Algorithmen für Ad-hoc- und Sensornetze

VL 07 – Data Gathering

Dr. rer. nat. Bastian Katz

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie

10. Juni 2009

(Version 4 vom 22. Juni 2009)

Heute

- Data Gathering (kurze Einführung)
  - Ein Name, viele Probleme
- Aggregation von Funktionen auf Sensorwerten
  - Anfragen und Funktionsklassen
  - Randomisierung und Schranken
- Scheduling von Data Gathering Trees
  - Einsammeln von großen Datenmengen
  - Energiesparen durch Koordination des Funkkanals

Data Gathering

„Einsammeln von Daten (in einer Senke)“ kann heißen

- Ereignismeldungen (mit etwas gutem Willen)
  - Knoten mit kritischen Werten teilen das mit
  - oft nur einzelne Pakete
  - $\Rightarrow$ Geringer Verkehr, Optimierungsziel z. B. geringe Latenz
- Antworten auf Anfragen
  - $\Rightarrow$ Verkehr stark abhängig von Art/Frequenz der Anfrage
- Permanenter Fluss von Daten
  - jeder Knoten schickt regelmäßig Sensorwerte Richtung Senke

Optimierungsebenen

- Topologiekontrolle/Power Control
  - MSTs, Shortest-Path-Trees, etc.
  - Auswahl einer geschickten Topologie kann jeweiligen Zweck unterstützen, Zeit/Energie sparen usw
- Datenaggregation
  - bei vielen Anfragen müssen nicht alle Sensordaten zur Senke, um eine Information bereitzustellen
- Scheduling von Übertragungen und Schlafzyklen
  - Absprache spart Energie: Kein ständiges Mithören von fremden Nachrichten!

Heute

- Data Gathering (kurze Einführung)
  - Ein Name, viele Probleme
- Aggregation von Funktionen auf Sensorwerten
  - Anfragen und Funktionsklassen
  - Randomisierung und Schranken
- Scheduling von Data Gathering Trees
  - Einsammeln von großen Datenmengen
  - Energiesparen durch Koordination des Funkkanals

Anfragen an ein Netzwerk

Anfragen an ein Sensornetz ähneln oft denen an Datenbanken:

„Gib mir die Koordinaten der zehn Knoten mit der höchsten Temperatur“

- TinyDB und ähnliche Services unterstützen Anfragesprachen mit SQL-ähnlicher Syntax.
- typischerweise Fluten der Anfrage, Aggregation der Antwort entlang eines Baumes
- beliebig komplexe Anfragen, aber einige Primitive
  - dazu gleich mehr

Aggregationsmodell

Voraussetzungen

- jeder Knoten enthält genau einen Sensorwert
  - (nur zur Vereinfachung)
- ein zur Senke gerichteter Spannbaum ist gegeben
  - Durchmesser D
- jede Nachricht darf nur 1-2 Elemente enthalten
  - (allgemeiner: $O(1)$ Elemente)
- Anfragefunktionen sind *unabhängig* davon, welcher Knoten welches Element hält

Gegeben ein (Anfrage-)Baum, in dem jeder Knoten genau einen Sensorwert hält, was ist die einfachste Anfrage, die man sich vorstellen kann?

Distributive Funktionen

Eine (berechenbare) Funktion f auf einer Menge M von Elementen (das sind die Sensorwerte!) heißt *distributiv*, wenn es ausreicht, wenn $f(M_1), f(M_2), \ldots$ für eine Partition $M_1, M_2, \ldots$ von M bekannt ist, um $f(M)$ zu berechnen[a].

[a] Partition: M ist die *disjunkte* Vereinigung der M_i

- Beispiele für distributive Funktionen:
  - Maximum: $f(M) = \max_i f(M_i)$ (Minimum genauso)
  - Summe: $f(M) = \sum_i f(M_i)$
  - Anzahl der Elemente mit bestimmter Eigenschaft: $f(M) = \sum_i f(M_i)$

Aggregation Distributiver Funktionen

Distributive Funktionen lassen sich mit $\Theta(n)$ Nachrichten in $\Theta(D)$ Schritten verteilt berechnen.

Klassischer Flooding-Echo-Ansatz

- Senke schickt Anfrage, jeder Knoten gibt die Anfrage an Kinder weiter.
- Blätter antworten mit Funktionswert
- Innere Knoten aggregieren ihren Funktionswert und den ihrer Teilbäume

Gegeben ein (Anfrage-)Baum, in dem jeder Knoten genau einen Sensorwert hält, was ist eine einfache nicht-distributive Anfrage?

Algebraische Funktionen

Eine (berechenbare) Funktion f auf einer Menge M von Elementen (das sind die Sensorwerte!) heißt *algebraisch*, wenn es ausreicht, die Funktionswerte einer *konstante Anzahl* distributiver Funktion für M zu kennen.

- Häufigstes Beispiel: Durchschnittlicher Wert
  - Summe / Anzahl aller Elemente
- k-größtes Element für festes k
  - distributive Funktionen hängen voneinander ab!
- Aggregation mit $\Theta(n)$ Nachrichten in $\Theta(D)$ Zeit.
- gibt es was Schlimmeres?

Holistische Funktionen

Berechenbare Funktionen auf einer Menge M von Elementen, die nicht algebraisch sind, werden *holistisch* genannt.

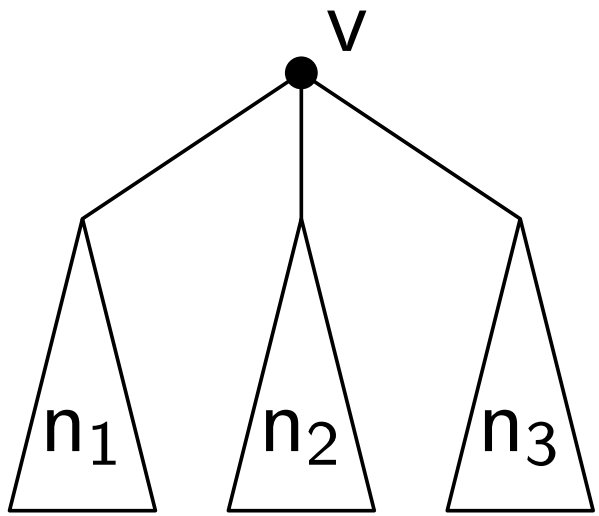
- Median, k-größtes Element
- z.B. in „Durchschnitt der 10% größten Werte“

Ist das wirklich schwerer? Vorschläge?

Aufwärmübung

Wie wähle ich in $\Theta(D)$ Runden *gleichverteilt* ein zufälliges Element aus?

- Berechne Anzahl der Elemente unter jedem Knoten
- Flute Anfrage nach Zufallselement, Knoten v mit Teilbaumgrößen $n_1, \ldots, n_k$
  - gibt mit Wahrscheinlichkeit $1/(1 + \sum_{i=1}^{k} n_i)$ seinen Wert zurück
  - leitet die Anfrage mit W'keit $n_j/(1 + \sum_{i=1}^{k} n_i)$ in Teilbaum j.



IN $O(D)$ Schritten kann ich auch ein zufälliges Element aus allen Elementen in einem Interval (a, b) ziehen! (klar?)

