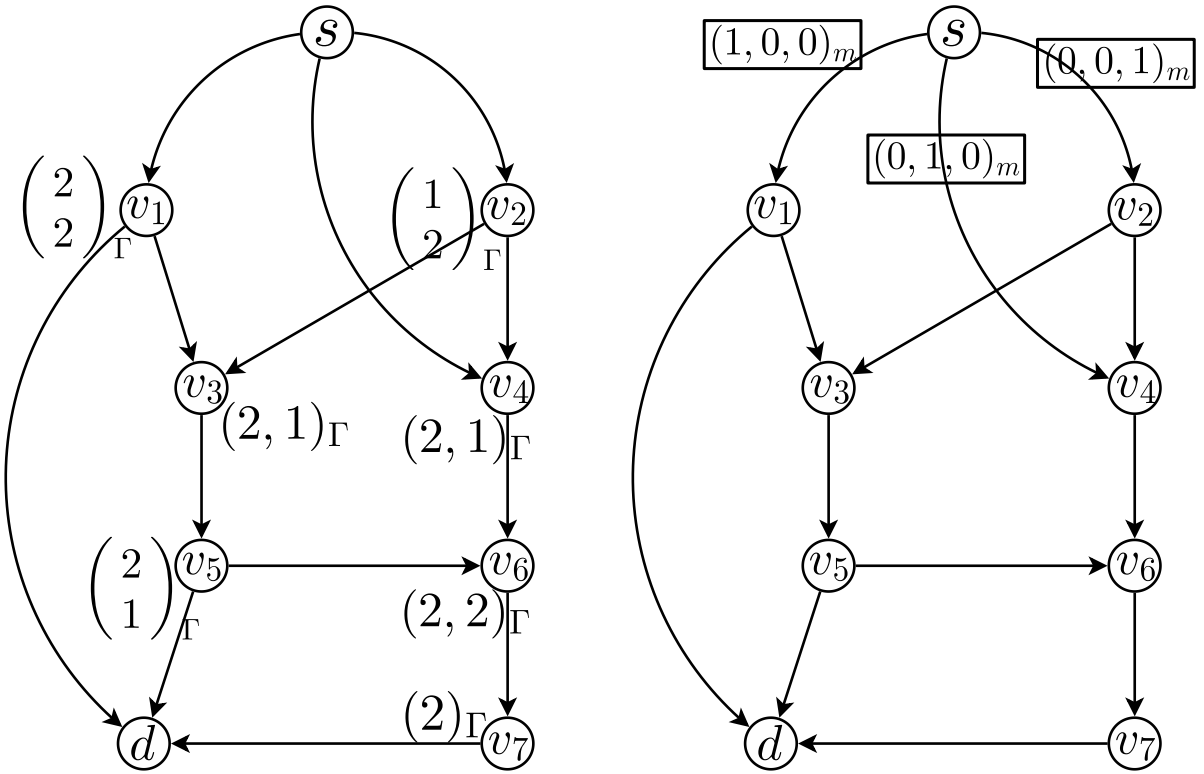


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

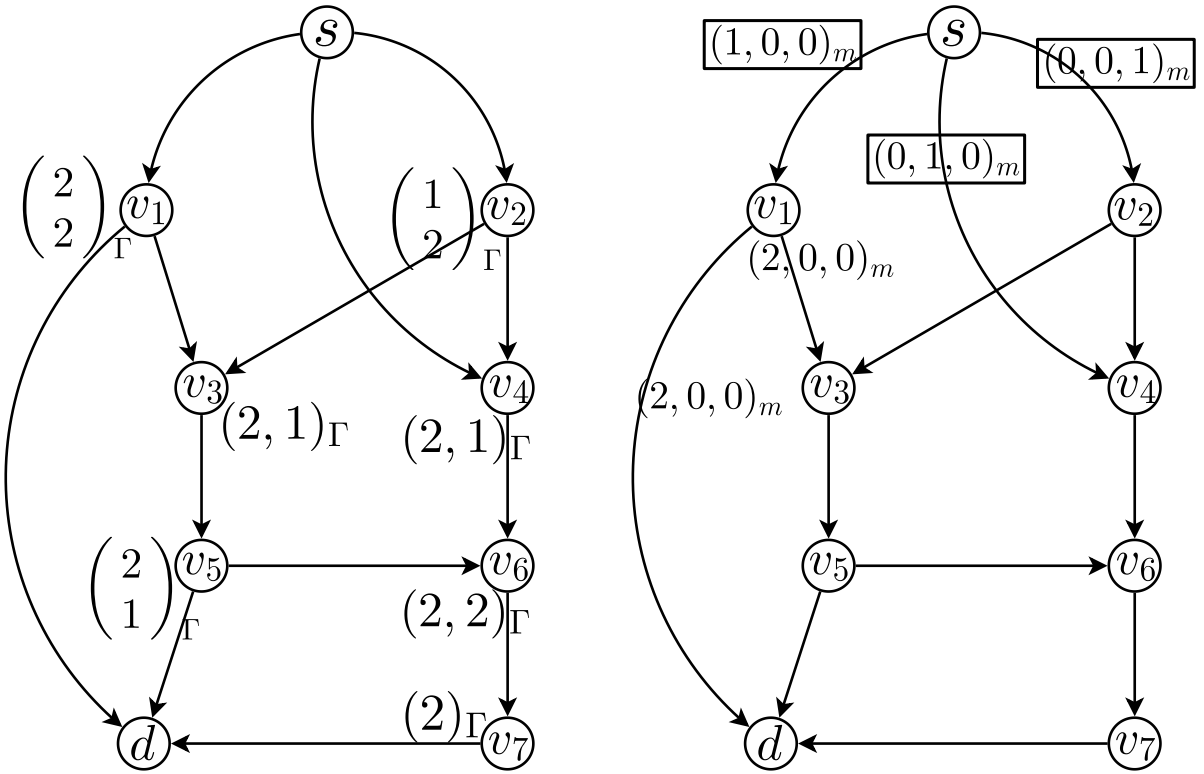


The Coded Feedback Approach

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

Step 2: Compute the coding vectors m_e

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

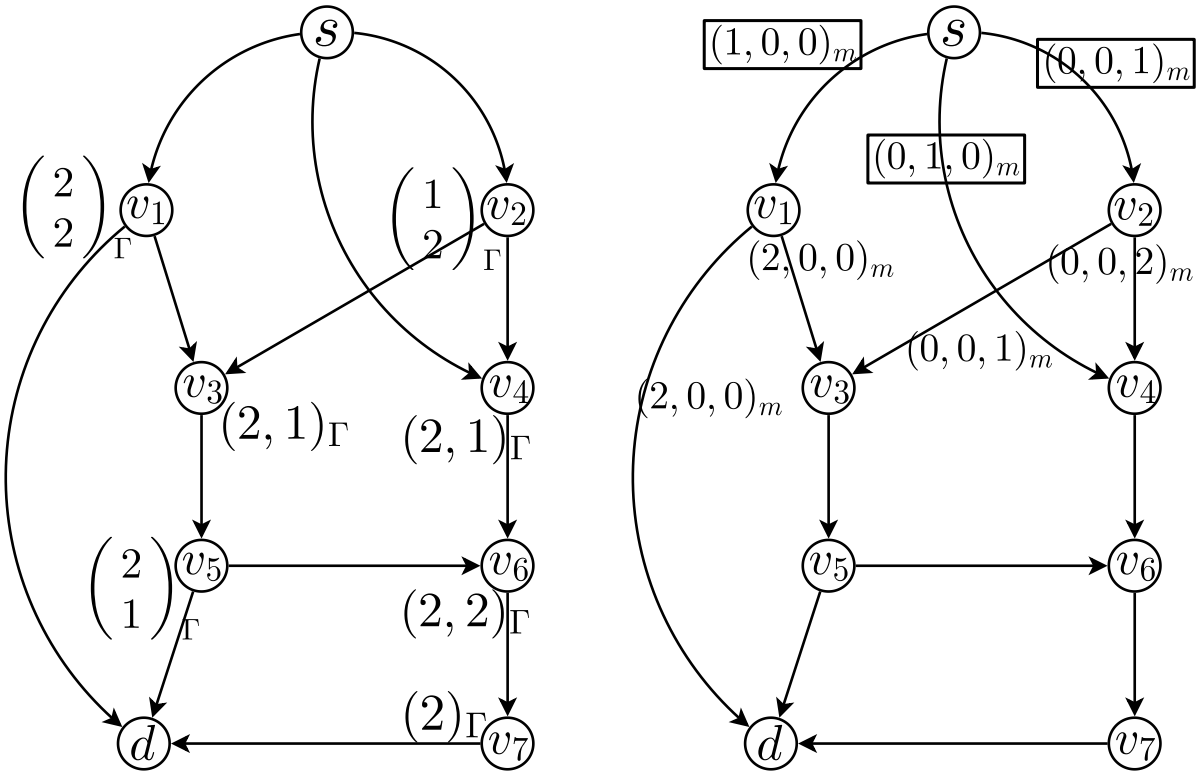


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

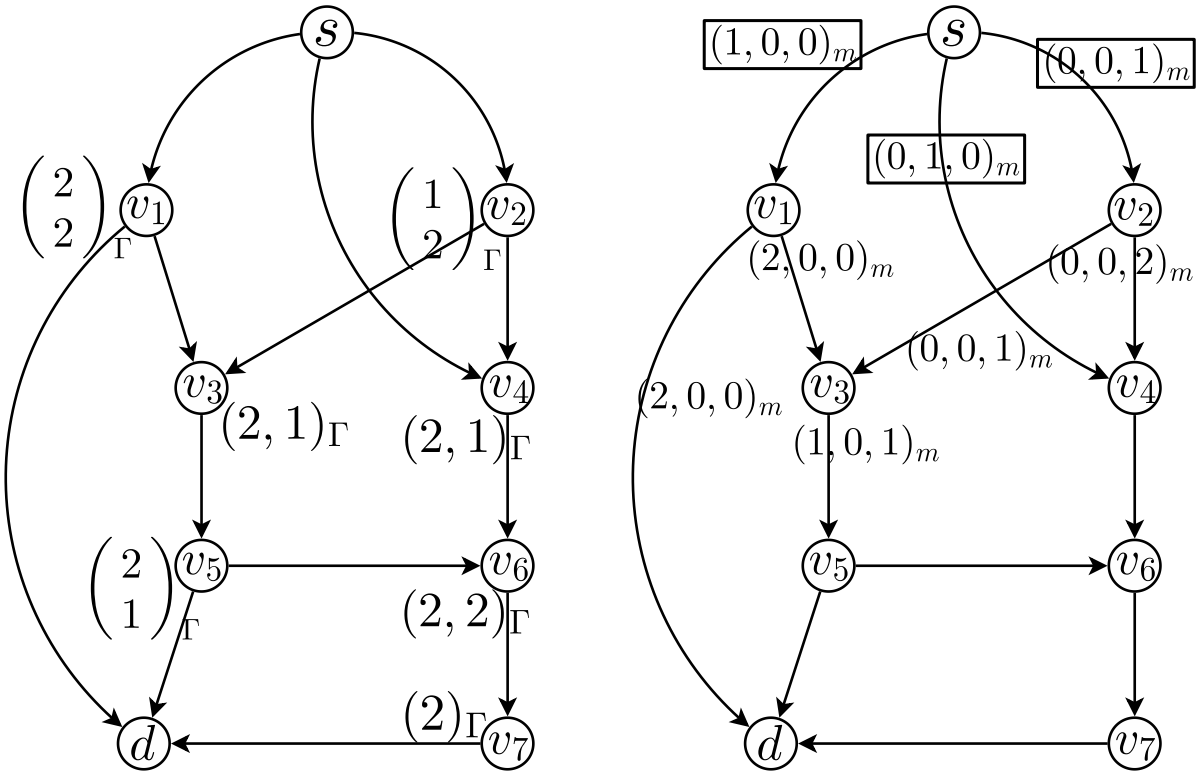


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

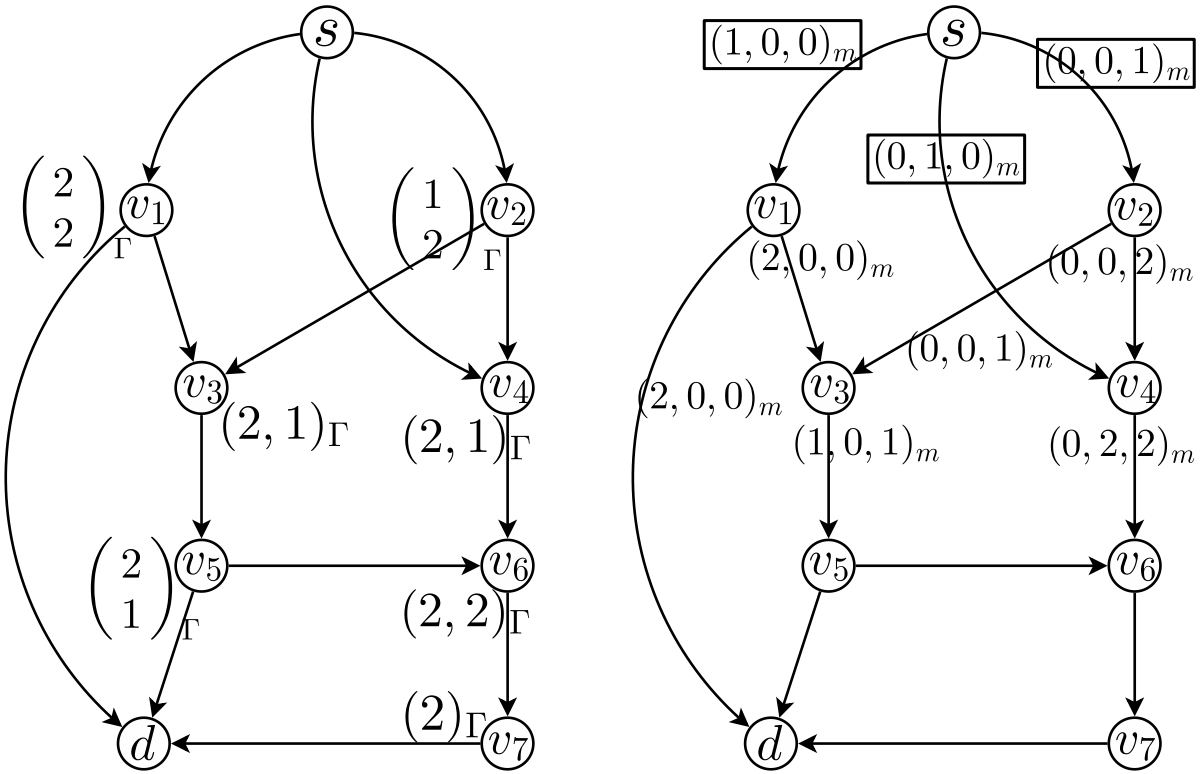


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

