Beyond the Butterfly — A Graph-Theoretic Characterization for Network Coding with Two Simple Unicast Sessions

Chih-Chun Wang and Ness B. Shroff

School of Electrical & Computer Engineering
Purdue University
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Directed Acyclic Graph

$s_1 ightarrow t_2$

$s_2 ightarrow t_1$
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Routing solutions

$\iff$ Edge disjoint paths
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Routing solutions
$\iff$ Edge disjoint paths

The existence of a butterfly
$\implies$ Network coding solutions
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Routing solutions
\[\iff\] Edge disjoint paths

The existence of a butterfly
\[\implies\] Network coding solutions

Vice versa?

\[s_1\]
\[X_1\]
\[s_2\]
\[X_2\]

\[\text{Red}\]

\[X_1\]
\[X_2\]

\[\text{Black}\]

\[X_1 + X_2\]

\[\text{Blue}\]

\[t_1\]
\[X_1\]
\[t_2\]
\[X_2\]

Wang & Shroff – p. 2/17
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Routing solutions
$\iff$ Edge disjoint paths

The existence of a butterfly
$\implies$ Network coding solutions
Vice versa?
Two Simple Unicast Sessions

When can we send $X_1$ and $X_2$ simultaneously?

Routing solutions
$\iff$ Edge disjoint paths

The existence of a butterfly
$\implies$ Network coding solutions
Vice versa?

Q: Network coding solutions $\iff$ ???
It is an ongoing research work!
It is an ongoing research work!

Review current understanding on network coding with multiple unicast/multicast sessions.
Content

- It is an ongoing research work!
- Review current understanding on network coding with multiple unicast/multicast sessions.
- Network coding with two simple unicasts
  - The setting
  - The main results & corollaries
  - The proofs
Content

- It is an ongoing research work!
- Review current understanding on network coding with multiple unicast/multicast sessions.
- Network coding with two simple unicasts
  - The setting
  - The main results & corollaries
  - The proofs
- Applications on distributed rate control algorithms.
Directed Cycles [1]

\[
\sum_{i \text{ separated by } e} r_i \leq c(e)
\]

Directed Cycles [1]

\[ \sum_{i \text{ separated by } e} r_i \leq c(e) \]

The undirected Okamura-Seymour example [1]

Network coding = routing. \( r = \frac{3}{4} \)

**Special Graphs w. Known Cap.**

- Directed Cycles [1]
  \[ \sum_{i \text{ separated by } e} r_i \leq c(e) \]

- The undirected Okamura-Seymour example [1]
  - Network coding = routing. \( r = \frac{3}{4} \)

- Directed, acyclic, degree 2, three-layer networks [2]

Bounds for Multiple Sessions

General graphs, $K \geq 2$ (Unicast) Sessions.
Bounds for Multiple Sessions

General graphs, $K \geq 2$ (Unicast) Sessions.

Pure inform.-theoretic approaches: Fundamental regions: [Song et al. 03], [Yan et al. 07], entropy calculus [Jain et al. 06]
Bounds for Multiple Sessions

- General graphs, $K \geq 2$ (Unicast) Sessions.
- Pure inform.-theoretic approaches: Fundamental regions: [Song et al. 03], [Yan et al. 07], entropy calculus [Jain et al. 06]
- Capacity outer bounds (nec. condition):
  - The cut conditions + Inform.-theoretic arguments
Bounds for Multiple Sessions

- General graphs, \( K \geq 2 \) (Unicast) Sessions.
- Pure inform.-theoretic approaches: Fundamental regions: [Song et al. 03], [Yan et al. 07], entropy calculus [Jain et al. 06]
- Capacity outer bounds (nec. condition):
  - The cut conditions + Inform.-theoretic arguments
  - The network-sharing bound [2], the information dominance condition [1], and the edge-cut bounds [Kramer et al. 06].
Bounds for Multiple Sessions

- General graphs, \( K \geq 2 \) (Unicast) Sessions.