Ein randomisierter Algorithmus

Mit dem Ziehen von zufälligen Elementen kann man eine binäre Suche nachbauen!

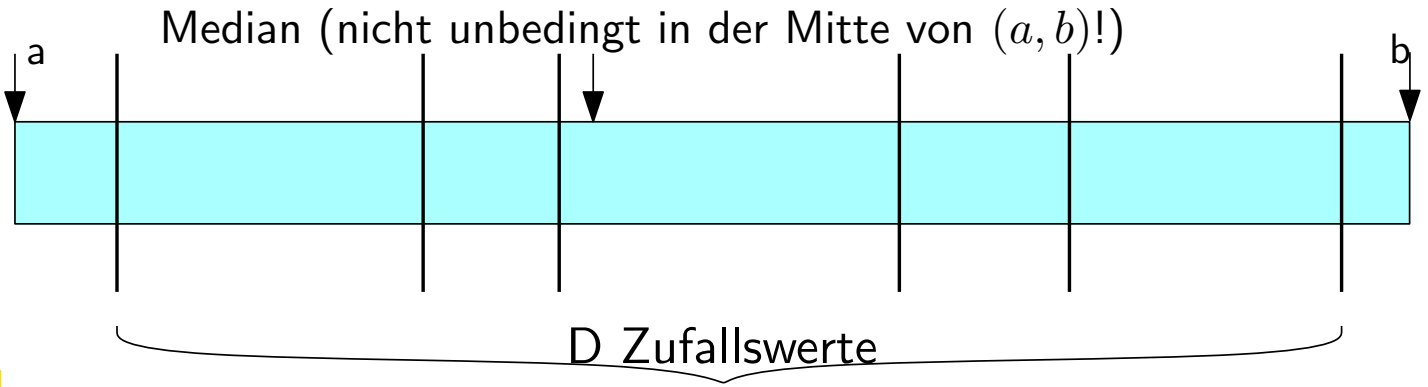
1. setze $a = -\infty, b = \infty$
2. bestimme zufälliges Element x mit Wert (a, b)
3. bestimme, wie viele Elemente größer als x sind
   - ≫ mehr als die Hälfte der Elemente: $a \leftarrow x$
   - ≫ weniger als die Hälfte der Elemente: $b \leftarrow x$
   - ≫ genau die Hälfte: x ist Median
4. gehe zu 2 (Schritte 2,3 sind eine *Runde*!)

Jede Runde dauert nur $\Theta(D)$ Schritte!

Pipelining und Analyse

In $\Theta(D)$ Schritten/Runde kann ich ja noch viel mehr machen!

- starte D Anfragen nach zufälligen Element in D aufeinanderfolgenden Schritten
  - letztes Element kommt nach $2D$ Runden an
- dasselbe bei der Bestimmung der größeren Elemente
- jede Runde reduziert die Anzahl der Elemente in (a, b) *erwartet* auf weniger als $1/D$-tel!



Was ist das wert?

Satz (ohne so richtig formalen Beweis)

Der randomisierte Algorithmus benötigt mit hoher Wahrscheinlichkeit nur $O(\log_D n)$ Runden, also $O(D \log_D n)$ Schritte und $O(Dn \log_D n)$ Nachrichten.

- Für $D \in \Omega(n^d)$ für ein $0 < d \leq 1$ ist

$$\log_D n \leq \log_{cn^d} n = \frac{\log_2 n}{\log_2(cn^d)} = \frac{\log_2 n}{\log_2 c + d \log_2 n} \leq 1/d$$

- $\Rightarrow$ außer bei *fast konstantem* Durchmesser in $O(D)$ Schritten und $O(Dn)$ Nachrichten!
  - das ist offensichtlich asymptotisch zeitoptimal
  - wie sieht es mit dem allgemeinen Fall aus?

Untere Schranke

Bemerkung

Bester bekannter deterministischer Algorithmus hat Laufzeit $O(D \log_D^2 n)$.

- Randomisierter Algorithmus ist *echt besser!*

Satz (beweisen wir gleich!)

Jeder deterministische Algorithmus benötigt zur Bestimmung des Medians im worst-case $\Omega(D \log_D n)$ Schritte.

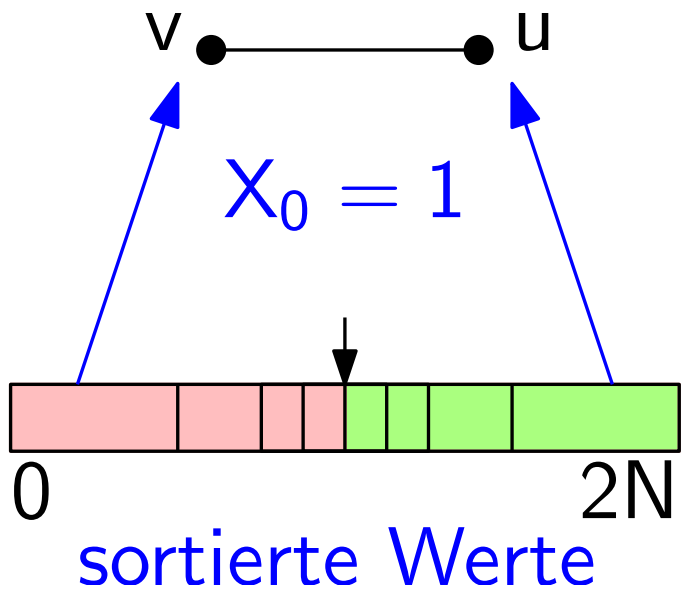
- gilt auch für randomisierte Algorithmen
  - ist *noch* schwerer zu analysieren
- der randomisierte Algorithmus ist also auch allgemein optimal!

Eine erste Näherung

Lemma

Zwei Knoten, die jeweils N Elemente halten und pro Runde B Elemente und beliebige Zusatzinformationen austauschen dürfen, benötigen im worst-case $\Omega(\log_{2B} N)$ Runden, um den gemeinsamen Median zu ermitteln.

- Nimm $2N$ beliebige Elemente, $N = 2^b$
- würfle b Zufallswerte $X_i \in \{0, 1\}$
  - Wenn $X_0 = 1$, teile $N/2$ größte Elemente zu u und $N/2$ kleinste zu v, sonst umgekehrt!
  - Wenn $X_1 = 1$, teile $N/4$ nächstgrößte Elemente zu u, $N/4$ nächstkleinste Elemente zu v, sonst umgekehrt, usw.