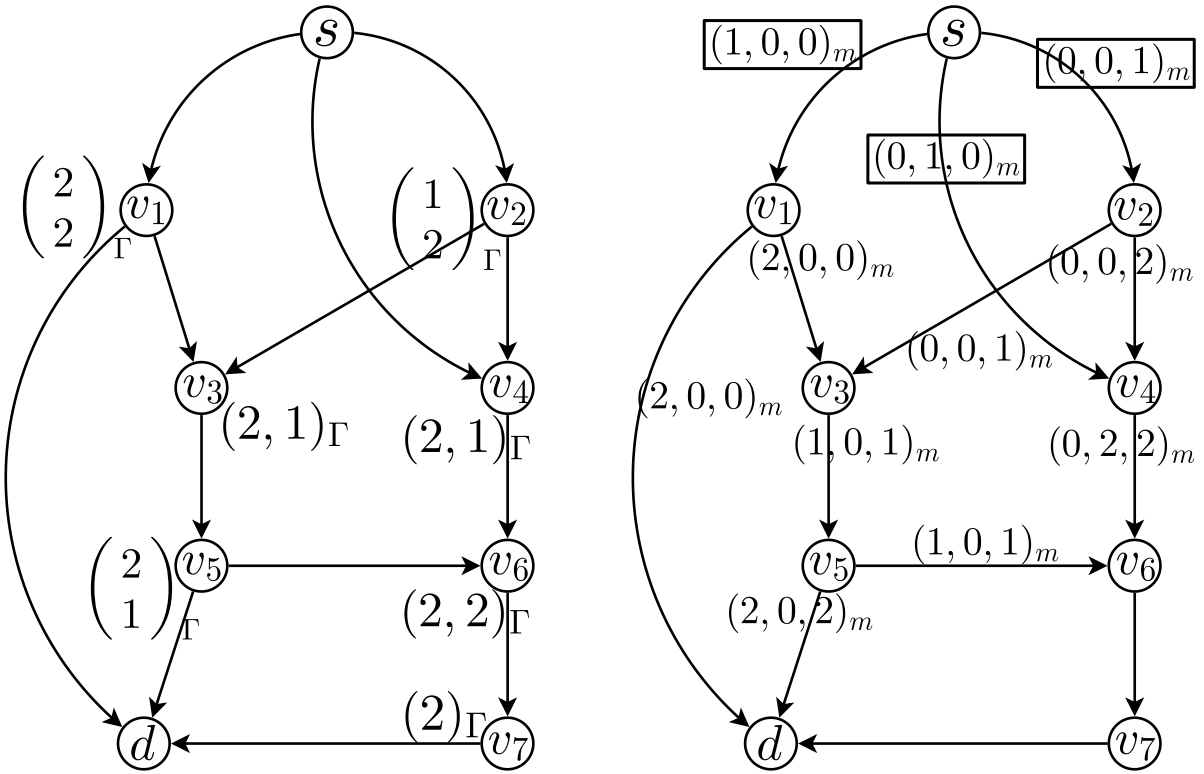


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

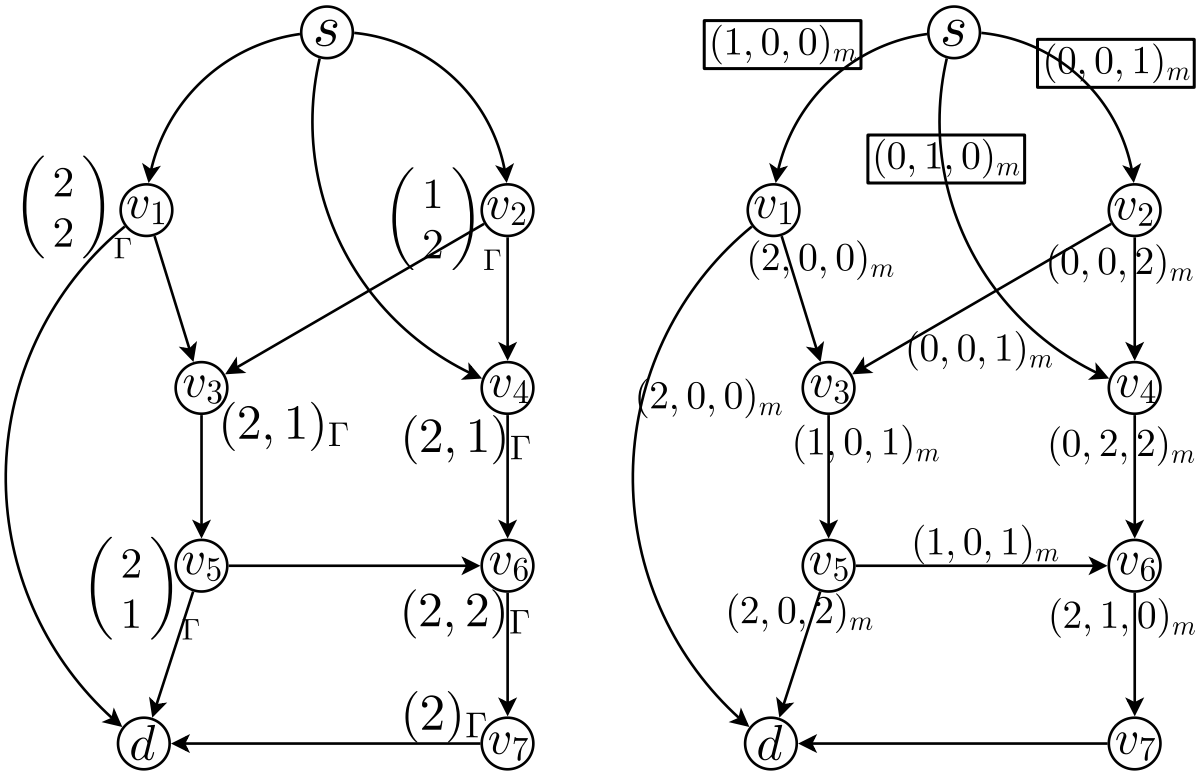


The Coded Feedback Approach

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 $\mathsf{GF}(3)$

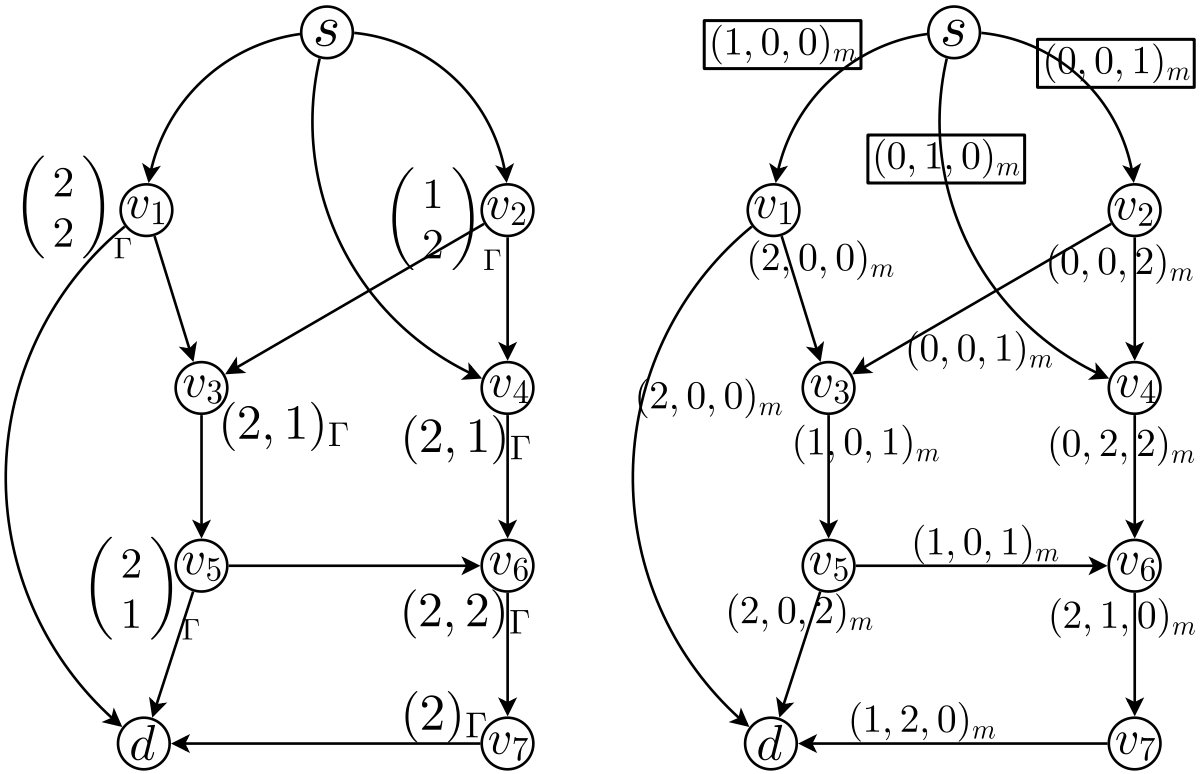


The Coded Feedback Approach

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 $\mathsf{GF}(3)$



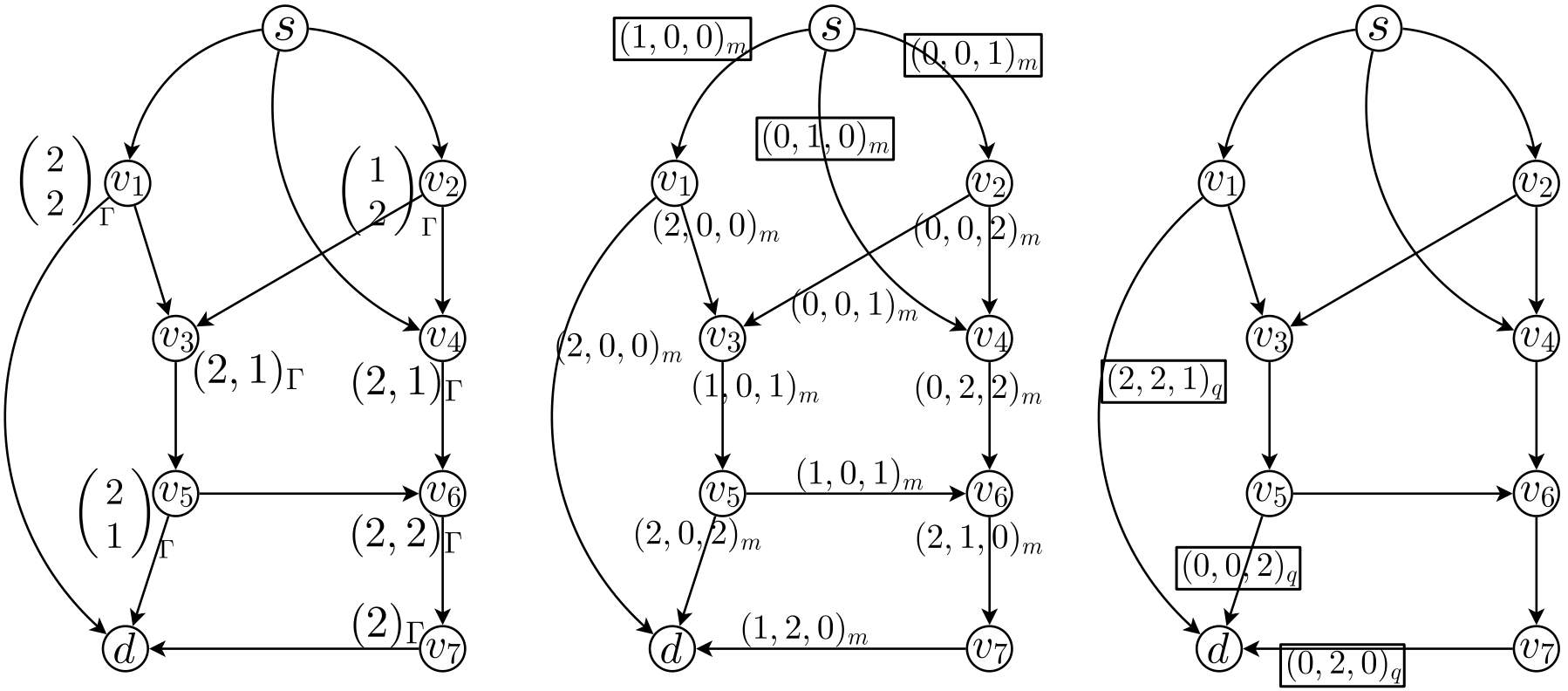
The Coded Feedback Approach

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

Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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