- Pure inform.-theoretic approaches: Fundamental regions: [Song et al. 03], [Yan et al. 07], entropy calculus [Jain et al. 06]

- Capacity outer bounds (nec. condition):
  - The cut conditions + Inform.-theoretic arguments
  - The network-sharing bound [2], the information dominance condition [1], and the edge-cut bounds [Kramer et al. 06].

- Capacity inner bound (suff. condition, achievability):
  - The modified flow conditions + Linear programming.
Bounds for Multiple Sessions

- General graphs, \( K \geq 2 \) (Unicast) Sessions.
- Pure inform.-theoretic approaches: Fundamental regions: [Song et al. 03], [Yan et al. 07], entropy calculus [Jain et al. 06]
- Capacity outer bounds (nec. condition):
  - The cut conditions + Inform.-theoretic arguments
  - The network-sharing bound [2], the information dominance condition [1], and the edge-cut bounds [Kramer et al. 06].
- Capacity inner bound (suff. condition, achievability):
  - The modified flow conditions + Linear programming.
  - Butterfly-based construction [Traskov et al. 06], pollution-treatment [Wu 06].
The Main Theorem

- Setting: General finite directed acyclic graphs, unit edge capacity, \((s_1, t_1) \& (s_2, t_2)\), two integer symbols \(X_1\) and \(X_2\).

- Number of Coinciding Paths of edge \(e\): \(\mathcal{P} = \{P_1, \cdots, P_k\}\), and \(\text{ncp}_{\mathcal{P}}(e) = |\{P \in \mathcal{P} : e \in P\}|\).
The Main Theorem

- Setting: General finite directed acyclic graphs, unit edge capacity, \((s_1, t_1) \& (s_2, t_2)\), two integer symbols \(X_1\) and \(X_2\).

Number of Coinciding Paths of edge \(e\): \(\mathcal{P} = \{P_1, \cdots, P_k\}\), and \(\text{ncp}_\mathcal{P}(e) = |\{P \in \mathcal{P} : e \in P\}|\).

**Theorem 1** Network coding \(\iff\) one of the following two holds.

1. \(\exists \mathcal{P} = \{P_{s_1,t_1}, P_{s_2,t_2}\},\) such that
   \[
   \max_{e \in E} \text{ncp}_\mathcal{P}(e) \leq 1.
   \]

2. \(\exists \mathcal{P} = \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\}\) and \(\mathcal{Q} = \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\}\) s.t.
   \[
   \max_{e \in E} \text{ncp}_\mathcal{P}(e) \leq 2 \text{ and } \max_{e \in E} \text{ncp}_\mathcal{Q}(e) \leq 2.
   \]
The Main Theorem

- Setting: General finite directed acyclic graphs, unit edge capacity, \((s_1, t_1) \& (s_2, t_2)\), two integer symbols \(X_1\) and \(X_2\).

- Number of Coinciding Paths of edge \(e\): \(\mathcal{P} = \{P_1, \ldots, P_k\}\), and \(\text{ncp}_\mathcal{P}(e) = |\{P \in \mathcal{P} : e \in P\}|\).

Theorem 1  \textit{Network coding} \iff \textit{one of the following two holds.}

1. \(\exists \mathcal{P} = \{P_{s_1,t_1}, P_{s_2,t_2}\}, \text{ such that } \max_{e \in E} \text{ncp}_\mathcal{P}(e) \leq 1\).

2. \(\exists \mathcal{P} = \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \text{ and } \mathcal{Q} = \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \text{ s.t. } \max_{e \in E} \text{ncp}_\mathcal{P}(e) \leq 2 \text{ and } \max_{e \in E} \text{ncp}_\mathcal{Q}(e) \leq 2\).