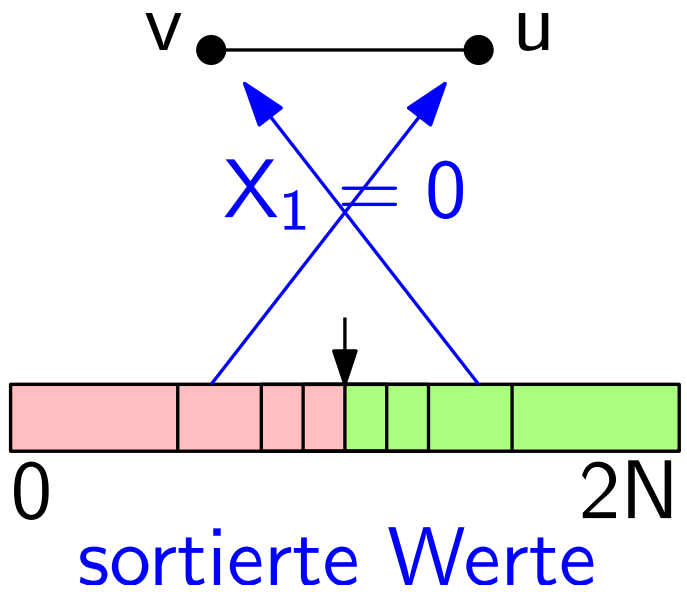


Eine erste Näherung

Lemma

Zwei Knoten, die jeweils N Elemente halten und pro Runde B Elemente und beliebige Zusatzinformationen austauschen dürfen, benötigen im worst-case $\Omega(\log_{2B} N)$ Runden, um den gemeinsamen Median zu ermitteln.

- Nimm $2N$ beliebige Elemente, $N = 2^b$
- würfle b Zufallswerte $X_i \in \{0, 1\}$
  - Wenn $X_0 = 1$, teile $N/2$ größte Elemente zu u und $N/2$ kleinste zu v, sonst umgekehrt!
  - Wenn $X_1 = 1$, teile $N/4$ nächstgrößte Elemente zu u, $N/4$ nächstkleinste Elemente zu v, sonst umgekehrt, usw.



Zwei kleine Beobachtungen..

Beobachtung I

Wenn ein Knoten den Median kennt, kennt er alle X_i!

- Zähle die Elemente, die in u größer sind.
  - $N/2$ oder mehr $\Rightarrow X_0 = 1$...

Beobachtung II

Die Zusatzinformationen können sich nur aus den Vergleichen zwischen Elementen ergeben, die einem Knoten bekannt sind.

- Vergleiche zwischen us Elementen und Elementen, die v schon geschickt hat
- Vergleiche zwischen vs Elementen und Elementen, die u schon geschickt hat

..und eine große Erkenntnis

Lemma

Zwei Knoten, die jeweils N Elemente halten und pro Runde B Elemente und beliebige Zusatzinformationen austauschen dürfen, benötigen im worst-case $\Omega(\log_{2B} N)$ Runden, um den gemeinsamen Median zu ermitteln.

Beweisskizze (nur für $B = 1$)

- betrachte „abwechselndes“ Senden von einzelnen Elementen
- egal, welches Element u an v zuerst schickt, im schlimmsten Fall enthält es keine Information über X_0 hinaus!
  - z.B.: als erstes schickt u Element aus seiner kleineren Hälfte, schlecht für $X_0 = 0$
  - danach gehts weiter für X_1

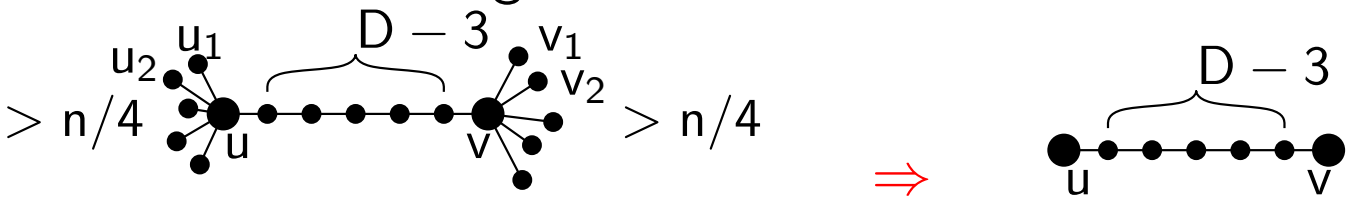
$\Rightarrow$ mindestens $\log_2 N$ Elemente müssen ausgetauscht werden

Zurück zur unteren Schranke

Lemma (im Prinzip ist das unser Satz)

Für jedes $D \geq 3$ gibt es eine Instanz, in der die Bestimmung des Medians $\Omega(D \log_D n)$ Schritte benötigt.

- für $D \in \Theta(n)$ ist das trivial (wir erinnern uns)
- also nehmen wir folgende Instanz



- die $D - 3$ Knoten enhalten keine Elemente (das macht es nur leichter!)
- in Runde 1 kann nichts sinnvolleres passieren, als das u und v die Elemente von allen Satelliten erhalten

Ein letzter Schritt

Lemma (im Prinzip ist das unser Satz)

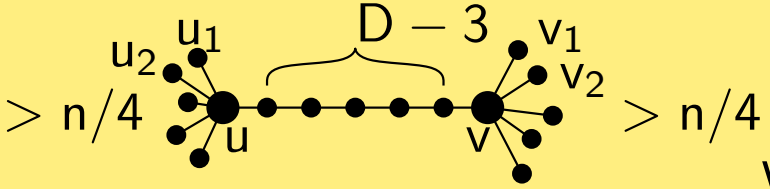
Für jedes $D \geq 3$ gibt es eine Instanz, in der die Bestimmung des Medians $\Omega(D \log_D n)$ Schritte benötigt.

Wir haben das auf eine Instanz zurückgeführt, in der zwei Knoten mind. $n/4$ Elemente halten und durch einen Pfad der Länge $D-2$ getrennt sind.

- $\gg$ Anschnallen! Wir nehmen an, dass u und v in je $D-2$ Runden $D-2$ Elemente an alle anderen Knoten schicken kann.
  - $\gg$ Daraus könnten u und v das Verhalten der Knoten auf dem Pfad sogar rekonstruieren!
  - $\Rightarrow$ Die Annahme kann das Problem nur vereinfachen!

und eine letzte Schlussfolgerung

Wir machen es uns für diese Instanz nur leichter,



wenn u und v alle Werte der Satelliten kennen *und* wenn man annimmt, dass die beiden sich in Runden je $D-2$ Schritten genau $D-2$ Elemente zuschicken können.

- Wir wissen, dass zwei Knoten mit je $N = n/4$ Elementen, die sich pro Runde $B = D-2$ Elemente schicken können,

$$\Omega(\log_{2D-4} n/4) \in \Omega(\log_D n)$$

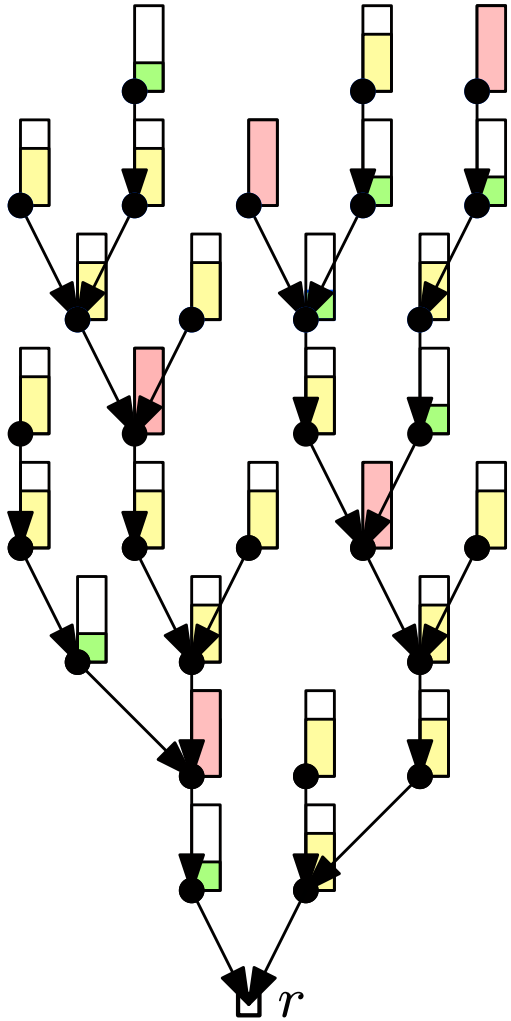
  Runden benötigen.
- unsere Runden hier dauern $D-2$ Schritte
- das beweist die untere Schranke von $\Omega(D \log_D n)$!

