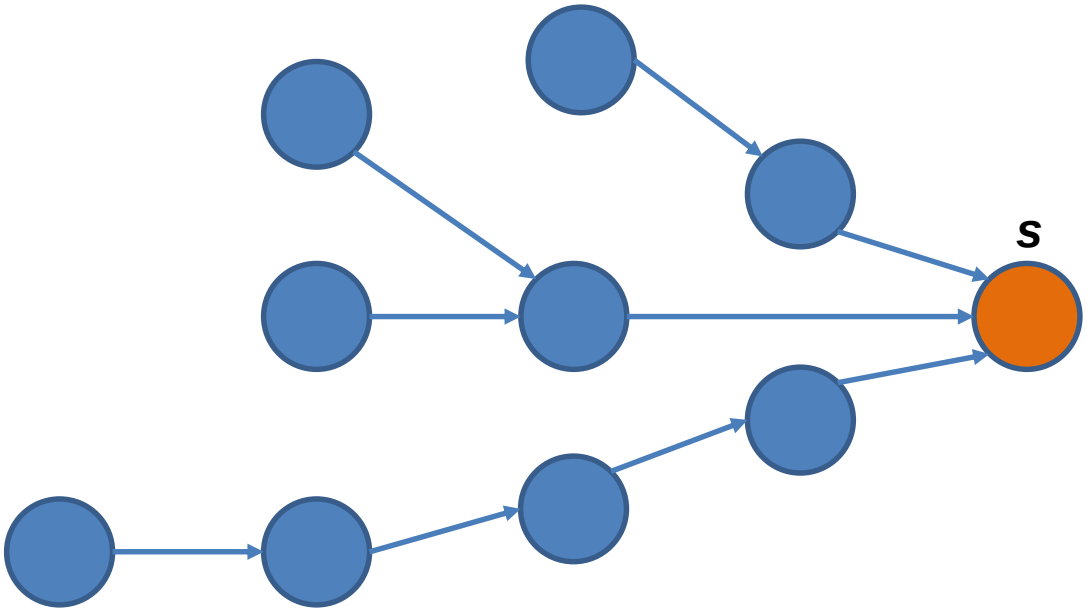
Optimal Local Buffer Management for Information Gathering with Adversarial Traffic

Stefan Dobrev, Slovak Academy of Sciences, Slovakia
Manuel Lafond, University of Ottawa, Canada
Lata Narayanan, Concordia University, Canada
Jaroslav Opatrny, Concordia University, Canada

Information gathering with adversary

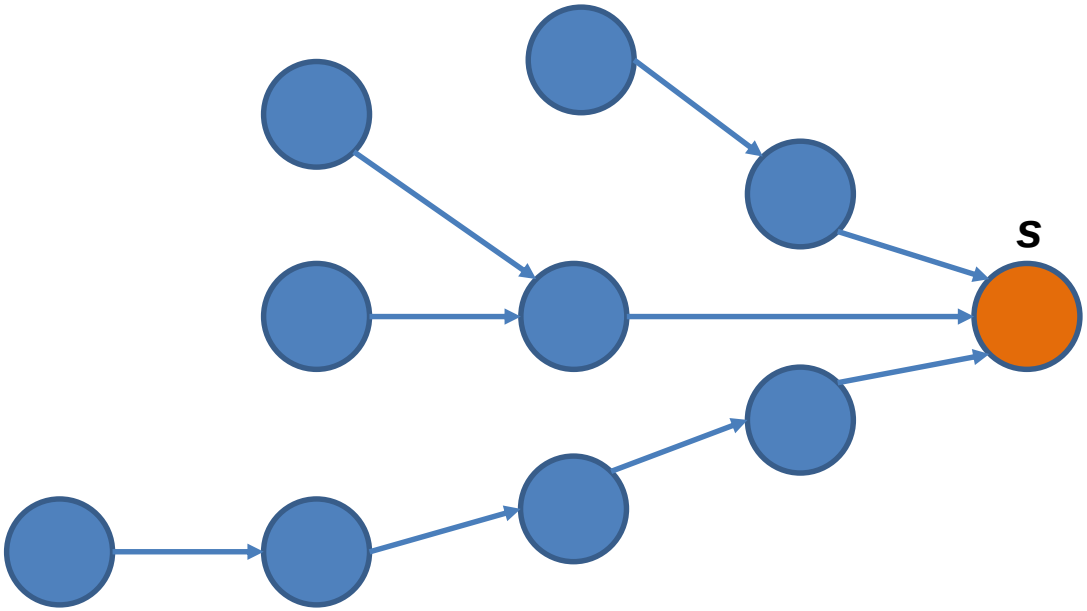
- Network has a special node ***s*** called the *sink.*
- Packets enter the network at discrete time steps.
- Each packet generated by the network is destined for the sink.
- Our networks are all trees directed towards ***s***.



Information gathering with adversary

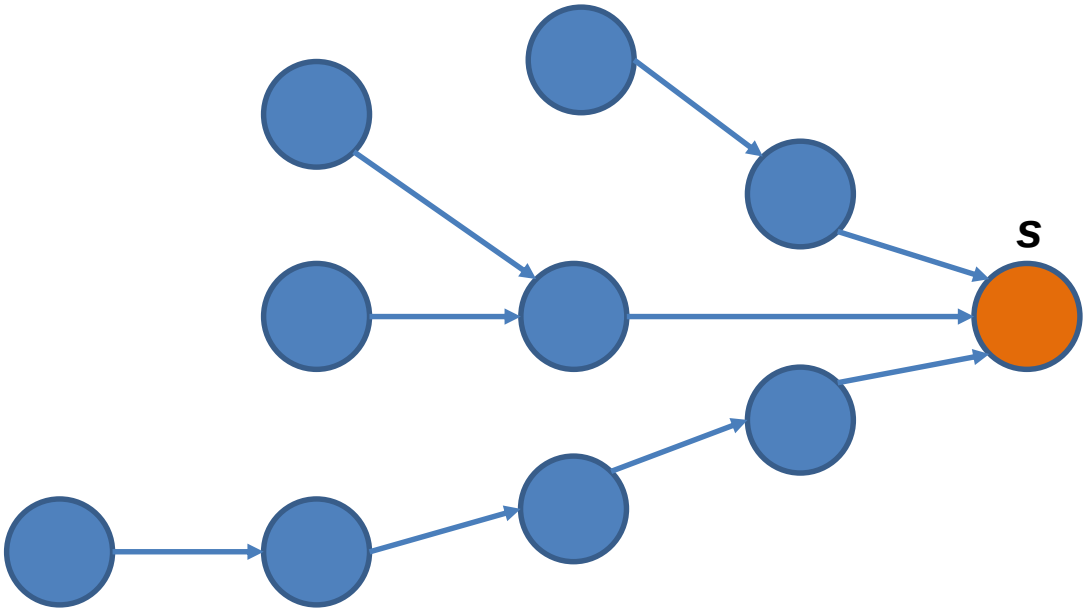
- Each arc has capacity c.
- Adversary can inject packets at a rate of c.

  Goal: fill up node buffers as much as possible.
- **Locality constraints**: each node can only see the state of the nodes at (undirected) distance at most ℓ.



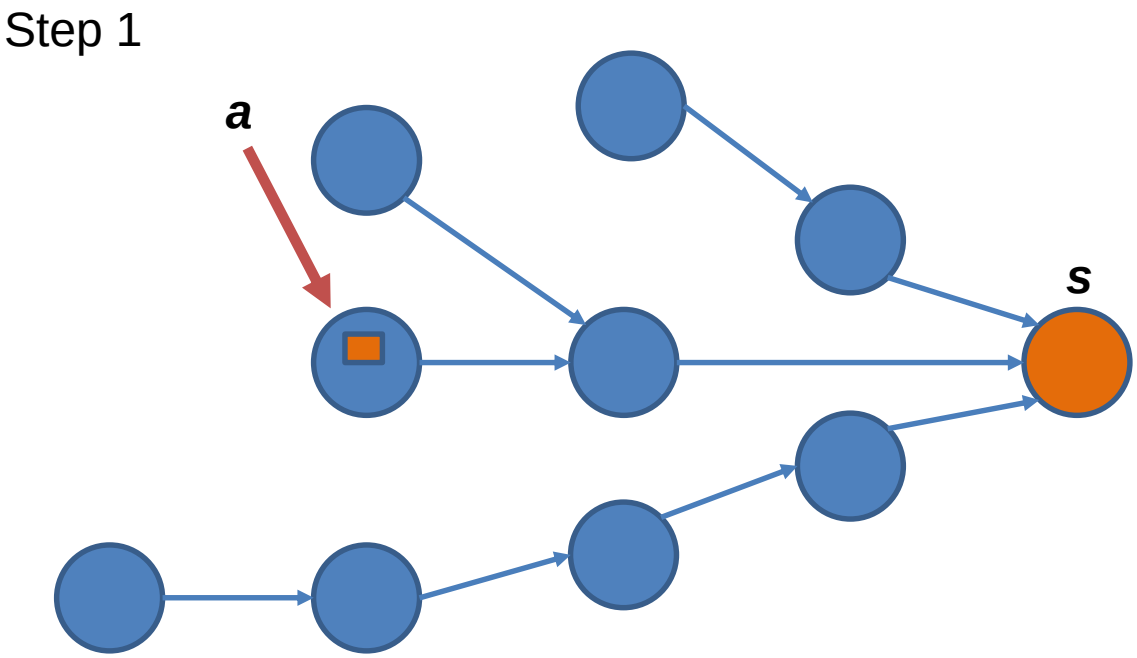
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.



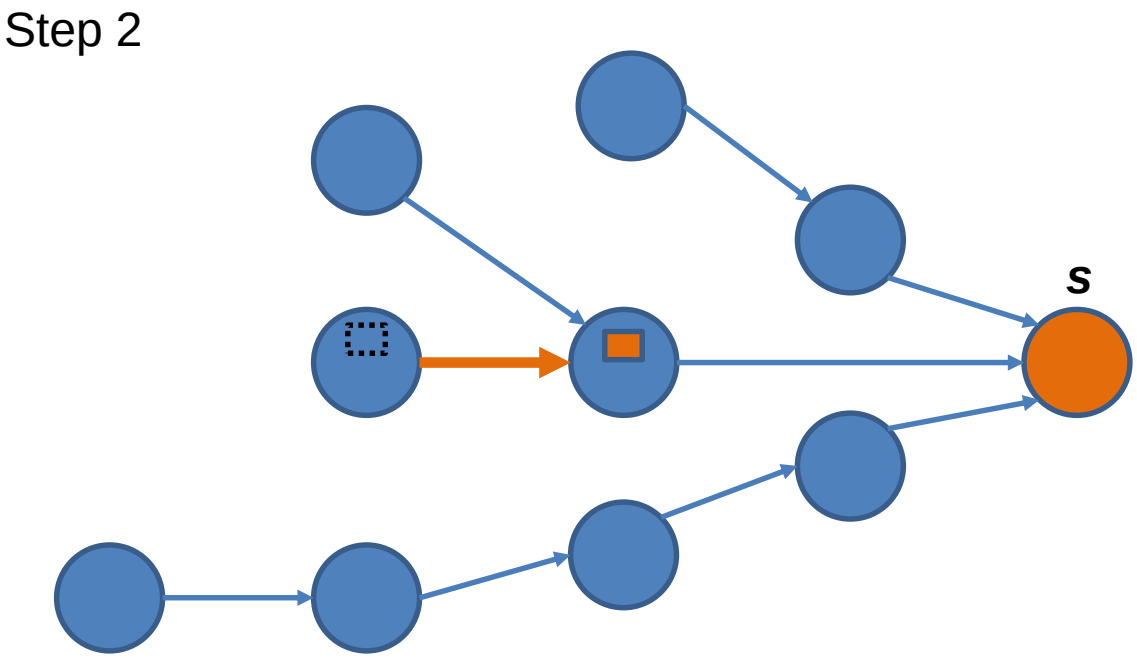
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