Feasible Example: The Butterfly

\[ Q = \{ Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2} \} \quad \mathcal{P} = \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \]
Feasible Example 2: The Grail

\[ Q = \{ Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2} \} \]

\[ P = \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \]
Infeasible Examples

\[ Q = \{ Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2} \} \]

\[ Q = \{ Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2} \} \]
Intuition & Corollaries

Edge disjointness $\rightarrow$ controlled overlap
Intuition & Corollaries

- Edge disjointness $\rightarrow$ controlled overlap

- The selection of $\mathcal{P}$ and $\mathcal{Q}$ are independent:
  Pairwise intersession network coding $\iff$ two half butterflies
Intuition & Corollaries

- Edge disjointness $\implies$ controlled overlap
- The selection of $P$ and $Q$ are independent:
  Pairwise intersession network coding $\iff$ two half butterflies

Corollaries for two simple unicast sessions w. directed acyclic graphs:
- Deciding the existence of a network coding solution is a polynomial-time problem.
Intuition & Corollaries

- Edge disjointness $\implies$ controlled overlap
- The selection of $P$ and $Q$ are independent:
  Pairwise intersession network coding $\iff$ two half butterflies

Corollaries for two simple unicast sessions w. directed acyclic graphs:

- Deciding the existence of a network coding solution is a polynomial-time problem.
  Proof: By the subgraph homeomorphism algorithm for directed acyclic graphs [Fortune et al. 79]
Intuition & Corollaries

- Edge disjointness $\implies$ controlled overlap
- The selection of $P$ and $Q$ are independent:
  Pairwise intersession network coding $\iff$ two half butterflies

Corollaries for two simple unicast sessions w. directed acyclic graphs:

- Deciding the existence of a network coding solution is a polynomial-time problem.
  Proof: By the subgraph homeomorphism algorithm for directed acyclic graphs [Fortune et al. 79]
- A network coding solution needs to use at most six paths.
Intuition & Corollaries

- Edge disjointness \(\rightarrow\) controlled overlap
- The selection of \(P\) and \(Q\) are independent:

  \[
  \text{Pairwise intersession network coding} \iff \text{two half butterflies}
  \]

Corollaries for two simple unicast sessions w. directed acyclic graphs:

- Deciding the existence of a network coding solution is a **polynomial-time** problem.
  
  Proof: By the subgraph homeomorphism algorithm for directed acyclic graphs [Fortune et al. 79]

- A network coding solution needs to use **at most six paths**.

- **Linear network coding** is sufficient, a byproduct of the proof.
A proof that doesn’t work

A first try on proving the necessity that *does not work*:

A network coding solution exists but not a routing one.
A proof that doesn’t work

A first try on proving the necessity that does not work:

A network coding solution exists but not a routing one.

\[ \sum_{i=1}^{n} X_i \]

\[ \sum_{j=1}^{m} X_j \]

\[ X_1 + X_2 \]

One intermediate node \( m_i \) for each \( t_i \) that is doing “decoding" to recover \( X_i \). Those intermediate nodes know both \( X_1 \) and \( X_2 \).
A proof that doesn’t work

A first try on proving the necessity that does not work:

A network coding solution exists but not a routing one.

One intermediate node $m_i$ for each $t_i$ that is doing "decoding" to recover $X_i$. Those intermediate nodes know both $X_1$ and $X_2$.

Use the result in [Fragouli et al. 06] that $(s_1, m_1)$ and $(s_2, m_2)$ must form two EDPs or a butterfly.

···
A proof that doesn’t work

A first try on proving the necessity that does not work:

A network coding solution exists but not a routing one.

One intermediate node $m_i$ for each $t_i$ that is doing “decoding" to recover $X_i$. Those intermediate nodes know both $X_1$ and $X_2$.

Use the result in [Fragouli et al. 06] that $(s_1, m_1)$ and $(s_2, m_2)$ must form two EDPs or a butterfly.

\[ \cdots \]
A proof of the necessity

Assume a linear network coding solution exists. ⇒ Construct \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \).
A proof of the necessity

Assume a linear network coding solution exists.
⇒ Construct \( \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \).

Construct \( P_{s_2,t_2} \) along non-zero \( X_2 \) messages.
A proof of the necessity

Assume a linear network coding solution exists. 
⇒ Construct \( \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \).

Construct \( P_{s_2,t_2} \) along non-zero \( X_2 \) messages.

\[ \Downarrow \]

Arbitrarily pick \( P_{s_1,t_1}^{(0)} \) and \( P_{s_2,t_1}^{(0)} \).
A proof of the necessity

Assume a linear network coding solution exists.
⇒ Construct \( \{ P_{s1,t1}, P_{s2,t2}, P_{s2,t1} \} \).