Heute

- Data Gathering (kurze Einführung)
  - Ein Name, viele Probleme
- Aggregation von Funktionen auf Sensorwerten
  - Anfragen und Funktionsklassen
  - Randomisierung und Schranken
- Scheduling von Data Gathering Trees
  - Einsammeln von großen Datenmengen
  - Energiesparen durch Koordination des Funkkanals

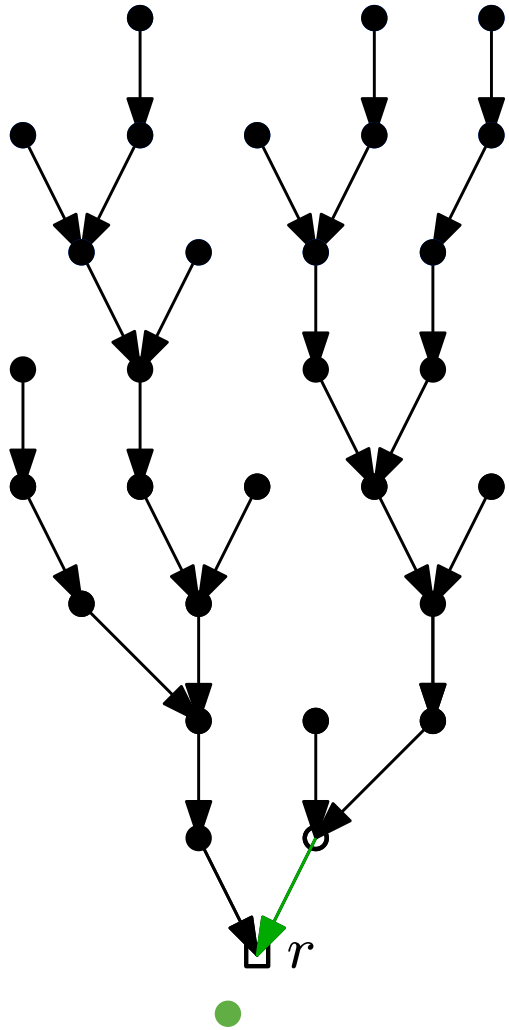
Einsammeln von Daten, Szenario

- Sensorknoten sammeln Daten
  - viele Pakete je Knoten, $\sum = N$
- Bei Bedarf wird eine Anfrage geflutet
  - spannt Kürzeste-Wege-Baum auf
  - Daten sollen schnell zur Senke fließen



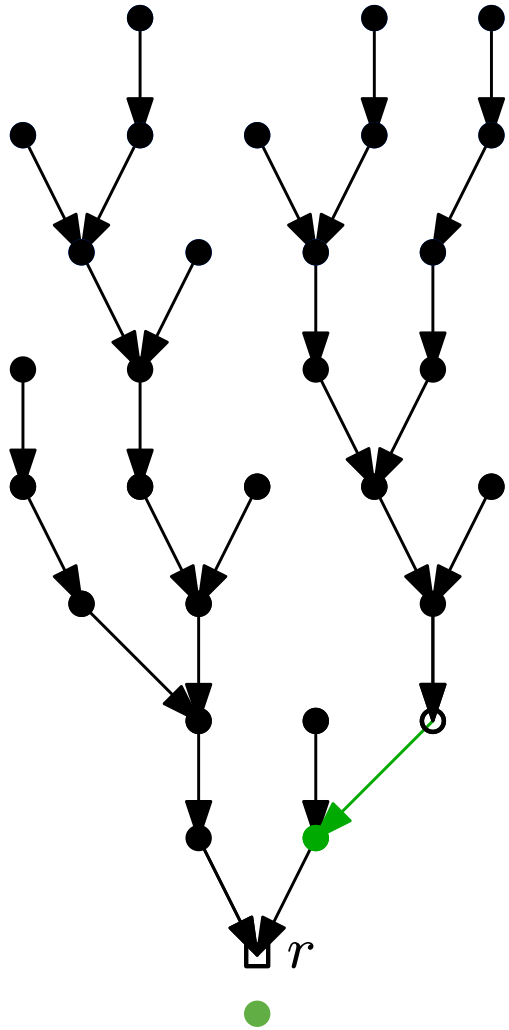
Einsammeln von Daten, Szenario

- Sensorknoten sammeln Daten
  - viele Pakete je Knoten, $\sum = N$
- Bei Bedarf wird eine Anfrage geflutet
  - spannt Kürzeste-Wege-Baum auf
  - Daten sollen schnell zur Senke fließen
- Einschränkungen:
  - Knoten haben keinen/wenig zusätzlichen Speicher (Rohdaten oft extern gespeichert)
  - Übertragungen können nicht beliebig parallel stattfinden, ohne sich zu stören (Interferenzmodell)



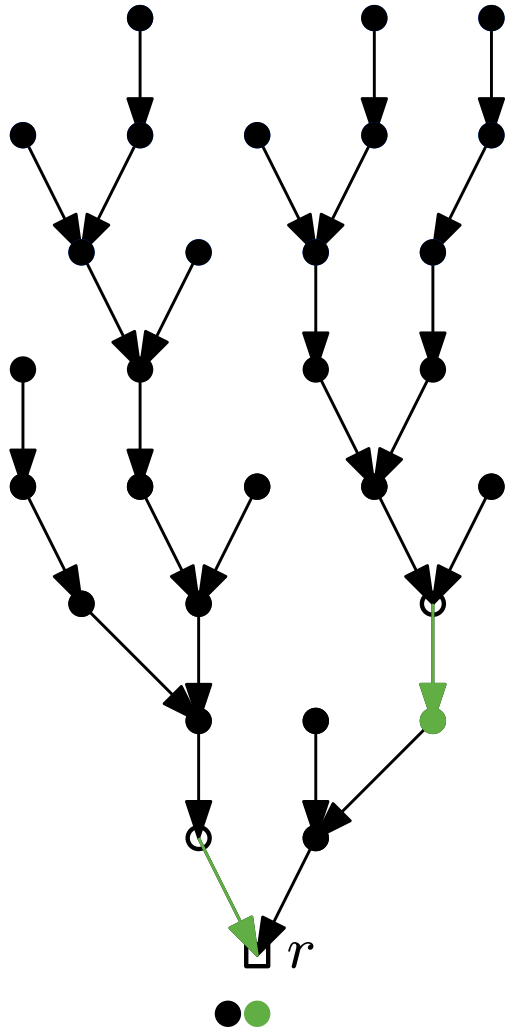
Einsammeln von Daten, Szenario

- Sensorknoten sammeln Daten
  - viele Pakete je Knoten, $\sum = N$
- Bei Bedarf wird eine Anfrage geflutet
  - spannt Kürzeste-Wege-Baum auf
  - Daten sollen schnell zur Senke fließen
- Einschränkungen:
  - Knoten haben keinen/wenig zusätzlichen Speicher (Rohdaten oft extern gespeichert)
  - Übertragungen können nicht beliebig parallel stattfinden, ohne sich zu stören (Interferenzmodell)