The Coded Feedback Approach

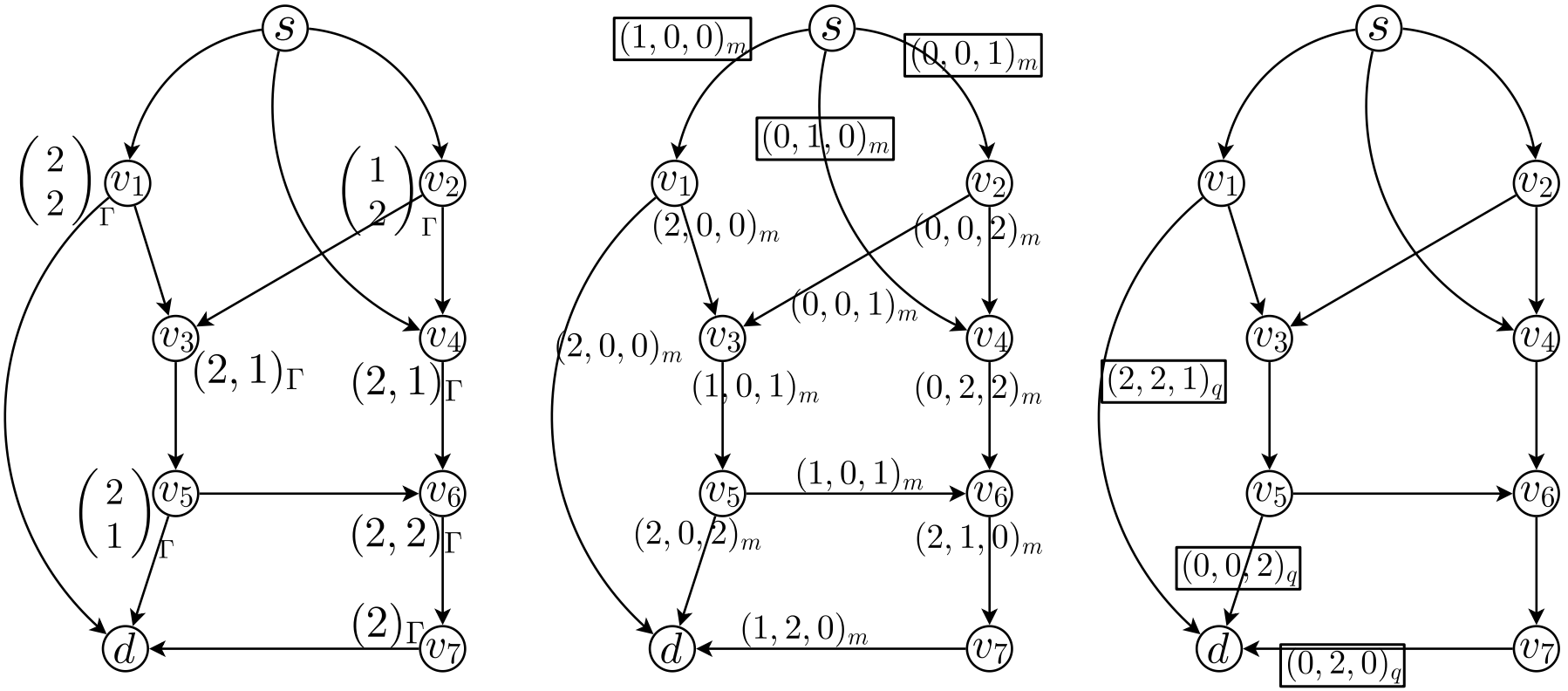
Step 1: Choose the $|\mathrm{Out}(v)| \times |\mathrm{In}(v)|$ mixing matrix $\Gamma(v)$

Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **Orthogonal Coded Feedback**



The Coded Feedback Approach

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

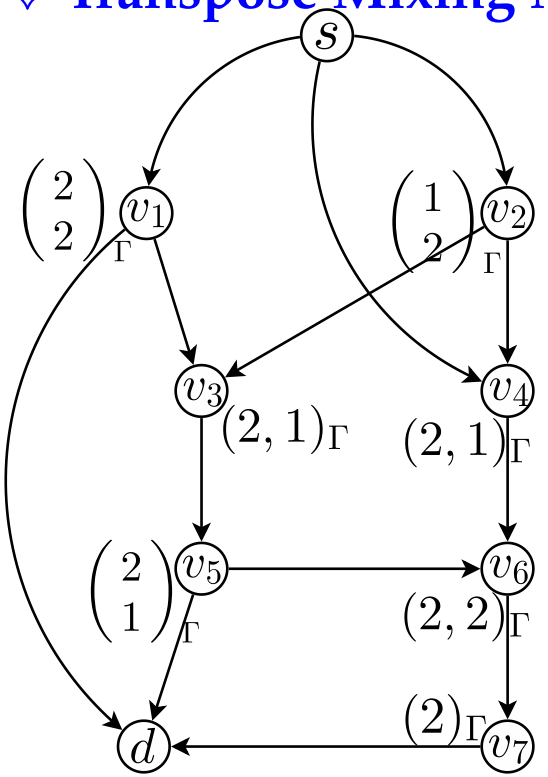
Step 2: Compute the coding vectors m_e

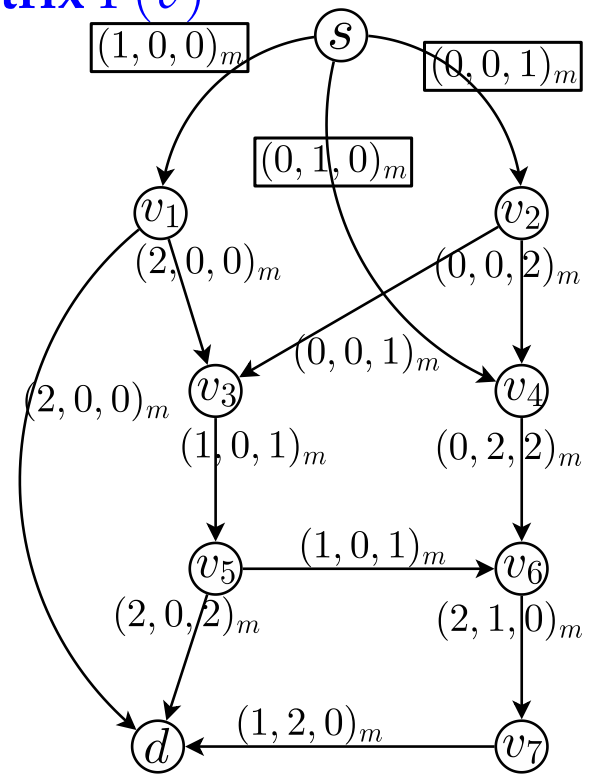
Step 3: Compute the coded feedback q_e

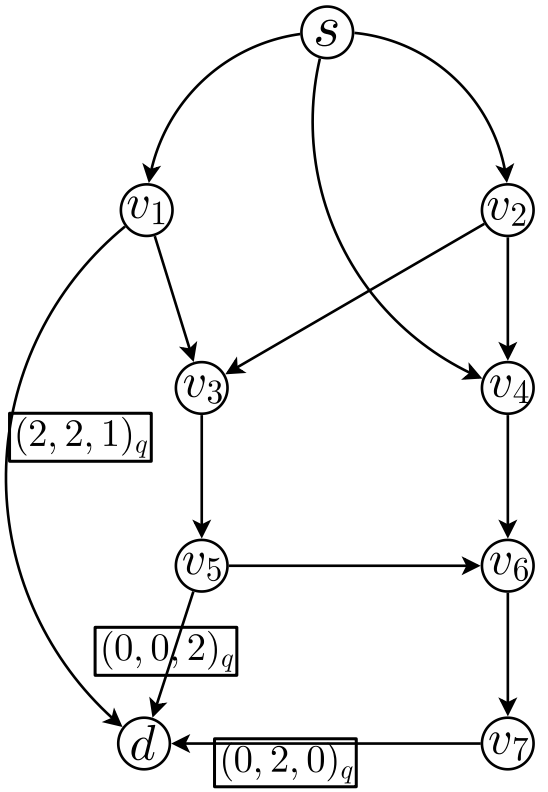
Network coding on $\mathrm{GF}(3)$

♡ **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 $\Gamma(v)$

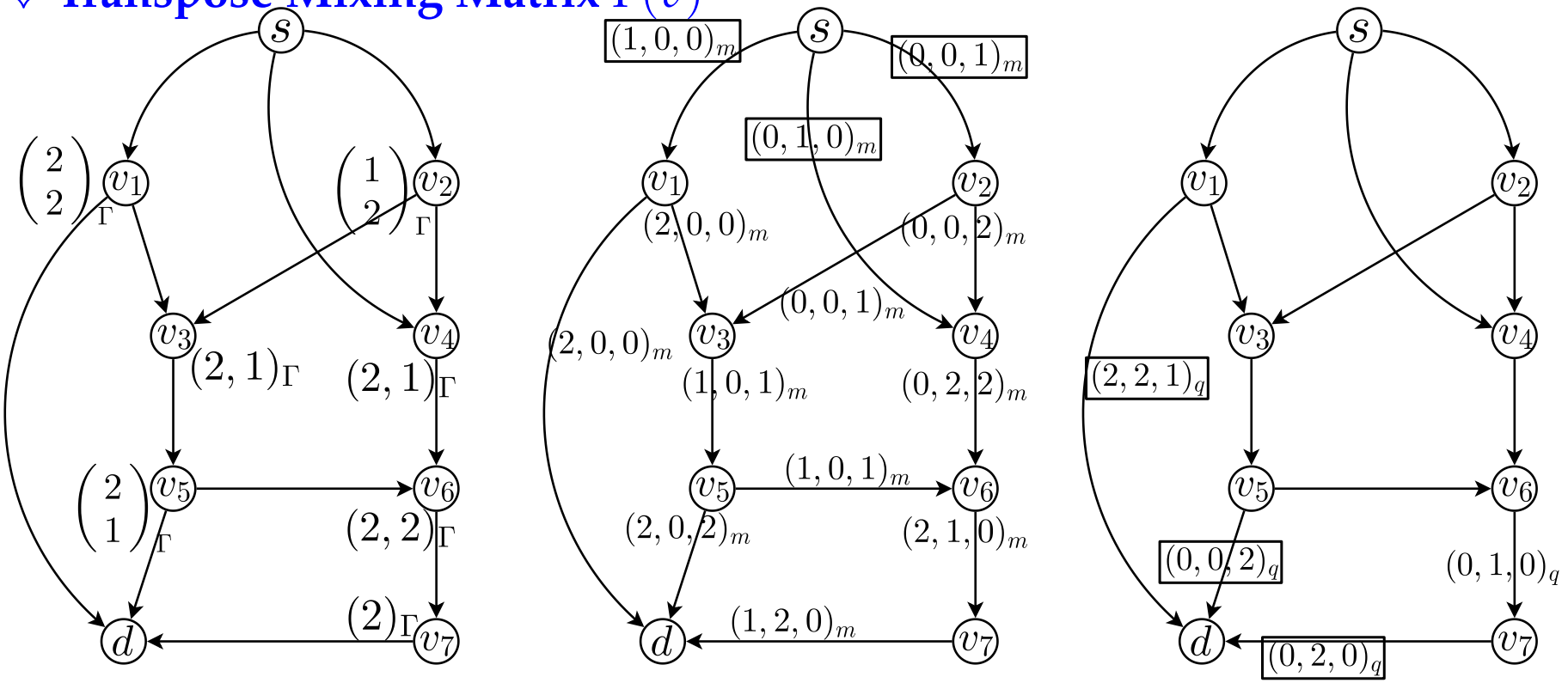
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **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 $\Gamma(v)$

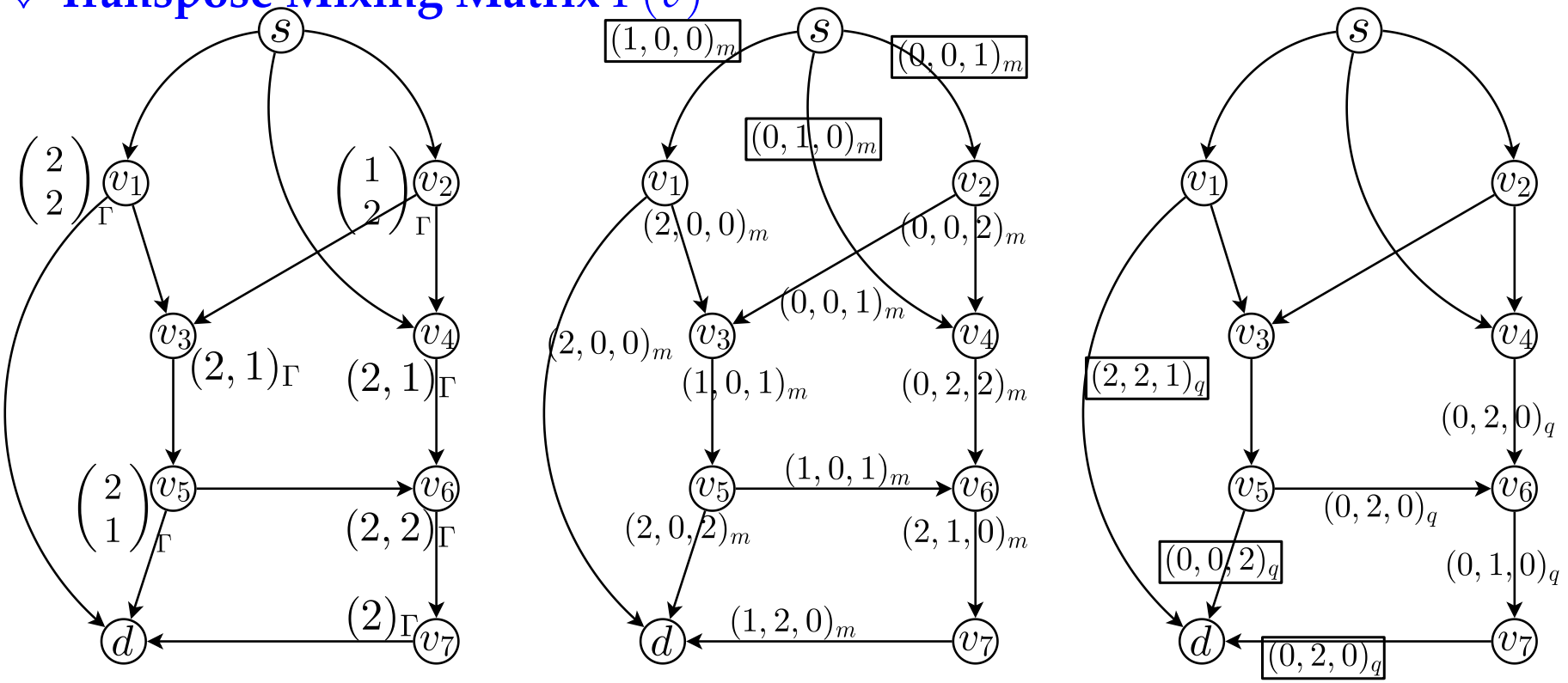
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **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 $\Gamma(v)$