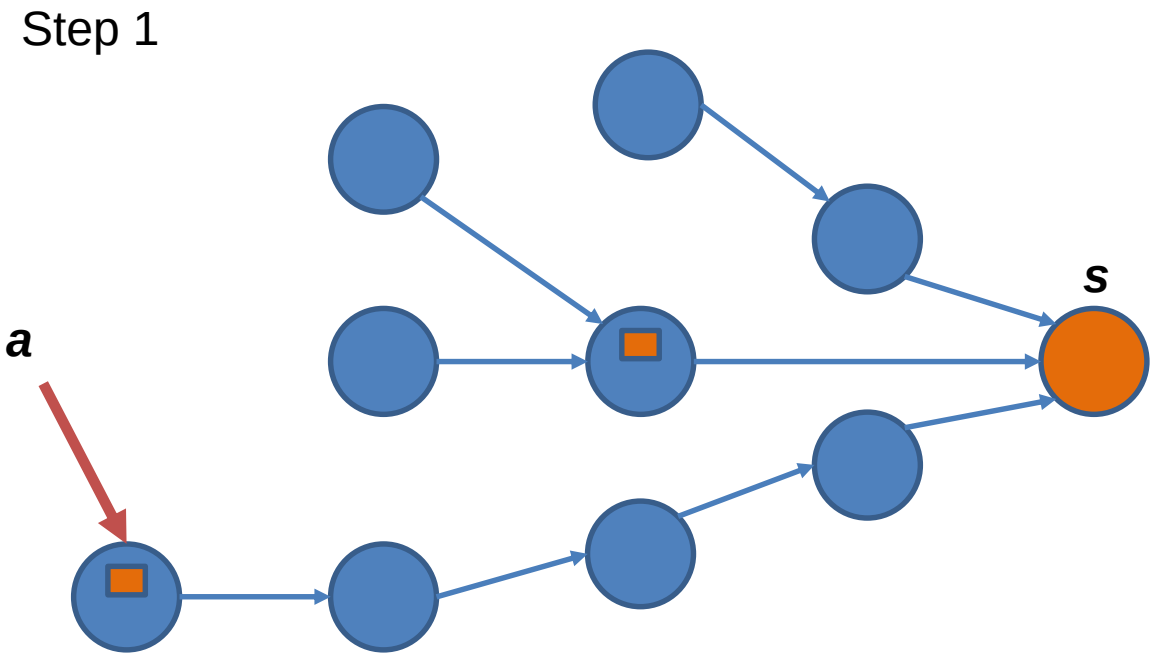
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



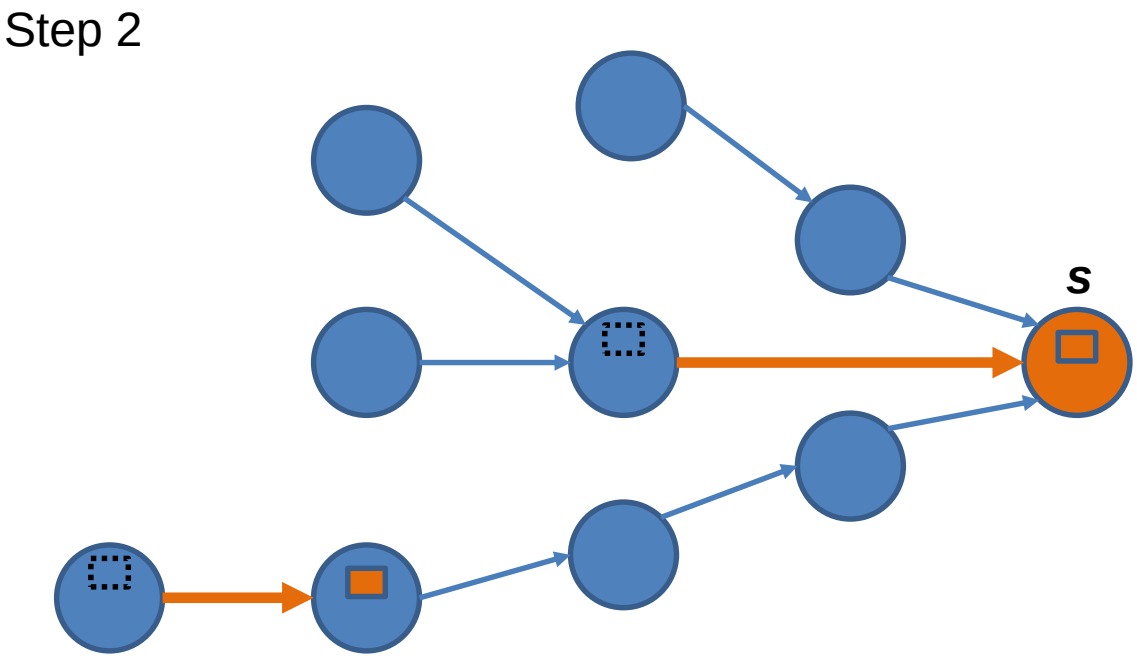
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



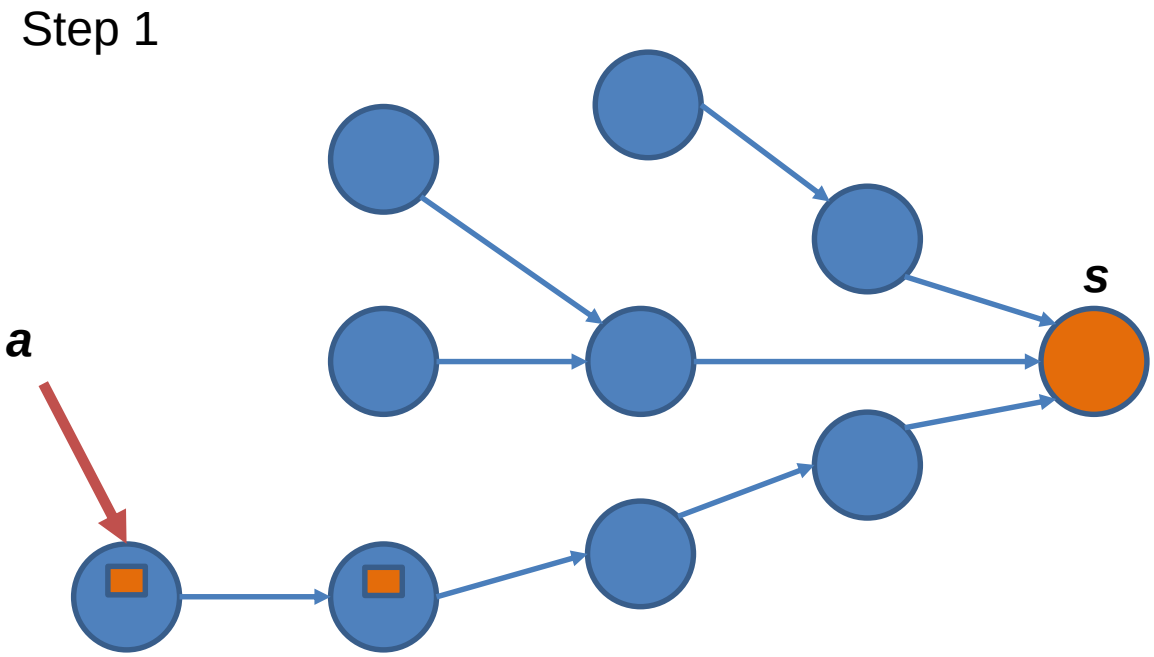
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



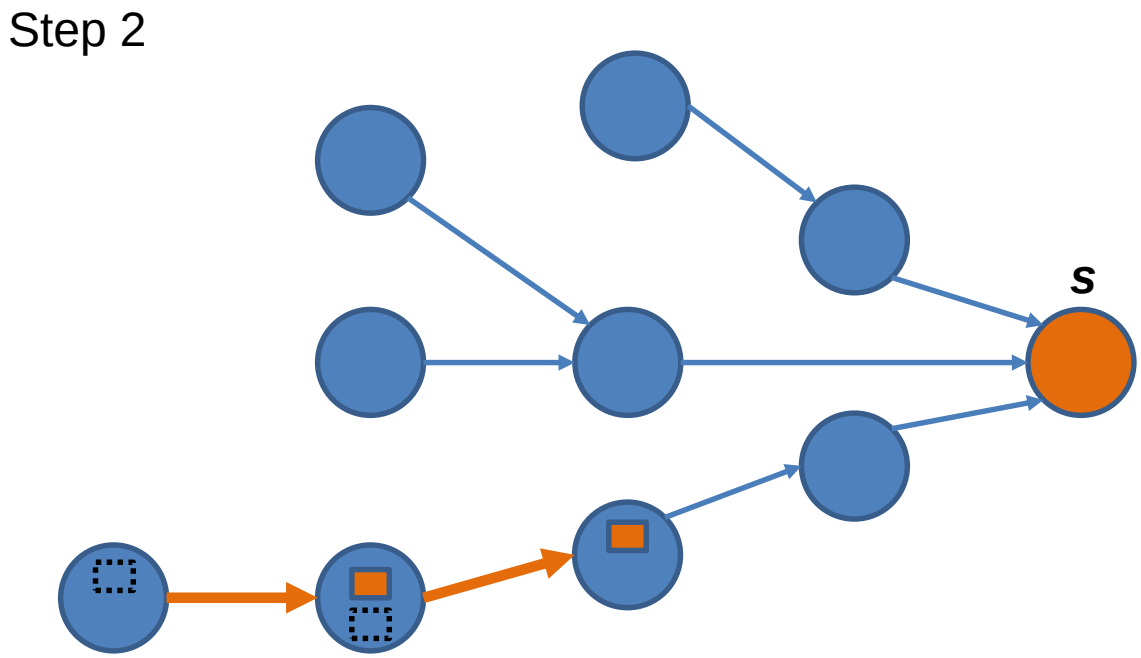
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



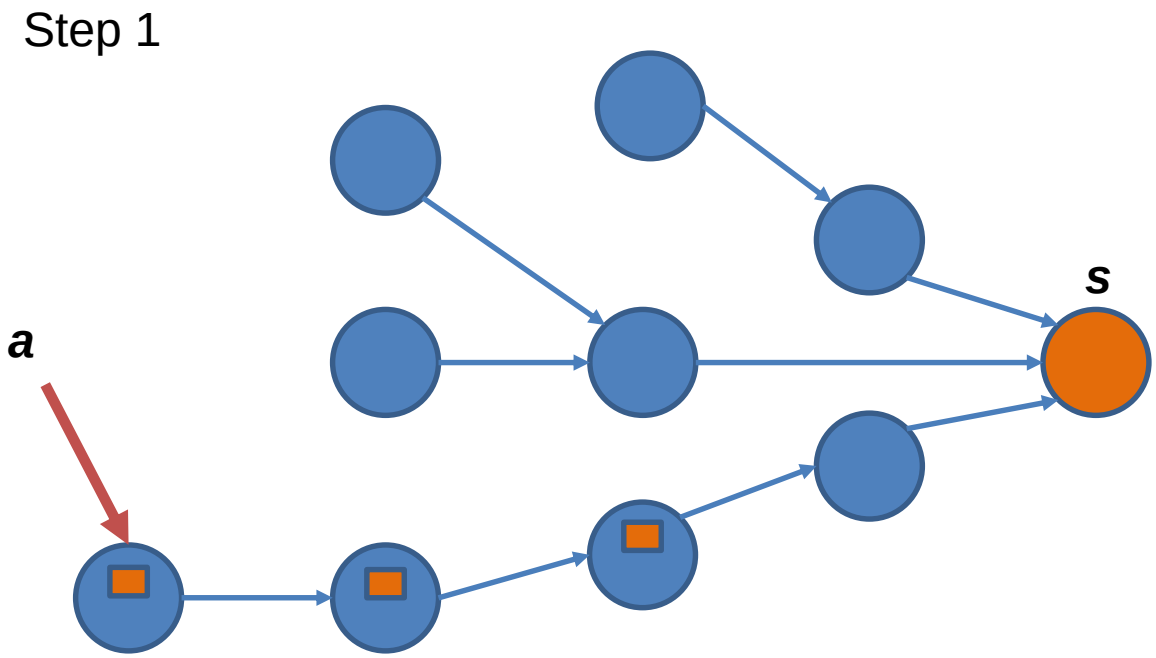
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



