Inhalt I

1. Introduction
   - Situation
   - The Model
   - Lower Bound
   - Proof
   - Application

2. Consistent Hashing
   - Introduction
   - Statements

3. Chord Network
   - Introduction
   - Statements

4. Randomized Oblivious Routing
   - Introduction

5. Path Selection
   - Path Selection on the Hypercube: Valiant’s Trick
   - Analyzing a random routing problem

6. Packet Scheduling for the Hypercube
   - The algorithm
   - Proof

7. Packet scheduling for general networks
   - The algorithm
   - Proof
Current Situation:

- Every permutation could be routed on a permutation network.
- The number of steps is proportional to the diameter.
- The algorithm was centralized and required global knowledge about sources and destinations of all packets.

We now want to devise local-control algorithms. Each node decides on the next step based on some local information.
Current Situation:

- every permutation could be routed
Current Situation:
- every permutation could be routed
- on permutation network and mesches
Current Situation:

- every permutation could be routed
- on permutation network and meshes
- number of steps proportional to the diameter
Current Situation:

- every permutation could be routed
- on permutation network and meshes
- number of steps proportional to the diameter
- algorithm was centralized
Current Situation:

- every permutation could be routed
- on permutation network and meshes
- number of steps proportional to the diameter
- algorithm was centralized
- needed global knowledge about the sources and destinations of all packets
Current Situation:

- every permutation could be routed
- on permutation network and mesches
- number of steps proportional to the diameter
- algorithm was centralized
- needed global knowledge about the sources and destinations of all packets

We now want to devise local-control algorithms.
Current Situation:
- every permutation could be routed
- on permutation network and meshes
- number of steps proportional to the diameter
- algorithm was centralized
- needed global knowledge about the sources and destinations of all packets

We now want to devise local-control algorithms.

Each node decides on the next step by some local information.
The network is modelled by a graph $G = (V, E)$. 
The Model

- The network is modelled by a graph $G = (V, E)$.
- A routing problem $\mathcal{R}$ on $G$ is defined by a finite set of packets each of which comes with a source and a destination node.
The Model

- The network is modelled by a graph \( G = (V, E) \).
- A **routing problem** \( \mathcal{R} \) on \( G \) is defined by a finite set of packets each of which comes with a source and a destination node.
- We assume that time proceeds in synchronous steps:
The network is modelled by a graph $G = (V, E)$.

A routing problem $\mathcal{R}$ on $G$ is defined by a finite set of packets each of which comes with a source and a destination node.

We assume that time proceeds in synchronous steps:

- Before the first step, each packet is placed at its source.
The network is modelled by a graph $G = (V, E)$.

A routing problem $\mathcal{R}$ on $G$ is defined by a finite set of packets each of which comes with a source and a destination node.

We assume that time proceeds in synchronous steps:

- Before the first step, each packet is placed at its source.
- In each step, each edge can forward at most one packet in each direction.
The network is modelled by a graph $G = (V, E)$.

A routing problem $\mathcal{R}$ on $G$ is defined by a finite set of packets each of which comes with a source and a destination node.

We assume that time proceeds in synchronous steps:

- Before the first step, each packet is placed at its source.
- In each step, each edge can forward at most one packet in each direction.
- The routing is completed as soon as all packets have reached their destination.
The network is modelled by a graph $G = (V, E)$.

A routing problem $\mathcal{R}$ on $G$ is defined by a finite set of packets each of which comes with a source and a destination node.

We assume that time proceeds in synchronous steps:

- Before the first step, each packet is placed at its source.
- In each step, each edge can forward at most one packet in each direction.
- The routing is completed as soon as all packets have reached their destination.

The number of steps $T$ taken by an algorithm to deliver all packets is referred to as routing time.
Oblivious routing

- Here: algorithms that belong to the class of oblivious routing algorithms:
Here: algorithms that belong to the class of *oblivious routing algorithms*:

- the path of each packet depends only on the source and the destination of this packet but not on the sources and destinations of other packets.
Oblivious routing

- Here: algorithms that belong to the class of **oblivious routing algorithms**:
  - the path of each packet depends only on the source and the destination of this packet but not on the sources and destinations of other packets.
  - One specifies a path system \( \mathcal{W} \) with a path \( P_{u,v} \) from \( u \) to \( v \), for every possible source-destination pair \( (u, v) \in V^2, u \neq v \).
Oblivious routing

- Here: algorithms that belong to the class of **oblivious routing algorithms**:
  - the path of each packet depends only on the source and the destination of this packet but not on the sources and destinations of other packets.
  - One specifies a path system $\mathcal{W}$ with a path $P_{u,v}$ from $u$ to $v$, for every possible source-destination pair $(u,v) \in V^2, u \neq v$.
  - Every packet with source $u$ and destination $v$ is sent along the path $P_{u,v}$.
Oblivious routing

- Here: algorithms that belong to the class of oblivious routing algorithms:
  - the path of each packet depends only on the source and the destination of this packet but not on the sources and destinations of other packets.
  - One specifies a path system $\mathcal{W}$ with a path $P_{u,v}$ from $u$ to $v$, for every possible source-destination pair $(u, v) \in V^2, u \neq v$.
  - Every packet with source $u$ and destination $v$ is sent along the path $P_{u,v}$.
- Example: bit-fixing paths on the hypercube
Theorem

Let $G = (V, E)$ be any graph and
Let $G = (V, E)$ be any graph and

$\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2, u \neq v$. 
Lower Bound by Borodin and Hopcroft

**Theorem**

- Let $G = (V, E)$ be any graph and
- $\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2$, $u \neq v$.
- Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$. 

Let $\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2$, $u \neq v$. 

Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$. 

There exists a permutation $\pi : V \to V$ and an edge $e^* \in E$ such that at least $\sqrt{n} / (2 \Delta^2) = \Omega(\sqrt{n} / \Delta)$ of the paths selected by $\pi$ from $\mathcal{W}$ contain the edge $e^* \in E$. 

This is very bad news about deterministic oblivious routing: The time complexity for permutation routing under this paradigm is lower bounded by $\Omega(\sqrt{n} / \Delta)$, which is polynomial in $n$. 

Even a small diameter, say logarithmic in $n$, does not help.
Lower Bound by Borodin and Hopcroft

Theorem

- Let $G = (V, E)$ be any graph and
- $\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2, u \neq v$.
- Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$. 

There exists a permutation $\pi : V \rightarrow V$ and an edge $e^* \in E$ such that at least $\sqrt{n}/(2 \Delta^2) = \Omega(\sqrt{n}/\Delta)$ of the paths selected by $\pi$ from $\mathcal{W}$ contain the edge $e^* \in E$. 

This is very bad news about deterministic oblivious routing: The time complexity for permutation routing under this paradigm is lower bounded by $\Omega(\sqrt{n}/\Delta)$, which is polynomial in $n$. Even a small diameter, say logarithmic in $n$, does not help.
Theorem

Let $G = (V, E)$ be any graph and

$\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2, u \neq v$.

Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$.

There exists a permutation $\pi : V \to V$ and an edge $e^* \in E$ such that at least

$$\sqrt{n}/(2\Delta^2) = \Omega(\sqrt{n}/\Delta)$$

of the paths selected by $\pi$ from $W$ contain the edge $e^* \in E$. 
Lower Bound by Borodin and Hopcroft

Theorem

- Let $G = (V, E)$ be any graph and $\mathcal{W}$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2, u \neq v$.
- Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$.

There exists a permutation $\pi : V \rightarrow V$ and an edge $e^* \in E$ such that at least

$$\sqrt{n}/(2\Delta^2) = \Omega(\sqrt{n}/\Delta)$$

of the paths selected by $\pi$ from $\mathcal{W}$ contain the edge $e^* \in E$.

This is very bad news about deterministic oblivious routing:
Lower Bound by Borodin and Hopcroft

**Theorem**

- Let $G = (V, E)$ be any graph and
- $W$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u,v) \in V^2$, $u \neq v$.
- Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$.

There exists a permutation $\pi : V \rightarrow V$ and an edge $e^* \in E$ such that at least

$$\sqrt{n}/(2\Delta^2) = \Omega(\sqrt{n}/\Delta)$$

of the paths selected by $\pi$ from $W$ contain the edge $e^* \in E$.

- This is very bad news about deterministic oblivious routing:
- The time complexity for permutation routing under this paradigm is lower bounded by $\Omega(\sqrt{n}/\Delta)$, which is polynomial in $n$. 
Lower Bound by Borodin and Hopcroft

**Theorem**

- Let $G = (V, E)$ be any graph and
- $W$ be any path system with a path $P_{u,v}$ from $u$ to $v$, for every $(u, v) \in V^2, u \neq v$.
- Let $n$ denote the number of nodes and $\Delta$ the maximum degree of $G$.

There exists a permutation $\pi : V \to V$ and an edge $e^* \in E$ such that at least

$$\sqrt{n}/(2\Delta^2) = \Omega(\sqrt{n}/\Delta)$$

of the paths selected by $\pi$ from $W$ contain the edge $e^* \in E$.

- This is very bad news about deterministic oblivious routing:
- The time complexity for permutation routing under this paradigm is lower bounded by $\Omega(\sqrt{n}/\Delta)$, which is polynomial in $n$.
- Even a small diameter, say logarithmic in $n$, does not help.
Proof of the lower bound by Borodin and Hopcroft

Definition

- For $v \in V$, let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$. 
Proof of the lower bound by Borodin and Hopcroft

Definition

- For $v \in V$, let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$.
- For a positive number $t$, a node $v \in V$, and an edge $e \in E$, we say that $e$ is $t$-popular for $v$ if at least $t$ paths from $\mathcal{W}_v$ contain $e$. 
Proof of the lower bound by Borodin and Hopcroft

Definition

- For $v \in V$, let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$.
- For a positive number $t$, a node $v \in V$, and an edge $e \in E$, we say that $e$ is $t$-popular for $v$ if at least $t$ paths from $\mathcal{W}_v$ contain $e$. 
Proof of the lower bound by Borodin and Hopcroft

**Definition**

1. For \( v \in V \), let \( \mathcal{W}_v = \{ P_{v,u} \mid u \in V \} \).
2. For a positive number \( t \), a node \( v \in V \), and an edge \( e \in E \), we say that \( e \) is \( t \)-popular for \( v \) if at least \( t \) paths from \( \mathcal{W}_v \) contain \( e \).

**Outline of the proof:**

1. First, we prove a lemma showing that, for any given node \( v \in V \), there are many edges that are "quite popular for \( v \)."
Proof of the lower bound by Borodin and Hopcroft

Definition

- For \( v \in V \), let \( \mathcal{W}_v = \{ P_{v,u} \mid u \in V \} \).
- For a positive number \( t \), a node \( v \in V \), and an edge \( e \in E \), we say that \( e \) is \( t \)-popular for \( v \) if at least \( t \) paths from \( \mathcal{W}_v \) contain \( e \).

Outline of the proof:

- First, we prove a lemma showing that, for any given node \( v \in V \), there are many edges that are "quite popular for \( v \)."
- Then we use the lemma to show that there is an edge \( e^* \) that is "quite popular for many nodes, that is, \( e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \)."
Definition

- For $v \in V$, let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$.
- For a positive number $t$, a node $v \in V$, and an edge $e \in E$, we say that $e$ is $t$-popular for $v$ if at least $t$ paths from $\mathcal{W}_v$ contain $e$.

Outline of the proof:

- First, we prove a lemma showing that, for any given node $v \in V$, there are many edges that are "quite popular for $v$.

- Then we use the lemma to show that there is an edge $e^*$ that is "quite popular for many nodes, that is, $e^*$ is $t$-popular for $t$ different nodes, for $t = \Omega(\sqrt{n}/\Delta)$.

- Given this, we will be able to construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$, which proves the lower bound.
Proof of the lower bound by Borodin and Hopcroft

Definition

- For $t > 0$, we define a 0-1 matrix $A(t)$:
Proof of the lower bound by Borodin and Hopcroft

**Definition**

- For \( t > 0 \), we define a 0-1 matrix \( A(t) \):
- The matrix has \( n \) rows and \( |E| \) columns.
Proof of the lower bound by Borodin and Hopcroft

Definition

- For \( t > 0 \), we define a 0-1 matrix \( A(t) \):
- The matrix has \( n \) rows and \( |E| \) columns.
- For \( v \in V \), and \( e \in E \), define

\[
A_{v,e}(t) = \begin{cases} 
1 & \text{if } e \text{ is } t\text{-popular for } v, \text{ and} \\
0 & \text{otherwise,}
\end{cases}
\]
Proof of the lower bound by Borodin and Hopcroft

Definition

- For $t > 0$, we define a 0-1 matrix $A(t)$:
  - The matrix has $n$ rows and $|E|$ columns.
- For $v \in V$, and $e \in E$, define
  $$A_{v,e}(t) = \begin{cases} 1 & \text{if } e \text{ is } t\text{-popular for } v, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$
- For $v \in V$, let $A_v(t) = \sum_{e \in E} A_{v,e}(t)$ denote the row sum of $v$. 
Proof of the lower bound by Borodin and Hopcroft

**Definition**

- For \( t > 0 \), we define a 0-1 matrix \( A(t) \):
  - The matrix has \( n \) rows and \(|E|\) columns.
  - For \( v \in V \), and \( e \in E \), define
    \[
    A_{v,e}(t) = \begin{cases} 
      1 & \text{if } e \text{ is } t\text{-popular for } v, \text{ and} \\
      0 & \text{otherwise},
    \end{cases}
    \]
  - For \( v \in V \), let \( A_v(t) = \sum_{e \in E} A_{v,e}(t) \) denote the row sum of \( v \).
  - For \( e \in E \), let \( A_e(t) = \sum_{v \in V} A_{v,e}(t) \) denote the column sum of \( e \).
One Lemma for the Proof of the lower bound

**Lemma**

∀v ∈ V and t ≤ (n − 1)/Δ : Av(t) ≥ \( \frac{n}{2\Delta t} \).

**Proof of lemma:**
One Lemma for the Proof of the lower bound

Lemma

\[ \forall v \in V \text{ and } t \leq (n - 1)/\Delta : A_v(t) \geq \frac{n}{2\Delta t}. \]

Proof of lemma:

- Let \( Q \subseteq V \) be the set of nodes from which there is a path to \( v \) that contains only edges that are \( t \)-popular for \( v \).
One Lemma for the Proof of the lower bound

**Lemma**

\[ \forall v \in V \text{ and } t \leq (n - 1)/\Delta \colon A_v(t) \geq \frac{n}{2\Delta t}. \]

**Proof of lemma:**

- Let \( Q \subseteq V \) be the set of nodes from which there is a path to \( v \) that contains only edges that are \( t \)-popular for \( v \).
- Let \( L = V - Q \) and \( B = E \cap (L \times Q) \), that is, \( B \) is the set of those edges connecting a node in \( L \) with a node in \( Q \).
One Lemma for the Proof of the lower bound

Lemma

\[ \forall v \in V \text{ and } t \leq (n - 1)/\Delta : A_v(t) \geq \frac{n}{2\Delta t}. \]

Proof of lemma:

- Let \( Q \subseteq V \) be the set of nodes from which there is a path to \( v \) that contains only edges that are \( t \)-popular for \( v \).
- Let \( L = V - Q \) and \( B = E \cap (L \times Q) \), that is, \( B \) is the set of those edges connecting a node in \( L \) with a node in \( Q \).
- It holds
One Lemma for the Proof of the lower bound

**Lemma**

$$\forall v \in V \text{ and } t \leq (n - 1)/\Delta : A_v(t) \geq \frac{n}{2\Delta t}.$$  

**Proof of lemma:**

- Let $Q \subseteq V$ be the set of nodes from which there is a path to $v$ that contains only edges that are $t$-popular for $v$.

- Let $L = V - Q$ and $B = E \cap (L \times Q)$, that is, $B$ is the set of those edges connecting a node in $L$ with a node in $Q$.

- It holds
  
  - $|B| \cdot (t - 1) \geq |L|$ because, for each node $u \in L$, the path $P_{v,u}$ leads through at least one edge in $B$ and these edges are not $t$-popular so that each of them can be contained in at most $t - 1$ paths from $W_v$. 


One Lemma for the Proof of the lower bound

**Lemma**

\[ \forall v \in V \text{ and } t \leq \frac{(n - 1)}{\Delta} : A_v(t) \geq \frac{n}{2\Delta t}. \]

**Proof of lemma:**

- Let \( Q \subseteq V \) be the set of nodes from which there is a path to \( v \) that contains only edges that are \( t \)-popular for \( v \).