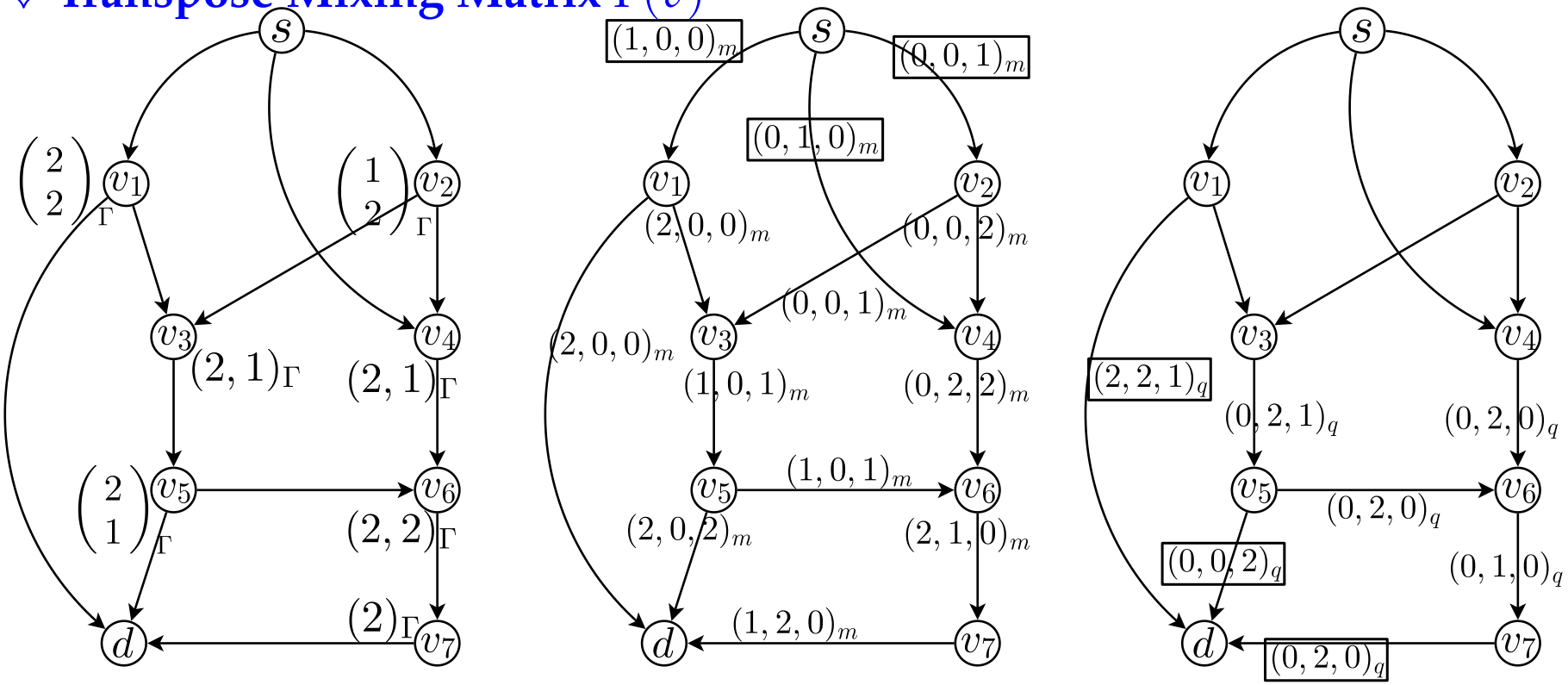
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **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 $\Gamma(v)$

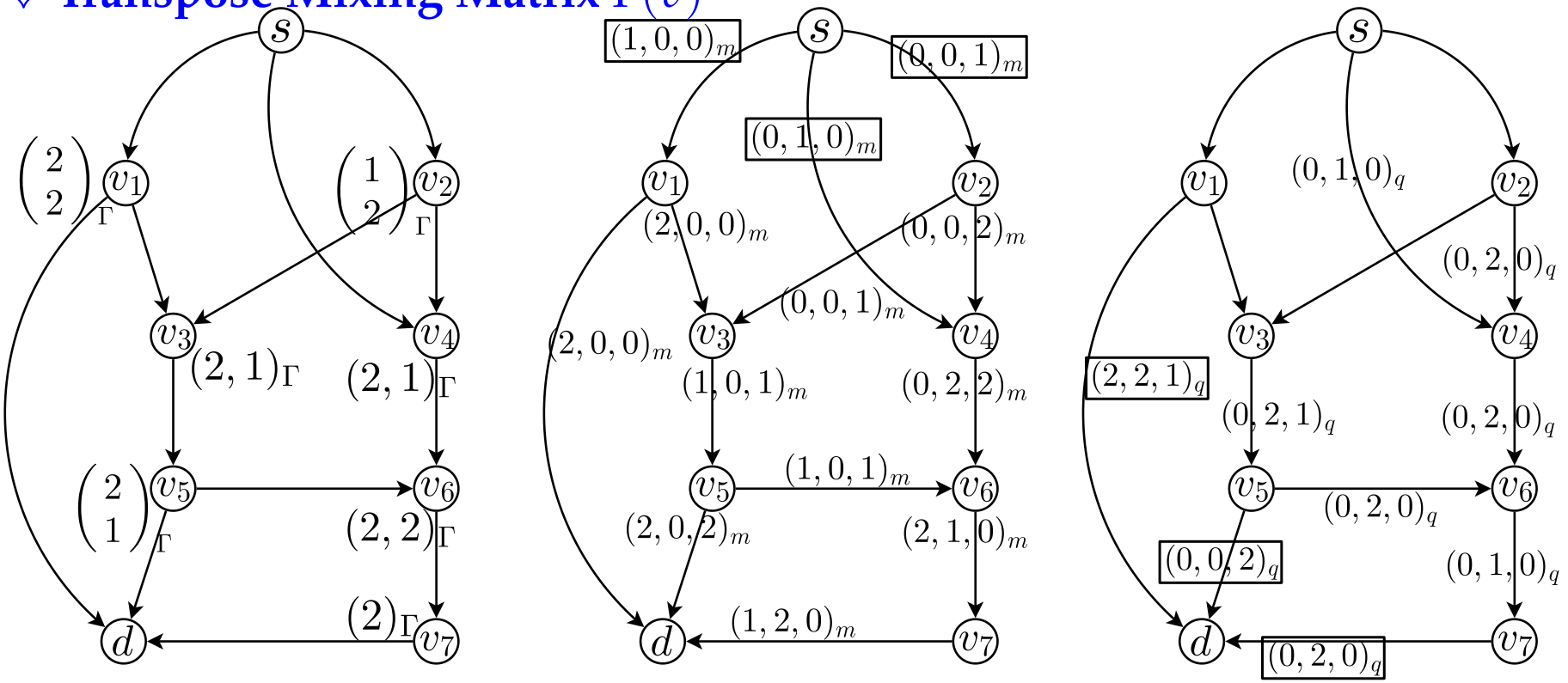
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **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 $\Gamma(v)$

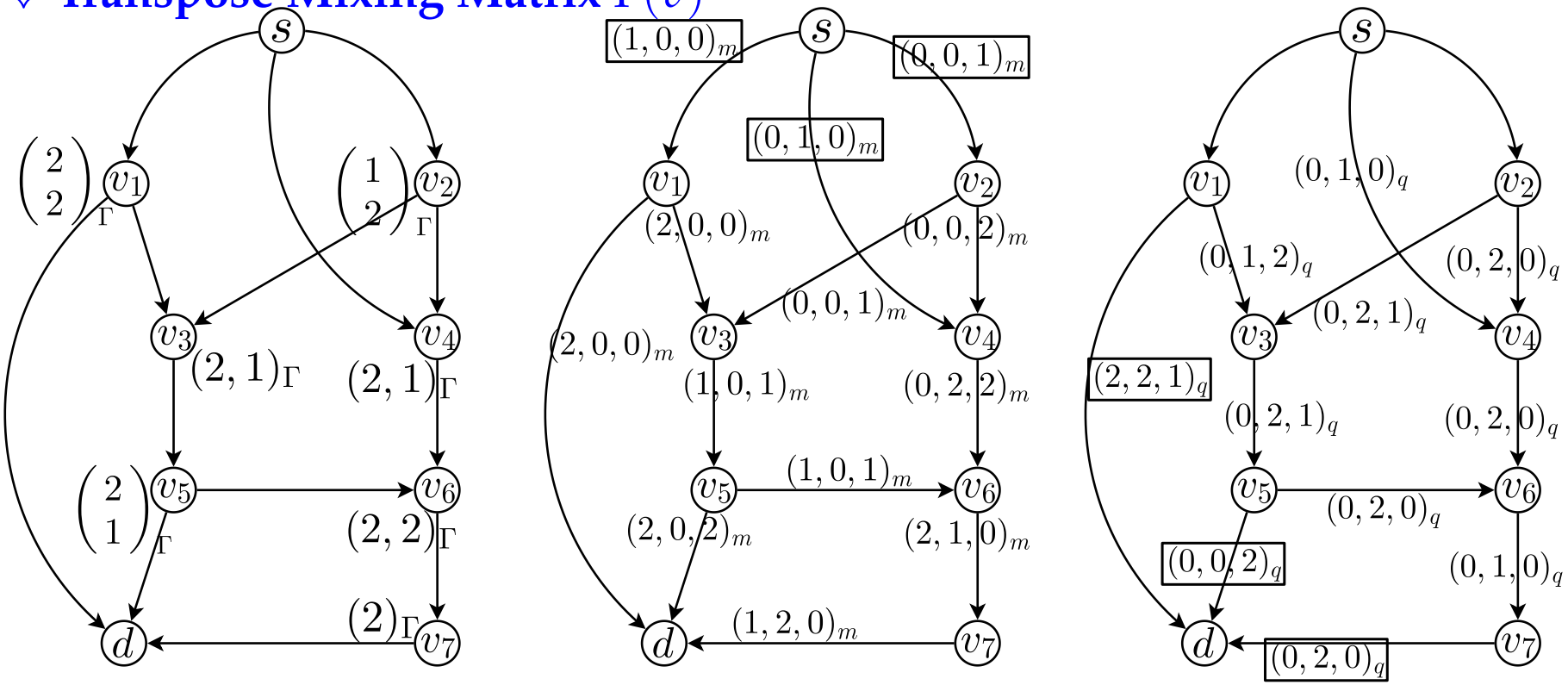
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