Construct \( P_{s2,t2} \) along non-zero \( X_2 \) messages.

\[ \Downarrow \]

Arbitrarily pick \( P_{s1,t1}^{(0)} \) and \( P_{s2,t1}^{(0)} \).

\[ \Downarrow \]

\( \forall l \), if \( \{ P_{s1,t1}^{(l)}, P_{s2,t1}^{(l)}, P_{s2,t2} \} \) is not good, then construct \( P_{s1,t1}^{(l+1)} \) and \( P_{s2,t1}^{(l+1)} \) from \( P_{s1,t1}^{(l)} \) and \( P_{s2,t1}^{(l)} \).
A proof of the necessity

Assume a linear network coding solution exists.
⇒ Construct \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \).

\[
\begin{align*}
\text{Construct } P_{s_2,t_2} \text{ along non-zero } X_2 \text{ messages.} \\
\downarrow \\
\text{Arbitrarily pick } P_{s_1,t_1}^{(0)} \text{ and } P_{s_2,t_1}^{(0)}. \\
\downarrow \\
\forall l, \text{ if } \{P_{s_1,t_1}^{(l)}, P_{s_2,t_1}^{(l)}, P_{s_2,t_2}\} \text{ is not good, then construct } P_{s_1,t_1}^{(l+1)} \text{ and } P_{s_2,t_1}^{(l+1)} \text{ from } P_{s_1,t_1}^{(l)} \text{ and } P_{s_2,t_1}^{(l)}.
\end{align*}
\]
A proof of the necessity

Assume a linear network coding solution exists.
⇒ Construct \( \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \).

Construct \( P_{s_2,t_2} \) along non-zero \( X_2 \) messages.

⇓

Arbitrarily pick \( P^{(0)}_{s_1,t_1} \) and \( P^{(0)}_{s_2,t_1} \).

⇓

\( \forall l \), if \( \{ P^{(l)}_{s_1,t_1}, P^{(l)}_{s_2,t_1}, P_{s_2,t_2} \} \) is not good, then construct \( P^{(l+1)}_{s_1,t_1} \) and \( P^{(l+1)}_{s_2,t_1} \) from \( P^{(l)}_{s_1,t_1} \) and \( P^{(l)}_{s_2,t_1} \).
A proof of the necessity

Assume a linear network coding solution exists.
⇒ Construct \( \{ P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1} \} \).

Construct \( P_{s_2,t_2} \) along non-zero \( X_2 \) messages.

\[ \Downarrow \]

Arbitrarily pick \( P^{(0)}_{s_1,t_1} \) and \( P^{(0)}_{s_2,t_1} \).

\[ \Downarrow \]

\( \forall l, \text{ if } \{ P^{(l)}_{s_1,t_1}, P^{(l)}_{s_2,t_1}, P_{s_2,t_2} \} \text{ is not good, then construct } P^{(l+1)}_{s_1,t_1} \text{ and } P^{(l+1)}_{s_2,t_1} \text{ from } P^{(l)}_{s_1,t_1} \text{ and } P^{(l)}_{s_2,t_1} \).

\[ \Downarrow \]

\( l \leftarrow l + 1. \text{ By the finiteness of } G, \text{ the iteration will halt.} \)
How about non-linear network coding? Assume a linear network coding solution exists.

$\Rightarrow$ Construct $\{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\}$.

Construct $P_{s_2,t_2}$ along non-zero $X_2$ messages.

$\Downarrow$

Arbitrarily pick $P_{s_1,t_1}^{(0)}$ and $P_{s_2,t_1}^{(0)}$.

$\Downarrow$

$\forall l$, if $\{P_{s_1,t_1}^{(l)}, P_{s_2,t_1}^{(l)}, P_{s_2,t_2}\}$ is not good, then construct $P_{s_1,t_1}^{(l+1)}$ and $P_{s_2,t_1}^{(l+1)}$ from $P_{s_1,t_1}^{(l)}$ and $P_{s_2,t_1}^{(l)}$.