- Let \( L = V - Q \) and \( B = E \cap (L \times Q) \), that is, \( B \) is the set of those edges connecting a node in \( L \) with a node in \( Q \).

- It holds
  
  - \( |B| \cdot (t - 1) \geq |L| \) because, for each node \( u \in L \), the path \( P_{v,u} \) leads through at least one edge in \( B \) and these edges are not \( t \)-popular so that each of them can be contained in at most \( t - 1 \) paths from \( \mathcal{W}_v \).
  
  - \( |B| \leq \Delta |Q| \) as each node in \( Q \) has at most \( \Delta \) incident edges.
Proof of the lemma

Combining the two equations, we obtain

\[ \Delta |Q| (t - 1) \geq |L| = n - |Q| \]
Proof of the lemma

- Combining the two equations, we obtain

$$\Delta |Q|(t - 1) \geq |L| = n - |Q|,$$

- which implies

$$\Delta |Q|t \geq n.$$
Proof of the lemma

- Combining the two equations, we obtain
  \[ \Delta |Q| (t - 1) \geq |L| = n - |Q| , \]

- which implies
  \[ \Delta |Q| t \geq n . \]

- and, hence,
  \[ |Q| \geq \frac{n}{\Delta t} . \]
Proof of the lemma

Combining the two equations, we obtain

$$\Delta |Q|(t-1) \geq |L| = n - |Q|,$$

which implies

$$\Delta |Q| t \geq n.$$

and, hence,

$$|Q| \geq \frac{n}{\Delta t}.$$

Next we will show $|Q| \leq 2A_v(t)$ which completes the proof of the lemma as it implies

$$A_v(t) \geq \frac{|Q|}{2} \geq \frac{n}{2\Delta t}.$$
Proof of the lemma

Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$. 
Proof of the lemma

Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

At first, we observe that $t \leq (n-1)/\Delta$ implies that $E' \neq \emptyset$. This is because
Proof of the lemma

Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because

- $v$ has at most $\Delta$ incident edges, and
Proof of the lemma

- Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

- At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because
  - $v$ has at most $\Delta$ incident edges, and
  - $\mathcal{W}_v$ contains $n - 1$ paths.
Proof of the lemma

Show: $|Q| \leq 2A_v(t)$

- Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

- At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because
  - $v$ has at most $\Delta$ incident edges, and
  - $\mathcal{W}_v$ contains $n - 1$ paths

- such that at least one of the edges incident to $v$ is contained in at least $(n - 1)/\Delta \geq z$ paths from $\mathcal{W}_v$. 
Proof of the lemma

Show: $|Q| \leq 2A_v(t)$

- Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

- At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because
  - $v$ has at most $\Delta$ incident edges, and
  - $\mathcal{W}_v$ contains $n - 1$ paths

- such that at least one of the edges incident to $v$ is contained in at least $(n - 1)/\Delta \geq z$ paths from $\mathcal{W}_v$.

- Therefore, there is at least one edge that is $t$-popular for $v$. 
Proof of the lemma

Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because

- $v$ has at most $\Delta$ incident edges, and
- $\mathcal{W}_v$ contains $n - 1$ paths

such that at least one of the edges incident to $v$ is contained in at least $(n - 1)/\Delta \geq z$ paths from $\mathcal{W}_v$.

Therefore, there is at least one edge that is $t$-popular for $v$.

Given that $E'$ is non-empty, each node in $Q$ is incident to an edge in $E'$. 

Proof of the lemma

- Let $E'$ denote the set of edges that are $t$-popular for $v$. To complete the proof of the lemma, we have to show $|Q| \leq 2|E'| = 2A_v(t)$.

- At first, we observe that $t \leq (n - 1)/\Delta$ implies that $E' \neq \emptyset$. This is because
  - $v$ has at most $\Delta$ incident edges, and
  - $\mathcal{W}_v$ contains $(n - 1)$ paths

such that at least one of the edges incident to $v$ is contained in at least $(n - 1)/\Delta \geq z$ paths from $\mathcal{W}_v$.

- Therefore, there is at least one edge that is $t$-popular for $v$.

- Given that $E'$ is non-empty, each node in $Q$ is incident to an edge in $E'$.

- Consequently, $|Q| \leq 2|E'|$ as each of the edges in $E'$ is incident to at most two nodes from $Q$. 

Show: $|Q| \leq 2A_v(t)$
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \).

Our next goal is to show that there exists an edge \( e^* \) that is \( t \)-popular for \( t \) nodes where \( t = \Omega(\sqrt{n}/\Delta) \).
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \).

- Our next goal is to show that there exists an edge \( e^* \) that is \( t \)-popular for \( t \) nodes where \( t = \Omega(\sqrt{n}/\Delta) \).

- We observe that

\[
\sum_{e \in E} A_e(t) = \sum_{e \in E} \sum_{v \in V} A_{e,v}(t) = \sum_{v \in V} \sum_{e \in E} A_{e,v}(t) = \sum_{v \in V} A_v(t) \geq \frac{n^2}{2\Delta t},
\]

where the inequality follows from the lemma.
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^* \): \( e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \).

- Our next goal is to show that there exists an edge \( e^* \) that is \( t \)-popular for \( t \) nodes where \( t = \Omega(\sqrt{n}/\Delta) \).

- We observe that

\[
\sum_{e \in E} A_e(t) = \sum_{e \in E} \sum_{v \in V} A_{e,v}(t) = \sum_{v \in V} \sum_{e \in E} A_{e,v}(t) = \sum_{v \in V} A_v(t) \geq \frac{n^2}{2\Delta t},
\]

where the inequality follows from the lemma.

- Because of the “pigeonhole principle”, there has to exist an edge \( e^* \in E \) such that

\[
A_{e^*}(t) \geq \left\lceil \frac{n^2}{|E| \cdot 2\Delta t} \right\rceil \geq \left\lceil \frac{n}{2\Delta^2 t} \right\rceil,
\]

where the last step follows from \( |E| \leq \Delta n \).
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \). We have \( A_{e^*}(t) \geq \lceil \frac{n}{2\Delta^2 t} \rceil \).

- Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2 t} \), that is, we set \( t = \sqrt{n}/(\sqrt{2}\Delta) \).
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \text{ is } t\text{-popular for } t \text{ different nodes, for } t = \Omega(\sqrt{n/\Delta}). \) We have \( A_{e^*}(t) \geq \lceil \frac{n}{2\Delta^2 t} \rceil. \)

- Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2 t} \), that is, we set \( t = \sqrt{n/\Delta}. \)
- Observe that \( t = \sqrt{n}/(\sqrt{2}\Delta) \) implies \( t \leq (n - 1)/\Delta \), for any \( n \geq 2 \),
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \). We have \( A_{e^*}(t) \geq \lceil \frac{n}{2\Delta^2 t} \rceil \).

- Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2 t} \), that is, we set \( t = \sqrt{n}/(\sqrt{2}\Delta) \).
- Observe that \( t = \sqrt{n}/(\sqrt{2}\Delta) \) implies \( t \leq (n - 1)/\Delta \), for any \( n \geq 2 \),
- so that the assumption about \( t \) that we made in the lemma is satisfied.
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \text{ is } t\text{-popular for } t \text{ different nodes, for } t = \Omega(\sqrt{n}/\Delta). \) We have \( A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2t} \right\rceil \).

Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2t} \), that is, we set \( t = \sqrt{n}/(\sqrt{2}\Delta) \).

- Observe that \( t = \sqrt{n}/(\sqrt{2}\Delta) \) implies \( t \leq (n - 1)/\Delta \), for any \( n \geq 2 \),
- so that the assumption about \( t \) that we made in the lemma is satisfied.

For this choice of \( t \), our analysis gives
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^* \): \( e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \). We have \( A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2 t} \right\rceil \).

Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2 t} \), that is, we set \( t = \frac{\sqrt{n}}{(\sqrt{2}\Delta)} \).

Observe that \( t = \frac{\sqrt{n}}{(\sqrt{2}\Delta)} \) implies \( t \leq \frac{(n - 1)}{\Delta} \), for any \( n \geq 2 \),

so that the assumption about \( t \) that we made in the lemma is satisfied.

For this choice of \( t \), our analysis gives

\[
A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2 \sqrt{n}/(\sqrt{2}\Delta)} \right\rceil = \left\lceil \frac{n\sqrt{2}\Delta}{2\Delta^2 \sqrt{n}} \right\rceil = \left\lceil \frac{\sqrt{n}}{\sqrt{2}\Delta} \right\rceil.
\]
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is \( t \)-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \). We have \( A_{e^*}(t) \geq \lceil \frac{n}{2\Delta^2} \rceil \).

- Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2} \), that is, we set \( t = \sqrt{n}/(\sqrt{2}\Delta) \).
- Observe that \( t = \sqrt{n}/(\sqrt{2}\Delta) \) implies \( t \leq (n-1)/\Delta \), for any \( n \geq 2 \), so that the assumption about \( t \) that we made in the lemma is satisfied.

For this choice of \( t \), our analysis gives

- \( A_{e^*}(t) \geq \lceil \frac{n}{2\Delta^2 \sqrt{n}/(\sqrt{2}\Delta)} \rceil = \lceil \frac{n\sqrt{2}\Delta}{2\Delta^2 \sqrt{n}} \rceil = \lceil \frac{\sqrt{n}}{\sqrt{2}\Delta} \rceil \).
- \( A_{e^*}(t) \geq \lceil t \rceil \), that is,
Proof of the lower bound by Borodin and Hopcroft

Show: \( \exists e^*: e^* \) is t-popular for \( t \) different nodes, for \( t = \Omega(\sqrt{n}/\Delta) \). We have \( A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2 t} \right\rceil \).

- Next we choose \( t \) such that \( t = \frac{n}{2\Delta^2 t} \), that is, we set \( t = \frac{\sqrt{n}}{(\sqrt{2}\Delta)} \).
- Observe that \( t = \frac{\sqrt{n}}{(\sqrt{2}\Delta)} \) implies \( t \leq \frac{n-1}{\Delta} \), for any \( n \geq 2 \), so that the assumption about \( t \) that we made in the lemma is satisfied.

- For this choice of \( t \), our analysis gives
  - \( A_{e^*}(t) \geq \left\lceil \frac{n}{2\Delta^2 \sqrt{n}/(\sqrt{2}\Delta)} \right\rceil = \left\lceil \frac{n\sqrt{2}\Delta}{2\Delta^2 \sqrt{n}} \right\rceil = \left\lceil \frac{\sqrt{n}}{\sqrt{2}\Delta} \right\rceil \).
  - \( A_{e^*}(t) \geq \left\lceil t \right\rceil \), that is,
  - \( e^* \) is \( \left\lceil t \right\rceil \)-popular for \( \left\lceil t \right\rceil \) nodes, where \( t = \frac{\sqrt{n}}{\sqrt{2}\Delta} \).
Proof of the lower bound by Borodin and Hopcroft

Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$. Finally, we construct a permutation $\pi$ such that $\lceil t \rceil$ of the paths selected by $\pi$ contain $e^*$:

- Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
- W.l.o.g., $V' = \{1, \ldots, \lceil t \rceil \}$. 
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$. Finally, we construct a permutation $\pi$ such that $\lceil t \rceil$ of the paths selected by $\pi$ contain $e^*$:

- Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
- W.l.o.g., $V' = \{1, \ldots, \lceil t \rceil\}$.
- For every $v \in V'$, there exists a subset $U_v \subseteq V$ of cardinality $\lceil t \rceil$ such that, for every $u \in U_v$, the path $P_{v,u}$ contains $e^*$.
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$. Finally, we construct a permutation $\pi$ such that $\lceil t \rceil$ of the paths selected by $\pi$ contain $e^*$:

- Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
- W.l.o.g., $V' = \{1, \ldots, \lceil t \rceil\}$.
- For every $v \in V'$, there exists a subset $U_v \subseteq V$ of cardinality $\lceil t \rceil$ such that, for every $u \in U_v$, the path $P_{v,u}$ contains $e^*$.
- For $v = 1$ to $\lceil t \rceil$, set $\pi(v) = u$ where $u$ is chosen arbitrarily from $U_v \setminus \{\pi(1), \ldots, \pi(v-1)\}$. 
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation \( \pi \) such that \( t \) of the paths selected by \( \pi \) contain \( e^* \).

Finally, we construct a permutation \( \pi \) such that \( \lceil t \rceil \) of the paths selected by \( \pi \) contain \( e^* \):

- Let \( V' \) denote a set of \( \lceil t \rceil \) nodes for which \( e^* \) is \( \lceil t \rceil \)-popular.
- W.l.o.g., \( V' = \{1, \ldots, \lceil t \rceil \} \).
- For every \( v \in V' \), there exists a subset \( U_v \subseteq V \) of cardinality \( \lceil t \rceil \) such that, for every \( u \in U_v \), the path \( P_{v,u} \) contains \( e^* \).
- For \( v = 1 \) to \( \lceil t \rceil \), set \( \pi(v) = u \) where \( u \) is chosen arbitrarily from \( U_v \setminus \{\pi(1), \ldots, \pi(v - 1)\} \).
- For \( v = \lceil t \rceil + 1 \) to \( n \), set \( \pi(v) = u \) where \( u \) is chosen arbitrarily from \( V \setminus \{\pi(1), \ldots, \pi(v - 1)\} \).
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$. Finally, we construct a permutation $\pi$ such that $\lceil t \rceil$ of the paths selected by $\pi$ contain $e^*$:

- Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
- W.l.o.g., $V' = \{1, \ldots, \lceil t \rceil\}$.
- For every $v \in V'$, there exists a subset $U_v \subseteq V$ of cardinality $\lceil t \rceil$ such that, for every $u \in U_v$, the path $P_{v, u}$ contains $e^*$.
- For $v = 1$ to $\lceil t \rceil$, set $\pi(v) = u$ where $u$ is chosen arbitrarily from $U_v \setminus \{\pi(1), \ldots, \pi(v - 1)\}$.
- For $v = \lceil t \rceil + 1$ to $n$, set $\pi(v) = u$ where $u$ is chosen arbitrarily from $V \setminus \{\pi(1), \ldots, \pi(v - 1)\}$. 
Proof of the lower bound by Borodin and Hopcroft

Construct a permutation $\pi$ such that $t$ of the paths selected by $\pi$ contain $e^*$. Finally, we construct a permutation $\pi$ such that $\lceil t \rceil$ of the paths selected by $\pi$ contain $e^*$:

- Let $V'$ denote a set of $\lceil t \rceil$ nodes for which $e^*$ is $\lceil t \rceil$-popular.
- W.l.o.g., $V' = \{1, \ldots, \lceil t \rceil\}$.
- For every $v \in V'$, there exists a subset $U_v \subseteq V$ of cardinality $\lceil t \rceil$ such that, for every $u \in U_v$, the path $P_{v,u}$ contains $e^*$.
- For $v = 1$ to $\lceil t \rceil$, set $\pi(v) = u$ where $u$ is chosen arbitrarily from $U_v \setminus \{\pi(1), \ldots, \pi(v-1)\}$.
- For $v = \lceil t \rceil + 1$ to $n$, set $\pi(v) = u$ where $u$ is chosen arbitrarily from $V \setminus \{\pi(1), \ldots, \pi(v-1)\}$.

By our construction, $\pi$ and $e^*$ satisfy the properties described in the theorem.
Application to the hypercube and Goal

- For the $d$-dimensional hypercube with $n = 2^d$ nodes, the lower bound of Borodin and Hopcroft implies a lower bound of $\Omega(\sqrt{n}/ \log n)$ for permutation routing.
Application to the hypercube and Goal

- For the $d$-dimensional hypercube with $n = 2^d$ nodes, the lower bound of Borodin and Hopcroft implies a lower bound of $\Omega(\sqrt{n}/\log n)$ for permutation routing.

- There is a permutation $\pi$ such that $\Omega(\sqrt{n})$ paths contain the same edge when using bit-fixing paths on the hypercube.
Application to the hypercube and Goal

- For the $d$-dimensional hypercube with $n = 2^d$ nodes, the lower bound of Borodin and Hopcroft implies a lower bound of $\Omega(\sqrt{n}/\log n)$ for permutation routing.

- There is a permutation $\pi$ such that $\Omega(\sqrt{n})$ paths contain the same edge when using bit-fixing paths on the hypercube.

- Consequently, when using bit-fixing paths the time complexity for permutation routing is $\Omega(\sqrt{n})$. 
Application to the hypercube and Goal

- For the $d$-dimensional hypercube with $n = 2^d$ nodes, the lower bound of Borodin and Hopcroft implies a lower bound of $\Omega(\sqrt{n}/\log n)$ for permutation routing.