♡ **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 $\Gamma(v)$

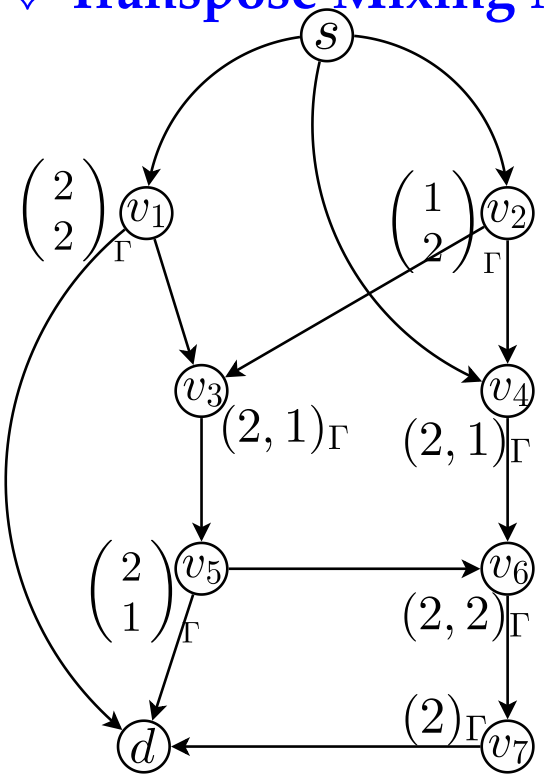
Step 2: Compute the coding vectors m_e

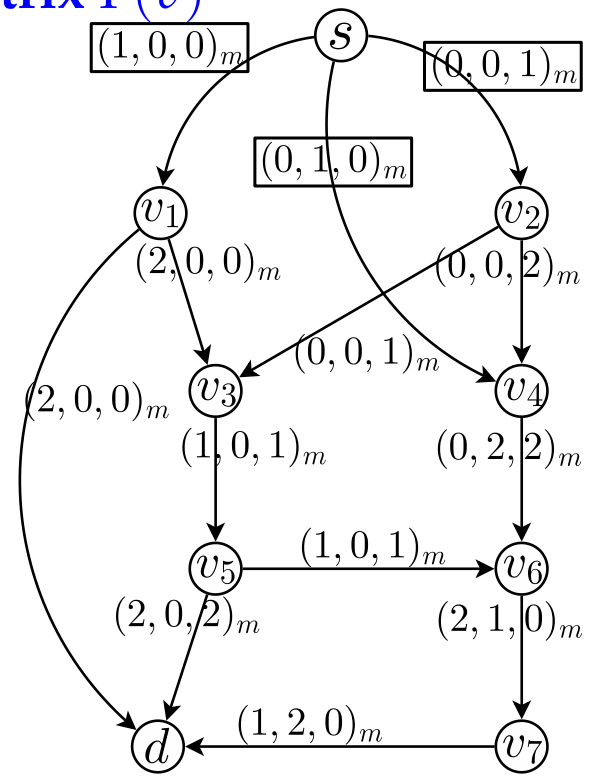
Step 3: Compute the coded feedback q_e

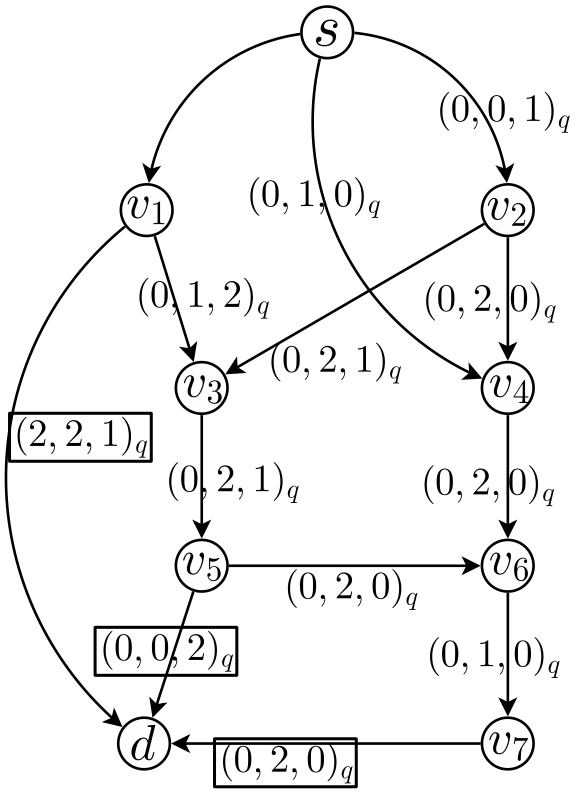
Network coding on $\mathrm{GF}(3)$

♡ **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 $\Gamma(v)$

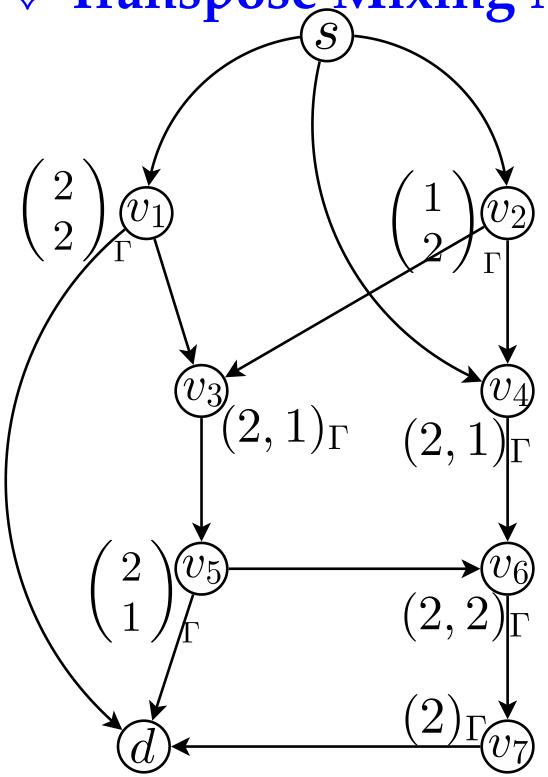
Step 2: Compute the coding vectors m_e

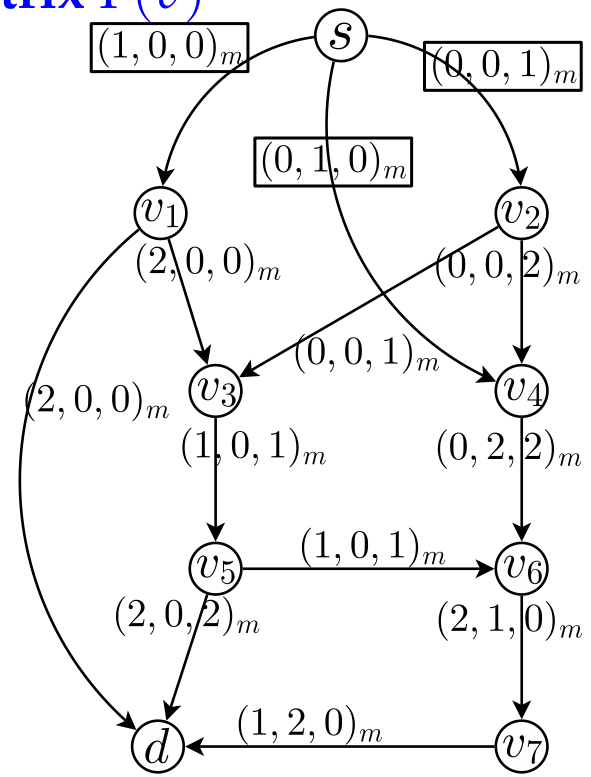
Step 3: Compute the coded feedback q_e

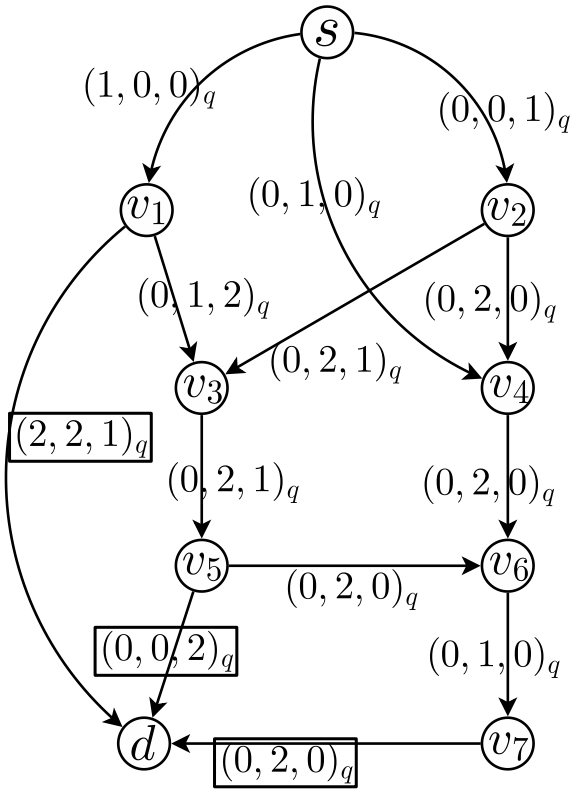
Network coding on $\mathrm{GF}(3)$

♡ **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 $\Gamma(v)$

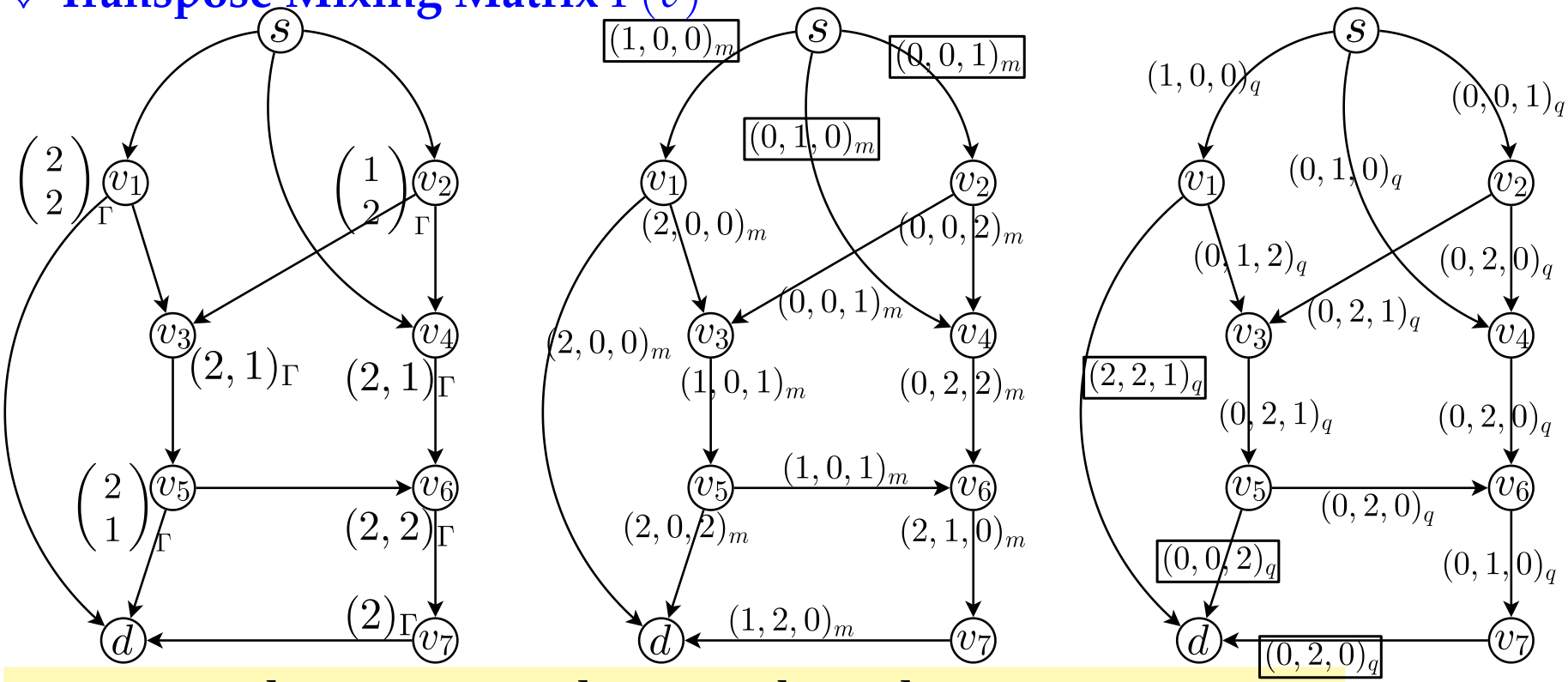
Step 2: Compute the coding vectors m_e

Step 3: Compute the coded feedback q_e

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

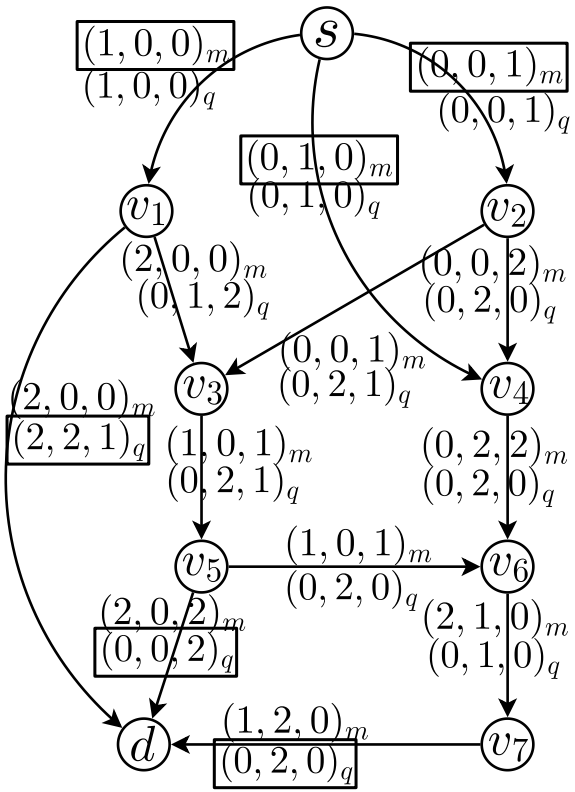
♡ **Orthogonal Coded Feedback**

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



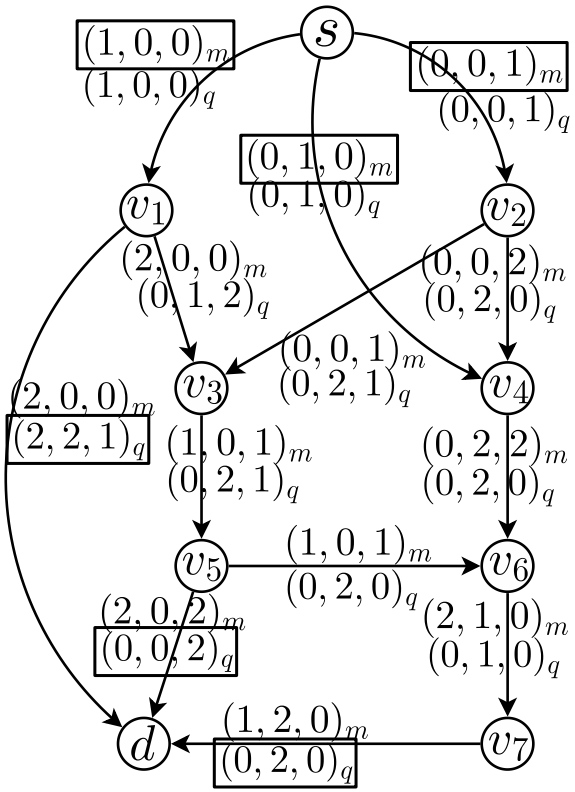
Steps 1 and 2 are Normal Network Coding. Step 3 is new.

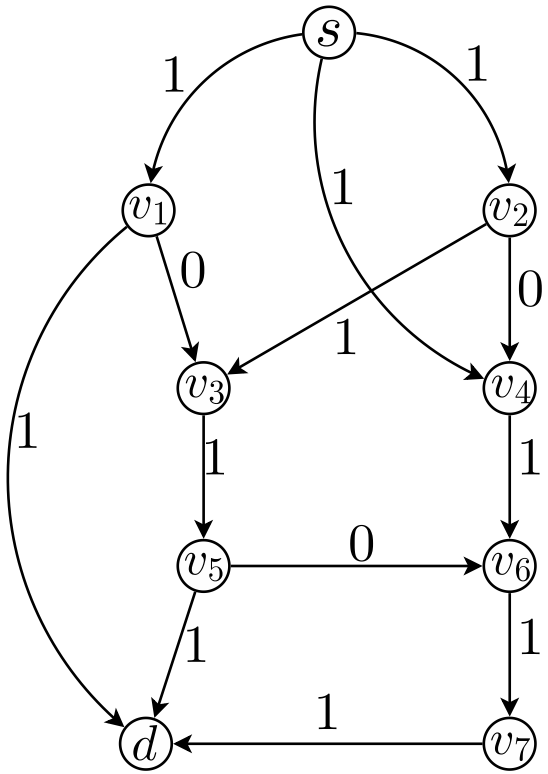
Cont'd


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
$(1,0,1)_m$ $(0,2,0)_q$
v_6
$(2,0,2)_m$ $(0,0,2)_q$
$(2,1,0)_m$ $(0,1,0)_q$
d
$(1,2,0)_m$ $(0,2,0)_q$
v_7