$\Downarrow$

$l \leftarrow l + 1$. By the finiteness of $G$, the iteration will halt.
A proof of the necessity

How about non-linear network coding? Assume a linear network coding solution exists. ⇒ Construct \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \).

Construct \( P_{s_2,t_2} \) along \( I(f_e(X_1, X_2), X_2|X_1) > 0 \)

\[ \begin{align*}
\downarrow \\
\text{Arbitrarily pick } P^{(0)}_{s_1,t_1} \text{ and } P^{(0)}_{s_2,t_1}. \\
\downarrow \\
\forall l, \text{ if } \{P^{(l)}_{s_1,t_1}, P^{(l)}_{s_2,t_1}, P_{s_2,t_2}\} \text{ is not good, then construct } P^{(l+1)}_{s_1,t_1} \text{ and } P^{(l+1)}_{s_2,t_1} \text{ from } P^{(l)}_{s_1,t_1} \text{ and } P^{(l)}_{s_2,t_1}. \\
\downarrow \\
l \leftarrow l + 1. \text{ By the finiteness of } G, \text{ the iteration will halt.}
\end{align*} \]
A proof of the sufficiency

Assume \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} and \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\}

⇒ Construct a network coding solution.

A two-staged, add-up-&-reset construction

1. The random add-up stage:
   - Maximizing the span of any set of messages without “erasing” its origins.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)
\[ \Rightarrow \] Construct a network coding solution.

A two-staged, add-up-&-reset construction

1. The random add-up stage:
   **Random add-up:**
   (a) If all \( M_{\text{IN}} \) messages are identical, then \( M_\epsilon = M_{\text{IN}} \).
   (b) Otherwise \( M_\epsilon = a_1 M_1 + \cdots + a_m M_m \) for \( a_i > 0 \), such that \( M_\epsilon \) is linearly indep. of any other messages \( M_{\epsilon'} \) for those \( e' \) not in the downstream of \( e \).
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \) 

\[ \Rightarrow \] Construct a network coding solution.

A two-staged, add-up-&-reset construction

Random add-up:

1. The random add-up stage:

   Maximizing the span of any set of messages without "erasing" its origins.

   (a) If all \( M_{IN} \) messages are identical, then \( M_e = M_{IN} \).

   (b) Otherwise \( M_e = a_1 M_1 + \cdots + a_m M_m \) for \( a_i > 0 \), such that \( M_e \) is linearly indep. of any other messages \( M_{e'} \) for those \( e' \) not in the downstream of \( e \).
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2} \} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2} \} \)

\[ \Rightarrow \] Construct a network coding solution.

A two-staged, add-up-&-reset construction

1. The random add-up stage:
   - Maximizing the span of any set of messages without “erasing" its origins.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

\[ \Rightarrow \] Construct a network coding solution.

A two-staged, **add-up-\&-reset** construction

1. The **random add-up** stage:
   - Maximizing the span of any set of messages without "erasing" its origins.

2. The **reset** stage:
   - Perform "reset-to-\(X_1\)" & "reset-to-\(X_2\)"
     sequentially in the topological order & in a need basis.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

⇒ Construct a network coding solution.

A two-staged, add-up-\&-reset construction

1. The random add-up stage:
   - Maximizing the span of any set of messages without “erasing” its origins.

2. The reset stage:
   - Perform “reset-to-\(X_1\)” \& “reset-to-\(X_2\)” sequentially in the topological order \& in a need basis.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

\[\Rightarrow\] Construct a network coding solution.

A two-staged, \textit{add-up-\&-reset} construction

1. The \textit{random add-up} stage:
   - Maximizing the span of any set of messages without “erasing" its origins.

2. The \textit{reset} stage:
   - Perform “reset-to-\(X_1\)" \& “reset-to-\(X_2\)"
     sequentially in the topological order \& in a need basis.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

\( \Rightarrow \) Construct a network coding solution.