- There is a permutation $\pi$ such that $\Omega(\sqrt{n})$ paths contain the same edge when using bit-fixing paths on the hypercube.

- Consequently, when using bit-fixing paths the time complexity for permutation routing is $\Omega(\sqrt{n})$.

- Our goal is to devise a distributed permutation routing algorithm with time complexity $O(\log n)$. 
Application to the hypercube and Goal

- For the $d$-dimensional hypercube with $n = 2^d$ nodes, the lower bound of Borodin and Hopcroft implies a lower bound of $\Omega(\sqrt{n}/ \log n)$ for permutation routing.

- There is a permutation $\pi$ such that $\Omega(\sqrt{n})$ paths contain the same edge when using bit-fixing paths on the hypercube.

- Consequently, when using bit-fixing paths the time complexity for permutation routing is $\Omega(\sqrt{n})$.

- Our goal is to devise a distributed permutation routing algorithm with time complexity $O(\log n)$.

- This will take some time.
Outline of the approach

- We build a dynamic system of storage devices supporting the addition and removal of storage devices using dynamic hashing:

\[\text{independently, uniformly at random}\]
Outline of the approach

- We build a dynamic system of storage devices supporting the addition and removal of storage devices using dynamic hashing:
  - devices are mapped i.u.r.\(^1\) to the ring \([0, 1)\), that is, each device \(i\) gets assigned a random address \(a(i) \in [0, 1)\)

\(^1\)independently, uniformly at random
Outline of the approach

- We build a dynamic system of storage devices supporting the addition and removal of storage devices using dynamic hashing:
  - *devices are mapped i.u.r.\(^1\) to the ring \([0, 1)\), that is, each device \(i\) gets assigned a random address \(a(i) \in [0, 1)\)*
  - *data objects are mapped to the ring using a random hash function \(h : U \rightarrow [0, 1)\), that is, object \(x\) is mapped to position \(h(x)\)*

\(^1\)independently, uniformly at random
Outline of the approach

- We build a dynamic system of storage devices supporting the addition and removal of storage devices using dynamic hashing:
  - devices are mapped i.u.r.\(^1\) to the ring \([0, 1)\), that is, each device \(i\) gets assigned a random address \(a(i) \in [0, 1)\)
  - data objects are mapped to the ring using a random hash function \(h : U \rightarrow [0, 1)\), that is, object \(x\) is mapped to position \(h(x)\)
  - data object \(x\) is stored on the device found next to \(h(x)\) in clock-wise direction on the ring

\(^1\)independently, uniformly at random
Outline of the approach

- We build a dynamic system of storage devices supporting the addition and removal of storage devices using dynamic hashing:
  - devices are mapped i.u.r.¹ to the ring $[0, 1)$, that is, each device $i$ gets assigned a random address $a(i) \in [0, 1)$
  - data objects are mapped to the ring using a random hash function $h : U \rightarrow [0, 1)$, that is, object $x$ is mapped to position $h(x)$
  - data object $x$ is stored on the device found next to $h(x)$ in clock-wise direction on the ring
- We assume an idealistic hash function, that is, the hash values are real numbers chosen i.u.r. from $[0, 1)$.

¹independently, uniformly at random
Definition of successors

- Let $V$ be the set of storage devices at some point of time, and let $n = |V|$. 


Definition of successors

- Let $V$ be the set of storage devices at some point of time, and let $n = |V|$.
- For address $A \in [0, 1)$, define

$$succ(A) = \begin{cases} \arg\min\{a(i) \geq A \mid i \in V\} & \text{if } \exists i \in V : a(i) \in [A, 1), \\ \arg\min\{a(i) \geq 0 \mid i \in V\} & \text{otherwise.} \end{cases}$$

$$pred(A) = \begin{cases} \arg\max\{a(i) < A \mid i \in V\} & \text{if } \exists i \in V : a(i) \in [0, A), \\ \arg\max\{a(i) < 1 \mid i \in V\} & \text{else.} \end{cases}$$
Definition of successors

- Let $V$ be the set of storage devices at some point of time, and let $n = |V|$.
- For address $A \in [0, 1)$, define
  
  $$
  \text{succ}(A) = \begin{cases} 
  \text{argmin}\{a(i) \geq A \mid i \in V\} & \text{if } \exists i \in V : a(i) \in [A, 1), \\
  \text{argmin}\{a(i) \geq 0 \mid i \in V\} & \text{otherwise}. 
  \end{cases}
  $$

  $$
  \text{pred}(A) = \begin{cases} 
  \text{argmax}\{a(i) < A \mid i \in V\} & \text{if } \exists i \in V : a(i) \in [0, A), \\
  \text{argmax}\{a(i) < 1 \mid i \in V\} & \text{else}. 
  \end{cases}
  $$

- Object $x \in U$ is mapped to device $\text{succ}(h(x))$. 
Quality of the load balancing

- The quality of the load balancing depends on the distribution of the sizes of the ring for which the storage devices are responsible.
The quality of the load balancing depends on the distribution of the sizes of the ring for which the storage devices are responsible.

Definition (weight of a device)

For device $i \in V$, define the weight of device $i$ by

$$W_i = \begin{cases} a(i) - a(pred(a(i))) & \text{if } a(pred(a(i))) < a(i), \\ 1 - (a(pred(a(i))) - a(i)) & \text{otherwise.} \end{cases}$$

Let $W = \max_{i \in [n]} W_i$. 
Quality of the load balancing

- The quality of the load balancing depends on the distribution of the sizes of the ring for which the storage devices are responsible.

**Definition (weight of a device)**

For device $i \in V$, define the weight of device $i$ by

$$W_i = \begin{cases} a(i) - a(pred(a(i))) & \text{if } a(pred(a(i))) < a(i), \\ 1 - (a(pred(a(i))) - a(i)) & \text{otherwise}. \end{cases}$$

Let $W = \max_{i \in [n]} W_i$.

- Ideally, we would have $W = W_0 = W_1 = \ldots = W_{n-1} = \frac{1}{n}$. 
Quality of the load balancing

- The quality of the load balancing depends on the distribution of the sizes of the ring for which the storage devices are responsible.

**Definition (weight of a device)**

For device $i \in V$, define the weight of device $i$ by

$$W_i = \begin{cases} a(i) - a(\text{pred}(a(i))) & \text{if } a(\text{pred}(a(i))) < a(i), \\ 1 - (a(\text{pred}(a(i))) - a(i)) & \text{otherwise}. \end{cases}$$

Let $W = \max_{i \in [n]} W_i$.

- Ideally, we would have $W = W_0 = W_1 = \ldots = W_{n-1} = \frac{1}{n}$.
- We will show that $W = O\left(\frac{\log n}{n}\right)$, w.h.p.$^2$

---

$^2$The term "w.h.p.äbbreviates "with high probabilityänd means with probability at least $1 - n^{-\alpha}$, for any constant $\alpha > 0$. 
Quality of the load balancing

**Lemma**

Let $T \subseteq [0, 1)$ and $t = |T|$ the mass (length) of $T$. Suppose that $M$ points are chosen i.u.r. from $[0, 1)$. The probability that none of these points is from $T$ is at most $e^{-tM}$.

**Proof:**

$$\Pr[\text{no point in } T] = (1 - t)^M = ((1 - t)^{1/t})^{tM} \leq e^{-tM}$$

as, for every $x > 0$, it holds $(1 - \frac{1}{x})^x \leq \frac{1}{e}$.
Quality of the load balancing

**Theorem**

\[ W = O\left(\frac{\log n}{n}\right), \text{ w.h.p.} \]

**Proof:**

- Fix \( j \in V \). Suppose \( j \)'s address \( a(j) \) is fixed arbitrarily.

\[ \Pr[\text{no point in } T] \leq e^{-tM}. \]
Quality of the load balancing

**Theorem**

\[ W = O\left(\frac{\log n}{n}\right), \text{ w.h.p.} \]

**Proof:**

- Fix \( j \in V \). Suppose \( j \)'s address \( a(j) \) is fixed arbitrarily.
- A necessary condition for the event \( W_j \geq t, \ t \in [0, 1) \), is that no addresses of the other \( n-1 \) devices falls into the interval from \( a(j) - t \) to \( a(j) \).
Quality of the load balancing

**Theorem**

\[ W = O\left(\frac{\log n}{n}\right), \text{ w.h.p.} \]

**Proof:**

- Fix \( j \in V \). Suppose \( j \)'s address \( a(j) \) is fixed arbitrarily.
- A necessary condition for the event \( W_j \geq t, \ t \in [0, 1) \), is that no addresses of the other \( n-1 \) devices falls into the interval from \( a(j) - t \) to \( a(j) \).
- For any \( \alpha > 0 \),
  \[
  \Pr\left[W_j \geq 2(\alpha + 1)\frac{\ln n}{n}\right] \leq e^{-2(\alpha + 1)\frac{\ln n}{n}(n-1)} \\
  \leq e^{-(\alpha + 1)\ln n} = n^{-(\alpha + 1)}
  \]
Quality of the load balancing

Pr [no point in $T$] $\leq e^{-tM}$.

**Theorem**

$W = O\left(\frac{\log n}{n}\right)$, w.h.p.

**Proof:**

- Fix $j \in V$. Suppose $j$’s address $a(j)$ is fixed arbitrarily.
- A necessary condition for the event $W_j \geq t$, $t \in [0, 1)$, is that no addresses of the other $n-1$ devices falls into the interval from $a(j) - t$ to $a(j)$.
- for any $\alpha > 0$,
  
  $$\Pr \left[ W_j \geq 2(\alpha + 1) \frac{\ln n}{n} \right] \leq e^{-2(\alpha + 1) \frac{\ln n}{n} (n-1)}$$

  $$\leq e^{-(\alpha+1) \ln n} = n^{-(\alpha+1)}$$

- and, hence,
  
  $$\Pr \left[ W \geq 2(\alpha + 1) \frac{\ln n}{n} \right] \leq \sum_{j \in V} \Pr \left[ W_j \geq 2(\alpha + 1) \frac{\ln n}{n} \right] \leq n^{-\alpha}.$$
Improved quality of the load balancing

We have $W = O\left(\frac{\log n}{n}\right)$, w.h.p.

- In order to improve the load balancing, we use $k$ virtual nodes for each device. Let $V'$ denote the set of $kn$ virtual nodes.
Improved quality of the load balancing

- In order to improve the load balancing, we use \( k \) virtual nodes for each device. Let \( V' \) denote the set of \( kn \) "virtual nodes.
- Each of these nodes gets an address from \([0,1)\) chosen i.u.r.

We have \( W = O\left(\frac{\log n}{n}\right) \), w.h.p.
Improved quality of the load balancing

We have $W = O\left(\frac{\log n}{n}\right)$, w.h.p.

- In order to improve the load balancing, we use $k$ virtual nodes for each device. Let $V'$ denote the set of $kn$ virtual nodes.
- Each of these nodes gets an address from $[0,1)$ chosen i.u.r.
- For address $A \in [0,1)$, re-define

\[
\text{succ}(A) = \begin{cases} 
\arg\min \{ a(i) \geq A \mid i \in V' \} & \text{if } \exists i \in V' : a(i) \in [A, 1), \\
\arg\min \{ a(i) \geq 0 \mid i \in V' \} & \text{otherwise}.
\end{cases}
\]
Improved quality of the load balancing

- In order to improve the load balancing, we use $k$ virtual nodes for each device. Let $V'$ denote the set of $kn$ virtual nodes.
- Each of these nodes gets an address from $[0,1)$ chosen i.u.r.
- For address $A \in [0,1)$, re-define
  \[
  \text{succ}(A) = \begin{cases} 
  \arg\min\{a(i) \geq A \mid i \in V'\} & \text{if } \exists i \in V' : a(i) \in [A,1), \\
  \arg\min\{a(i) \geq 0 \mid i \in V'\} & \text{otherwise.}
  \end{cases}
  \]
- Object $x \in U$ is mapped to node $\text{succ}(h(x))$ and stored on the device to which this node belongs.

We have $W = O\left(\frac{\log n}{n}\right)$, w.h.p.
Improved quality of the load balancing

We have $W = O\left(\frac{\log n}{n}\right)$, w.h.p.

- In order to improve the load balancing, we use $k$ virtual nodes for each device. Let $V'$ denote the set of $kn$ virtual nodes.

- Each of these nodes gets an address from $[0,1)$ chosen i.u.r.

- For address $A \in [0,1)$, re-define

  $$\text{succ}(A) = \begin{cases} \arg\min\{a(i) \geq A \mid i \in V'\} & \text{if } \exists i \in V' : a(i) \in [A, 1), \\ \arg\min\{a(i) \geq 0 \mid i \in V'\} & \text{otherwise.} \end{cases}$$

- Object $x \in U$ is mapped to node $\text{succ}(h(x))$ and stored on the device to which this node belongs.

- Let $W_i$ denote the weight of device $i$, i.e., the sum of the lengths of the intervals corresponding to $i$'s nodes, and $W = \max_{i \in [n]} W_i$. 
Improved quality of the load balancing

**Theorem**

For any $k \geq 1$, $W = \frac{1}{n} \cdot O\left(1 + \frac{\log n}{k}\right)$, w.h.p.
Improved quality of the load balancing

**Theorem**

For any $k \geq 1$, $W = \frac{1}{n} \cdot O\left(1 + \frac{\log n}{k}\right)$, w.h.p.

**Corollary**

If $k \geq \log n$ then $W = O\left(\frac{1}{n}\right)$, w.h.p.
Improved quality of the load balancing

**Theorem**

For any $k \geq 1$, $W = \frac{1}{n} \cdot O\left(1 + \frac{\log n}{k}\right)$, w.h.p.

**Corollary**

If $k \geq \log n$ then $W = O\left(\frac{1}{n}\right)$, w.h.p.

**Proof of the Theorem:**

- Consider device $j$ and suppose the address of the $k$ nodes of this device are fixed arbitrarily.
Improved quality of the load balancing

**Theorem**

For any $k \geq 1$, $W = \frac{1}{n} \cdot O(1 + \frac{\log n}{k})$, w.h.p.

**Corollary**

If $k \geq \log n$ then $W = O\left(\frac{1}{n}\right)$, w.h.p.

**Proof of the Theorem:**

- Consider device $j$ and suppose the address of the $k$ nodes of this device are fixed arbitrarily.

- For any $t \in [0, 1)$, we want to upper-bound $\Pr[W_j \geq t]$. 
Improved quality of the load balancing

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- these intervals have a total length of $t$, and
Improved quality of the load balancing

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- these intervals have a total length of $t$, and
- none of the other $k(n - 1)$ nodes have an address that falls into these intervals.
Improved quality of the load balancing

Exact condition:

The event \( W_j \geq t \) happens if and only if there are \( k \) intervals left of the \( k \) addresses of \( j \)'s nodes so that

- these intervals have a total length of \( t \), and
- none of the other \( k(n - 1) \) nodes have an address that falls into these intervals.
**Improved quality of the load balancing**

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that:

- these intervals have a total length of $t$, and
- none of the other $k(n - 1)$ nodes have an address that falls into these intervals.

In order to be able to enumerate all possibilities for choosing these $k$ intervals, we look at a slightly stronger **necessary condition** for the event $W_j \geq t$. 
Improved quality of the load balancing

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- these intervals have a total length of $t$, and
- none of the other $k(n-1)$ nodes have an address that falls into these intervals.

**Necessary condition:**

If the event $W_j \geq t$ happens then there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- the length of each of these intervals is a multiple of $\frac{1}{kn}$.
Improved quality of the load balancing

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- these intervals have a total length of $t$, and
- none of the other $k(n-1)$ nodes have an address that falls into these intervals.

**Necessary condition:**

If the event $W_j \geq t$ happens then there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- the length of each of these intervals is a multiple of $\frac{1}{kn}$
- these intervals have a total length of $t'$ where $t'$ is the largest multiple of $\frac{1}{kn}$ such that $t' \leq t - \frac{1}{n}$, and
Improved quality of the load balancing

**Exact condition:**

The event $W_j \geq t$ happens if and only if there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- these intervals have a total length of $t$, and
- none of the other $k(n - 1)$ nodes have an address that falls into these intervals.