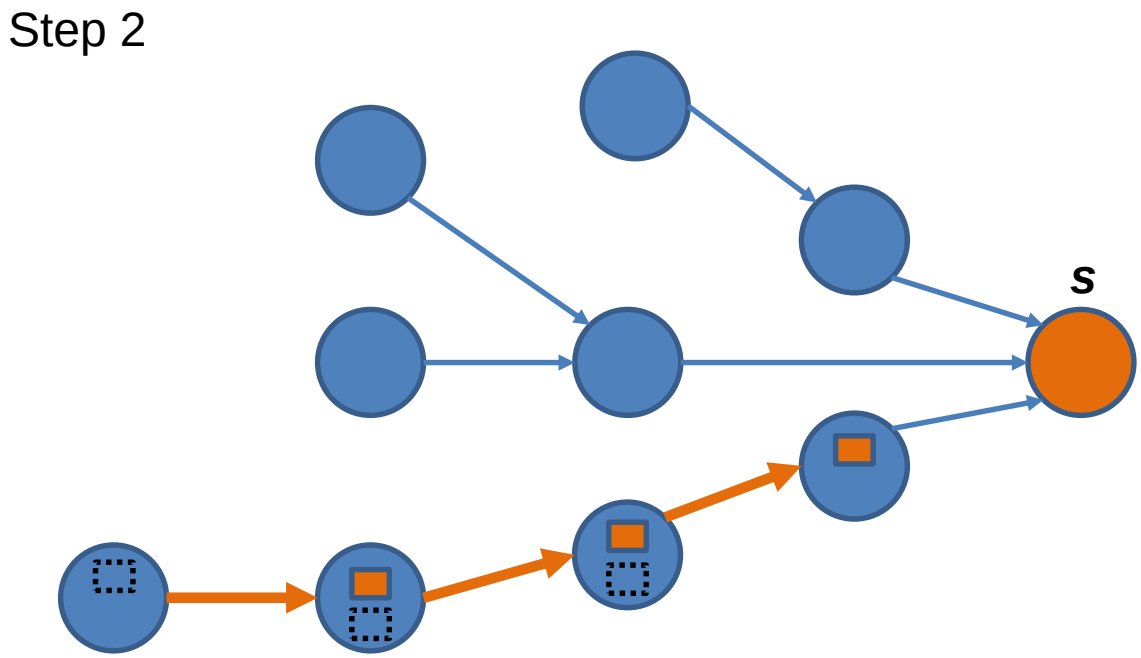
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



Information gathering with adversary

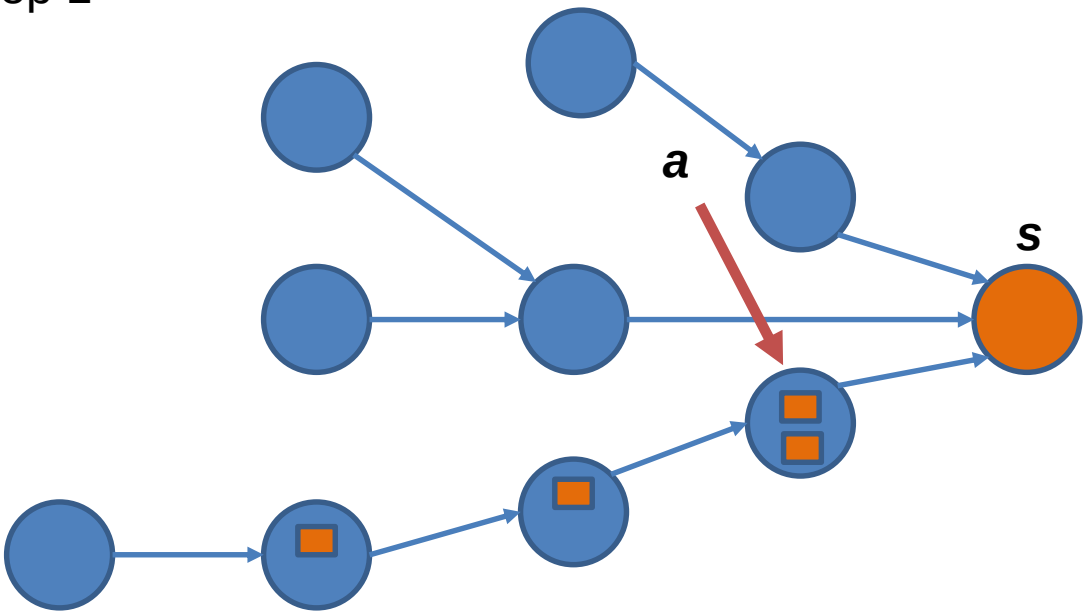
- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



Information gathering with adversary

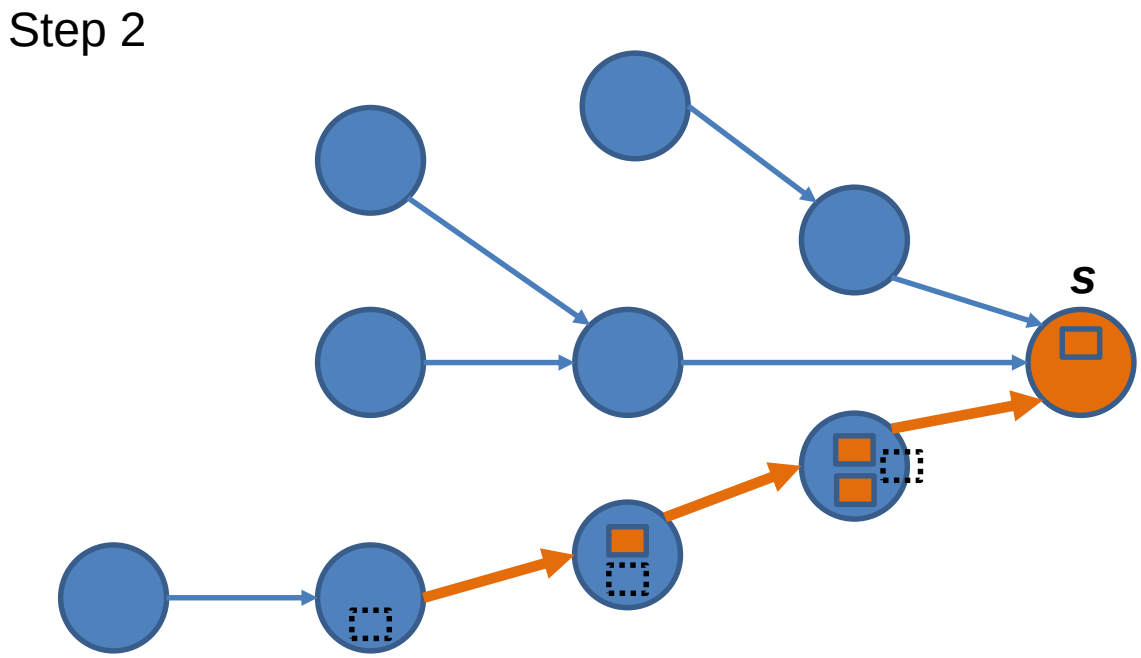
- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*

Step 1



Information gathering with adversary

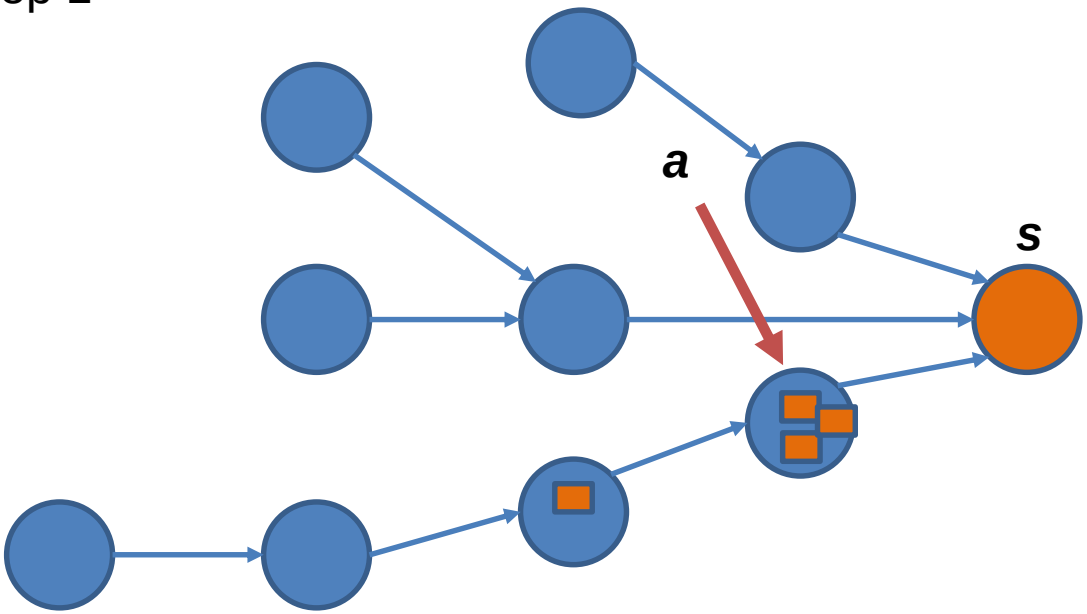
- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



Information gathering with adversary

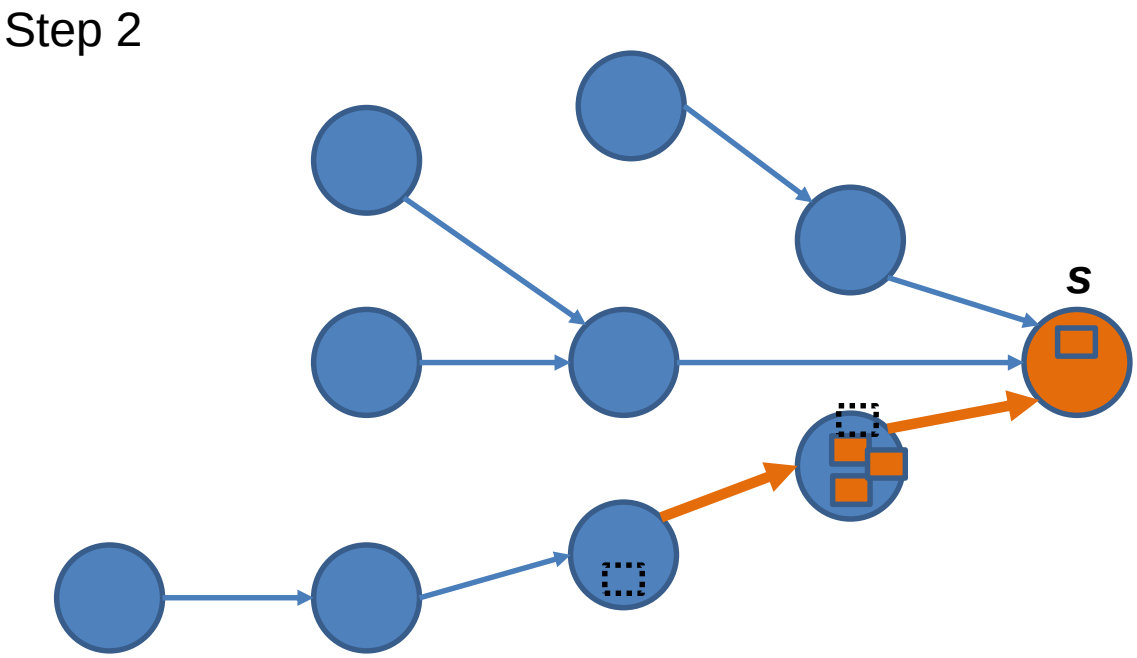
- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*