Cont'd

Step 4: Compare the inner products

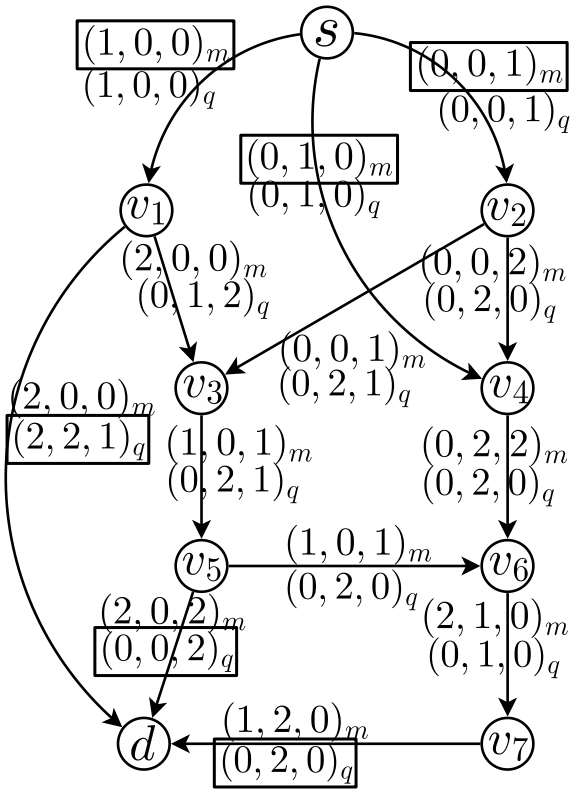


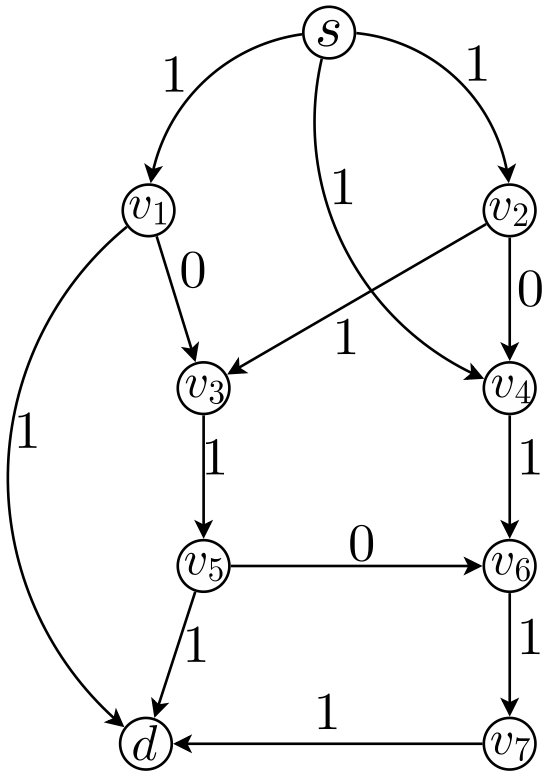


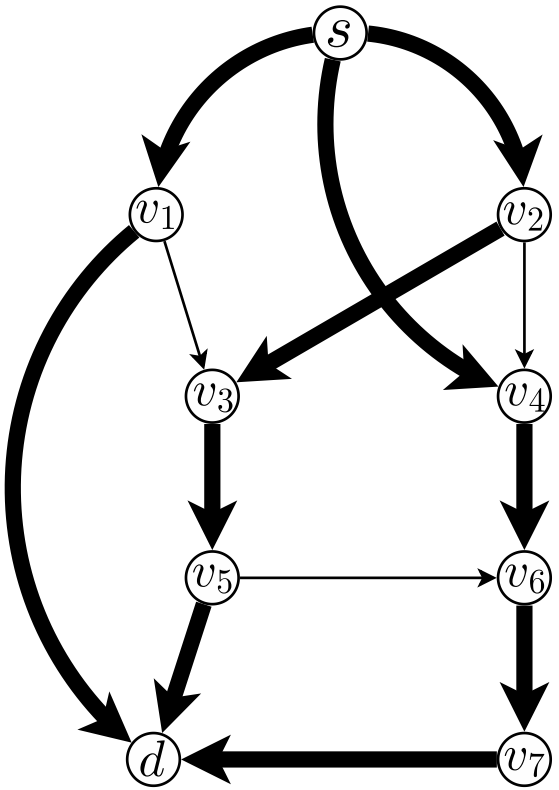
Cont'd

Step 4: Compare the inner products

Comparison to the true max flow found offline



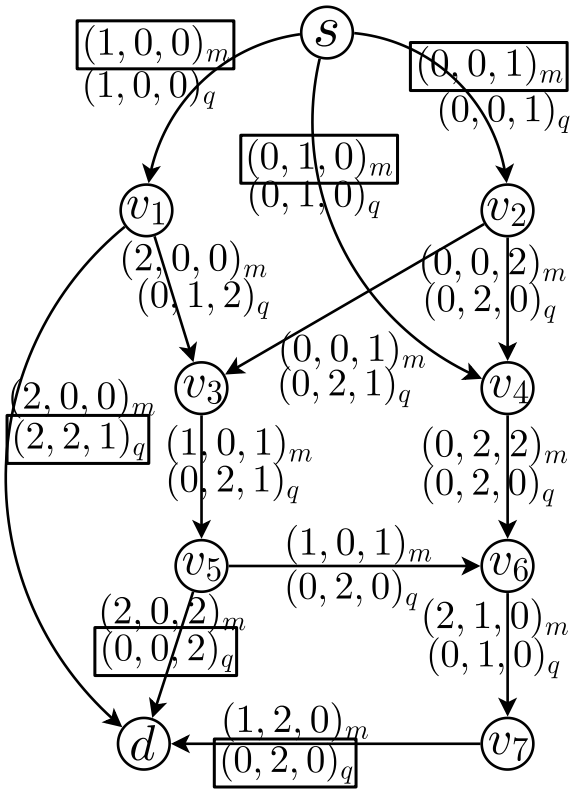


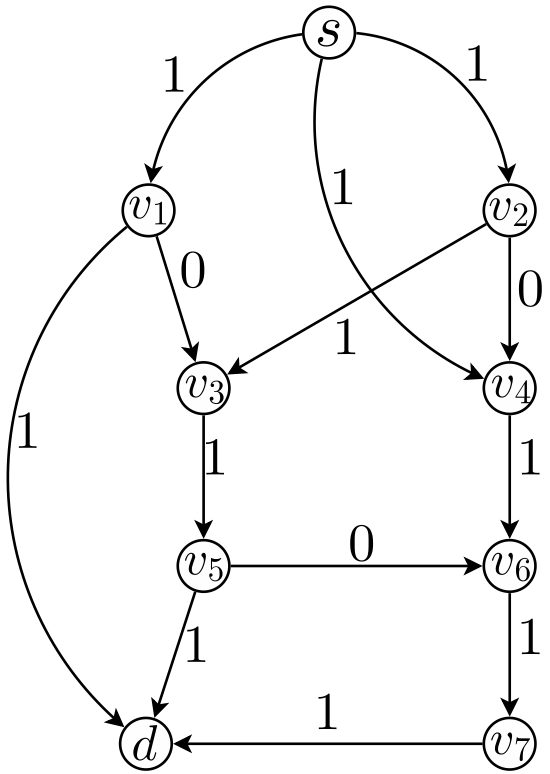


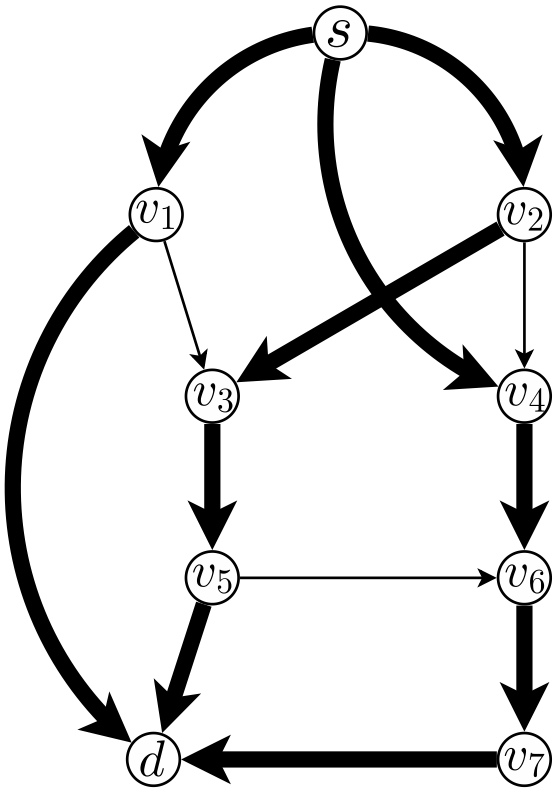
Cont'd

Step 4: Compare the inner products

Comparison to the true max flow found offline





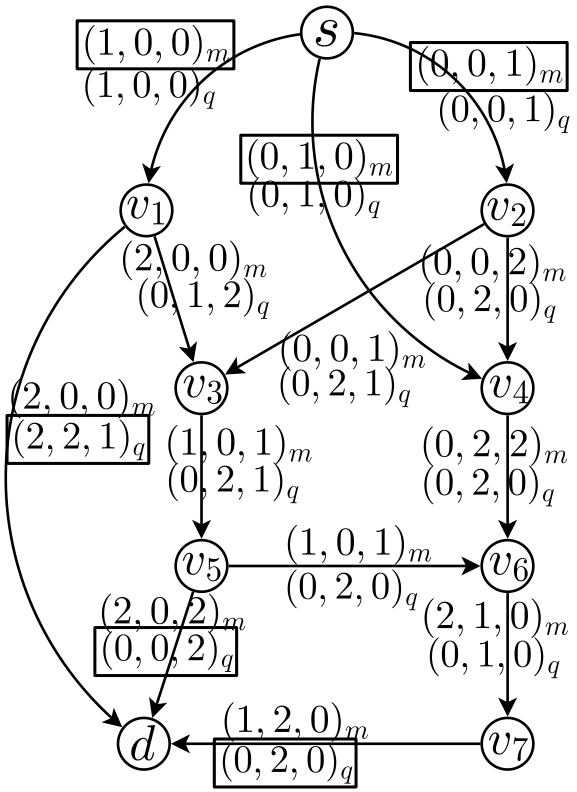


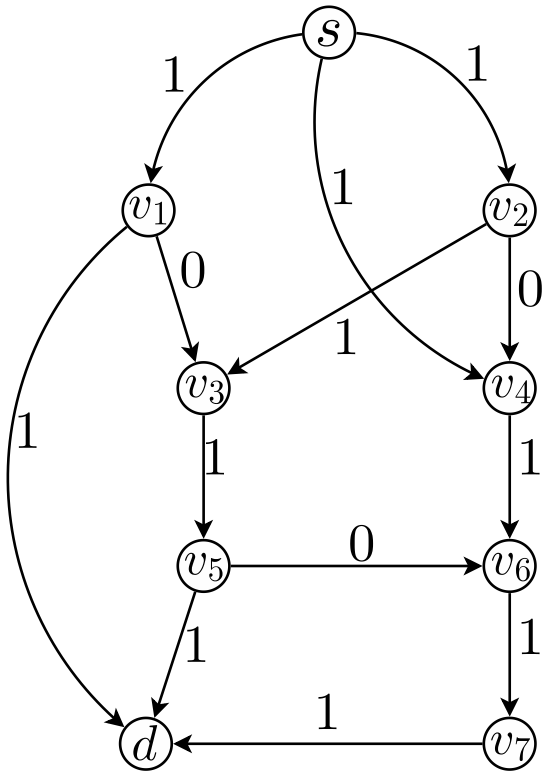
Voila!

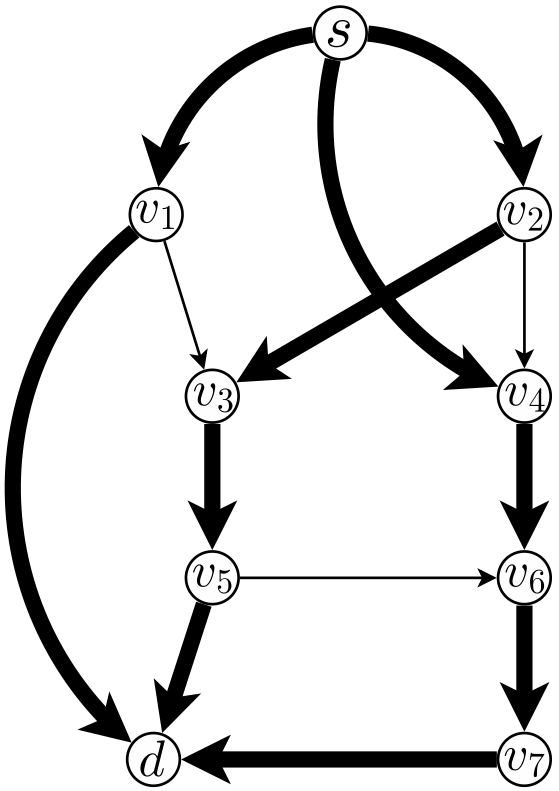
Cont'd

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.

Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***

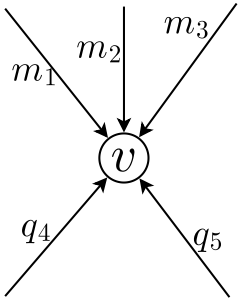
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

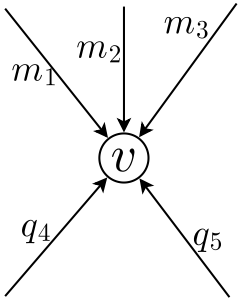
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ

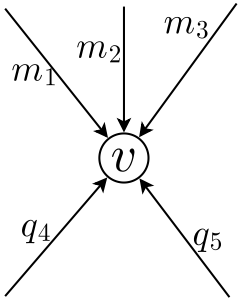
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ
$\Rightarrow$ the useful edges

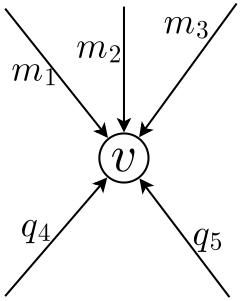
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ
$\Rightarrow$ the useful edges
$\Rightarrow$ The complement being redundant $\Rightarrow E_R(v)$

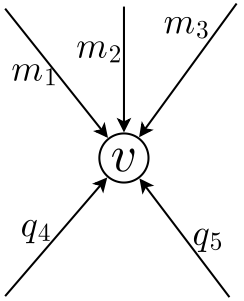
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ
$\Rightarrow$ the useful edges
$\Rightarrow$ The complement being redundant $\Rightarrow E_R(v)$

[New Results] $\heartsuit$ Arbitrarily search any v. $\heartsuit$

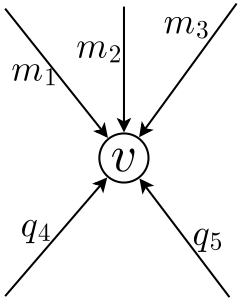
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

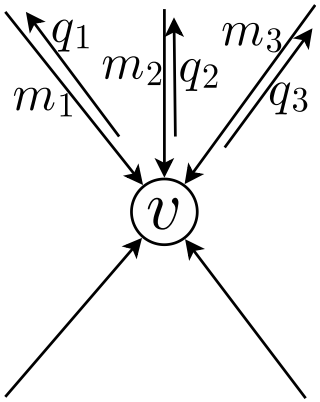
[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ
$\Rightarrow$ the useful edges
$\Rightarrow$ The complement being redundant $\Rightarrow E_R(v)$

[New Results] ♡ Arbitrarily search any v. ♡



For any $\Xi \subseteq \{e_1, e_2, e_3\}$ (say $\Xi = \{e_1, e_2\}$)

Let $$\Pi_\Xi = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}} \right)$$

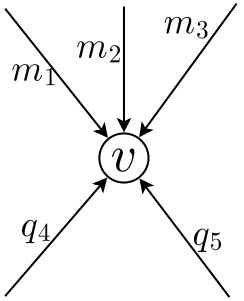
Two Coded-Feedback Algorithms

High-level description:

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback q_e
5: Find redundant edge set $E_R(v)$
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove $E_R(v)$.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Find redundant edge set $E_R(v)$:

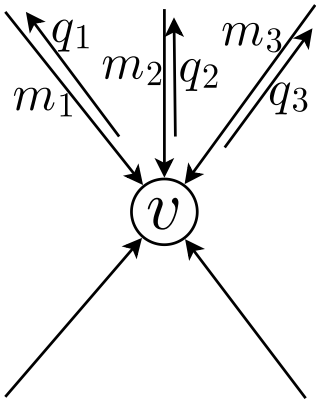
[ISIT08] *** Sequentially check from the down-stream to the upstream nodes. ***



$$\Phi = \begin{pmatrix} q_4 \\ q_5 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}}, m_3^{\mathrm{T}} \right)$$