**Necessary condition:**

If the event $W_j \geq t$ happens then there are $k$ intervals left of the $k$ addresses of $j$’s nodes so that

- the length of each of these intervals is a multiple of $\frac{1}{kn}$
- these intervals have a total length of $t'$ where $t'$ is the largest multiple of $\frac{1}{kn}$ such that $t' \leq t - \frac{1}{n}$, and
- none of the other $k(n - 1)$ nodes have an address that falls into these intervals.
Improved quality of the load balancing

- The number of possibilities to choose these intervals corresponds to the number of possibilities to choose $k$ integers $q_1, \ldots, q_k$ such that $\sum_{i=1}^{k} q_i = q$, for $q = t' kn$. 
Improved quality of the load balancing

- The number of possibilities to choose these intervals corresponds to the number of possibilities to choose $k$ integers $q_1, \ldots, q_k$ such that $\sum_{i=1}^{k} q_i = q$, for $q = t'kn$.
- The $q_i$’s can be encoded bijectively into binary strings with $k - 1$ ones and $q$ zeros.
**Improved quality of the load balancing**

- The number of possibilities to choose these intervals corresponds to the number of possibilities to choose $k$ integers $q_1, \ldots, q_k$ such that $\sum_{i=1}^{k} q_i = q$, for $q = t'kn$.

- The $q_i$’s can be encoded bijectively into binary strings with $k - 1$ ones and $q$ zeros.

- Thus, the number of possibilities to choose the $q_i$’s and, hence, the intervals is at most
  
  $$\binom{q + k - 1}{k - 1} \leq \binom{q + k}{k} \leq \left(\frac{e(q + k)}{k}\right)^k.$$
Improved quality of the load balancing

- The number of possibilities to choose these intervals corresponds to the number of possibilities to choose $k$ integers $q_1, \ldots, q_k$ such that $\sum_{i=1}^{k} q_i = q$, for $q = t'kn$.

- The $q_i$’s can be encoded bijectively into binary strings with $k - 1$ ones and $q$ zeros.

- Thus, the number of possibilities to choose the $q_i$’s and, hence, the intervals is at most

$$\binom{q + k - 1}{k - 1} \leq \binom{q + k}{k} \leq \left(\frac{e(q + k)}{k}\right)^k.$$ 

- Now $q + k = t'kn + k \leq (t - \frac{1}{n})kn + k = tkn$, so that this number is at most

$$\left(\frac{etkn}{k}\right)^k = (etn)^k.$$
Improved quality of the load balancing

Once the intervals are fixed, the probability that these intervals with a total length of $t' \geq t - \frac{2}{n}$ are not hit by one other $k(n - 1)$ addresses is at most

$$e^{-t'k(n-1)} \leq e^{-(t - \frac{2}{n})k(n-1)}$$
Improved quality of the load balancing

- Once the intervals are fixed, the probability that these intervals with a total length of $t' \geq t - \frac{2}{n}$ are not hit by one other $k(n - 1)$ addresses is at most

$$e^{-t'k(n-1)} \leq e^{-(t - \frac{2}{n})k(n-1)}$$

- which follows analogously to the lemma on slide 17.
Improved quality of the load balancing

- Once the intervals are fixed, the probability that these intervals with a total length of $t' \geq t - \frac{2}{n}$ are not hit by one other $k(n - 1)$ addresses is at most
  \[ e^{-t'k(n-1)} \leq e^{-(t - \frac{2}{n})k(n-1)} \]

- which follows analogously to the lemma on slide 17.

- This gives
  \[ \Pr [W_i \geq t] \leq (etn)^k \cdot e^{-(t - \frac{2}{n})k(n-1)}. \]
Improved quality of the load balancing

- Once the intervals are fixed, the probability that these intervals with a total length of $t' \geq t - \frac{2}{n}$ are not hit by one other $k(n - 1)$ addresses is at most
  \[ e^{-t'k(n-1)} \leq e^{-\left(t - \frac{2}{n}\right)k(n-1)} \]

- which follows analogously to the lemma on slide 17.

- This gives
  \[ \Pr[W_i \geq t] \leq (etn)^k \cdot e^{-\left(t - \frac{2}{n}\right)k(n-1)} . \]

- Now choose $t = \frac{\beta}{n}$, where the value for $\beta$ will be specified later.
Improved quality of the load balancing

- Once the intervals are fixed, the probability that these intervals with a total length of $t' \geq t - \frac{2}{n}$ are not hit by one other $k(n - 1)$ addresses is at most
  \[ e^{-t'k(n-1)} \leq e^{-(t - \frac{2}{n})k(n-1)} \]

- which follows analogously to the lemma on slide 17.

- This gives
  \[ \Pr[W_i \geq t] \leq (etn)^k \cdot e^{-(t - \frac{2}{n})k(n-1)} . \]

- Now choose $t = \frac{\beta}{n}$, where the value for $\beta$ will be specified later.

- This gives $etn = e\beta$ and
  \[ \left( t - \frac{2}{n} \right) (n - 1) \geq \left( t - \frac{2}{n} \right) \frac{n}{2} = \frac{\beta}{2} - 1 \]
  assuming $n \geq 2$. 
Improved quality of the load balancing

- Consequently,

\[ \Pr \left[ W_i \geq \frac{\beta}{n} \right] \leq \left( e\beta \cdot e^{-\beta/2+1} \right)^k \]

\[ = \left( e\beta \cdot e^{-\beta/2+1} \cdot \left( \frac{4}{3} \right)^\beta \right)^k \left( \frac{3}{4} \right)^{\beta k} \]
**Improved quality of the load balancing**

**Consequently,**

$$\Pr \left[ \mathcal{W}_i \geq \frac{\beta}{n} \right] \leq \left( e^{\beta} \cdot e^{-\beta/2+1} \right)^k$$

$$= \left( e^{\beta} \cdot e^{-\beta/2+1} \cdot \left( \frac{4}{3} \right)^\beta \right)^k \left( \frac{3}{4} \right)^{\beta k}.$$

**Now observe** that $e^{\beta} \cdot e^{-\beta/2+1} \cdot \left( \frac{4}{3} \right)^\beta$ decreases exponentially in $\beta$ since $e^{\frac{1}{2}} \geq \frac{4}{3}$. For $\beta \geq 25$, this term is less than 1. Consequently,

$$\Pr \left[ \mathcal{W}_i \geq \frac{\beta}{n} \right] \leq \left( \frac{3}{4} \right)^{\beta k} \leq \left( \frac{3}{4} \right)^{(\alpha+1) \log_4 \frac{4}{3} n} = n^{-(\alpha+1)},$$

for $\beta \geq \frac{(\alpha+1) \log_4 \frac{4}{3} n}{k}$. 
Improved quality of the load balancing

- Consequently,

\[ \Pr \left[ W_i \geq \frac{\beta n}{n} \right] \leq \left( e \beta \cdot e^{-\beta/2+1} \right)^k \]

\[ = \left( e \beta \cdot e^{-\beta/2+1} \cdot \left( \frac{4}{3} \right)^{\beta} \right)^k \left( \frac{3}{4} \right)^{\beta k} . \]

- Now observe that \( e \beta \cdot e^{-\beta/2+1} \cdot \left( \frac{4}{3} \right)^{\beta} \) decreases exponentially in \( \beta \) since \( e^{\frac{1}{2}} \geq \frac{4}{3} \). For \( \beta \geq 25 \), this term is less than 1. Consequently,

\[ \Pr \left[ W_i \geq \frac{\beta n}{n} \right] \leq \left( \frac{3}{4} \right)^{\beta k} \leq \left( \frac{3}{4} \right)^{(\alpha+1) \log_{4/3} n} = n^{-(\alpha+1)} , \]

for \( \beta \geq \frac{(\alpha+1) \log_{4/3} n}{k} . \)

- It follows \( \Pr \left[ W \geq \frac{\beta n}{n} \right] \leq n^{-\alpha} \), for \( \beta = O(1 + \frac{\log n}{k}) \), which proves the theorem.
Now we connect the nodes from the consistent hashing scheme by an overlay network called Chord running on top of the Internet.
Overlay network

- Now we connect the nodes from the consistent hashing scheme by an overlay network called Chord running on top of the Internet.

- Each node holds a so-called finger table, i.e., a table with the IP addresses of only a few other nodes.
Overlay network

- Now we connect the nodes from the consistent hashing scheme by an overlay network called Chord running on top of the Internet.
- Each node holds a so-called finger table, i.e., a table with the IP addresses of only a few other nodes.
- We say that node $v$ has a link to node $u$ if $u$'s IP address is stored in the finger table of $v$. $^3$

$^3$For practical purposes it might be useful that nodes do not only store addresses of outgoing but also of incoming links.
Overlay network

- Now we connect the nodes from the consistent hashing scheme by an overlay network called Chord running on top of the Internet.

- Each node holds a so-called finger table, i.e., a table with the IP addresses of only a few other nodes.

- We say that node $v$ has a link to node $u$ if $u$’s IP address is stored in the finger table of $v$. \(^3\)

- The Chord network allows that devices enter and leave the system dynamically and supports the efficient search for data objects.

---

\(^3\)For practical purposes it might be useful that nodes do not only store addresses of outgoing but also of incoming links.
Definition of the Chord edges

- Let $V$ denote the set of (virtual) nodes at some point of time.
Definition of the Chord edges

- Let $V$ denote the set of (virtual) nodes at some point of time.
- The set of links (directed edges) is defined by

$$E = \{(v, \text{succ}(a(v)) + 2^{-i}) | v \in V, i \in \mathbb{N}\}.$$
Definition of the Chord edges

- Let $V$ denote the set of (virtual) nodes at some point of time.
- The set of links (directed edges) is defined by

$$E = \{ (v, \text{succ}(a(v) + 2^{-i})) | v \in V, i \in \mathbb{N} \} =: e(v,i)$$

- The parameter $i$ is called the index of the link.
Definition of the Chord edges

- Let $V$ denote the set of (virtual) nodes at some point of time.
- The set of links (directed edges) is defined by
  \[
  E = \left\{ (v, \text{succ}(a(v)) + 2^{-i}) \mid v \in V, i \in \mathbb{N} \right\} .
  \]
  \[=: e(v,i)\]
- The parameter $i$ is called the index of the link.
- Observe that the set of links is finite. For $v \in V$, let $d(v)$ denote the smallest integer such that
  \[
  \forall i \in \mathbb{N}, i \geq d(v) : e(v, i) = (v, \text{succ}(a(v))) .
  \]
Definition of the Chord edges

- Let $V$ denote the set of (virtual) nodes at some point of time.
- The set of links (directed edges) is defined by
  \[ E = \left\{ (v, \text{succ}(a(v) + 2^{-i})) \mid v \in V, i \in \mathbb{N} \right\} =: e(v, i) \]
  where $a(v)$ is the anchor of node $v$.
- The parameter $i$ is called the **index** of the link.
- Observe that the set of links is finite. For $v \in V$, let $d(v)$ denote the smallest integer such that
  \[ \forall i \in \mathbb{N}, \ i \geq d(v) : e(v, i) = (v, \text{succ}(a(v))) \]
- The outdegree of $v$ is at most $d(v)$. Let $D = \max\{d(v) \mid v \in V\}$. 

Upper-bounding the outdegree

**Theorem**

\[ D = O(\log n), \text{ w.h.p., where } n = |V|. \]

**Proof:**
- Consider any node \( v \in V \).
Theorem

\[ D = O(\log n), \text{ w.h.p., where } n = |V|. \]

Proof:

- Consider any node \( v \in V \).
- Let \( \ell(v) \) denote the length of the interval (ring segment) from \( a(v) \) to \( a(\text{succ}(a(v))) \).
Upper-bounding the outdegree

Theorem

\[ D = O(\log n), \text{ w.h.p., where } n = |V|. \]

Proof:

- Consider any node \( v \in V \).
- Let \( \ell(v) \) denote the length of the interval (ring segment) from \( a(v) \) to \( a(\text{succ}(a(v))) \).
- All edges \( e(v, i) \) with \( i \geq d(v) \) point to \( \text{succ}(a(v)) \).
Upper-bounding the outdegree

**Theorem**

\[ D = O(\log n), \text{ w.hp., where } n = |V|. \]

**Proof:**

- Consider any node \( v \in V \).
- Let \( \ell(v) \) denote the length of the interval (ring segment) from \( a(v) \) to \( a(\text{succ}(a(v))) \).
- All edges \( e(v, i) \) with \( i \geq d(v) \) point to \( \text{succ}(a(v)) \).
- In particular, it holds \( 2^{-d(v)} \leq \ell(v) \), which gives

\[
d(v) = \left\lceil \log \left( \frac{1}{\ell(v)} \right) \right\rceil.
\]
Upper-bounding the outdegree

- Fix the address of $a(v)$ on the ring arbitrarily.
Upper-bounding the outdegree

- Fix the address of $a(v)$ on the ring arbitrarily.
- For any $\beta \in [0, 1],$
  \[
  \Pr[\ell(v) \leq \beta] \leq (n - 1)\beta \leq n\beta
  \]
  as $\ell(v) \leq \beta$
Upper-bounding the outdegree

- Fix the address of $a(v)$ on the ring arbitrarily.
- For any $\beta \in [0, 1]$,
  \[
  \Pr[\ell(v) \leq \beta] \leq (n - 1)\beta \leq n\beta
  \]
  as $\ell(v) \leq \beta$
- if at least one of the other $n - 1$ nodes falls into the interval $[a(v), a(v) + \beta)$ which, for each of these nodes, happens with probability $\beta$. 
Upper-bounding the outdegree

Now let $\alpha > 0$ be chosen arbitrarily. We obtain

$$\Pr[d(v) \geq (\alpha + 3) \log n] \leq \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) \geq (\alpha + 3) \log n \right]$$

$$\leq \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) > (\alpha + 2) \log n \right]$$

$$= \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) \geq (\alpha + 2) \log n \right]$$

$$\leq \Pr \left[ \ell(v) \leq n^{-(\alpha+2)} \right]$$

$$\leq n \cdot n^{-(\alpha+2)} \leq n^{-(\alpha+1)}.$$
Upper-bounding the outdegree

Now let $\alpha > 0$ be chosen arbitrarily. We obtain

$$\Pr[d(v) \geq (\alpha + 3) \log n] \leq \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) \geq (\alpha + 3) \log n \right]$$

$$\leq \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) > (\alpha + 2) \log n \right]$$

$$= \Pr \left[ \log \left( \frac{1}{\ell(v)} \right) \geq (\alpha + 2) \log n \right]$$

$$\leq \Pr \left[ \ell(v) \leq n^{-(\alpha+2)} \right]$$

$$\leq n \cdot n^{-(\alpha+2)} \leq n^{-(\alpha+1)}.$$

Hence, the probability that there exists a node $v \in V$ for which $d(v) \geq (\alpha + 3) \log n$ is at most $n^{-\alpha}$. \qed
Routing in Chord

Suppose a node $v$ (or the device corresponding to $v$) wants to access a data object $x$. 
Routing in Chord

1. Suppose a node $v$ (or the device corresponding to $v$) wants to access a data object $x$.

2. The object can be found by applying the following routing algorithm:
Routing in Chord

- Suppose a node $v$ (or the device corresponding to $v$) wants to access a data object $x$.

- The object can be found by applying the following routing algorithm:
  - First, $v$ checks whether $\text{succ}(h(x)) = v$. If yes, then stop.
Routing in Chord

- Suppose a node \( v \) (or the device corresponding to \( v \)) wants to access a data object \( x \).

- The object can be found by applying the following routing algorithm:
  - First, \( v \) checks whether \( \text{succ}(h(x)) = v \). If yes, then stop.
  - Otherwise, \( v \) sends a message along the outgoing link with smallest index such that the link does not overlap \( h(x) \) on the ring \([0, 1)\).
Routing in Chord

Suppose a node \( v \) (or the device corresponding to \( v \)) wants to access a data object \( x \).

The object can be found by applying the following routing algorithm:

- First, \( v \) checks whether \( \text{succ}(h(x)) = v \). If yes, then stop.
- Otherwise, \( v \) sends a message along the outgoing link with smallest index such that the link does not overlap \( h(x) \) on the ring \([0, 1)\).
- The node receiving the message continues the routing in the same fashion recursively until the node holding \( x \) is found.
Routing in Chord

- Suppose a node \( v \) (or the device corresponding to \( v \)) wants to access a data object \( x \).

- The object can be found by applying the following routing algorithm:
  - First, \( v \) checks whether \( \text{succ}(h(x)) = v \). If yes, then stop.
  - Otherwise, \( v \) sends a message along the outgoing link with smallest index such that the link does not overlap \( h(x) \) on the ring \( [0, 1) \).
  - The node receiving the message continues the routing in the same fashion recursively until the node holding \( x \) is found.