Step 1



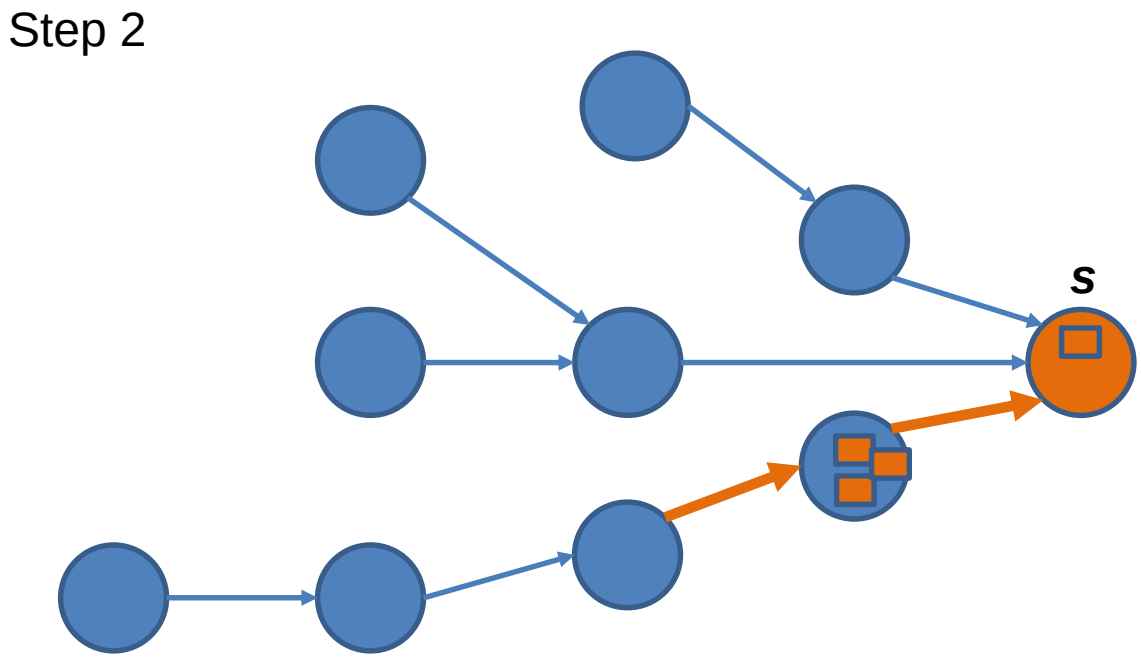
Information gathering with adversary

- 2 mini-steps model: each round has 2 steps
  - Step 1: adversary injects up to *c* packets into the network.
  - Step 2: each node sends up to *c* packets forward.
- Example with *c = 1* *(always send policy)*



Information gathering with adversary

- Node buffer has size 3.
- Could we have done better, using another policy?
- What buffer size is sufficient against any adversarial strategy?
  - Depends on policy. So, which policy requires minimum buffer size?



Related work

- Adversarial queueing theory introduced in [Borodin et al., 2001]
  - Each packet can have its own destination + forced route.
  - *Stability* of a policy: are buffer sizes bounded by some *f(n)* for all input streams? Greedy policies are stable for all DAGs when *c = 1*.
  - There exist *universally stable* policies (stable on *any* network) when *c = 1* [Andrews et al., 2001] (though *f(n)* can be exponential in *n*).

Related work

- Adversarial queueing theory introduced in [Borodin et al., 2001]
  - Each packet can have its own destination + forced route.
  - *Stability* of a policy: are buffer sizes bounded by some *f(n)* for all input streams? Greedy policies are stable for all DAGs when *c = 1*.
  - There exist *universally stable* policies (stable on *any* network) when *c = 1* [Andrews et al., 2001] (though *f(n)* can be exponential in *n*).
- Competitive Network Throughput model [Aiello et al., 2003]
  - Buffer sizes are fixed to some constant *B*.
  - Goal: minimize number of *dropped* packets.
  - For *B = 1*, any online deterministic algorithm is $\Omega(n)$-competitive.
  - For *B > 1*, $O(\sqrt{n})$-competitiveness can be achieved.

Related work

- Maximum buffer size for information gathering studied in [Kothapalli and Scheideler, 2003] on ***undirected*** paths
  - More powerful adversary that turns edges on/off each round.
  - *Θ(log n)*-competitiveness upper/lower bound.

Related work

- Maximum buffer size for information gathering studied in [Kothapalli and Scheideler, 2003] on ***undirected*** paths
  - More powerful adversary that turns edges on/off each round.
  - *Θ(log n)*-competitiveness upper/lower bound.
- Maximum buffer size on ***directed*** paths (our setting) [Miller and Patt-Shamir, DISC 2016]
  - With no locality constraints (every node can see the whole network), *O(c)* buffer size is sufficient.
  - With locality constraints:
    - "Always send" requires *Θ(n)* buffer size.
    - "Forward iff successor empty" requires *Θ(r)* packets after *r* rounds.
    - "Local downhill", which forwards iff successor has strictly less packets in its buffer, requires *Θ(n)* buffer size.

Our results

- Information gathering on directed paths and trees with locality ℓ and injection rate c
  - $\Omega(c \log n/\ell)$ lower bound on required buffer sizes
    - more precisely $c(1 + (\log n - 2 \log \ell)/(2\ell))$
  - Asymptotic lower bound also holds for undirected paths
  - For $c = 1$ and $\ell = 1$, upper bound of $O(\log n)$ on directed paths
    - More precisely $\log n + 3$ upper bound
  - For $c = 1$ and $\ell = 2$, upper bound of $O(\log n)$ on trees

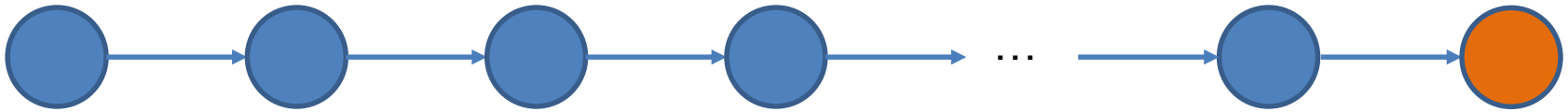
Our results

- Information gathering on directed paths and trees with locality ℓ and injection rate c
  - $\Omega(c \log n/\ell)$ lower bound on required buffer sizes
    - more precisely $c(1 + (\log n - 2 \log \ell)/(2\ell))$
  - Asymptotic lower bound also holds for undirected paths
  - For $c = 1$ and $\ell = 1$, upper bound of $O(\log n)$ on directed paths
    - More precisely $\log n + 3$ upper bound
  - *For $c = 1$ and $\ell = 2$,* upper bound of $O(\log n)$ on trees

- Patt-Shamir and Rosenbaum present essentially the same results in their PODC 2017 paper (!)
  - Same algorithms, different analysis
  - Stronger bounds of $O(\log diam(G))$ for trees

Lower bound on directed path

- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$



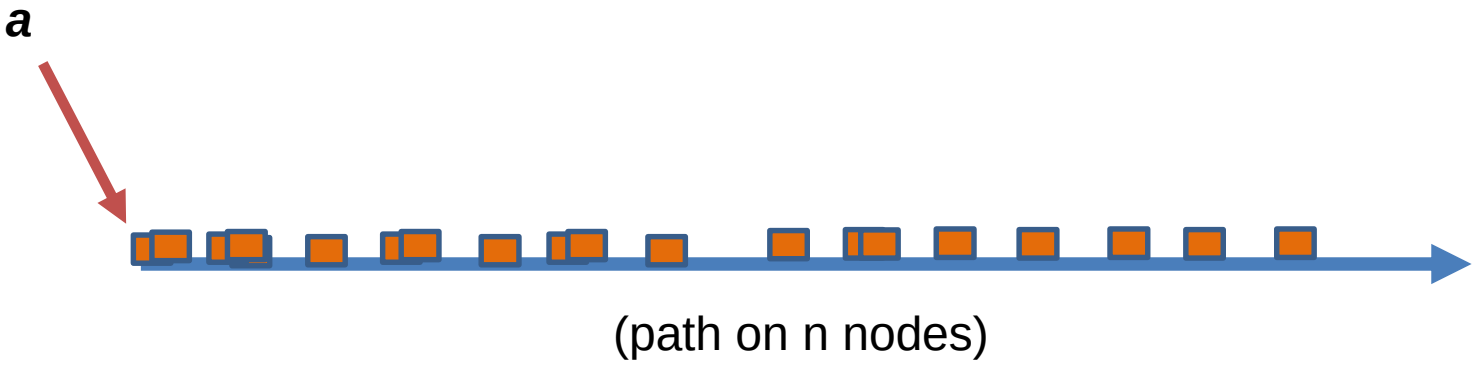
Lower bound on directed path

- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$

(path on n nodes)

Lower bound on directed path

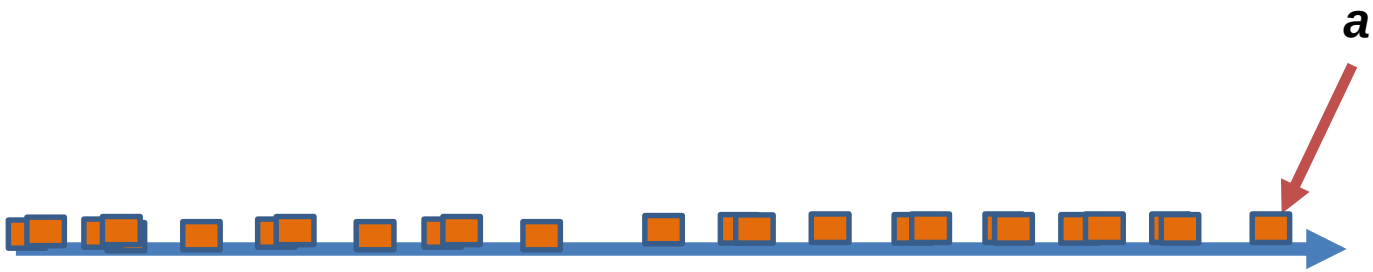
- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$



1. Adversary successively injects n packets at the tail. No packet has time to get to the sink => The *packet density* becomes
   d = #packets / #nodes = n/n = 1

Lower bound on directed path

- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$

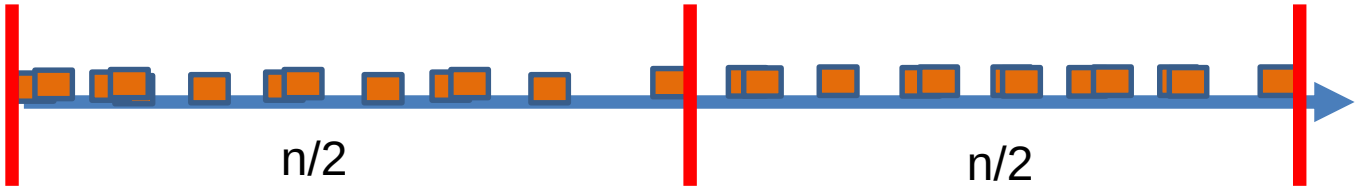


(path on n nodes)

1. Adversary successively injects n packets at the tail. No packet has time to get to the sink => The *packet density* becomes
   d = #packets / #nodes = n/n = 1
2. Next, adversary successively injects at the head. The head can never empty its buffer.
   At some point, every node (except the head) will stop sending forward.

Lower bound on directed path

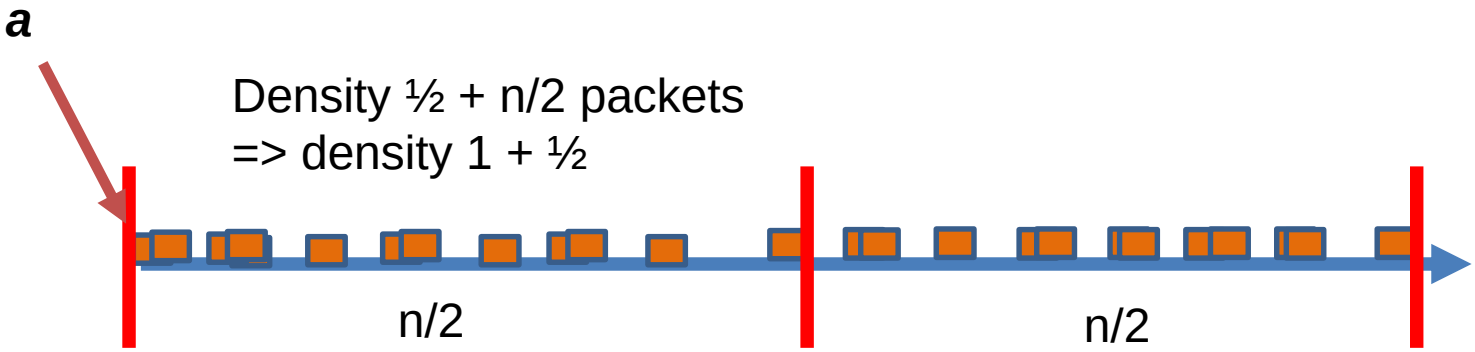
- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$



3. The next step is to obtain an interval of n/2 nodes with density 1 + ½. If right half already has this density, we are done. If not, the left half must have density at least ½.