A two-staged, add-up-\&-reset construction

1. The random add-up stage:
   - Maximizing the span of any set of messages without “erasing” its origins.

2. The reset stage:
   - Perform “reset-to-\(X_1\)” \& “reset-to-\(X_2\)” sequentially in the topological order \& in a need basis.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

\[ \Rightarrow \text{Construct a network coding solution.} \]

A two-staged, add-up-\&-reset construction

1. The random add-up stage:
   - Maximizing the span of any set of messages without “erasing” its origins.

2. The reset stage:
   - Perform “reset-to-\( X_1 \)” & “reset-to-\( X_2 \)” sequentially in the topological order \& in a need basis.
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

\[ \Rightarrow \text{Construct a network coding solution.} \]

A two-staged, \textit{add-up-\&-reset} construction

1. The \textbf{random add-up} stage:
   \begin{itemize}
   \item Maximizing the span of any set of messages without “erasing” its origins.
   \end{itemize}

2. The \textbf{reset} stage:
   \begin{itemize}
   \item Perform “reset-to-}X_1\text{" \& “reset-to-}X_2\text{" sequentially in the topological order \& in a need basis.
   \item Controlled overlap condition \[ \Rightarrow \text{the feasibility.} \]
   \end{itemize}
A proof of the sufficiency

Assume \( \{P_{s_1,t_1}, P_{s_2,t_2}, P_{s_2,t_1}\} \) and \( \{Q_{s_1,t_1}, Q_{s_2,t_2}, Q_{s_1,t_2}\} \)

⇒ Construct a network coding solution.

A two-staged, **add-up-\&-reset** construction

\[
\text{ncp}\{P_{s_1,t_1}, P_{s_2,t_1}, Q_{s_1,t_1}\}(e) = 3,
\]

\[
\text{ncp}\{P_{s_1,t_1}, P_{s_2,t_1}, P_{s_2,t_2}\}(e) = 2,
\]

⇒ \( e \notin P_{s_2,t_2} \)

⇒ Messages along \( P_{s_2,t_2} \) are not affected.

\[
\text{ncp}\{P_{s_1,t_1}, P_{s_2,t_1}, P_{s_2,t_2}\}(e) = 2,
\]

\[
\Rightarrow e \notin P_{s_2,t_2}
\]

⇒ Messages along \( P_{s_2,t_2} \) are not affected.

Controlled overlap condition ⇒ the feasibility.

Wang & Shroff – p. 13/17
Improved Capacity Region

- Existing results: Search for butterfly coding opportunities via linear/integer programming. [Traskov et al. 06]
Existing results: Search for butterfly coding opportunities via linear/integer programming. [Traskov et al. 06]
Improved Capacity Region

- Existing results: Search for butterfly coding opportunities via linear/integer programming. [Traskov et al. 06]
- Now, we should search for the grail structure as well.
Improved Capacity Region

- Existing results: Search for butterfly coding opportunities via linear/integer programming. [Traskov et al. 06]
- Now, we should search for the grail structure as well.
Improved Capacity Region

- Existing results: Search for butterfly coding opportunities via linear/integer programming. [Traskov et al. 06]

- Now, we should search for the grail structure as well.

- The capacity region is strictly improved.
Pattern-based construction vs. path-based construction
Pattern-based construction vs. path-based construction

- \((s_1, t_1)\): 1 path, \((s_2, t_2)\): 3 paths, \((s_3, t_3)\): 4 paths,
- \((s_1, t_2)\): 3 paths, \((s_2, t_3)\): 5 paths, \((s_1, t_3)\): 2 paths,
- \((s_2, t_1)\): 3 paths, \((s_3, t_2)\): 1 path, \((s_3, t_1)\): 3 paths.
Pattern-based construction vs. path-based construction

- \((s_1, t_1)\): 1 path,
- \((s_2, t_2)\): 3 paths,
- \((s_3, t_3)\): 4 paths,
- \((s_1, t_2)\): 3 paths,
- \((s_2, t_3)\): 5 paths,
- \((s_1, t_3)\): 2 paths,
- \((s_2, t_1)\): 3 paths,
- \((s_3, t_2)\): 1 path,
- \((s_3, t_1)\): 3 paths.