- The number of hops needed for finding an object is at most \( D \) and, thus, \( O(\log n) \), w.h.p., because the index of the outgoing links is increasing with every hop on the routing path.
Definition

- One specifies a path system $\mathcal{W}$ containing a set of paths $W_{u,v}$ from $u$ to $v$
Oblivious Randomized Routing

Definition

- One specifies a path system \( \mathcal{W} \) containing a set of paths \( W_{u,v} \) from \( u \) to \( v \)
- together with a probability distribution \( D_{u,v} : W_{u,v} \rightarrow [0, 1] \),
Oblivious Randomized Routing

Definition

- One specifies a path system $\mathcal{W}$ containing a set of paths $\mathcal{W}_{u,v}$ from $u$ to $v$
- together with a probability distribution $\mathcal{D}_{u,v} : \mathcal{W}_{u,v} \to [0, 1]$,
- for every possible source-destination pair $(u, v) \in V^2$. 
Oblivious Randomized Routing

Definition

- One specifies a path system \( W \) containing a set of paths \( W_{u,v} \) from \( u \) to \( v \)
- together with a probability distribution \( D_{u,v} : W_{u,v} \rightarrow [0, 1] \),
- for every possible source-destination pair \( (u, v) \in V^2 \).
- For each packet with source \( u \) and destination \( v \) one chooses a path \( P \in W_{u,v} \)
Oblivious Randomized Routing

Definition

- One specifies a path system $\mathcal{W}$ containing a set of paths $W_{u,v}$ from $u$ to $v$ together with a probability distribution $D_{u,v} : W_{u,v} \rightarrow [0, 1]$,
- for every possible source-destination pair $(u, v) \in V^2$.
- For each packet with source $u$ and destination $v$ one chooses a path $P \in W_{u,v}$ independently at random with probability $D_{u,v}(P)$ and forwards the packet along $P$ to its destination.
Oblivious Randomized Routing

Definition

- One specifies a path system $\mathcal{W}$ containing a set of paths $W_{u,v}$ from $u$ to $v$
- together with a probability distribution $D_{u,v} : W_{u,v} \rightarrow [0, 1]$,
- for every possible source-destination pair $(u, v) \in V^2$.
- For each packet with source $u$ and destination $v$ one chooses a path $P \in W_{u,v}$
- independently at random with probability $D_{u,v}(P)$ and forwards the packet along $P$ to its destination.
Oblivious Randomized Routing

**Definition**

- One specifies a path system \( \mathcal{W} \) containing a set of paths \( W_{u,v} \) from \( u \) to \( v \)
- together with a probability distribution \( D_{u,v} : W_{u,v} \rightarrow [0, 1] \),
- for every possible source-destination pair \( (u, v) \in V^2 \).
- For each packet with source \( u \) and destination \( v \) one chooses a path \( P \in W_{u,v} \)
- independently at random with probability \( D_{u,v}(P) \) and forwards the packet along \( P \) to its destination.

**Example:**

- For any two nodes \( u, v \in V, u \neq v \), one specifies two alternative paths, that is, \( |W_{u,v}| = 2 \).
Oblivious Randomized Routing

Definition

- One specifies a path system $\mathcal{W}$ containing a set of paths $W_{u,v}$ from $u$ to $v$
- together with a probability distribution $D_{u,v} : W_{u,v} \rightarrow [0, 1]$,
- for every possible source-destination pair $(u, v) \in V^2$.
- For each packet with source $u$ and destination $v$ one chooses a path $P \in W_{u,v}$
- independently at random with probability $D_{u,v}(P)$ and forwards the packet along $P$ to its destination.

Example:

- For any two nodes $u, v \in V, u \neq v$, one specifies two alternative paths, that is, $|W_{u,v}| = 2$.
- Let $D_{u,v}$ denote the uniform distribution on $W_{u,v}$. 
Oblivious Randomized Routing

**Definition**

- One specifies a path system \( \mathcal{W} \) containing a set of paths \( W_{u,v} \) from \( u \) to \( v \) together with a probability distribution \( \mathcal{D}_{u,v} : W_{u,v} \to [0,1] \), for every possible source-destination pair \( (u,v) \in V^2 \).
- For each packet with source \( u \) and destination \( v \) one chooses a path \( P \in W_{u,v} \) independently at random with probability \( \mathcal{D}_{u,v}(P) \) and forwards the packet along \( P \) to its destination.

**Example:**

- For any two nodes \( u, v \in V, u \neq v \), one specifies two alternative paths, that is, \( |W_{u,v}| = 2 \).
- Let \( \mathcal{D}_{u,v} \) denote the uniform distribution on \( W_{u,v} \).
- When sending a packet from \( u \) to \( v \) choose \( P \in W_{u,v} \) with probability \( \mathcal{D}_{u,v}(P) = 1/2 \).
Packet Scheduling Problem and Scheduling Policies

**Definition (Packet Scheduling Problem)**

- Input: collection of paths \( P \), one for each packet.
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem)

- **Input**: collection of paths \( P \), one for each packet.

- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem)

- **Input:** collection of paths $\mathcal{P}$, one for each packet.
- **Task:** One needs to specify which packet should be forwarded along which edge in which time step.

- We will address the packet scheduling problem by describing a scheduling policy.
Packet Scheduling Problem and Scheduling Policies

**Definition (Packet Scheduling Problem)**

- **Input**: collection of paths $\mathcal{P}$, one for each packet.

- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.

- We will address the packet scheduling problem by describing a **scheduling policy**

- specifying which packet can go first and
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem)

- **Input:** collection of paths $\mathcal{P}$, one for each packet.
- **Task:** One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a scheduling policy

- specifying which packet can go first and
- which packets have to wait if two or more packets are contending for the same edge.
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem):

- **Input**: collection of paths $\mathcal{P}$, one for each packet.
- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a **scheduling policy**

- specifying which packet can go first and
- which packets have to wait if two or more packets are contending for the same edge.

Examples:

- FCFS (first-come-first-serve)
- FTG (Farthest-to-go)
- Random Rank (as defined later)
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem)

- **Input**: collection of paths $\mathcal{P}$, one for each packet.
- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a **scheduling policy**

- specifying which packet can go first and
- which packets have to wait if two or more packets are contending for the same edge.

Examples:

- **FCFS** (first-come-first-serve)
Packet Scheduling Problem and Scheduling Policies

**Definition (Packet Scheduling Problem)**

- Input: collection of paths $\mathcal{P}$, one for each packet.
- Task: One needs to specify which packet should be forwarded along which edge in which time step.

- We will address the packet scheduling problem by describing a scheduling policy
- specifying which packet can go first and
- which packets have to wait if two or more packets are contending for the same edge.

**Examples:**

- FCFS (first-come-first-serve)
- FTG (Farthest-to-go)
Packet Scheduling Problem and Scheduling Policies

**Definition (Packet Scheduling Problem)**

- **Input**: collection of paths $\mathcal{P}$, one for each packet.
- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a **scheduling policy**

- specifying which packet can go first and
- which packets have to wait if two or more packets are contending for the same edge.

**Examples:**

- **FCFS** (first-come-first-serve)
- **FTG** (Farthest-to-go)
- **Random Rank** (as defined later)
Packet Scheduling Problem and Scheduling Policies

Definition (Packet Scheduling Problem)

- **Input**: collection of paths $P$, one for each packet.
- **Task**: One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a scheduling policy specifying which packet can go first and which packets have to wait if two or more packets are contending for the same edge.

Examples:

- FCFS (first-come-first-serve)
- FTG (Farthest-to-go)
- Random Rank (as defined later)

A scheduling policy is called greedy if a packet $p$ has to wait in a step $t$ before using the next edge $e$ on its path only because there is another packet $p'$ using $e$ in this step.
Packet Scheduling Problem and Scheduling Policies

**Definition (Packet Scheduling Problem)**

- **Input:** collection of paths $\mathcal{P}$, one for each packet.
- **Task:** One needs to specify which packet should be forwarded along which edge in which time step.

We will address the packet scheduling problem by describing a **scheduling policy** specifying which packet can go first and which packets have to wait if two or more packets are contending for the same edge.

**Examples:**

- **FCFS** (first-come-first-serve)
- **FTG** (Farthest-to-go)
- Random Rank (as defined later)

A scheduling policy is called **greedy** if a packet $p$ has to wait in a step $t$ before using the next edge $e$ on its path only because there is another packet $p'$ using $e$ in this step.

We say that $p$ is delayed by $p'$ at edge $e$ in time step $t$. 
Congestion and Dilation

**Definition (Dilation)**

The dilation $D$ of a path collection $\mathcal{P}$ is the length (number of edges) on the longest path in $\mathcal{P}$.

**Definition (Congestion)**

The congestion $C$ of a path collection $\mathcal{P}$ is the maximum number of paths from $\mathcal{P}$ that share the same edge (in the same direction).
Definition (Dilation)

The dilation $D$ of a path collection $\mathcal{P}$ is the length (number of edges) on the longest path in $\mathcal{P}$.

Definition (Congestion)

The congestion $C$ of a path collection $\mathcal{P}$ is the maximum number of paths from $\mathcal{P}$ that share the same edge (in the same direction).

- In the following, we assume that every undirected edge is replaced by two edges in opposite direction.
Congestion and Dilation

Definition (Dilation)

The dilation $D$ of a path collection $\mathcal{P}$ is the length (number of edges) on the longest path in $\mathcal{P}$.

Definition (Congestion)

The congestion $C$ of a path collection $\mathcal{P}$ is the maximum number of paths from $\mathcal{P}$ that share the same edge (in the same direction).

- In the following, we assume that every undirected edge is replaced by two edges in opposite direction.
- For a (directed) edge $e \in E$, $C(e)$ denotes the number of paths from $\mathcal{P}$ that contain $e$. 
Congestion and Dilation

Definition (Dilation)
The dilation $D$ of a path collection $\mathcal{P}$ is the length (number of edges) on the longest path in $\mathcal{P}$.

Definition (Congestion)
The congestion $C$ of a path collection $\mathcal{P}$ is the maximum number of paths from $\mathcal{P}$ that share the same edge (in the same direction).

- In the following, we assume that every undirected edge is replaced by two edges in opposite direction.
- For a (directed) edge $e \in E$, $C(e)$ denotes the number of paths from $\mathcal{P}$ that contain $e$.
- The congestion is thus defined by $C = \max_{e \in E} C(e)$. 
Trivial bounds on the routing time

Observation (Lower Bound)

The routing time needed by any scheduling policy is at least
\[ \max\{C, D\} = \Omega(C + D) \] because

- there is a packet which has path length \(D\) and thus needs at least \(D\) steps to reach its destination, and
Trivial bounds on the routing time

Observation (Lower Bound)

The routing time needed by any scheduling policy is at least \( \max\{C, D\} = \Omega(C + D) \) because

- there is a packet which has path length \( D \) and thus needs at least \( D \) steps to reach its destination, and
- there is an edge that needs to forward at least \( C \) packets which takes at least \( C \) steps.
Trivial bounds on the routing time

Observation (Lower Bound)

The routing time needed by any scheduling policy is at least
\[ \max\{C, D\} = \Omega(C + D) \] because

- there is a packet which has path length \( D \) and thus needs at least \( D \) steps to reach its destination, and
- there is an edge that needs to forward at least \( C \) packets which takes at least \( C \) steps.
Trivial bounds on the routing time

**Observation (Lower Bound)**

*The routing time needed by any scheduling policy is at least*

$$\max\{C, D\} = \Omega(C + D)$$

*because*

- *there is a packet which has path length $D$ and thus needs at least $D$ steps to reach its destination, and*

- *there is an edge that needs to forward at least $C$ packets which takes at least $C$ steps.*

**Observation (Upper Bound)**

*The routing time needed by any greedy scheduling policy is at most $C \cdot D$ steps because each packet can be delayed at most for $C - 1$ steps on each edge on its routing path.*
Valiant’s trick

- We study permutation routing on the $d$-dimensional hypercube with $n = 2^d$ nodes.
Valiant’s trick

- We study permutation routing on the $d$-dimensional hypercube with $n = 2^d$ nodes.
- For each packet $p$, with source node $s_p$ and destination $d_p$ we pick a node $v_p$ independently, uniformly at random from $V$. 
Valiant’s trick

- We study permutation routing on the $d$-dimensional hypercube with $n = 2^d$ nodes.
- For each packet $p$, with source node $s_p$ and destination $d_p$ we pick a node $v_p$ independently, uniformly at random from $V$.
  - The packet is routed first along bit-fixing paths from $s_p$ to $v_p$. 
Valiant’s trick

- We study permutation routing on the $d$-dimensional hypercube with $n = 2^d$ nodes.
- For each packet $p$, with source node $s_p$ and destination $d_p$ we pick a node $v_p$ independently, uniformly at random from $V$.
  - The packet is routed first along bit-fixing paths from $s_p$ to $v_p$.
  - Then the packet is routed along bit-fixing paths from $v_p$ to $d_p$. 
We study permutation routing on the $d$-dimensional hypercube with $n = 2^d$ nodes.

For each packet $p$, with source node $s_p$ and destination $d_p$ we pick a node $v_p$ independently, uniformly at random from $V$.

- The packet is routed first along bit-fixing paths from $s_p$ to $v_p$.
- Then the packet is routed along bit-fixing paths from $v_p$ to $d_p$.

The node $v_p$ is thus used as intermediate destination.
Valiant's trick

- To simplify our analysis, we assume that Valiant's routing algorithm proceeds into two phases:
To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:

- **Phase 1**: All packets are routed from their source nodes to their intermediate destinations.
Valiant’s trick

- To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:
  - **Phase 1:** All packets are routed from their source nodes to their intermediate destinations.
  - **Phase 2:** All packets are routed from their intermediate destinations to their final destinations.
Valiant’s trick

- To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:
  - **Phase 1**: All packets are routed from their source nodes to their intermediate destinations.
  - **Phase 2**: All packets are routed from their intermediate destinations to their final destinations.

- Valiant’s trick reduces a "worst-case permutation routing problem to two random routing problems":

Valiant’s trick

- To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:
  - **Phase 1**: All packets are routed from their source nodes to their intermediate destinations.
  - **Phase 2**: All packets are routed from their intermediate destinations to their final destinations.

- Valiant’s trick reduces a "worst-case permutation routing problem to two random routing problems":
  - One with randomly picked destination nodes (phase 1) and
Valiant’s trick

To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:

- **Phase 1**: All packets are routed from their source nodes to their intermediate destinations.
- **Phase 2**: All packets are routed from their intermediate destinations to their final destinations.

Valiant’s trick reduces a "worst-case permutation routing problem to two random routing problems":

- One with randomly picked destination nodes (phase 1) and
- one with randomly picked source nodes (phase 2).
Valiant’s trick

- To simplify our analysis, we assume that Valiant’s routing algorithm proceeds into two phases:
  - **Phase 1**: All packets are routed from their source nodes to their intermediate destinations.
  - **Phase 2**: All packets are routed from their intermediate destinations to their final destinations.

- Valiant’s trick reduces a "worst-case permutation routing problem to two random routing problems":
  - One with randomly picked destination nodes (phase 1) and
  - one with randomly picked source nodes (phase 2).

- Observe that Valiant’s trick follows the paradigm of randomized oblivious routing.
Analyzing a random routing problem

- In the following, we present an analysis of phase 1.
In the following, we present an analysis of phase 1.

The same result can be shown for phase 2.
Analyzing a random routing problem

- In the following, we present an analysis of phase 1.
- The same result can be shown for phase 2.
Analyzing a random routing problem

- In the following, we present an analysis of phase 1.
- The same result can be shown for phase 2.

**Lemma**

The congestion $C$ in phase 1 (phase 2) is $O(\log n / \log \log n)$, w.h.p.
Proof of the lemma

- Let $e$ be an edge of dimension $i$, i.e., an edge that flips the $i$-th bit.
Proof of the lemma

- Let \( e \) be an edge of dimension \( i \), i.e., an edge that flips the \( i \)-th bit.
- Let \( \text{IN}(e) \) be the set of nodes from which \( e \) is reachable by a bit-fixing path. It holds \( |\text{IN}(e)| = 2^{d-i-1} \).
Proof of the lemma

- Let $e$ be an edge of dimension $i$, i.e., an edge that flips the $i$-th bit.
- Let $IN(e)$ be the set of nodes from which $e$ is reachable by a bit-fixing path. It holds $|IN(e)| = 2^{d-i-1}$.
- Let $OUT(e)$ be the set of nodes that are reachable from $e$ by a bit-fixing path. It holds $|OUT(e)| = 2^i$. 
Proof of the lemma

- Let $e$ be an edge of dimension $i$, i.e., an edge that flips the $i$-th bit.
- Let $IN(e)$ be the set of nodes from which $e$ is reachable by a bit-fixing path. It holds $|IN(e)| = 2^{d-i-1}$.
- Let $OUT(e)$ be the set of nodes that are reachable from $e$ by a bit-fixing path. It holds $|OUT(e)| = 2^i$.
- Fix any node in $IN(e)$. The path of the packet starting at $v$ contains $e$ if the packet’s intermediate destination is in $OUT(e)$. 