Lower bound on directed path

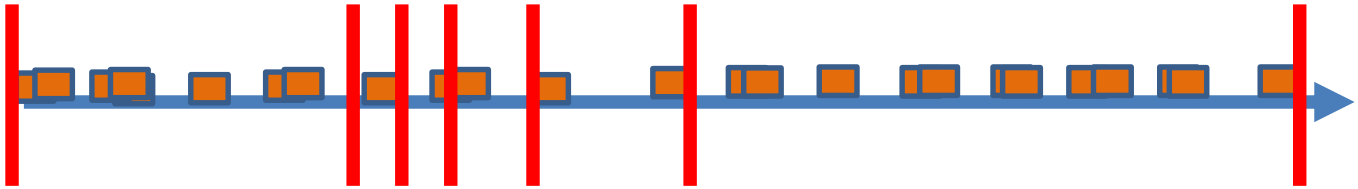
- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$



3. The next step is to obtain an interval of n/2 nodes with density 1 + ½. If right half already has this density, we are done. If not, the left half must have density at least ½.
4. In this case, adversary injects n/2 packets at the tail. Recall that every node has stopped sending forward until something changes. The last node of the left half won't catch on until n/2 injections. The density of the left half becomes at least ½ + 1.

Lower bound on directed path

- How can adversary enforce *O(log n)* buffer size on at least one node?
  - Rough idea for c = 1, $\ell = 1$



5. Adversary can repeat this procedure on the 1 + ½ interval. This can be applied up to *log n* times, after which the density is 1 + ½ log n, implying that some node has at least this buffer size.

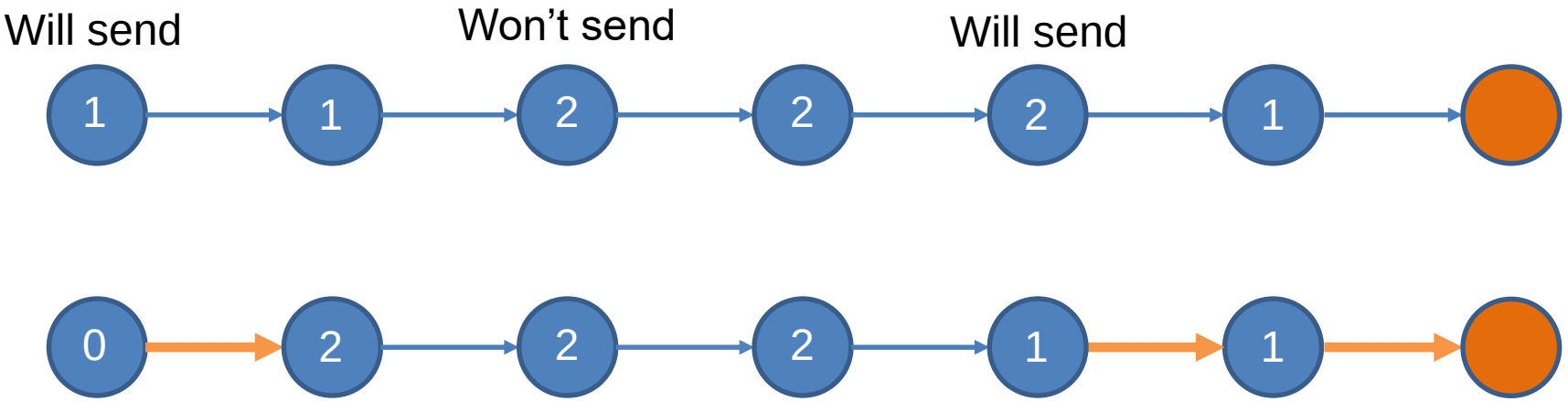
Upper bound on directed path

- How can we ensure that buffer size $O(\log n)$ is enough?
  - $c = 1, \ell = 1$
- Simple algorithm for a node v with successor v':
  - If v currently has **odd** buffer size, send to v' iff ***bufsize(v) ≥ bufsize(v')***
  - If v currently has **even** buffer size, send to v' iff ***bufsize(v) > bufsize(v')***

- **Claim:** if every node runs this algorithm, $O(\log n)$ buffer size is enough.

Upper bound on directed path

- Example of before and after step 2
  - If v currently has **odd** buffer size, send to v' iff ***bufsize(v) ≥ bufsize(v')***
  - If v currently has **even** buffer size, send to v' ***iff bufsize(v) > bufsize(v')***



Upper bound on directed path

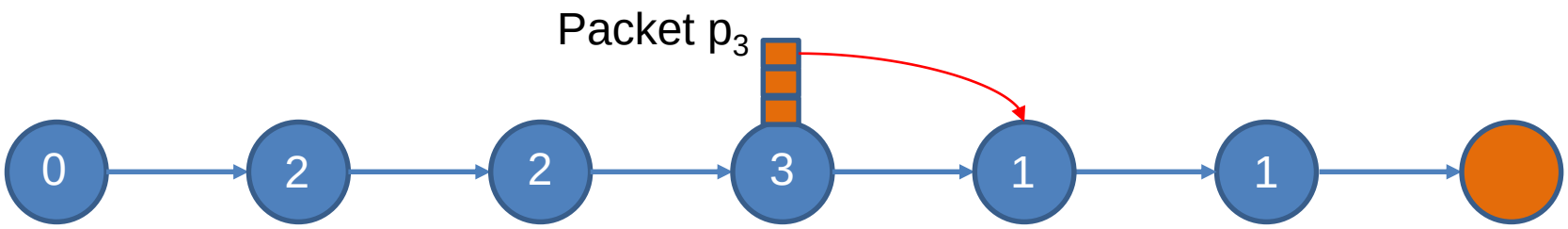
- Simple algorithm for a node v with successor v':
  - If v currently has **odd** buffer size, send to v' iff ***bufsize(v) ≥ bufsize(v')***
  - If v currently has **even** buffer size, send to v' ***iff bufsize(v) > bufsize(v')***

- **Claim:** if every node runs this algorithm, *O(log n)* buffer size is enough.
- **Intuition:** if only "send when ≥" is applied, packets accumulate in the front on the path. If only "send when >" is applied, packets accumulate in the back of the path.

  Alternating between the two policies spreads out packets in the middle.

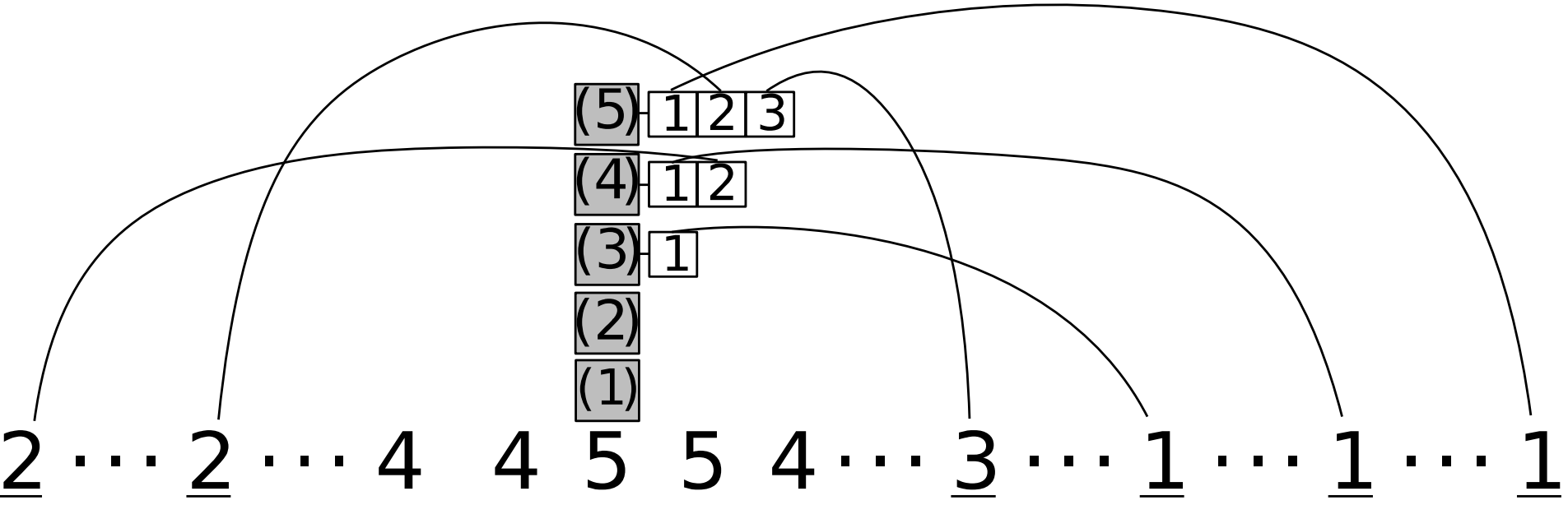
Upper bound on directed path

- Order the packets $p_1, p_2, \ldots, p_k$ of a node v arbitrarily (at end of round)
- Call $h(v) = k$ the height of v (i.e. its number of packets)
- Idea: for $i \geq 3$, for packet p_i to exist, there must be nodes $u_1, u_2, \ldots, u_{k-2}$ of heights $1,2,\ldots,i-2$, respectively.
- "Attach" each packet p_i to nodes $u_1, u_2, \ldots, u_{i-2}$



Upper bound on directed path

- Idea: for $i \geq 3$, for the i-th packet to exist, there must be nodes u_1, u_2, …, u_{k-2} of heights $1,2,\ldots,i-2$, respectively.
- "Attach" each i-th packet to nodes u_1, u_2, …, u_{i-2}



Upper bound on directed path

- Attachment scheme:
  - For every node *v,* each of its packets p_i is attached to nodes of height *1,2,…,i-2* (for every $i \geq 3$)
  - No two nodes are attached to the same packet.

- ***Lemma:*** if the even-odd algorithm is used, at the end of every round, there exists an attachment scheme.

  Proof idea: oh jeez…

- ***Theorem:*** the maximum size of a buffer is *log n + 3*

  Proof idea: counting argument using attachment scheme. If a node has a packet $p_{\log n + 4}$, this implies the existence of distinct attached nodes of height *log n + 2, log n + 1, …, 1.* Plus the nodes attached to $p_{\log n + 3}$, *to* $p_{\log n + 2}$, etc. Plus the nodes attached to these attached nodes, and so on. By counting appropriately, we end up with $> n$ nodes, a contradiction.

Upper bound on directed path

- Attachment scheme:
  - For every node *v,* each of its packets p_i is attached to nodes of height *1,2,…,i-2* (for every $i \geq 3$)
  - No two nodes are attached to the same packet.

- ***Lemma:*** if the even-odd algorithm is used, at the end of every round, there exists an attachment scheme.

  Proof idea: oh jeez…

  How do we prove this?!

Upper bound on directed path

- Attachment scheme:
  - For every node *v,* each of its packets p_i is attached to nodes of height *1,2,…,i-2* (for every $i \geq 3$)
  - No two nodes are attached to the same packet.

- ***Lemma:*** if the even-odd algorithm is used, at the end of every round, there exists an attachment scheme.

  Proof idea: oh jeez…

  How do we prove this?!

  Induction on the number of rounds. True at round 0 when every node has height 0.

Upper bound on directed path

- ***Lemma:*** if the even-odd algorithm is used, at the end of every round, there exists an attachment scheme.

  Proof idea: oh jeez…

  How do we prove this?!

  Induction on the number of rounds. True at round 0 when every node has height 0.

  For the induction step, start the round with a valid attachment scheme.

  Process the round, retain attachments that are still valid.