Einsammeln von Daten, Szenario

- Sensorknoten sammeln Daten
  - viele Pakete je Knoten, $\sum = N$
- Bei Bedarf wird eine Anfrage geflutet
  - spannt Kürzeste-Wege-Baum auf
  - Daten sollen schnell zur Senke fließen
- Einschränkungen:
  - Knoten haben keinen/wenig zusätzlichen Speicher (Rohdaten oft extern gespeichert)
  - Übertragungen können nicht beliebig parallel stattfinden, ohne sich zu stören (Interferenzmodell)



Noch mehr Einschränkungen

Die meiste Energie in Sensornetzen geht nicht verloren, weil so viel Daten zu übertragen sind, sondern, weil Knoten zu oft wach sind und den Kanal überwachen!

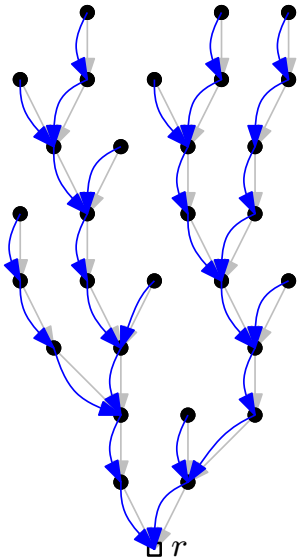
- ≫ Feste *Schedules* verhindern das!
- ≫ Jeder Knoten weiß, wann er hören und wann er senden muss
- ≫ Henne-Ei-Problem: Wie verabredet man ein Schedule?

Schedule-Set-Up

Modell

In einem Anfragebaum darf jeder Knoten einmalig ein Paket mit $O(\log N)$ Bits an jeden Nachbarn schicken, um sich auf ein Schedule zu einigen.

- Einzig sinnvolles Schema:
  - Convergecast zur Senke hin
  - Broadcast von der Senke weg

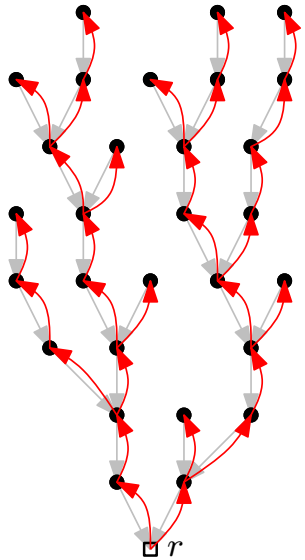


Schedule-Set-Up

Modell

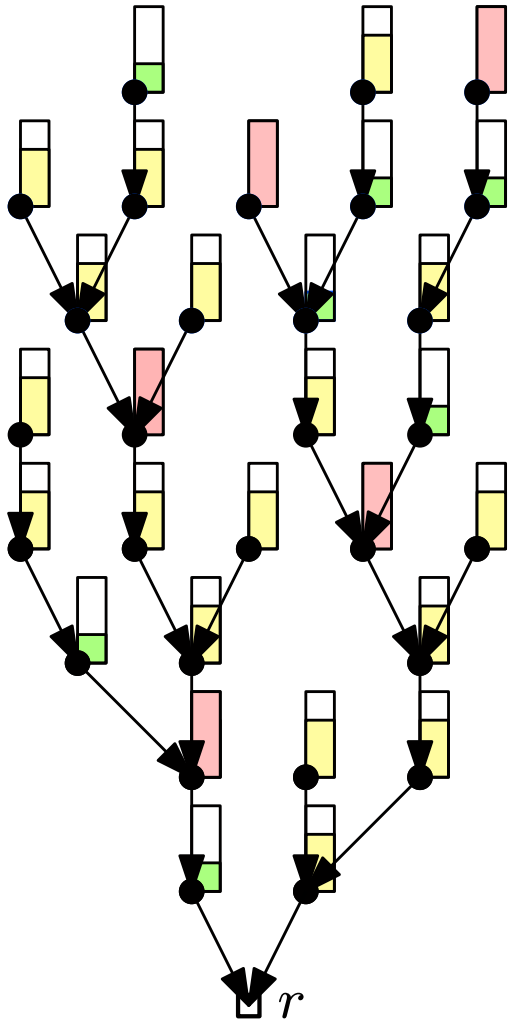
In einem Anfragebaum darf jeder Knoten einmalig ein Paket mit $O(\log N)$ Bits an jeden Nachbarn schicken, um sich auf ein Schedule zu einigen.

- Einzig sinnvolles Schema:
  - Convergecast zur Senke hin
  - Broadcast von der Senke weg



Erinnerung/Wiedereinstieg

- Knoten haben Pakete
  - ggf viele pro Knoten, $\sum = N$
- Kürzeste-Wege-Baum (in Hops) gegeben
- gesucht: Zeitplan, wer wann sendet/weiterleitet
  - leicht zu verabreden (das werden wir nicht zeigen)
  - kein Knoten soll mehr als ein Paket puffern müssen
  - parallele Übertragungen dürfen sich nicht stören

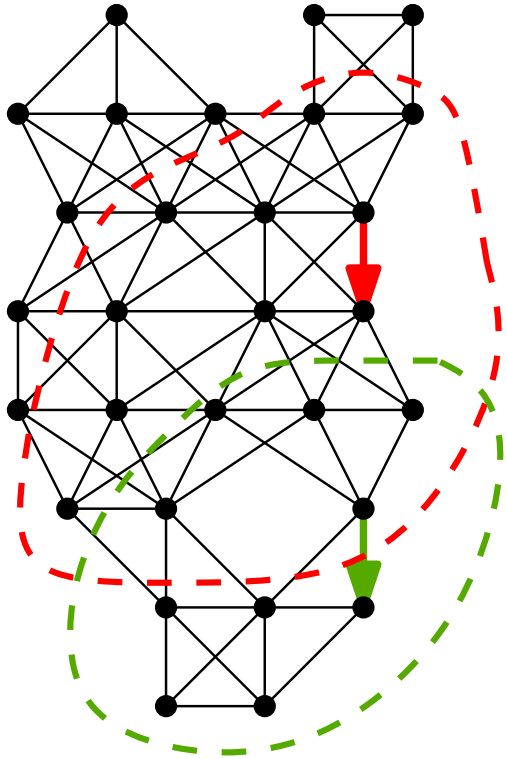


k-Hop-Interferenz

k-Hop-Interferenz

Im Modell der *k*-Hop-Interferenz kann ein Knoten eine Nachricht eines Nachbarn genau dann empfangen, wenn der Nachbar der einzige aktive Sender im Radius von *k* Hops ist.