Proof of the lemma

- Let $e$ be an edge of dimension $i$, i.e., an edge that flips the $i$-th bit.

- Let $\text{IN}(e)$ be the set of nodes from which $e$ is reachable by a bit-fixing path. It holds $|\text{IN}(e)| = 2^{d-i-1}$.

- Let $\text{OUT}(e)$ be the set of nodes that are reachable from $e$ by a bit-fixing path. It holds $|\text{OUT}(e)| = 2^i$.

- Fix any node in $\text{IN}(e)$. The path of the packet starting at $v$ contains $e$ if the packet’s intermediate destination is in $\text{OUT}(e)$.

- As intermediate destinations are picked uniformly at random

$$\Pr[\text{v's packet traverses } e] = \frac{|\text{OUT}(e)|}{n} = \frac{2^i}{2^d} = 2^{i-d}.$$
Proof of the lemma

- For a subset $X \subseteq \text{IN}(e)$, let $A(X, e)$ denote the event that the paths of all packets starting from $X$ contain $e$. 
Proof of the lemma

- For a subset $X \subseteq \text{IN}(e)$, let $A(X, e)$ denote the event that the paths of all packets starting from $X$ contain $e$.

- Let $C(e)$ be a random variable describing the congestion at edge $e$, i.e., $C(e)$ is the number of paths containing $e$. 
Proof of the lemma

- For a subset $X \subseteq IN(e)$, let $A(X, e)$ denote the event that the paths of all packets starting from $X$ contain $e$.

- Let $C(e)$ be a random variable describing the congestion at edge $e$, i.e., $C(e)$ is the number of paths containing $e$.

- Let $k$ be any natural number.

$$\Pr[C(e) \geq k] = \Pr[\exists X \subseteq IN(e), |X| = k : A(X, e)]$$

(Union Bound)

$$\leq \sum_{X \subseteq IN(e), |X| = k} \Pr[A(X, e)]$$

$$= \sum_{X \subseteq IN(e), |X| = k} \left(2^{i-d}\right)^k$$

$$= \binom{|IN(e)|}{k} \left(2^{i-d}\right)^k.$$
Proof of lemma

- Binomial coefficients can be estimated by

\[
\left( \frac{a}{b} \right)^b \leq \binom{a}{b} \leq \left( \frac{e \cdot a}{b} \right)^b,
\]

where \( e = 2.71 \ldots \) is the Eulerian number.
Proof of lemma

- Binomial coefficients can be estimated by

\[
\left( \frac{a}{b} \right)^b \leq \binom{a}{b} \leq \left( \frac{e \cdot a}{b} \right)^b,
\]

where \( e = 2.71\ldots \) is the Eulerian number.

- This gives

\[
\Pr [C(e) \geq k] \leq \left( \frac{e|\ln(e)|}{k} \right)^k \left( 2^{i-d} \right)^k \\
= \left( \frac{e2^{d-i-1}}{k} \right)^k \left( 2^{i-d} \right)^k \\
= \left( \frac{e}{2k} \right)^k.
\]
Proof of lemma

- The congestion is defined to be $C = \max\{C(e) | e \in E\}$.

\[
\Pr[C \geq k] = \Pr[\exists e \in E : C(e) \geq k] \\
\leq \sum_{e \in E} \Pr[C(e) \geq k] \\
\leq |E| \left(\frac{e}{2k}\right)^k \leq n^2 \left(\frac{1}{2}\right)^k.
\]

The last bound follows from $|E| \leq dn \leq n^2$ and $\frac{e}{2k} \leq \frac{1}{2}$, where we assume $k \geq 3$. 
Proof of lemma

- The congestion is defined to be \( C = \max \{ C(e) | e \in E \} \).

\[
\begin{align*}
\Pr [ C \geq k ] &= \Pr [ \exists e \in E : C(e) \geq k ] \\
&\leq \sum_{e \in E} \Pr [ C(e) \geq k ] \\
&\leq |E| \left( \frac{e}{2k} \right)^k \leq n^2 \left( \frac{1}{2} \right)^k .
\end{align*}
\]

The last bound follows from \( |E| \leq dn \leq n^2 \) and \( \frac{e}{2k} \leq \frac{1}{2} \), where we assume \( k \geq 3 \).

- Now we choose \( k \) such that \( \Pr [ C \geq k ] \leq n^{-\alpha} \), for constant \( \alpha > 0 \).
Proof of lemma

- The congestion is defined to be $C = \max\{C(e)|e \in E\}$.

\[
\Pr[C \geq k] = \Pr[\exists e \in E: C(e) \geq k] \\
\leq \sum_{e \in E} \Pr[C(e) \geq k] \\
\leq |E| \left(\frac{e}{2k}\right)^k \leq n^2 \left(\frac{1}{2}\right)^k.
\]

The last bound follows from $|E| \leq dn \leq n^2$ and $\frac{e}{2k} \leq \frac{1}{2}$, where we assume $k \geq 3$.

- Now we choose $k$ such that $\Pr[C \geq k] \leq n^{-\alpha}$, for constant $\alpha > 0$.

- In particular, we set $k = \lceil (\alpha + 2) \log n \rceil \geq 3$ which gives

\[
\Pr[C \geq k] \leq n^2 2^{-(\alpha+2)\log n} \leq n^2 n^{-(\alpha+2)} = n^{-\alpha},
\]

which shows $C = O(\log n)$, w.h.p.
Proof of lemma

- We have shown $C = O(\log n)$, w.h.p., which is slightly weaker than the bound in the lemma.
Proof of lemma

- We have shown $C = O(\log n)$, w.h.p., which is slightly weaker than the bound in the lemma.
- In order to show $C = O(\log n / \log \log n)$, w.h.p., we need to choose $k$ in a more clever way.
Proof of lemma

- We have shown $C = O(\log n)$, w.h.p., which is slightly weaker than the bound in the lemma.
- In order to show $C = O(\log n / \log \log n)$, w.h.p., we need to choose $k$ in a more clever way.
- We set

$$k = \max \left\{ \frac{e}{2} \sqrt{d}, 2(\alpha + 2) \frac{d}{\log d} \right\} = O \left( \frac{\log n}{\log \log n} \right)$$
Proof of lemma

- We have shown $C = O(\log n)$, w.h.p., which is slightly weaker than the bound in the lemma.
- In order to show $C = O(\log n / \log \log n)$, w.h.p., we need to choose $k$ in a more clever way.
- We set
  
  $$k = \max \left\{ \frac{e}{2} \sqrt{d}, 2(\alpha + 2) \frac{d}{\log d} \right\} = O\left( \frac{\log n}{\log \log n} \right)$$

which gives

$$\Pr [ C \geq k ] \leq n^2 \left( \frac{e}{2k} \right)^k \leq n^2 \left( \frac{1}{\sqrt{d}} \right)^k \leq n^2 \left( \left( \frac{1}{\sqrt{d}} \right)^{\frac{2}{\log d}} \right)^{(\alpha+2)d} = n^2 \left( \frac{1}{2} \right)^{(\alpha+2)d} = n^2 \cdot n^{-(\alpha+2)} = n^{-\alpha}.$$
Congestion of $h$-relations

**Definition ($h$-to-$h$-routing problem)**

An $h$-relation is a routing problem in which every node is the source of $h$ packets and the destination of $h$ packets.

- Observe that a "1-relation" is a permutation routing problem."
Congestion of $h$-relations

**Definition ($h$-to-$h$-routing problem)**

An $h$-relation is a routing problem in which every node is the source of $h$ packets and the destination of $h$ packets.

- Observe that a "1-relation" is a permutation routing problem."
Congestion of $h$-relations

**Definition ($h$-to-$h$-routing problem)**

An $h$-relation is a routing problem in which every node is the source of $h$ packets and the destination of $h$ packets.

- Observe that a "1-relation" is a permutation routing problem.

**Lemma**

*Suppose we use Valiant’s trick for routing an arbitrary $h$-relation on the hypercube. The congestion $C$ is $O(\log n + h)$, w.h.p.*

Proof: Exercise
Scheduling on the hypercube

We study the problem of forwarding packets along prespecified paths on the $d$-dimensional hypercube.

**Theorem**

- Suppose we are given a set of packets each of which coming with a bit-fixing path along which it should be sent from its source to its destination.
Scheduling on the hypercube

We study the problem of forwarding packets along prespecified paths on the $d$-dimensional hypercube.

**Theorem**

- Suppose we are given a set of packets each of which coming with a bit-fixing path along which it should be sent from its source to its destination.
- Let $C$ denote the congestion of the paths.
Scheduling on the hypercube

We study the problem of forwarding packets along prespecified paths on the $d$-dimensional hypercube.

**Theorem**

- Suppose we are given a set of packets each of which coming with a bit-fixing path along which it should be sent from its source to its destination.
- Let $C$ denote the congestion of the paths.
Scheduling on the hypercube

We study the problem of forwarding packets along prespecified paths on the \(d\)-dimensional hypercube.

**Theorem**

- Suppose we are given a set of packets each of which coming with a bit-fixing path along which it should be sent from its source to its destination.
- Let \(C\) denote the congestion of the paths.

There is a distributed, randomized scheduling protocol that delivers all packets in time \(O(C + \log n)\), \(w.h.p.\).
Scheduling on the hypercube

We study the problem of forwarding packets along prespecified paths on the $d$-dimensional hypercube.

**Theorem**

- Suppose we are given a set of packets each of which coming with a bit-fixing path along which it should be sent from its source to its destination.
- Let $C$ denote the congestion of the paths.

There is a distributed, randomized scheduling protocol that delivers all packets in time $O(C + \log n)$, w.h.p.

Combining this result with Valiant’s trick gives:

**Corollary**

There is a distributed algorithm that routes any $h$-relation in time $O(h + \log n)$, w.h.p., on the hypercube.
Randomized scheduling policy

The random rank protocol:

- Let $R$ denote a sufficiently large integer whose value will be specified later.
Randomized scheduling policy

The random rank protocol:

- Let $R$ denote a sufficiently large integer whose value will be specified later.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$. 
Randomized scheduling policy

The random rank protocol:

- Let $R$ denote a sufficiently large integer whose value will be specified later.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Besides every packet is assumed to have a unique integer ident number (id).
Randomized scheduling policy

The random rank protocol:

- Let $R$ denote a sufficiently large integer whose value will be specified later.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Besides every packet is assumed to have a unique integer ident number (id).
- If two or more packets contend for the same edge in a step, then the one with smallest rank is forwarded and the others have to wait.
Randomized scheduling policy

The random rank protocol:

- Let $R$ denote a sufficiently large integer whose value will be specified later.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Besides every packet is assumed to have a unique integer ident number (id).
- If two or more packets contend for the same edge in a step, then the one with smallest rank is forwarded and the others have to wait.
- In case of equal ranks, packets with smaller ids are preferred.
Delay sequence analysis

Our analysis uses the following witness structure.

**Definition (delay sequence)**

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)
Delay sequence analysis

Our analysis uses the following witness structure.

Definition (delay sequence)

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)

2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;
Delay sequence analysis

Our analysis uses the following **witness** structure.

**Definition (delay sequence)**

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)

2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;

3. $s + 1$ distinct delay packets $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packet $p_{i-1}$ and packet $p_i$;
Delay sequence analysis

Our analysis uses the following witness structure.

Definition (delay sequence)

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)

2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;

3. $s + 1$ distinct delay packets $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packet $p_{i-1}$ and packet $p_i$;

4. $s + 1$ numbers $k_0, k_1, \ldots, k_s \in [R]$ with $k_0 \geq k_1 \geq \cdots \geq k_s$. 
Delay sequence analysis

Our analysis uses the following witness structure.

Definition (delay sequence)
A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)
2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;
3. $s + 1$ distinct delay packets $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packet $p_{i-1}$ and packet $p_i$;
4. $s + 1$ numbers $k_0, k_1, \ldots, k_s \in [R]$ with $k_0 \geq k_1 \geq \cdots \geq k_s$. 

Definition (active delay sequence)
$DS$ is called active if $r(p_i) = k_i$, for $0 \leq i \leq s$. 

Delay sequence analysis

Our analysis uses the following witness structure.

Definition (delay sequence)

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, $1 \leq L \leq d$, with edges of increasing dimension (like a bit-fixing path in reverse order)

2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;

3. $s + 1$ distinct delay packets $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packet $p_{i-1}$ and packet $p_i$;

4. $s + 1$ numbers $k_0, k_1, \ldots, k_s \in [R]$ with $k_0 \geq k_1 \geq \cdots \geq k_s$.

Definition (active delay sequence)

$DS$ is called active if $r(p_i) = k_i$, for $0 \leq i \leq s$. 
Delay sequence analysis

Lemma

If the random rank protocol needs $T > d$ steps, then there exists an active DS of length at least $T - d$.

Proof:

- Consider any packet packet arriving at its destination in step $T$. As $T > d$, this packet must have been delayed for at least one step. We call this packet $p_0$. 
Delay sequence analysis

Lemma

If the random rank protocol needs $T > d$ steps, then there exists an active DS of length at least $T - d$.

Proof:

- Consider any packet packet arriving at its destination in step $T$. As $T > d$, this packet must have been delayed for at least one step. We call this packet $p_0$.
- We follow the path of $p_0$ backwards from its destination until we reach an edge where it has been delayed by a packet that we call $p_1$. 
Delay sequence analysis

**Lemma**

*If the random rank protocol needs $T > d$ steps, then there exists an active DS of length at least $T - d$.***

**Proof:**

- Consider any packet packet arriving at its destination in step $T$. As $T > d$, this packet must have been delayed for at least one step. We call this packet $p_0$.
- We follow the path of $p_0$ backwards from its destination until we reach an edge where it has been delayed by a packet that we call $p_1$.
- Now we follow $p_1$ backwards through time until we reach a time step where this packet has been delayed before by another packet that we call $p_2$ (possibly at the same edge).
Lemma

If the random rank protocol needs $T > d$ steps, then there exists an active DS of length at least $T - d$.

Proof:

- Consider any packet $p_0$ arriving at its destination in step $T$. As $T > d$, this packet must have been delayed for at least one step. We call this packet $p_0$.

- We follow the path of $p_0$ backwards from its destination until we reach an edge where it has been delayed by a packet that we call $p_1$.

- Now we follow $p_1$ backwards through time until we reach a time step where this packet has been delayed before by another packet that we call $p_2$ (possibly at the same edge).

- Next we follow packet $p_2$ and so on until we reach a packet $p_s$, $s \geq 1$, that was not delayed before. We follow this packet back to its source.
Delay sequence analysis

Lemma

*If the random rank protocol needs $T > d$ steps, then there exists an active DS of length at least $T - d$.*

**Proof:**

- Consider any packet $p_0$ arriving at its destination in step $T$. As $T > d$, this packet must have been delayed for at least one step. We call this packet $p_0$.

- We follow the path of $p_0$ backwards from its destination until we reach an edge where it has been delayed by a packet that we call $p_1$.

- Now we follow $p_1$ backwards through time until we reach a time step where this packet has been delayed before by another packet that we call $p_2$ (possibly at the same edge).

- Next we follow packet $p_2$ and so on until we reach a packet $p_s$, $s \geq 1$, that was not delayed before. We follow this packet back to its source.

- Our tour backward through time covers $T$ steps and we observed $s$ delays. Let $L$ denote the number of edges on the recorded path.
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.
Delay sequence analysis

From this tour backwards through time, we can now construct an active DS as follows.

1. The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$. 
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.

1. The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
2. The packets \( p_0, \ldots, p_s \) are defined to be the delay packets. By our construction, these packets are distinct.
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.

1. The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
2. The packets \( p_0, \ldots, p_s \) are defined to be the delay packets. By our construction, these packets are distinct.
3. For \( 1 \leq i \leq s \), we choose \( \ell_i \in \{1, \ldots, L\} \) so \( e(\ell_i) \) is the edge on which \( p_{i-1} \) was delayed by \( p_i \).
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.

  1. The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
  2. The packets \( p_0, \ldots, p_s \) are defined to be the delay packets. By our construction, these packets are distinct.
  3. For \( 1 \leq i \leq s \), we choose \( \ell_i \in \{1, \ldots, L\} \) so \( e(\ell_i) \) is the edge on which \( p_{i-1} \) was delayed by \( p_i \).
  4. Observe that the path of \( p_{i-1} \) and the path of packet \( p_i \) traverse edge \( e(\ell_i) \), and \( \ell_1 \leq \ell_2 \leq \cdots \leq \ell_s \).
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.