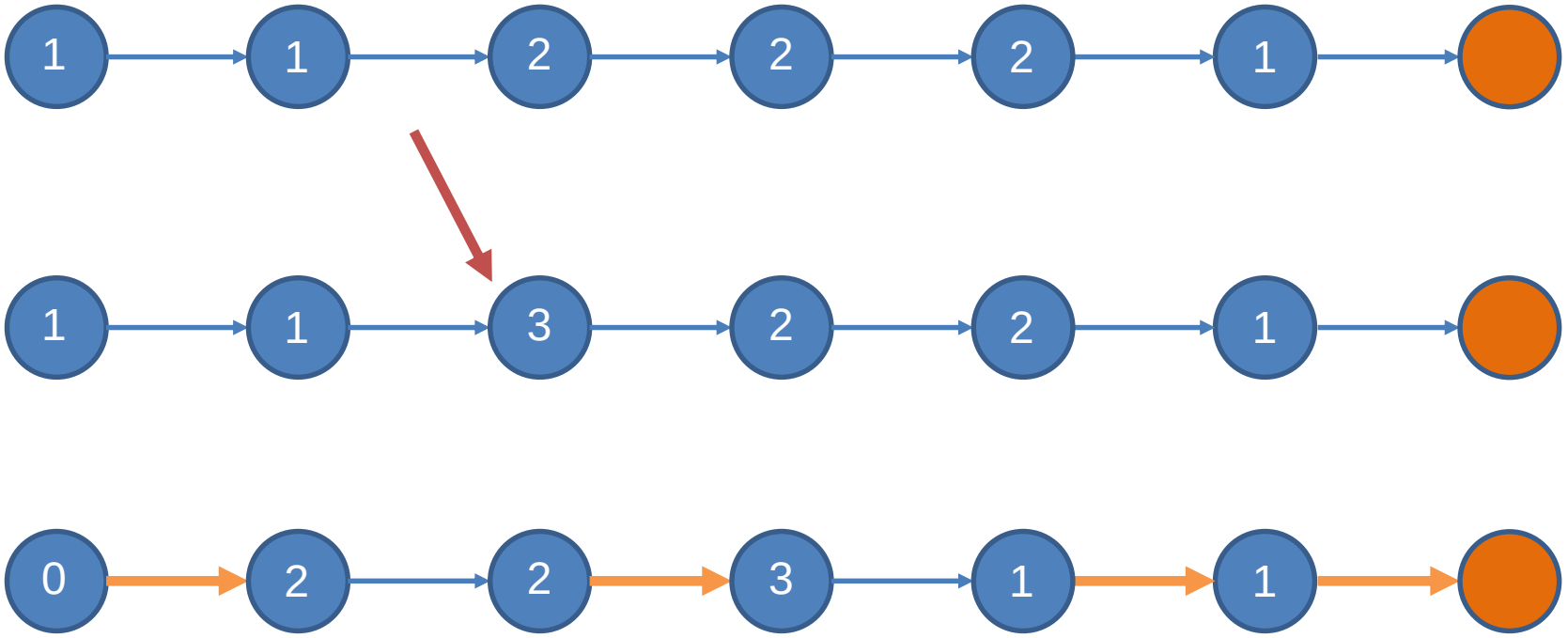
  Some node have gained height ("up nodes") and must find attachments.

  Some nodes have lost height ("down nodes") and have attachments to give.

  Match each up node with a distinct down node, and pass unused attachments.

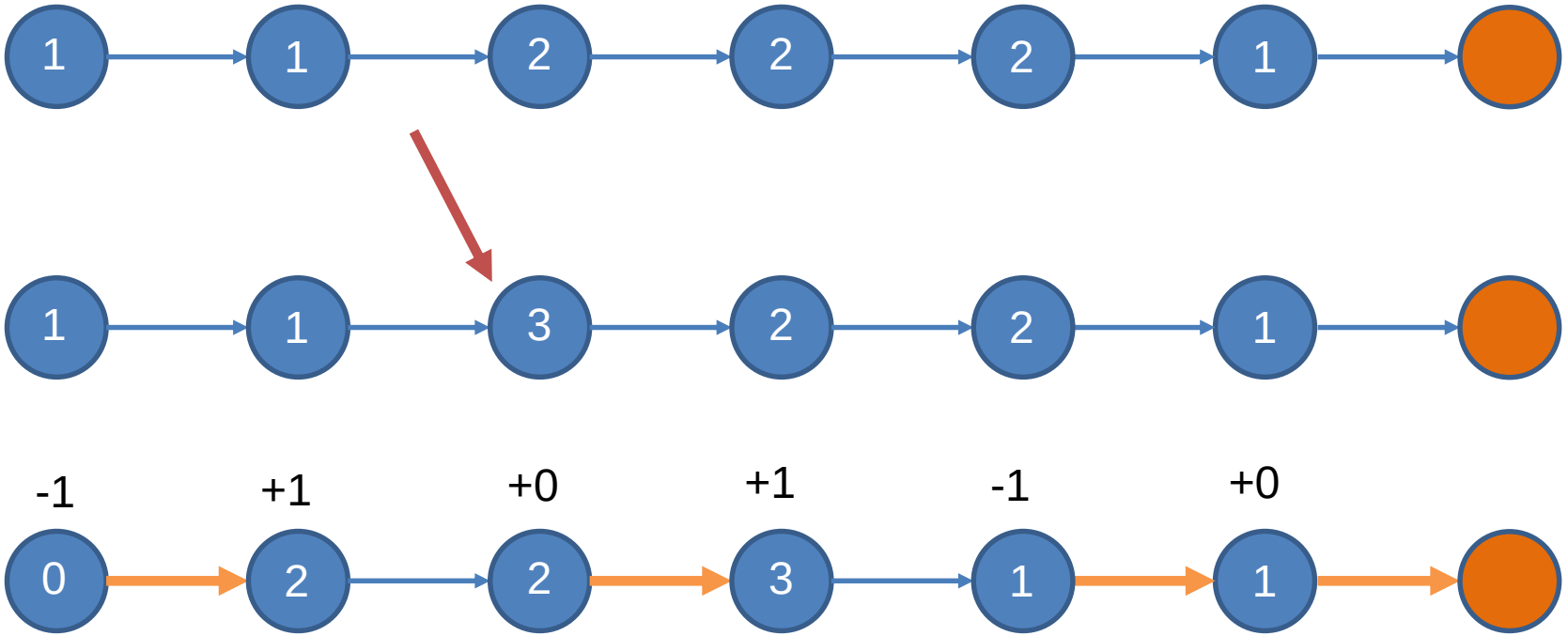
Up nodes and down nodes

- Example of before-step-1, after-step-1, after-step-2



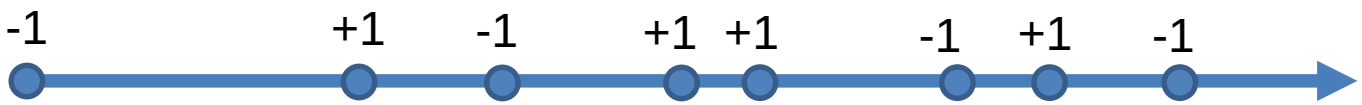
Up nodes and down nodes

- After each round of two mini-steps, each node has either -1, +1, +2 or +0 buffer size.



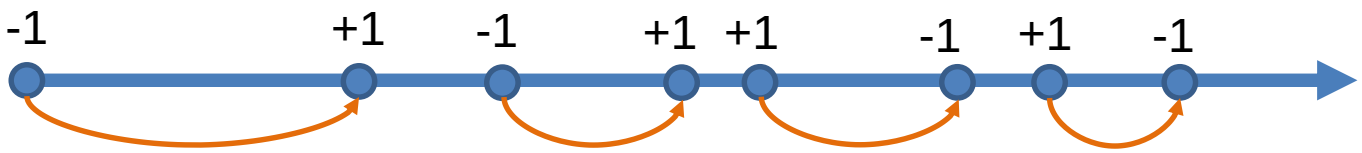
Up nodes and down nodes

- After each round of two mini-steps, each node has either -1, +1, +2 or +0 buffer size.
- Focus on the +1 and -1 nodes. (to simplify things, assume no +2)



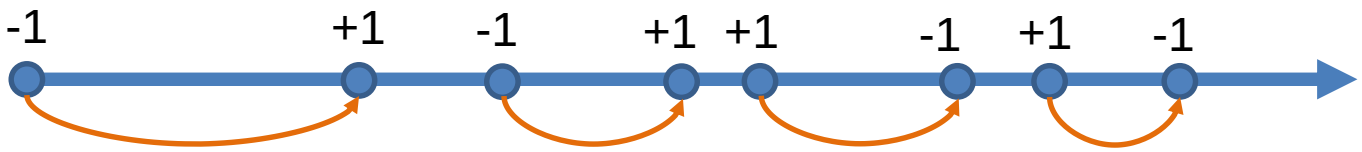
Up nodes and down nodes

- After each round of two mini-steps, each node has either -1, +1, +2 or +0 buffer size.
- Focus on the +1 and -1 nodes. (to simplify things, assume no +2)
- Make +1/-1 node pairs by traversing from left to right, matching the unmatched nodes encountered with their nearest right neighbor.



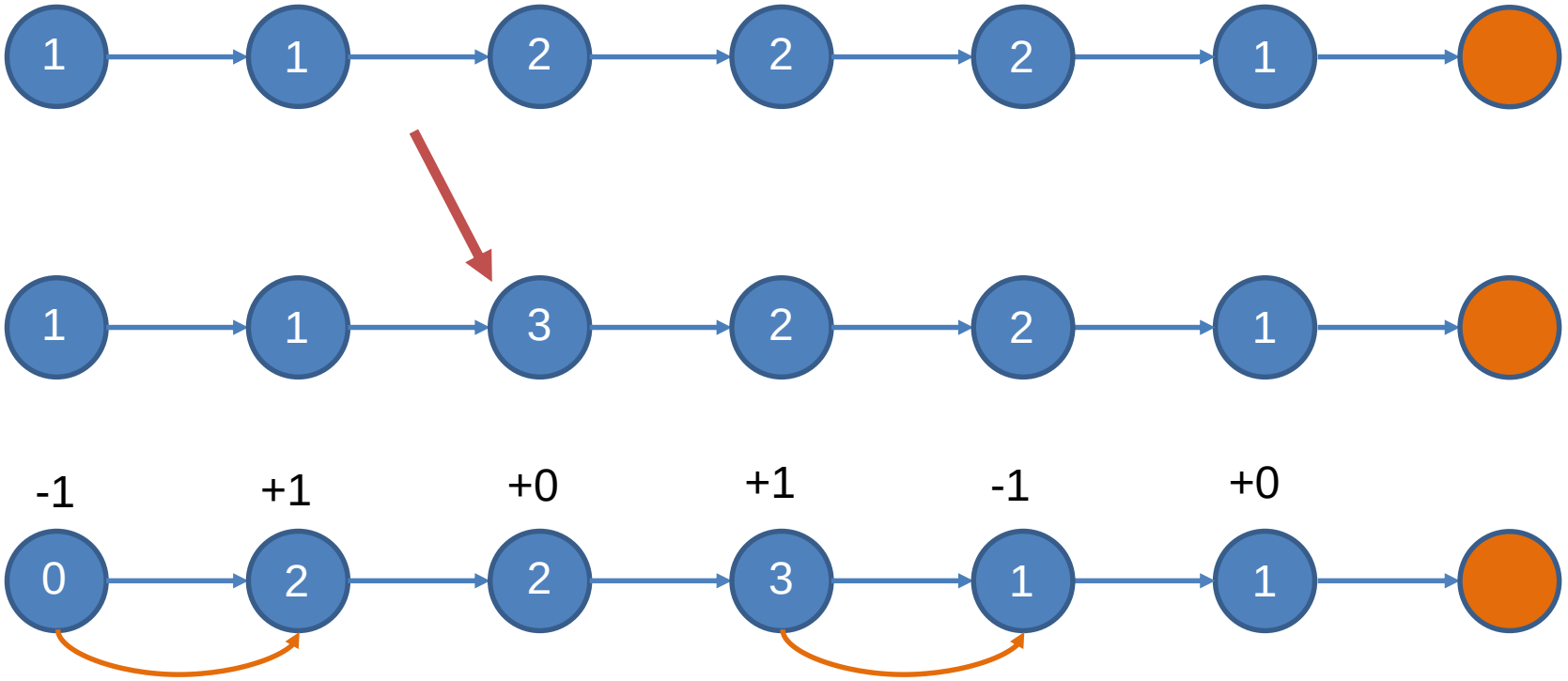
Up nodes and down nodes

- After each round of two mini-steps, each node has either -1, +1, +2 or +0 buffer size.
- Focus on the +1 and -1 nodes. (to simplify things, assume no +2)
- Make +1/-1 node pairs by traversing from left to right, matching the unmatched nodes encountered with their nearest right neighbor.
- ***Lemma:*** every matched pair has a +1 and a -1.