≫ Wir suchen nur Schedules, die in diesem Modell kollisionsfrei sind!

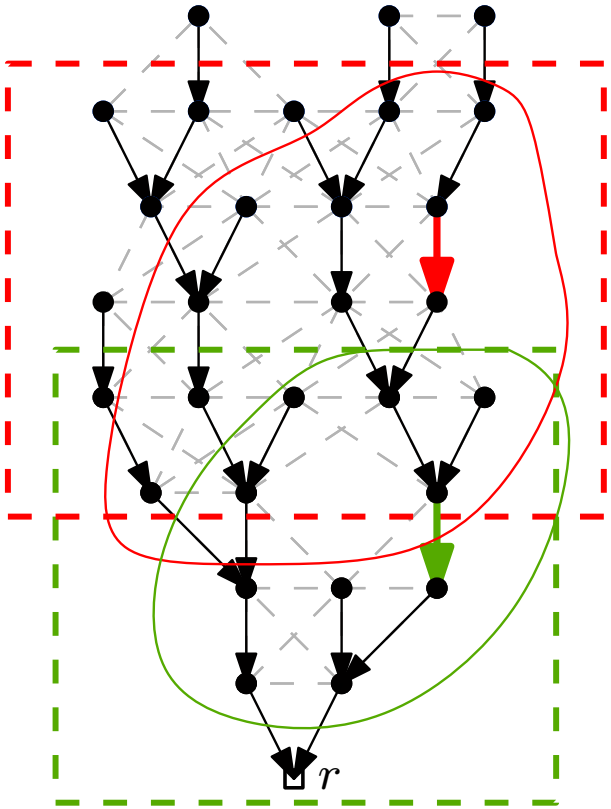


Bäume mit k-lagenbeschränkter Interferenz

k-lagenbeschränkte Interferenz

Ein Anfragebaum hat k-lagenbeschränkte Interferenz, wenn ein Knoten u die Nachricht eines Kindes immer empfangen kann, wenn v der einzige Aktive Sender in den Lagen $h_u - k \ldots h_u + k$ ist.

- Kürzeste-Wege-Bäume haben bei k-Hop-Interferenz k-lagenbeschränkte Interferenz!
- Jeder Sender, der einen Knoten u stören könnte, hat maximal Abstand k im Graphen und kann dann auch nur eine Höhendifferenz von k im Baum haben!



Untere Schranke

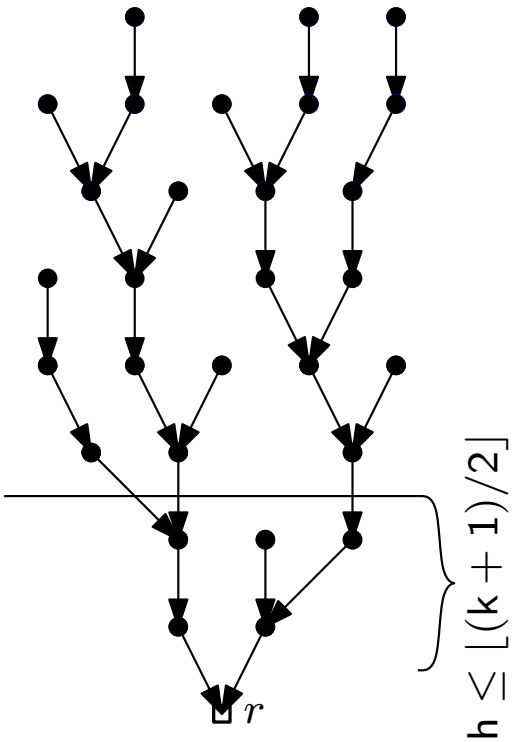
Lemma

In jedem gültigen Schedule darf in der $\lfloor (k+1)/2 \rfloor$-Nachbarschaft der Senke zu jedem Zeitpunkt nur maximal ein aktiver Sender sein.

- Beweis:
  - Die Empfänger haben jeweils nur Abstand $\lfloor (k+1)/2 \rfloor - 1$ zur Senke.
  - damit haben „falsche“ Sender-Empfänger-Paare Abstand

$$2 \cdot \lfloor (k+1)/2 \rfloor - 1 \leq k$$

  - $\Rightarrow$ Die würden sich stören!



k = 3

Untere Schranke

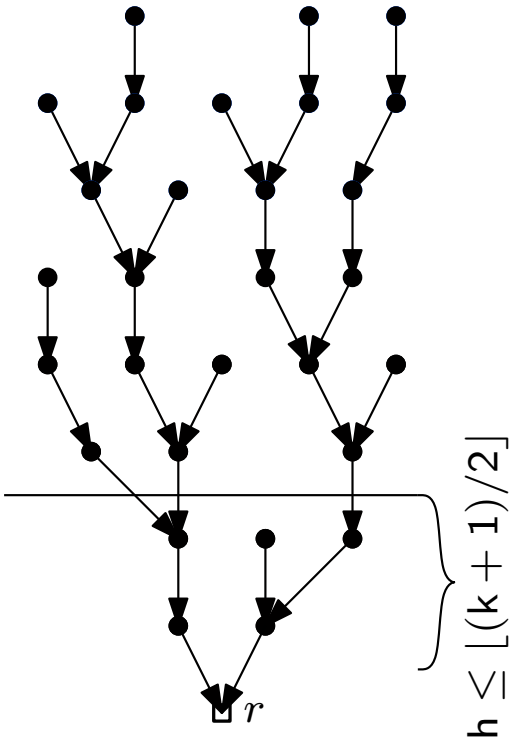
Lemma

Bezeichnet h_u die Entfernung von Knoten u zur Senke und p_u die Anzahl der Pakete, die u am Anfang enthält, benötigt jedes gültige Schedule mindestens

$$\sum_{u \in V} (p_u \cdot \min\{h_u, \lfloor (k+1)/2 \rfloor\})$$

Zeiteinheiten.

- Jedes Paket von einem Knoten u in Entfernung h_u zur Senke schlägt mit mindestens $\min\{h_u, \lfloor (k+1)/2 \rfloor\}$ Zeiteinheiten „zu Buche“, in denen es von einem Knoten in der $\lfloor (k+1)/2 \rfloor$-Nachbarschaft der Senke gesendet wird!