1. The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$.
2. The packets $p_0, \ldots, p_s$ are defined to be the delay packets. By our construction, these packets are distinct.
3. For $1 \leq i \leq s$, we choose $\ell_i \in \{1, \ldots, L\}$ so $e(\ell_i)$ is the edge on which $p_{i-1}$ was delayed by $p_i$.
4. Observe that the path of $p_{i-1}$ and the path of packet $p_i$ traverse edge $e(\ell_i)$, and $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$.
5. For $0 \leq i \leq s$, we set $k_i = r(p_i)$. Observe that this gives $k_0 \geq k_1 \geq \cdots \geq k_s$ as packet $p_{i-1}$ is delayed by packet $p_i$ and the protocol prefers packets with smaller rank.
Delay sequence analysis

- From this tour backwards through time, we can now construct an active DS as follows.
  1. The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$.
  2. The packets $p_0, \ldots, p_s$ are defined to be the delay packets. By our construction, these packets are distinct.
  3. For $1 \leq i \leq s$, we choose $\ell_i \in \{1, \ldots, L\}$ so $e(\ell_i)$ is the edge on which $p_{i-1}$ was delayed by $p_i$.
  4. Observe that the path of $p_{i-1}$ and the path of packet $p_i$ traverse edge $e(\ell_i)$, and $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$.
  5. For $0 \leq i \leq s$, we set $k_i = r(p_i)$. Observe that this gives $k_0 \geq k_1 \geq \cdots \geq k_s$ as packet $p_{i-1}$ is delayed by packet $p_i$ and the protocol prefers packets with smaller rank.

- This ends the proof of the lemma.
Delay sequence analysis

Now we bound the probability that there exists an active DS. Our analysis begins with counting delay sequences.

Lemma

The number of delay sequences of length $s$ is at most

$$n^2 \cdot \binom{L - 1 + s}{s} \cdot C^{s+1} \cdot \binom{R + s}{s + 1}.$$
Delay sequence analysis

Now we bound the probability that there exists an active DS. Our analysis begins with counting delay sequences.

Lemma

The number of delay sequences of length \( s \) is at most

\[
n^2 \cdot \binom{L - 1 + s}{s} \cdot C^{s+1} \cdot \binom{R + s}{s + 1}.
\]

Proof:

1) Counting delay paths:
   The number of ways to choose a delay path is \( n(n - 1) \leq n^2 \) as this path corresponds to a bit-fixing path (in reverse order) that is determined by specifying the first and the last node on the path.
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

2) Counting the ways to choose the $l_i$’s and the $k_i$’s:
How many ways are there to choose the integers $l_1, \ldots, l_s$ such that $1 \leq l_1 \leq l_2 \leq \cdots \leq l_s \leq d$?
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L - 1 + s}{s} \cdot \binom{C^s + 1}{s + 1} \cdot \binom{R + s}{s + 1}$.

2) Counting the ways to choose the $\ell_i$'s and the $k'_i$'s:
   How many ways are there to choose the integers $\ell_1, \ldots, \ell_s$ such that $1 \leq \ell_1 \leq \ell_2 \leq \cdots \leq \ell_s \leq d$?

   These integers can be encoded into a binary string as follows

   $0^{\ell_1 - 1}10^{\ell_2 - \ell_1}10^{\ell_3 - \ell_2}1 \cdots 10^{\ell_s - \ell_{s-1}}10^{d - \ell_s}$.
Delay sequence analysis

The number of delay sequences of length \( s \) is at most \( n^2 \cdot \binom{L-1+s}{s} \cdot C^s+1 \cdot \binom{R+s}{s+1} \).

2) Counting the ways to choose the \( \ell_i \)'s and the \( k_i \)'s:
How many ways are there to choose the integers \( \ell_1, \ldots, \ell_s \) such that \( 1 \leq \ell_1 \leq \ell_2 \leq \cdots \leq \ell_s \leq d \)?

These integers can be encoded into a binary string as follows:

\[
0^{\ell_1-1}10^{\ell_2-\ell_1}10^{\ell_3-\ell_2}1\ldots10^{\ell_s-\ell_{s-1}}10^{d-\ell_s}.
\]

Observe that this string contains \( s \) ones and the number of zeros in this string is

\[
\ell_1 - 1 + \left( \sum_{i=2}^{s} (\ell_i - \ell_{i-1}) \right) + d - \ell_s = d - 1.
\]
Delay sequence analysis

The number of delay sequences of length \( s \) is at most

\[
n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}.
\]

Consequently, there is a one-to-one mapping between the \( \ell_i \)'s and the binary strings with \( d - 1 \) zeros and \( s \) ones. Hence, the number of ways to choose the \( \ell_i \)'s corresponds to the number of such strings which is

\[
\binom{d - 1 + s}{s}.
\]
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

- Consequently, there is a one-to-one mapping between the $\ell_i$’s and the binary strings with $d - 1$ zeros and $s$ ones. Hence, the number of ways to choose the $\ell_i$’s corresponds to the number of such strings which is

  $$\binom{d - 1 + s}{s}.$$

- Analogously the number of ways to choose $k_0, \ldots, k_s \in [R]$ such that $k_0 \geq k_1 \geq \cdots \geq k_s$ is equal to the number of binary strings consisting of $R - 1$ zeroes and $s + 1$ ones, which is

  $$\binom{R + s}{s + 1}.$$
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

3) Counting the ways to choose delay packets:
Now suppose that the delay path $P$ and the $\ell_i$'s are fixed.
The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

3) **Counting the ways to choose delay packets:**
Now suppose that the delay path $P$ and the $\ell_i$'s are fixed.

Then, for each delay packet, we know an edge that is contained in its path: In particular, we know that packet $p_i$, for $1 \leq i \leq s$, uses edge $e(\ell_i)$ and packet $p_0$ uses edge $e(\ell_1)$.
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

3) Counting the ways to choose delay packets:
Now suppose that the delay path $P$ and the $\ell_i$'s are fixed.

Then, for each delay packet, we know an edge that is contained in its path: In particular, we know that packet $p_i$, for $1 \leq i \leq s$, uses edge $e(\ell_i)$ and packet $p_0$ uses edge $e(\ell_1)$.

How may possibilities are there to choose a packet whose path is leading through a known edge? – At most $C$ since each edge is contained in the paths of at most $C$ packets.
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

3) **Counting the ways to choose delay packets:**
Now suppose that the delay path $P$ and the $\ell_i$'s are fixed.

Then, for each delay packet, we know an edge that is contained in its path: In particular, we know that packet $p_i$, for $1 \leq i \leq s$, uses edge $e(\ell_i)$ and packet $p_0$ uses edge $e(\ell_1)$.

How may possibilities are there to choose a packet whose path is leading through a known edge? – At most $C$ since each edge is contained in the paths of at most $C$ packets.

Hence, there are at most $C$ possibilities to choose $p_i$ and, hence, at most $C^{s+1}$ possibilities to choose all delay packets $p_0, \ldots, p_s$. 
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

3) Counting the ways to choose delay packets:
Now suppose that the delay path $P$ and the $\ell_i$'s are fixed.

Then, for each delay packet, we know an edge that is contained in its path: In particular, we know that packet $p_i$, for $1 \leq i \leq s$, uses edge $e(\ell_i)$ and packet $p_0$ uses edge $e(\ell_1)$.

How may possibilities are there to choose a packet whose path is leading through a known edge? – At most $C$ since each edge is contained in the paths of at most $C$ packets.

Hence, there are at most $C$ possibilities to choose $p_i$ and, hence, at most $C^{s+1}$ possibilities to choose all delay packets $p_0, \ldots, p_s$.

This ends the proof of the lemma.
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1}$.

**Lemma**

The probability that a given DS of length $s$ is active is $R^{-(s+1)}$.

**Proof:**

- For every delay packet $p_i$, the probability that the packet’s rank is $k_i$ is $1/R$ because ranks are chosen uniformly at random from $[R]$. 
Delay sequence analysis

The number of delay sequences of length \( s \) is at most \( n^2 \cdot \binom{L-1+s}{s} \cdot C^{s+1} \cdot \binom{R+s}{s+1} \).

**Lemma**

*The probability that a given DS of length \( s \) is active is \( R^{-(s+1)} \).*

**Proof:**

- For every delay packet \( p_i \), the probability that the packet’s rank is \( k_i \) is \( 1/R \) because ranks are chosen uniformly at random from \([R]\).

- Thus, the probability that all \( s + 1 \) delay packets have the prescribed rank is \( R^{-(s+1)} \) because the ranks of different packets are chosen independently.
Delay sequence analysis

The number of delay sequences of length $s$ is at most $n^2 \cdot \binom{L-1+s}{s} \cdot C_s^{s+1} \cdot \binom{R+s}{s+1}$.

Lemma

The probability that a given DS of length $s$ is active is $R^{-(s+1)}$.

Proof:

- For every delay packet $p_i$, the probability that the packet’s rank is $k_i$ is $1/R$ because ranks are chosen uniformly at random from $[R]$.

- Thus, the probability that all $s+1$ delay packets have the prescribed rank is $R^{-(s+1)}$ because the ranks of different packets are chosen independently.

- This ends the proof of the lemma.
By the first Lemma, if the algorithm needs $T \geq d + s$ steps, then there exists an active delay sequence of length at least $s$. 
Delay sequence analysis

- By the first Lemma, if the algorithm needs $T \geq d + s$ steps, then there exists an active delay sequence of length at least $s$.

- Cutting this sequence after packet $p_s$ gives an active delay sequence of length exactly $s$. 
Delay sequence analysis

By the first Lemma, if the algorithm needs $T \geq d + s$ steps, then there exists an active delay sequence of length at least $s$.

Cutting this sequence after packet $p_s$ gives an active delay sequence of length exactly $s$.

Let $\mathcal{DS}(s)$ denote the set of delay sequences of length $s$. It holds

$$\Pr[T \geq d + s] \leq \Pr[\exists DS \in \mathcal{DS}(s) : DS \text{ is active}] \leq \sum_{DS \in \mathcal{DS}(s)} \Pr[DS \text{ is active}] \quad \text{(third Lemma)}$$

$$\leq \sum_{DS \in \mathcal{DS}(s)} R^{-(s+1)} \quad \text{(second Lemma)}$$

$$\leq n^2 \cdot \binom{d - 1 + s}{s} \cdot C^{s+1} \cdot \binom{R + s}{s + 1} \cdot R^{-(s+1)}.$$
Delay sequence analysis

- We have so far:

\[
\Pr [T \geq d + s] \leq n^2 \cdot \binom{d - 1 + s}{s} \cdot C^{s+1} \cdot \binom{R + s}{s + 1} \cdot R^{-(s+1)}
\]
Delay sequence analysis

- We have so far:

\[
\Pr[T \geq d + s] \leq n^2 \cdot \left(\frac{d - 1 + s}{s}\right) \cdot C^{s+1} \cdot \left(\frac{R + s}{s + 1}\right) \cdot R^{-(s+1)}
\]

- Using \(\binom{a}{b} \leq 2^a\) and \(\binom{a}{b} \leq \left(\frac{ea}{b}\right)^b\) to upper-bound the binomial coefficients, we obtain

\[
\Pr[T \geq d + s] \leq n^2 \cdot 2^{d-1+s} \cdot C^{s+1} \cdot \left(\frac{e(R + s)}{s + 1}\right)^{s+1} \cdot R^{-(s+1)}
\]

\[
\leq n^3 \cdot \left(\frac{2eC(R + s)}{(s + 1)R}\right)^{s+1}
\]
Delay sequence analysis

We have so far:

\[
\Pr\left[ T \geq d + s \right] \leq n^2 \cdot \left( \frac{d - 1 + s}{s} \right) \cdot C^{s+1} \cdot \left( \frac{R + s}{s + 1} \right) \cdot R^{-(s+1)}
\]

Using \( \binom{a}{b} \leq 2^a \) and \( \binom{a}{b} \leq \left( \frac{e^a}{b} \right)^b \) to upper-bound the binomial coefficients, we obtain

\[
\Pr\left[ T \geq d + s \right] \leq n^2 \cdot 2^{d-1+s} \cdot C^{s+1} \cdot \left( \frac{e(R + s)}{s + 1} \right)^{s+1} \cdot R^{-(s+1)}
\]

\[
\leq n^3 \cdot \left( \frac{2eC(R + s)}{(s + 1)R} \right)^{s+1}
\]

Choosing \( R \geq s \) yields \( R + s \leq 2R \) and, hence,

\[
\Pr\left[ T \geq d + s \right] \leq n^3 \cdot \left( \frac{4eC}{s + 1} \right)^{s+1}
\]
Delay sequence analysis

We have so far:

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{4eC}{s + 1} \right)^{s+1} \]
We have so far:

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{4eC}{s + 1} \right)^{s+1} \]

Let us choose \( s = \lceil \max\{8eC, (\alpha + 3) \log n\} \rceil - 1 = O(C + \log n) \).
Delay sequence analysis

- We have so far:

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{4eC}{s + 1} \right)^{s+1} \]

- Let us choose \( s = \lceil \max\{8eC, (\alpha + 3) \log n\} \rceil - 1 = O(C + \log n). \)

- This gives

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{1}{2} \right)^{s+1} \leq n^3 \cdot \left( \frac{1}{2} \right)^{(\alpha+3) \log n} \leq n^{-\alpha}. \]
Delay sequence analysis

- We have so far:

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{4eC}{s + 1} \right)^{s+1} \]

- Let us choose \( s = \lceil \max\{8eC, (\alpha + 3) \log n\} \rceil - 1 = O(C + \log n) \).

- This gives

\[ \Pr[T \geq d + s] \leq n^3 \cdot \left( \frac{1}{2} \right)^{s+1} \leq n^3 \cdot \left( \frac{1}{2} \right)^{(\alpha+3) \log n} \leq n^{-\alpha}. \]

- Hence, with probability at least \( 1 - n^{-\alpha} \), the random rank protocol needs at most \( d + s - 1 = O(C + \log n) \) steps.
We have so far:

$$\Pr[T \geq d + s] \leq n^3 \cdot \left(\frac{4eC}{s+1}\right)^{s+1}$$

Let us choose $s = \lceil \max\{8eC, (\alpha + 3) \log n\} \rceil - 1 = O(C + \log n)$.

This gives

$$\Pr[T \geq d + s] \leq n^3 \cdot \left(\frac{1}{2}\right)^{s+1} \leq n^3 \cdot \left(\frac{1}{2}\right)^{(\alpha+3)\log n} \leq n^{-\alpha}.$$  

Hence, with probability at least $1 - n^{-\alpha}$, the random rank protocol needs at most $d + s - 1 = O(C + \log n)$ steps.

This end the proof of the theorem.
Scheduling along shortest paths in general networks

Consider any $n$-node network $G = (V, E)$.\(^4\) We study the problem of forwarding packets along arbitrary shortest paths in $G$.

\begin{shaded}
\begin{theorem}
Suppose we are given a set of $N \geq n$ packets each of which coming with a shortest path in $G$ along which it should be sent from its source to its destination.

Let $C$ and $D$ denote the congestion and the dilation of the paths, respectively. There is a distributed, randomized scheduling protocol that delivers all packets in time $O(C + D + \log N)$, w.h.p.
\end{theorem}
\end{shaded}

\(^4\)Recall that edges are assumed to be directed. In order to represent an undirected network, one replaces each edge by two directed edges in opposite direction.
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let $R$ denote a sufficiently large integer being a multiple of $D$. 
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let \( R \) denote a sufficiently large integer being a multiple of \( D \).
- Every packet \( p \) is assigned independently and uniformly at random a rank \( r(p) \in [R] \).
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let $R$ denote a sufficiently large integer being a multiple of $D$.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Whenever the packet moves along an edge, its rank is increased by the value $R/D$. 
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let $R$ denote a sufficiently large integer being a multiple of $D$.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Whenever the packet moves along an edge, its rank is increased by the value $R/D$.
- Besides every packet is assumed to have a unique integer ident number (id).
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let $R$ denote a sufficiently large integer being a multiple of $D$.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Whenever the packet moves along an edge, its rank is increased by the value $R/D$.
- Besides every packet is assumed to have a unique integer ident number (id).
- If two or more packets contend for the same edge in a step, then the one with smallest rank is forwarded and the others have to wait.
Randomized scheduling policy with increasing ranks

The growing rank protocol:

- Let $R$ denote a sufficiently large integer being a multiple of $D$.
- Every packet $p$ is assigned independently and uniformly at random a rank $r(p) \in [R]$.
- Whenever the packet moves along an edge, its rank is increased by the value $R/D$.
- Besides every packet is assumed to have a unique integer ident number (id).
- If two or more packets contend for the same edge in a step, then the one with smallest rank is forwarded and the others have to wait.
- In case of equal ranks, packets with smaller ids are preferred.
Randomized scheduling policy

**Observation**

As the initial rank is at most $R - 1$ and the rank of a packet is increased at most $D$ times by $R/D$, the final rank of a packet is at most $2R - 1$.