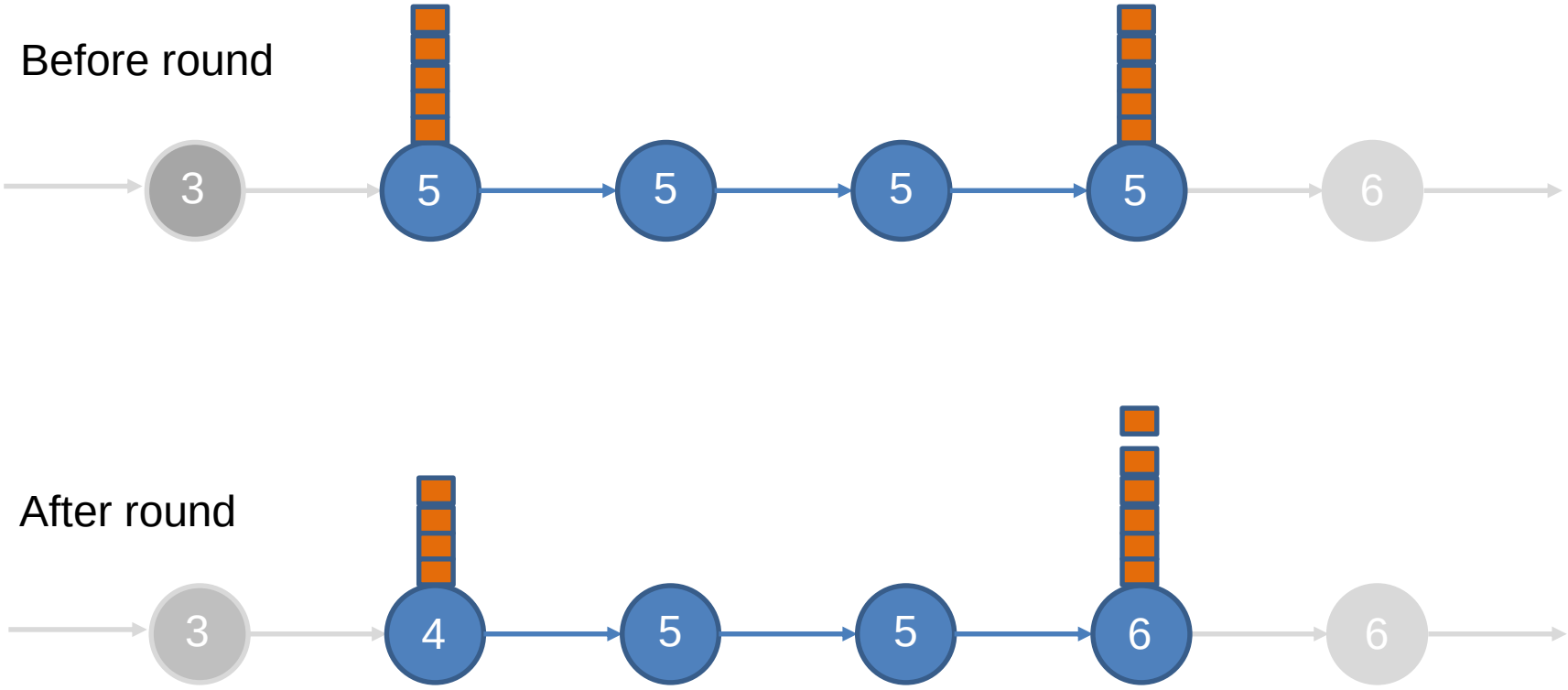


Up nodes and down nodes

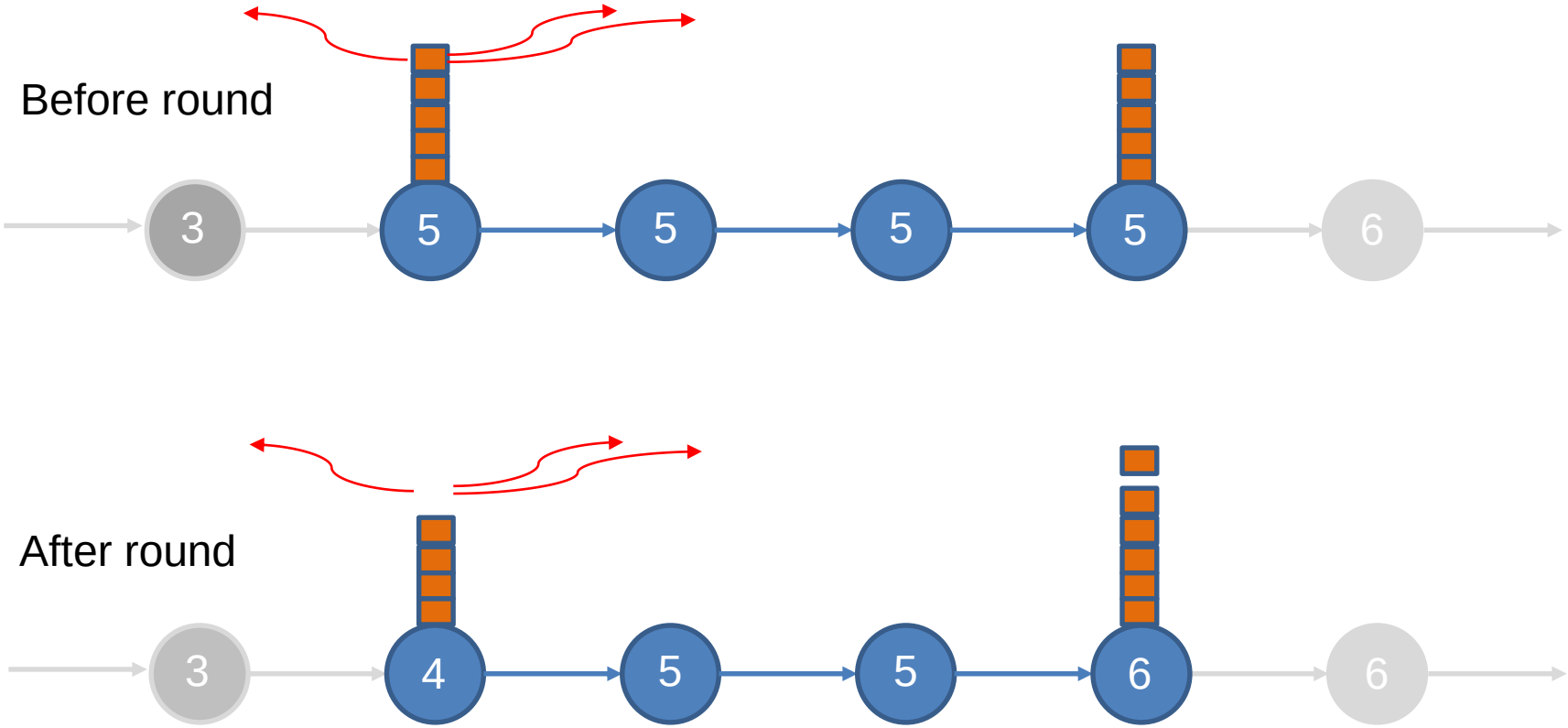
- For example…



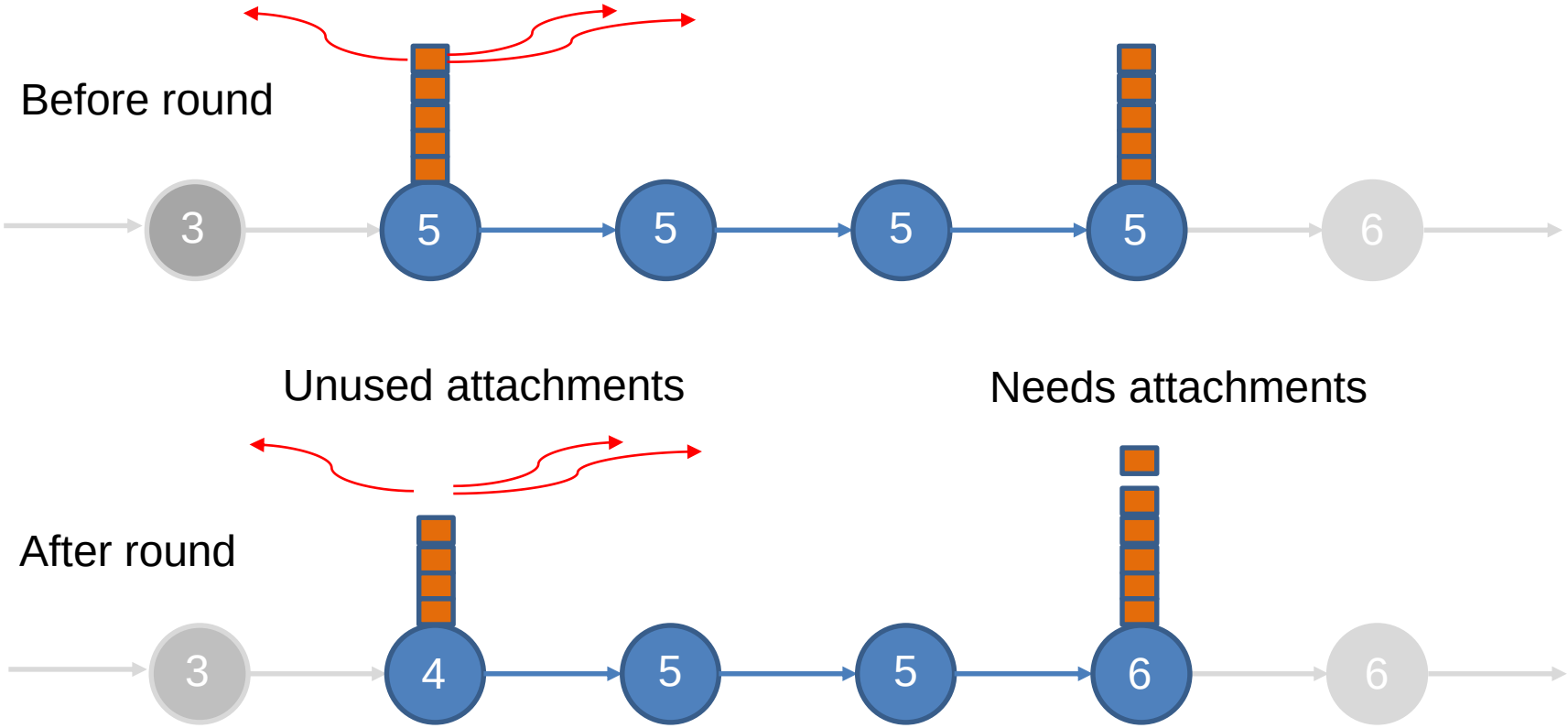
Handling a -1/+1 pair



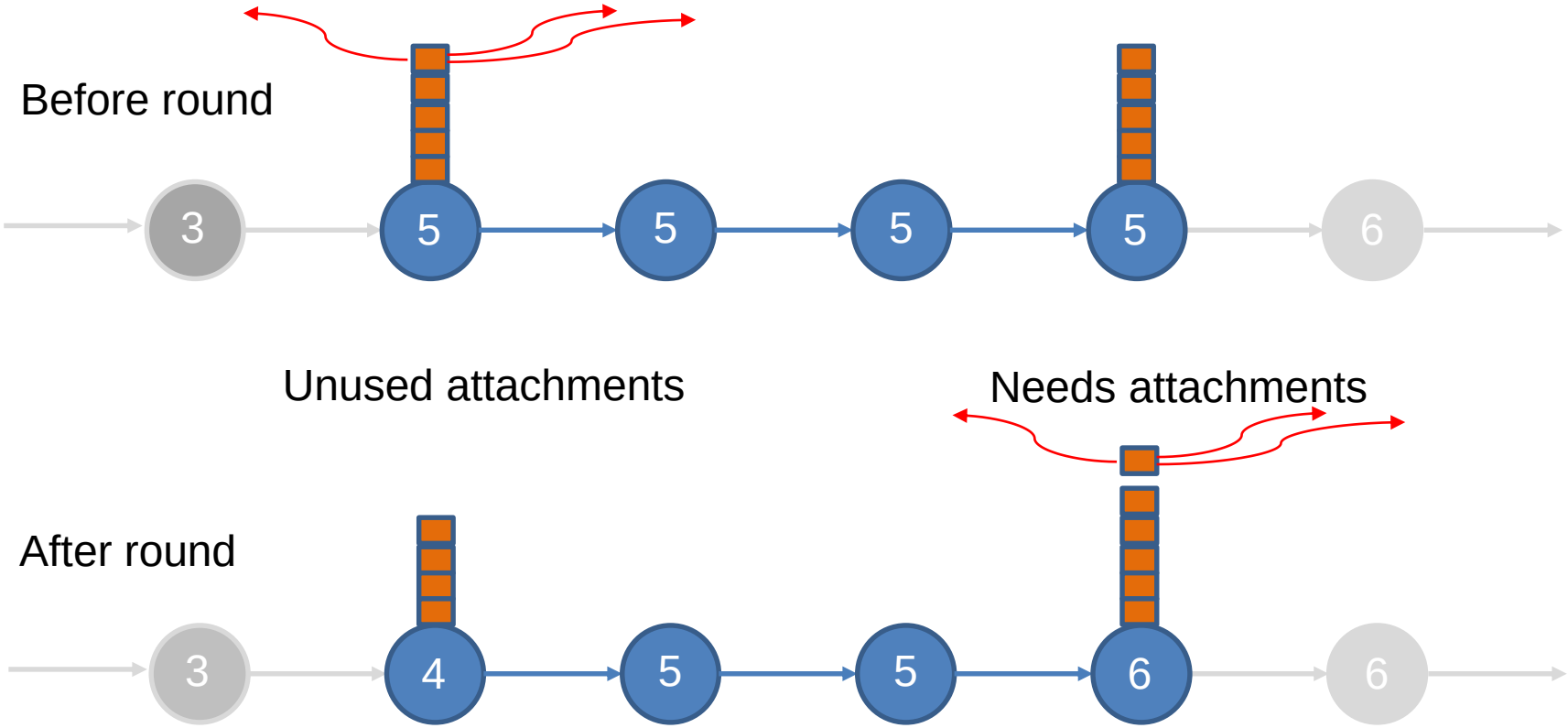
Handling a -1/+1 pair



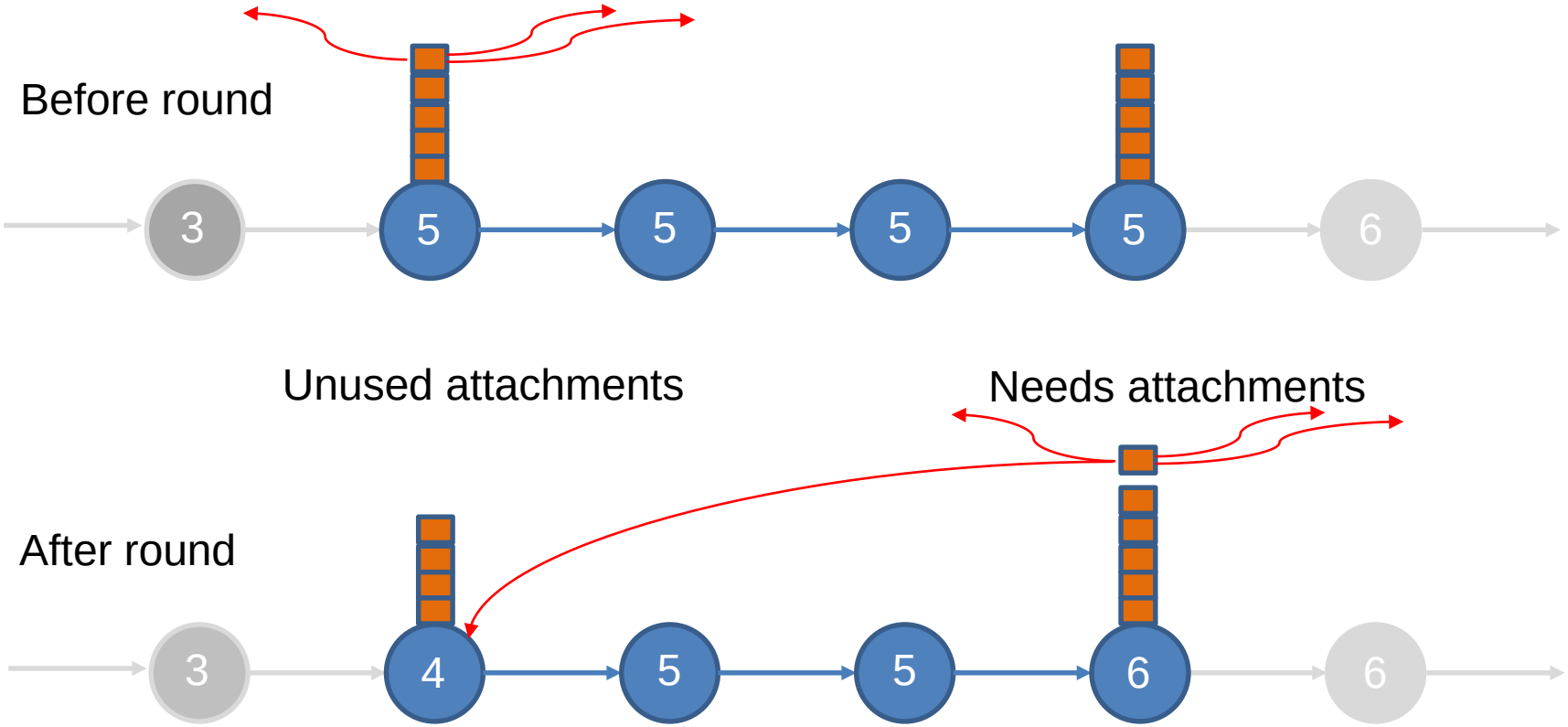
Handling a -1/+1 pair



Handling a -1/+1 pair

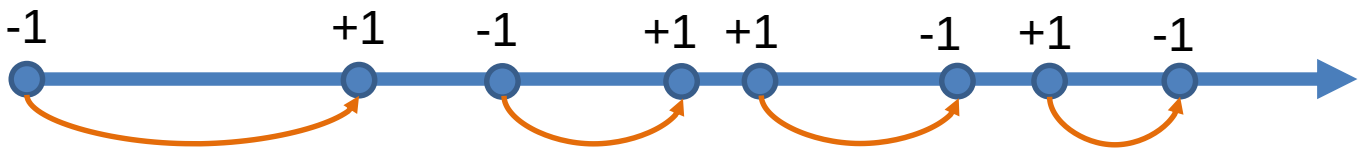


Handling a -1/+1 pair



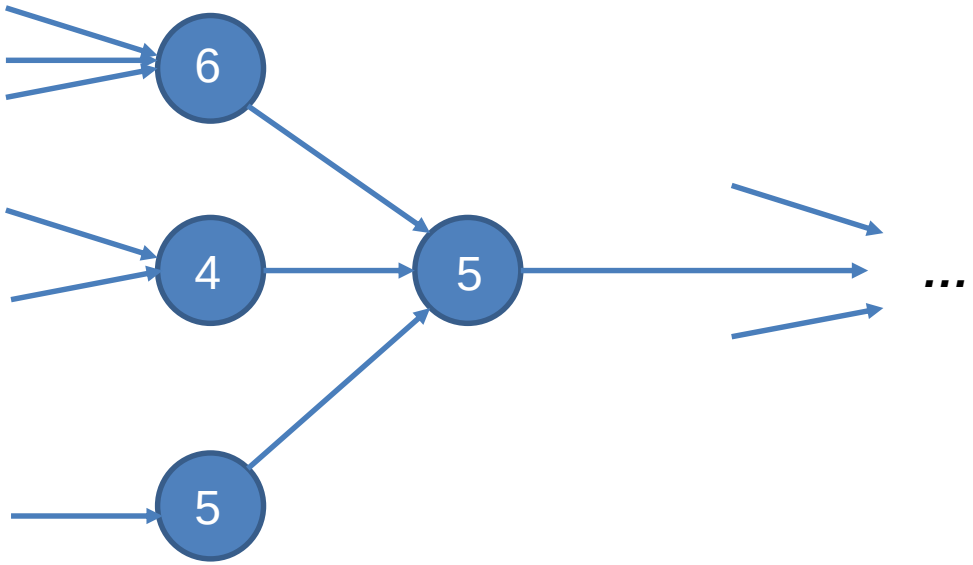
Up nodes and down nodes

- Process each +1/-1 subpath independently. After each has been handled, we have obtained a valid attachment scheme.
- Many, many subcases and annoying details to handle in the attachments passing (e.g. +2 nodes).
- The odd-even algorithm is **needed** for this proof to work – in some special cases of attachment passing.
  - Unfortunately, too technical to describe here. Sorry!



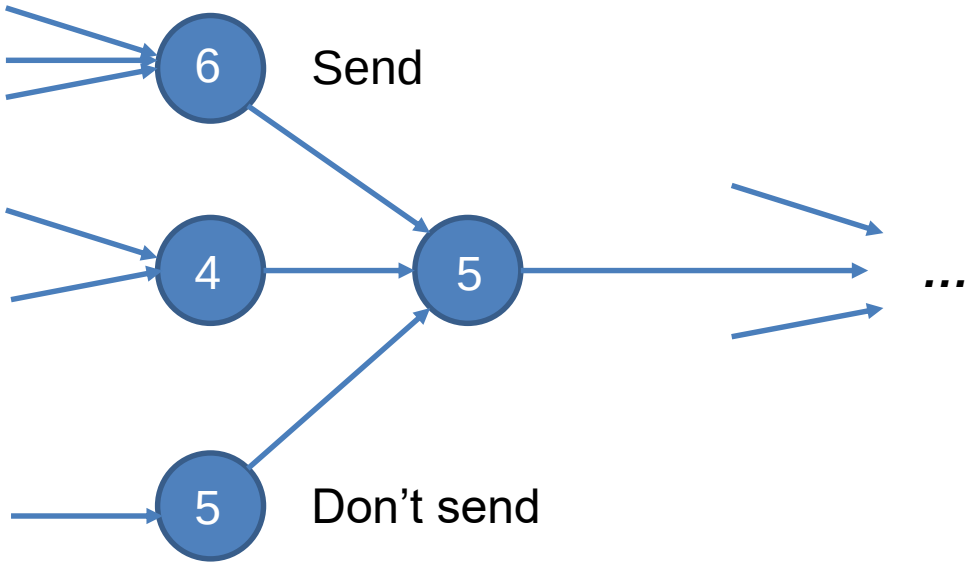
Algorithm on trees

- Slight modification of the odd-even algorithm required on trees.
- For nodes of indegree > 1, give priority to the highest node.
- Apply the same odd-even algorithm on the nodes that have priority.



Algorithm on trees

- Slight modification of the odd-even algorithm required on trees.
- For nodes of indegree > 1, give priority to the highest node.
- Apply the same odd-even algorithm on the nodes that have priority.



Algorithm on trees

- Locality 2 is needed to determine who has priority.
  - With locality 1 on trees, $\sqrt{n}$ buffer size lower bound.
- Our algorithm partitions the tree into priority paths. The path analysis can be applied to each path individually.
  - Just more cases to handle.

Conclusion

- We have shown a $\Omega(\log n)$ lower bound + matching $O(\log n)$ lower bound for paths when $c=1$, $\ell=1$ + trees when $\ell=2$.
  - What about $c > 1$? See Patt-Shamir & Rosenbaum paper.
- What of non-uniform edge capacities?
- We only cared about ***the*** node with highest buffer size. But adversary can't make ***every*** buffer of size $\log n$. Alternative criterion for space requirements?
- What if we allow randomization…
  - In the node algorithms?
  - In the packet injection model (instead of an adversary)?
- Competitiveness of information gathering in general DAGs?