k = 3

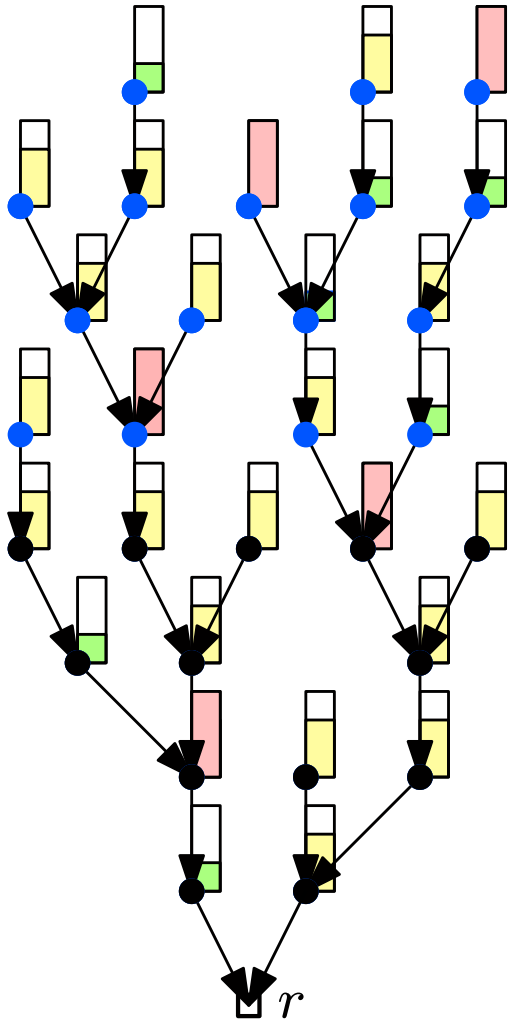
2-Phasen-Algorithmus

Nur Pakete, die *dicht* an der Senke sind, schlagen in unterer Schranke mit ihrer Entfernung zu Buche, weit entfernte Pakete trotzdem nur mit $\lfloor (k+1)/2 \rfloor$ Zeitslots.

Idee!

Das können wir nachbauen! Wir geben zwei Schedules an, die nacheinander ablaufen.

1. Einsammeln aller weit entfernter Pakete mit „Pipelining“
2. Einsammeln aller Pakete in der $k+2$-Hop-Nachbarschaft der Senke *ohne parallele Übertragungen*



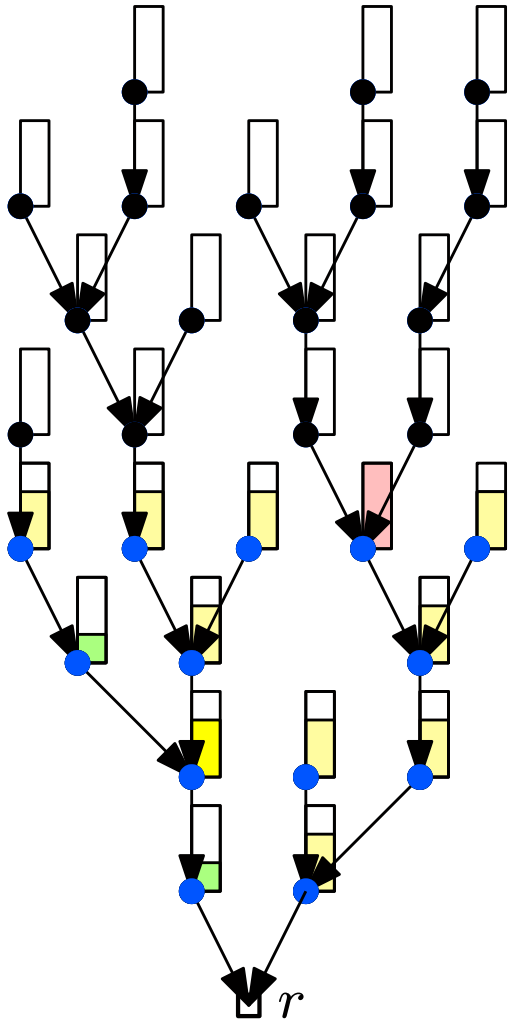
2-Phasen-Algorithmus

Nur Pakete, die *dicht* an der Senke sind, schlagen in unterer Schranke mit ihrer Entfernung zu Buche, weit entfernte Pakete trotzdem nur mit $\lfloor (k+1)/2 \rfloor$ Zeitslots.

Idee!

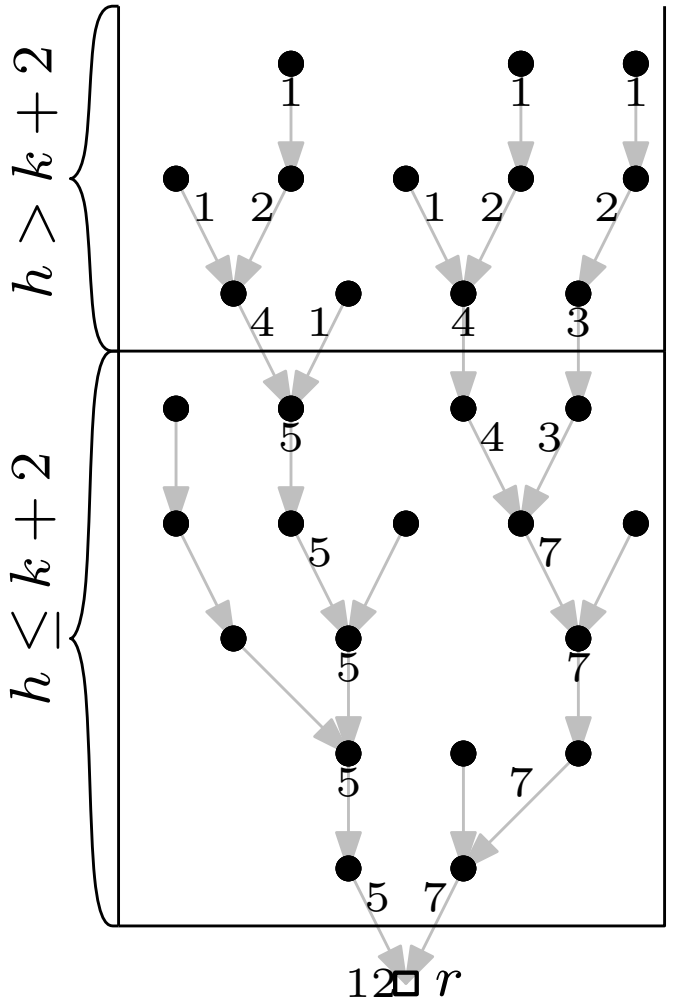
Das können wir nachbauen! Wir geben zwei Schedules an, die nacheinander ablaufen.

1. Einsammeln aller weit entfernter Pakete mit „Pipelining“
2. Einsammeln aller Pakete in der $k+2$-Hop-Nachbarschaft der Senke *ohne parallele Übertragungen*