Full rank submatrix of Φ
$\Rightarrow$ the useful edges
$\Rightarrow$ The complement being redundant $\Rightarrow E_R(v)$

[New Results] ♡ Arbitrarily search any v. ♡

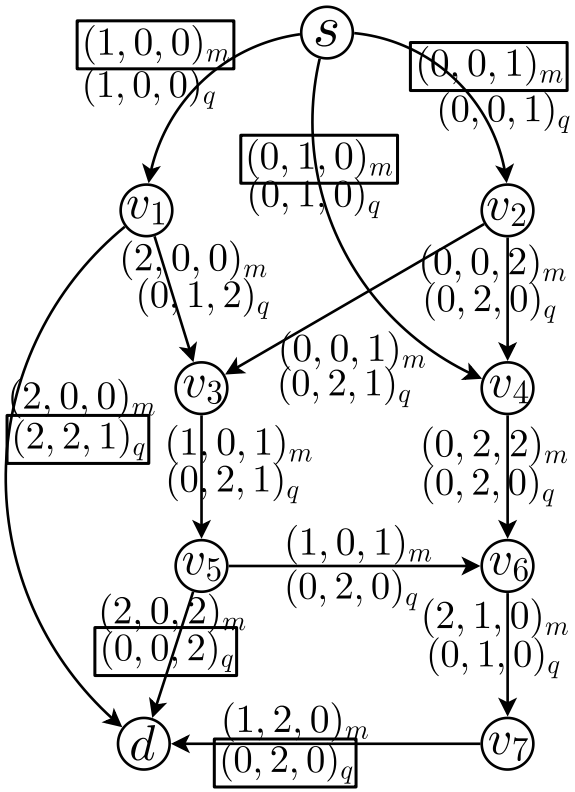


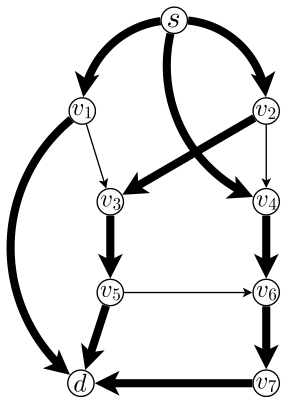
For any $\Xi \subseteq \{e_1, e_2, e_3\}$ (say $\Xi = \{e_1, e_2\}$)

Let $$\Pi_\Xi = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \left(m_1^{\mathrm{T}}, m_2^{\mathrm{T}} \right)$$

Check "Is $I_{|\Xi|} - \Pi_\Xi$ of full rank?"
$\Rightarrow$ If yes, then all edges in Ξ are redundant.

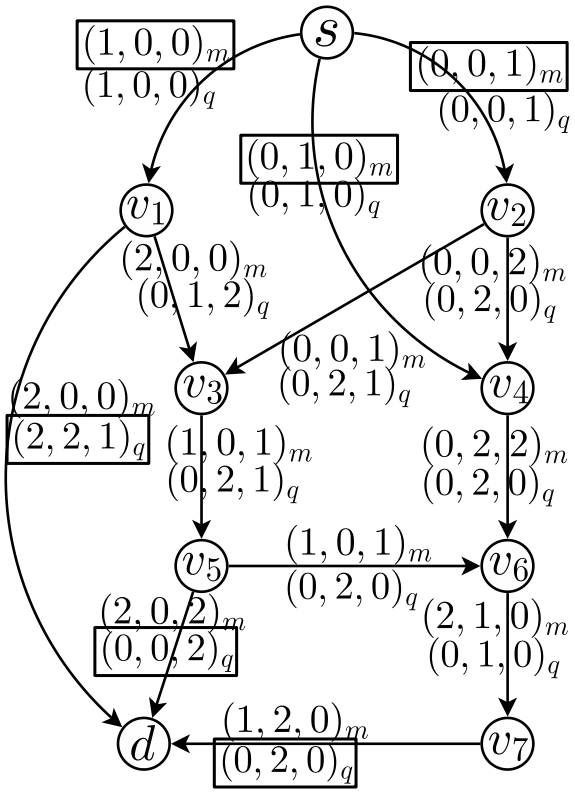
An Illustrative Example

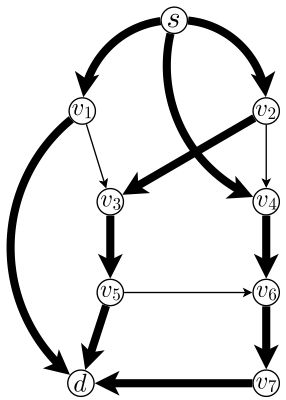




An Illustrative Example

Search v_3:

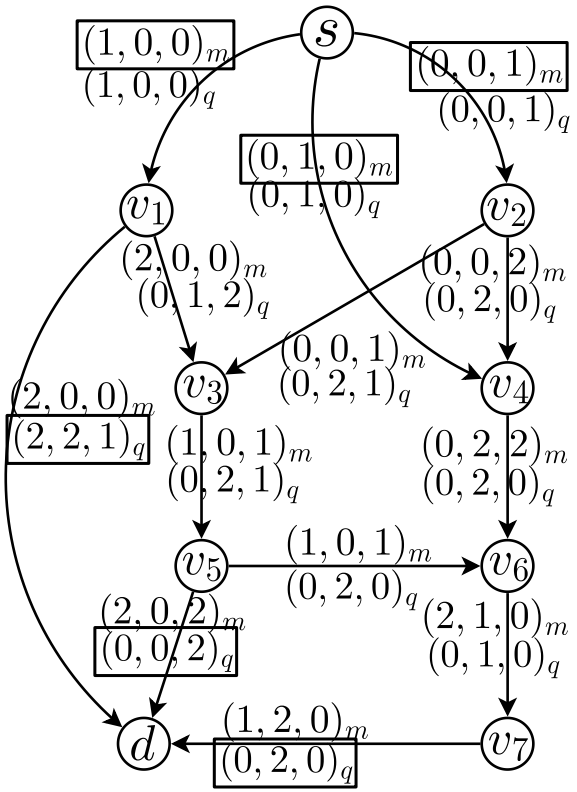


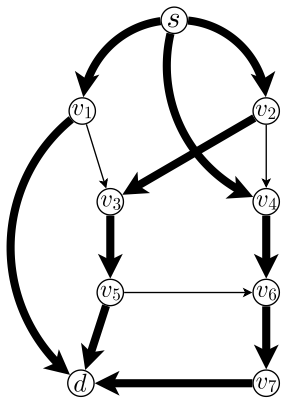


An Illustrative Example

Search v_3: we have three Ξ choices:

$$\Xi_1 = \{(v_1, v_3)\}$$





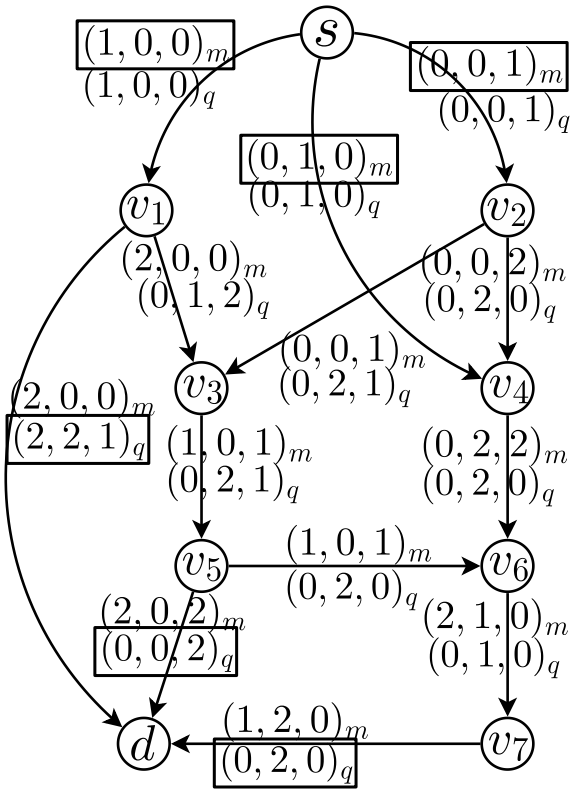
$$\Xi_2 = \{(v_2, v_3)\}$$

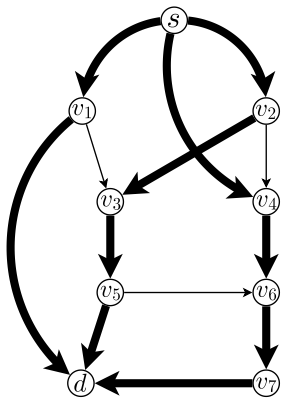
$$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$$

An Illustrative Example

Search v_3: we have three Ξ choices:

$$\Xi_1 = \{(v_1, v_3)\}$$
$$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$$

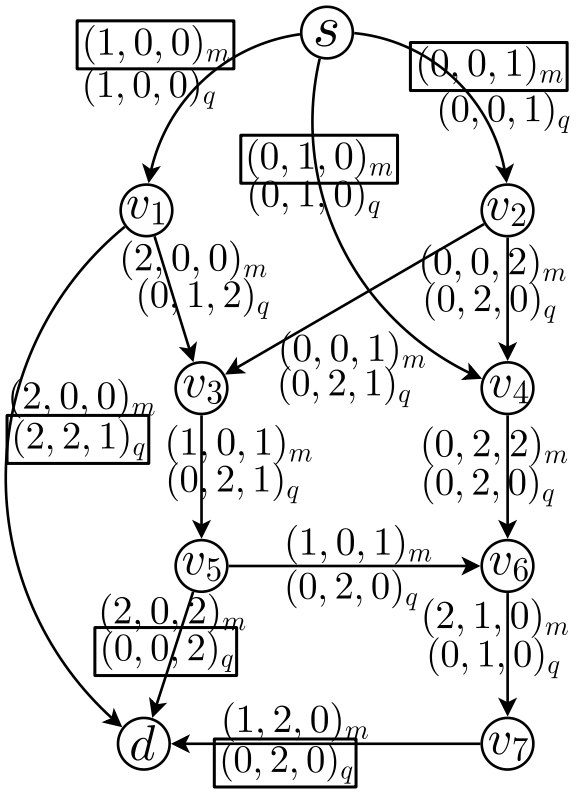




$$\Xi_2 = \{(v_2, v_3)\}$$

$$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$$

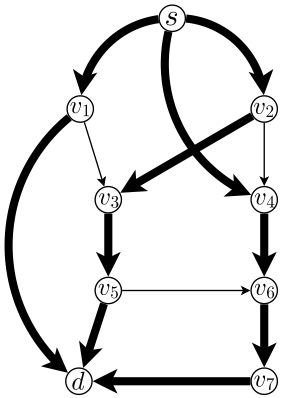
An Illustrative Example



Search v_3: we have three Ξ choices:

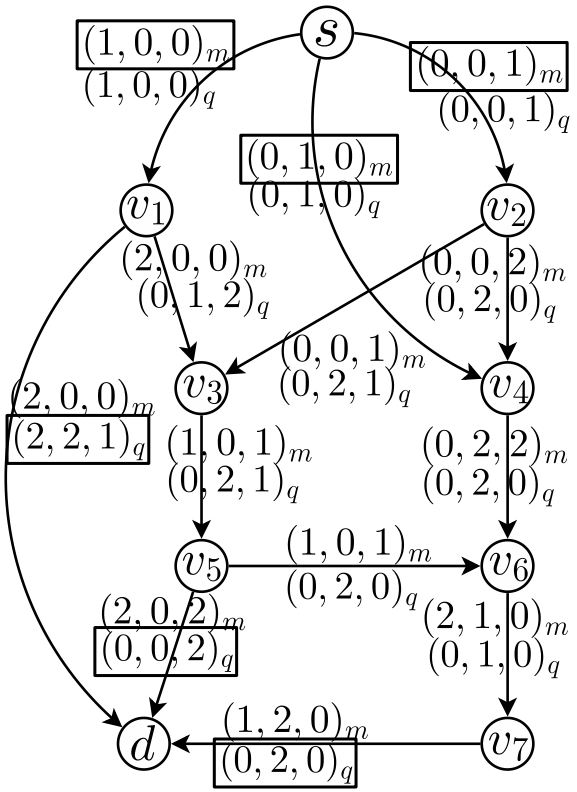
$\Xi_1 = \{(v_1, v_3)\}$
$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$
$\Leftrightarrow I - \Pi_1 = 1 \Rightarrow$ full rank
$\Rightarrow (v_1, v_3)$ is redundant.

$\Xi_2 = \{(v_2, v_3)\}$



$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$

An Illustrative Example

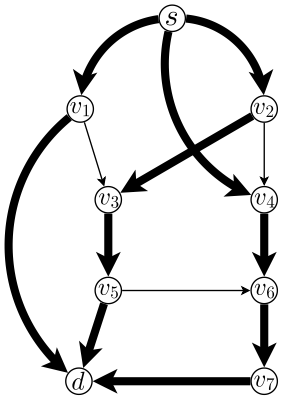


Search v_3: we have three Ξ choices:

$\Xi_1 = \{(v_1, v_3)\}$
$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$
$\Leftrightarrow I - \Pi_1 = 1 \Rightarrow$ full rank
$\Rightarrow (v_1, v_3)$ is redundant.

$\Xi_2 = \{(v_2, v_3)\}$
$\Pi_2 = [0,2,1][0,0,1]^{\mathrm{T}} = 1$

$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$



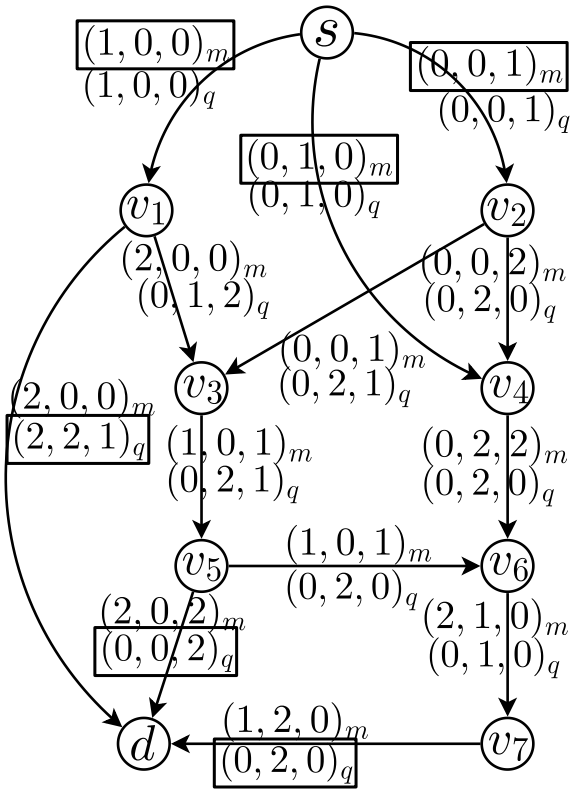
An Illustrative Example

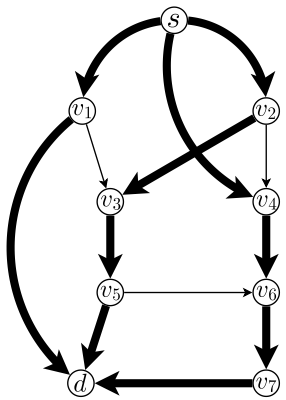
Search v_3: we have three Ξ choices:

$\Xi_1 = \{(v_1, v_3)\}$
$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$
$\Leftrightarrow I - \Pi_1 = 1 \Rightarrow$ full rank
$\Rightarrow (v_1, v_3)$ is redundant.