Bottleneck identification for all path combinations.
Capacity Region (Cont’d)

- **Pattern-based** construction vs. **path-based** construction

  \( (s_1, t_1) \): 1 path, 
  \( (s_2, t_2) \): 3 paths, 
  \( (s_3, t_3) \): 4 paths, 
  \( (s_1, t_2) \): 3 paths, 
  \( (s_2, t_3) \): 5 paths, 
  \( (s_1, t_3) \): 2 paths

- Bottleneck identification for all path combinations.

- **Distributed path-based** network optimization with arbitrary utility function. [Submitted to Infocom 08]
Other implications

- The network-sharing bound in [Yan et al. 06]
  - Cut-based outer bound for $K$-pair unicasts.
  - Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
  - Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
Other implications

- The network-sharing bound in [Yan et al. 06]
  - Cut-based outer bound for $K$-pair unicasts.
  - Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
  - Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
The network-sharing bound in [Yan et al. 06]

- Cut-based outer bound for $K$-pair unicasts.
- Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
- Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
Other implications

- The network-sharing bound in [Yan et al. 06]
  - Cut-based outer bound for $K$-pair unicasts.
  - Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
  - Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
Other implications

- The network-sharing bound in [Yan et al. 06]
  - Cut-based outer bound for $K$-pair unicasts.
  - Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
  - Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
The network-sharing bound in [Yan et al. 06]

- Cut-based outer bound for $K$-pair unicasts.
- Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
- Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
Other implications

- The network-sharing bound in [Yan et al. 06]
  - Cut-based outer bound for $K$-pair unicasts.
  - Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
  - Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$. 
The network-sharing bound in [Yan et al. 06]

- Cut-based outer bound for $K$-pair unicasts.
- Relabel the subscripts of $(s_i, t_i)$ according to an arbitrary permutation.
- Exclude the edges of which the upstream $s_i$ have indices strictly smaller than the downstream $t_j$.

**Corollary 1** The network-sharing bound is tight. Namely, if the network-sharing bound is $\geq 2$ for all permutation and for all cuts, then network coding is feasible.
Network coding w. two simple unicastst 
$\iff$ Path selections $\mathcal{P}$ and $\mathcal{Q}$ w. controlled overlap
Discussion

Network coding w. two simple unicasts
\[\iff\] Path selections $\mathcal{P}$ and $\mathcal{Q}$ w. controlled overlap

- A flow-based characterization for general directed acyclic graphs.
Network coding w. two simple unicasts \iff Path selections $\mathcal{P}$ and $\mathcal{Q}$ w. controlled overlap

- A flow-based characterization for general directed acyclic graphs.
- Is it the right form?
- Probably ...
Discussion

Network coding w. two simple unicasts
\iffalse\right\iffalse Path selections $\mathcal{P}$ and $\mathcal{Q}$ w. controlled overlap

- A \textit{flow-based} characterization for general directed acyclic graphs.

\textbf{Is it the right form?}

- \textbf{Probably ...}
  - Applicable to general directed acyclic graphs,
Discussion

Network coding w. two simple unicasts
⇔ Path selections $\mathcal{P}$ and $\mathcal{Q}$ w. controlled overlap

- A flow-based characterization for general directed acyclic graphs.
- **Is it the right form?**
- Probably ...
  - Applicable to general directed acyclic graphs,
  - Of a form similar to the min-cut max-flow theorem,
Discussion

Network coding w. two simple unicasts
\[ \iff \] Path selections \( P \) and \( Q \) w. controlled overlap

- A flow-based characterization for general directed acyclic graphs.
- Is it the right form?
- Probably ...

  - Applicable to general directed acyclic graphs,
  - Of a form similar to the min-cut max-flow theorem,
  - It can be generalized to two simple multicast sessions

[submitted to Allerton 07]
Send \( X_1 \) and \( X_2 \) along \( (s_1, \{t_{1,i}\}) \) and \( (s_2, \{t_{2,j}\}) \) where \( \{t_{1,i}\} \cap \{t_{2,j}\} \neq \emptyset \).