Let $r_e(p) \in [2R]$ denote the rank of packet $p$ in those time steps in which $p$ contends for being forwarded along edge $e$. 
Delay sequence analysis

We adapt the definition of a delay sequence as follows.

**Definition (delay sequence)**

A delay sequence $DS$ of length $s$ consists of

1. a delay path $P = (e(1), \ldots, e(L))$, for $L \leq 2D$, with edges in reverse direction, that is, $(e(L), \ldots, e(1))$ is a path in $G$;

2. $s$ numbers $\ell_1, \ldots, \ell_s \in \{1, \ldots, L\}$ with $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_s$;

3. $s + 1$ distinct delay packets $p_0, p_1, \ldots, p_s$ such that, for $1 \leq i \leq s$, edge $e(\ell_i)$ is contained in the paths of packet $p_{i-1}$ and packet $p_i$;

4. $s + 1$ numbers $k_0, k_1, \ldots, k_s \in [2R]$ with $k_0 \geq k_1 \geq \cdots \geq k_s$.

**Definition (active delay sequence)**

$DS$ is active if $r_{e(\ell_i)}(p_i) = k_i$, for $1 \leq i \leq s$, and $r_{e(\ell_1)}(p_0) = k_0$. 
Delay sequence analysis

Lemma
If the growing rank protocol needs $T \geq 2D$ steps, then there exists an active DS of length at least $T - 2D$.

Proof:
Consider any packet packet arriving at its destination in step $T$. As $T \geq 2D$, this packet must have been delayed for at least one step. We call this packet $p_0$. We follow the path of $p_0$ backwards through time from its destination until we reach an edge where it has been delayed by a packet that we call $p_1$. Now we follow $p_1$ backwards through time until we reach a time step where this packet has been delayed before by another packet that we call $p_2$, and so on ... ... until we reach a packet $p_s$, for some $s \geq 1$, that was not delayed before. We follow this packet back to its source.
Delay sequence analysis

From this tour backwards through time, we can now construct an active $DS$ as follows.

- The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$.

\footnote{This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in $G$.}
Delay sequence analysis

From this tour backwards through time, we can now construct an active $DS$ as follows.

- The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$.
- The packets $p_0, \ldots, p_s$ are defined to be the delay packets.

---

5This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in $G$. 
Delay sequence analysis

From this tour backwards through time, we can now construct an active DS as follows.

- The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
- The packets \( p_0, \ldots, p_s \) are defined to be the delay packets.
- For \( 1 \leq i \leq s \), we choose \( \ell_i \in \{1, \ldots, L\} \) so that \( e(\ell_i) \) is the edge on which \( p_{i-1} \) was delayed by \( p_i \).

This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in \( G \).
Delay sequence analysis

From this tour backwards through time, we can now construct an active DS as follows.

- The path that we have recorded by this process in reverse order gives us the delay path $P = (e(1), \ldots, e(L))$.
- The packets $p_0, \ldots, p_s$ are defined to be the delay packets.
- For $1 \leq i \leq s$, we choose $\ell_i \in \{1, \ldots, L\}$ so that $e(\ell_i)$ is the edge on which $p_{i-1}$ was delayed by $p_i$.
- For $1 \leq i \leq s$, we set $k_i = r_{e(\ell_i)}(p_i)$, and $k_0 = r_{e(\ell_1)}(p_0)$.

---

5This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in $G$. 

Delay sequence analysis

From this tour backwards through time, we can now construct an active DS as follows.

- The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
- The packets \( p_0, \ldots, p_s \) are defined to be the delay packets.
- For \( 1 \leq i \leq s \), we choose \( \ell_i \in \{1, \ldots, L\} \) so that \( e(\ell_i) \) is the edge on which \( p_{i-1} \) was delayed by \( p_i \).
- For \( 1 \leq i \leq s \), we set \( k_i = r_{e(\ell_i)}(p_i) \), and \( k_0 = r_{e(\ell_1)}(p_0) \).

\(^5\)This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in \( G \).
Delay sequence analysis

From this tour backwards through time, we can now construct an active DS as follows.

- The path that we have recorded by this process in reverse order gives us the delay path \( P = (e(1), \ldots, e(L)) \).
- The packets \( p_0, \ldots, p_s \) are defined to be the delay packets.
- For \( 1 \leq i \leq s \), we choose \( \ell_i \in \{1, \ldots, L\} \) so that \( e(\ell_i) \) is the edge on which \( p_{i-1} \) was delayed by \( p_i \).
- For \( 1 \leq i \leq s \), we set \( k_i = r_{e(\ell_i)}(p_i) \), and \( k_0 = r_{e(\ell_1)}(p_0) \).

**Exercise:** Show that the packets \( p_0, \ldots, p_s \) are distinct, that is, no packet appears more than once in the delay sequence.\(^5\)

---

\(^5\)This is the only part of the analysis where we need to assume that the paths of the packets are shortest paths in \( G \).
Delay sequence analysis

Observe that $k_0 \geq k_1 \geq \cdots \geq k_s$ as the ranks of the delay packets do not increase on our tour. More specifically:

- whenever we switch from packet $p_i$ to packet $p_{i+1}$ on our tour, the rank of $p_{i+1}$ is not larger than the rank of packet $p_i$ because $p_{i+1}$ delays $p_i$ and the protocol prefers packets with smaller rank, and
Delay sequence analysis

Observe that $k_0 \geq k_1 \geq \cdots \geq k_s$ as the ranks of the delay packets do not increase on our tour. More specifically:

- whenever we switch from packet $p_i$ to packet $p_{i+1}$ on our tour, the rank of $p_{i+1}$ is not larger than the rank of packet $p_i$ because $p_{i+1}$ delays $p_i$ and the protocol prefers packets with smaller rank, and

- whenever we add an edge to the delay path and follow this edge, the rank of the currently observed packet is decreased (by $R/d$) as we follow the packet backwards in time.
Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$. 
Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
- In particular, whenever we add an edge to the delay path, the rank of the packet that we follow is decreased by $R/D$. 


Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
- In particular, whenever we add an edge to the delay path, the rank of the packet that we follow is decreased by $R/D$.
- Hence, the rank of packet $p_s$ at its source is at most $2R - 1 - L(R/D)$. 

Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
- In particular, whenever we add an edge to the delay path, the rank of the packet that we follow is decreased by $R/D$.
- Hence, the rank of packet $p_s$ at its source is at most $2R - 1 - L(R/D)$.
- As ranks are non-negative, we obtain $(2R - 1) - L(R/D) \geq 0$ which gives $L \leq (2R - 1)/(R/D) \leq 2D$. 
Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
- In particular, whenever we add an edge to the delay path, the rank of the packet that we follow is decreased by $R/D$.
- Hence, the rank of packet $p_s$ at its source is at most $2R - 1 - L(R/D)$.
- As ranks are non-negative, we obtain $(2R - 1) - L(R/D) \geq 0$ which gives $L \leq (2R - 1)/(R/D) \leq 2D$.
- Finally, $T = L + s$ implies $s = T - L \geq T - 2D$. 
Delay sequence analysis

It only remains to prove $L \leq 2D$ and $s \geq T - 2D$.

- The final rank of $p_0$ is at most $2R - 1$.
- During our tour backwards through time, the sequence of observed ranks is not increasing.
- In particular, whenever we add an edge to the delay path, the rank of the packet that we follow is decreased by $R/D$.
- Hence, the rank of packet $p_s$ at its source is at most $2R - 1 - L(R/D)$.
- As ranks are non-negative, we obtain $(2R - 1) - L(R/D) \geq 0$ which gives $L \leq (2R - 1)/(R/D) \leq 2D$.
- Finally, $T = L + s$ implies $s = T - L \geq T - 2D$.
- This ends the proof of the lemma.
Delay sequence analysis

Now we bound the probability that there exists an active DS. Our analysis begins with counting delay sequences.

Lemma

*The number of delay sequences of length s is at most*

\[
\binom{2D - 1 + s}{s} \binom{2R + s}{s + 1} N C^s.
\]
Delay sequence analysis

Now we bound the probability that there exists an active DS. Our analysis begins with counting delay sequences.

**Lemma**

The number of delay sequences of length $s$ is at most

$$\binom{2D-1+s}{s} \binom{2R+s}{s+1} N^s C^s.$$

**Proof:**

Analogously to the analysis for the hypercube the number of ways to choose the $\ell_i$’s and the $k_i'$ can be bounded by

$$\binom{2D-1+s}{s} \binom{2R+s}{s+1}.$$. 
Delay sequence analysis

Now we assume that the $\ell_i$’s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$. 
Delay sequence analysis

Now we assume that the $\ell_i$’s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$.
- Once $p_0$ is fixed, we can construct the first part of the delay path from edge $e(1)$ up edge $e(\ell_1)$ by following the path of $p_0$ backwards from its destination.
Delay sequence analysis

Now we assume that the $\ell_i$'s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$.

- Once $p_0$ is fixed, we can construct the first part of the delay path from edge $e(1)$ up edge $e(\ell_1)$ by following the path of $p_0$ backwards from its destination.

- Now, as the path of $p_1$ contains $e(\ell_1)$, there are at most $C$ possibilities to choose $p_1$. 
Delay sequence analysis

Now we assume that the $\ell_i$’s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$.
- Once $p_0$ is fixed, we can construct the first part of the delay path from edge $e(1)$ up edge $e(\ell_1)$ by following the path of $p_0$ backwards from its destination.
- Now, as the path of $p_1$ contains $e(\ell_1)$, there are at most $C$ possibilities to choose $p_1$.
- Once $p_1$ is fixed, we can determine the delay path up to $e(\ell_2)$. 
Delay sequence analysis

Now we assume that the $\ell_i$’s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$.

- Once $p_0$ is fixed, we can construct the first part of the delay path from edge $e(1)$ up edge $e(\ell_1)$ by following the path of $p_0$ backwards from its destination.

- Now, as the path of $p_1$ contains $e(\ell_1)$, there are at most $C$ possibilities to choose $p_1$.

- Once $p_1$ is fixed, we can determine the delay path up to $e(\ell_2)$.

- As the path of $p_2$ contains $e(\ell_2)$, there are again at most $C$ possibilities to choose $p_2$, and so on until packet $p_s$. 

Delay sequence analysis

Now we assume that the \( \ell_i \)'s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are \( N \) possibilities to choose packet \( p_0 \).
- Once \( p_0 \) is fixed, we can construct the first part of the delay path from edge \( e(1) \) up edge \( e(\ell_1) \) by following the path of \( p_0 \) backwards from its destination.
- Now, as the path of \( p_1 \) contains \( e(\ell_1) \), there are at most \( C \) possibilities to choose \( p_1 \).
- Once \( p_1 \) is fixed, we can determine the delay path up to \( e(\ell_2) \).
- As the path of \( p_2 \) contains \( e(\ell_2) \), there are again at most \( C \) possibilities to choose \( p_2 \), and so on until packet \( p_s \).
- Thus, the number of possibilities to choose the delay packets and to construct the delay path is at most \( NC^s \).
Delay sequence analysis

Now we assume that the $\ell_i$’s are fixed and we count the number of ways to choose the delay packets and the edges on the delay path:

- There are $N$ possibilities to choose packet $p_0$.
- Once $p_0$ is fixed, we can construct the first part of the delay path from edge $e(1)$ up edge $e(\ell_1)$ by following the path of $p_0$ backwards from its destination.
- Now, as the path of $p_1$ contains $e(\ell_1)$, there are at most $C$ possibilities to choose $p_1$.
- Once $p_1$ is fixed, we can determine the delay path up to $e(\ell_2)$.
- As the path of $p_2$ contains $e(\ell_2)$, there are again at most $C$ possibilities to choose $p_2$, and so on until packet $p_s$.
- Thus, the number of possibilities to choose the delay packets and to construct the delay path is at most $NC^s$.
- This ends the proof of the lemma.
Lemma

The probability that a DS of length $s$ is active is at most $R^{-(s+1)}$.

Proof:

- Suppose $e(\ell_i)$ is the $j$th edge on the path of packet $p_i$. 

Delay sequence analysis

Lemma

The probability that a DS of length \( s \) is active is at most \( R^{-(s+1)} \).

Proof:

- Suppose \( e(\ell_i) \) is the \( j \)th edge on the path of packet \( p_i \).

- The rank at edge \( e(\ell_i) \) is equal to \( k_i \) if its initial rank is equal to \( k'_i = k_i - (j - 1) \cdot R/D \), which happens with probability \( 1/R \) if \( k'_i \in [R] \), and probability 0, otherwise.
Delay sequence analysis

Lemma

The probability that a DS of length $s$ is active is at most $R^{-(s+1)}$.

Proof:

- Suppose $e(\ell_i)$ is the $j$th edge on the path of packet $p_i$.
- The rank at edge $e(\ell_i)$ is equal to $k_i$ if its initial rank is equal to $k'_i = k_i - (j - 1) \cdot R/D$, which happens with probability $1/R$ if $k'_i \in [R]$, and probability 0, otherwise.
- That is, the probability that the rank of $p_i$ at edge $e(\ell_i)$ is equal to $k_i$ is at most $1/R$. 
Delay sequence analysis

**Lemma**

*The probability that a DS of length s is active is at most $R^{-(s+1)}$.***

**Proof:**

- Suppose $e(\ell_i)$ is the $j$th edge on the path of packet $p_i$.
- The rank at edge $e(\ell_i)$ is equal to $k_i$ if its initial rank is equal to $k'_i = k_i - (j - 1) \cdot R/D$, which happens with probability $1/R$ if $k'_i \in [R]$, and probability 0, otherwise.
- That is, the probability that the rank of $p_i$ at edge $e(\ell_i)$ is equal to $k_i$ is at most $1/R$.
- Consequently, the probability that all $s + 1$ delay packets have the prescribed rank is at most $R^{-(s+1)}$. 
Delay sequence analysis

Lemma

The probability that a DS of length $s$ is active is at most $R^{-(s+1)}$.

Proof:

- Suppose $e(\ell_i)$ is the $j$th edge on the path of packet $p_i$.
- The rank at edge $e(\ell_i)$ is equal to $k_i$ if its initial rank is equal to $k'_i = k_i - (j-1) \cdot R/D$, which happens with probability $1/R$ if $k'_i \in [R]$, and probability 0, otherwise.
- That is, the probability that the rank of $p_i$ at edge $e(\ell_i)$ is equal to $k_i$ is at most $1/R$.
- Consequently, the probability that all $s + 1$ delay packets have the prescribed rank is at most $R^{-(s+1)}$.
- This ends the proof of the lemma.
Delay sequence analysis

Now we proceed analogously to the analysis for the hypercube.

\[
\Pr[T \geq 2D + s] \leq \Pr[\exists DS \in DS(s) : DS \text{ is active}]
\]
\[
\leq \sum_{DS \in DS(s)} \Pr[DS \text{ is active}]
\]
\[
\leq \left(\frac{2D - 1 + s}{s}\right) \left(\frac{2R + s}{s + 1}\right)^{N C^s R^{-(s+1)}}
\]
\[
\leq 2^{2D-1+s} \left(\frac{e(2R + s)}{s + 1}\right)^{s+1} N C^s R^{-(s+1)}
\]
\[
\leq 2^{2D} \left(\frac{6Ce}{s + 1}\right)^{s+1} N ,
\]

where the last equation assumes \( R \geq s \).
Delay sequence analysis

Finally, we set $s = \lceil \max\{12eC, (\alpha + 1) \log N + 2D\} \rceil - 1 = O(C + D + \log N)$. This gives

$$\Pr[T \geq 2D + s] \leq 2^{2D} N \left(\frac{1}{2}\right)^{s+1} \leq 2^{2D} N \left(\frac{1}{2}\right)^{\alpha + 1} \log N + 2D \leq N^{-\alpha} \leq n^{-\alpha}$$

using $n \leq N$.

Hence, with probability at least $1 - n^{-\alpha}$, the growing rank protocol needs at most $2D + s - 1 = O(C + D + \log N)$ steps.
Literature