$\Xi_2 = \{(v_2, v_3)\}$
$\Pi_2 = [0,2,1][0,0,1]^{\mathrm{T}} = 1$
$\Leftrightarrow I - \Pi_2 = 0 \Rightarrow$ NOT of full rank
$\Rightarrow (v_2, v_3)$ is NOT redundant.

$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$

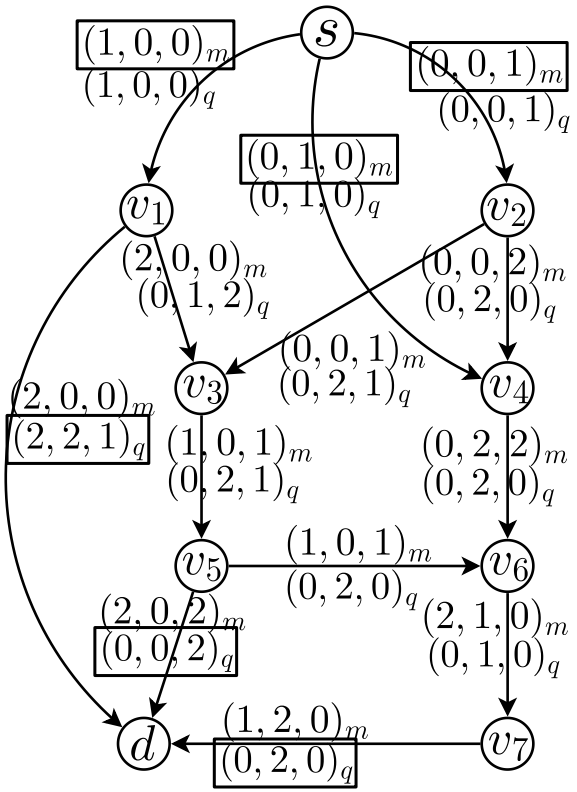


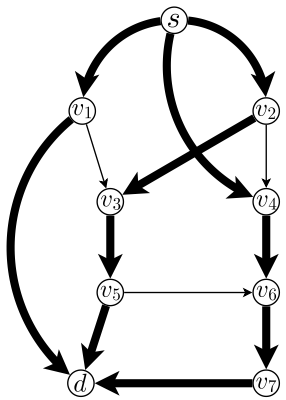


An Illustrative Example

Search v_3: we have three Ξ choices:

$\Xi_1 = \{(v_1, v_3)\}$
$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$
$\Leftrightarrow I - \Pi_1 = 1 \Rightarrow$ full rank
$\Rightarrow (v_1, v_3)$ is redundant.



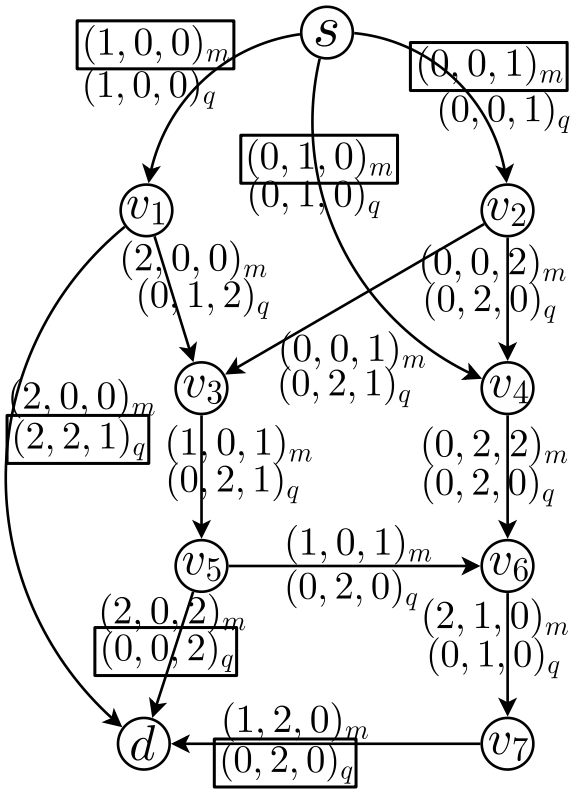


$\Xi_2 = \{(v_2, v_3)\}$
$\Pi_2 = [0,2,1][0,0,1]^{\mathrm{T}} = 1$
$\Leftrightarrow I - \Pi_2 = 0 \Rightarrow$ NOT of full rank
$\Rightarrow (v_2, v_3)$ is NOT redundant.

$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$

$$\Pi_3 = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix}$$

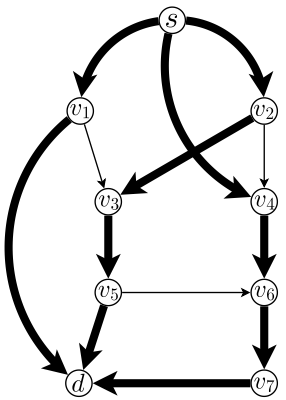
An Illustrative Example



Search v_3: we have three Ξ choices:

$\Xi_1 = \{(v_1, v_3)\}$
$\Pi_1 = [0,1,2][2,0,0]^{\mathrm{T}} = 0$
$\Leftrightarrow I - \Pi_1 = 1 \Rightarrow$ full rank
$\Rightarrow (v_1, v_3)$ is redundant.

$\Xi_2 = \{(v_2, v_3)\}$
$\Pi_2 = [0,2,1][0,0,1]^{\mathrm{T}} = 1$
$\Leftrightarrow I - \Pi_2 = 0 \Rightarrow$ NOT of full rank
$\Rightarrow (v_2, v_3)$ is NOT redundant.



$\Xi_3 = \{(v_1, v_3), (v_2, v_3)\}$

$$\Pi_3 = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix}$$

$$\Leftrightarrow I - \Pi_3 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \text{ in } \mathsf{GF}(3)$$

$\Rightarrow$ NOT of full rank
$\Rightarrow (v_1, v_3)$ and (v_2, v_3) are NOT jointly redundant.

Provable Correctness

- Assume m_e and q_e are row vectors.

Provable Correctness

- Assume m_e and q_e are row vectors.
- Using $\Gamma(v)^{\mathrm{T}}$, q_e carries the transfer matrix from d back to e.

Provable Correctness

- Assume m_e and q_e are row vectors.
- Using $\Gamma(v)^{\mathrm{T}}$, q_e carries the transfer matrix from d back to e.
- The impact of deletion: $q_e^{\mathrm{T}} \cdot \delta m_e = -q_e^{\mathrm{T}} m_e$.

Provable Correctness

- Assume m_e and q_e are row vectors.
- Using $\Gamma(v)^{\mathrm{T}}$, q_e carries the transfer matrix from d back to e.
- The impact of deletion: $q_e^{\mathrm{T}} \cdot \delta m_e = -q_e^{\mathrm{T}} m_e$.
- A strengthened Sylvester's determinant theorem:

$$\mathrm{Rank}(I_n) - \mathrm{Rank}(I_n - q_e^{\mathrm{T}} m_e) = \mathrm{Rank}(I_1) - \mathrm{Rank}(I_1 - q_e m_e^{\mathrm{T}})$$

Provable Correctness

- Assume m_e and q_e are row vectors.
- Using $\Gamma(v)^{\mathrm{T}}$, q_e carries the transfer matrix from d back to e.
- The impact of deletion: $q_e^{\mathrm{T}} \cdot \delta m_e = -q_e^{\mathrm{T}} m_e$.
- A strengthened Sylvester's determinant theorem:

$$\mathrm{Rank}(I_n) - \mathrm{Rank}(I_n - q_e^{\mathrm{T}} m_e) = \mathrm{Rank}(I_1) - \mathrm{Rank}(I_1 - q_e m_e^{\mathrm{T}})$$

- [**A sufficient condition**] Any $\Xi \subseteq \mathrm{In}(v)$ satisfying $I_{|\Xi|} - \Pi_\Xi$ being of full rank $\Longrightarrow$ redundant.
- [**A necessary condition**] Suppose $\mathrm{Rank}(d) = n$. Then any redundant $\Xi \subseteq \mathrm{In}(v) \Longrightarrow I_{|\Xi|} - \Pi_\Xi$ being of full rank.

Provable Correctness (Cont'd)

- **Convergence:** If each time a maximal $E_R(v)$ is identified & removed, the algorithm stops in $\mathcal{O}(|V|^2)$ seconds.
  - The distributed push-&-relabel algorithm converges in $\mathcal{O}(|V|^2)$.

Provable Correctness (Cont'd)

- **Convergence:** If each time a maximal $E_R(v)$ is identified & removed, 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 Correctness (Cont'd)

- **Convergence:** If each time a maximal $E_R(v)$ is identified & removed, 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 locally minimal.

Provable Correctness (Cont'd)

- **Convergence:** If each time a maximal $E_R(v)$ is identified & removed, 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 locally minimal.
- **Correctness with random network coding:** When $\mathsf{GF}(q)$ is large, the output is a max flow with close-to-1 probability.

Locally Min-Cost Multicast Codes

- ♡ We can search v in any arbitrary order ♡.

Locally Min-Cost Multicast Codes

- ♡ We can search v in any arbitrary order ♡.
- A single multicast session $(s, \{d_i\})$. Each edge e has cost c_e.

1: Choose $\Gamma(v)$
2: **loop**
3: Compute Forward Messages m_e
4: Compute Coded Feedback $q_e(i)$ for all d_i
5: Find redundant edge set $E_R(v) = \bigcap_i E_R(v,i)$.
6: **if** $E_R(v) \neq \emptyset$ **then**
7: Remove such $E_R(v)$ with the highest cost per edge.
8: **else**
9: **return** the remaining graph G
10: **end if**
11: **end loop**

Results

- Output a locally min-cost multicast code.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.
- Practical Advantages:
  - Monotonic traffic reduction.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.
- Practical Advantages:
  - Monotonic traffic reduction.
  - Limited exchange of control packets $q_e(i)$ enables straightforward distributed implementation.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.
- Practical Advantages:
  - Monotonic traffic reduction.
  - Limited exchange of control packets $q_e(i)$ enables straightforward distributed implementation.
  - q_e in the opposite direction enables easy piggyback. Use q_e to encode reverse data traffic, ex: video conferencing.

Results

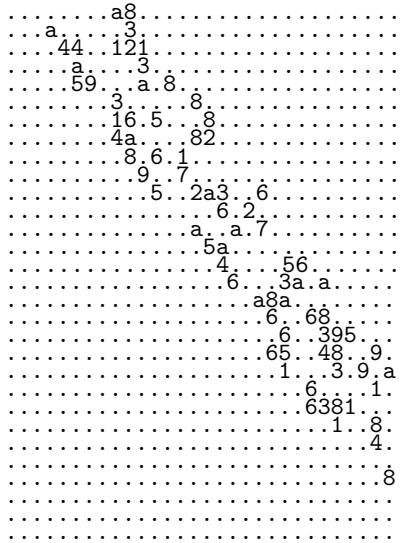
- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.
- Practical Advantages:
  - Monotonic traffic reduction.
  - Limited exchange of control packets $q_e(i)$ enables straightforward distributed implementation.
  - q_e in the opposite direction enables easy piggyback. Use q_e to encode reverse data traffic, ex: video conferencing.
  - No extra hardware requirement. Only linear operations.

Results

- Output a locally min-cost multicast code.
- Applicable even for small $\mathrm{GF}(q)$ — suboptimal sometimes.
- No interruption to forward coded traffic.
- Practical Advantages:
  - Monotonic traffic reduction.
  - Limited exchange of control packets $q_e(i)$ enables straightforward distributed implementation.
  - q_e in the opposite direction enables easy piggyback. Use q_e to encode reverse data traffic, ex: video conferencing.
  - No extra hardware requirement. Only linear operations.
  - Fully distributed implementation.

Simulations

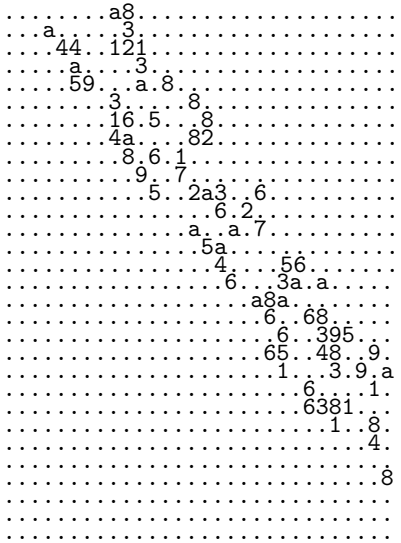
A 30-node network with incidence matrix



Minimize $\sum_e c_e$ where $c_e = \frac{1}{\text{multiplicity of } e}$.
I.e. the percentage of active time of a physical variable-rate link.

Simulations

A 30-node network with incidence matrix



Minimize $\sum_e c_e$ where $c_e = \frac{1}{\text{multiplicity of } e}$.
I.e. the percentage of active time of a physical variable-rate link.

$(s, \{d_i\})$	Optimal LP	Locally Min-Cost	Union of arb. max-flows
$(1, 30)$	10.0226	11.0028	11.7496
$(1, \{29, 30\})$	17.2036	18.8500	24.6210
$(1, \{28, 29, 30\})$	18.2036	19.8500	27.3294

Conclusion

- A coding-theoretic approach of constructing locally min-cost multicast network codes.
- Provably correct properties and fast convergence speed
- Maintains the delay minimality of network coding
- Many practical advantages as only coded feedback is used.