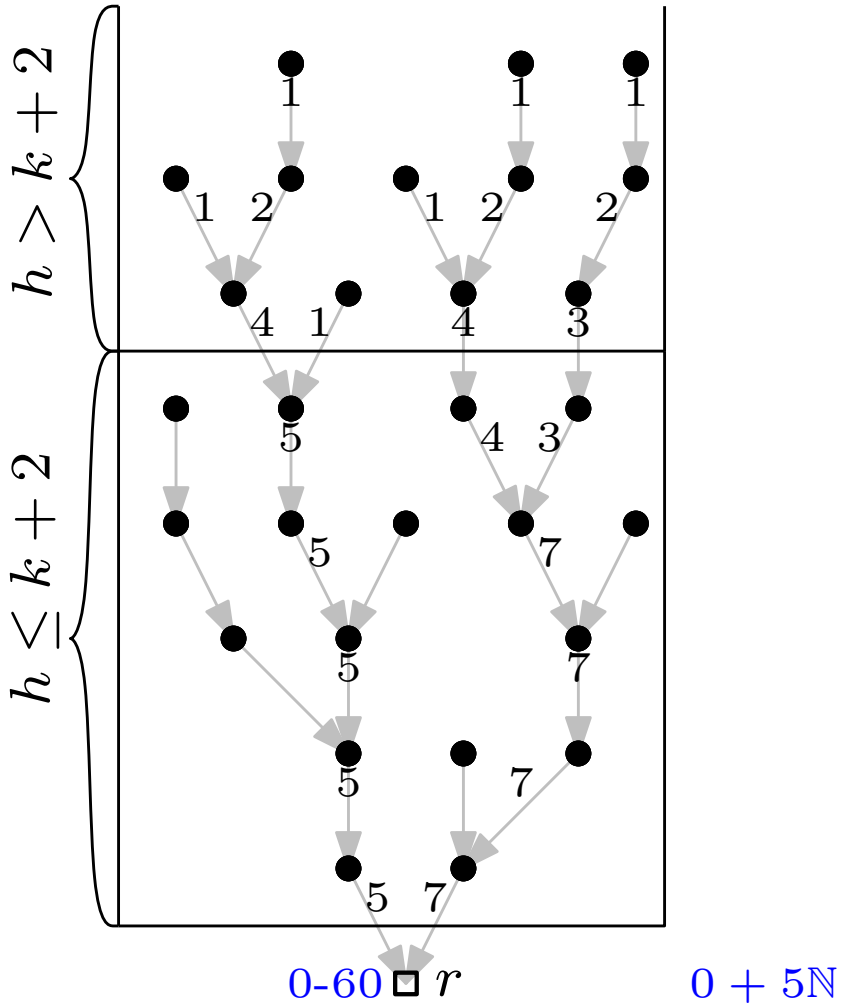
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet



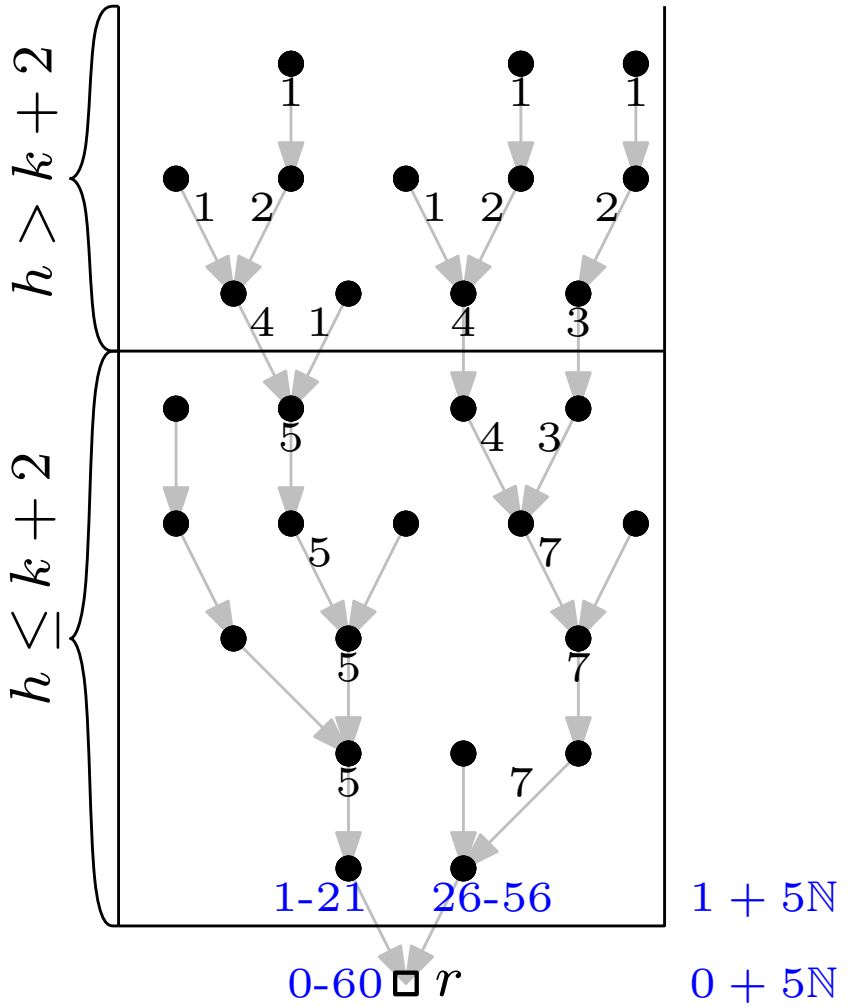
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



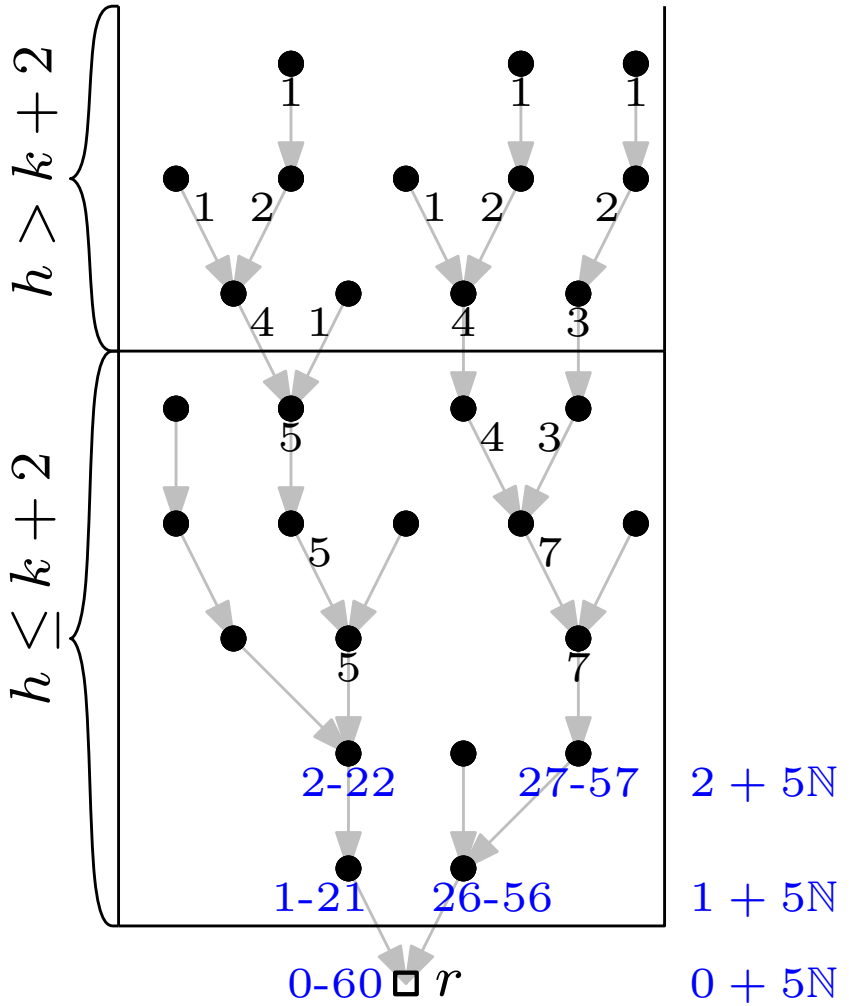
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



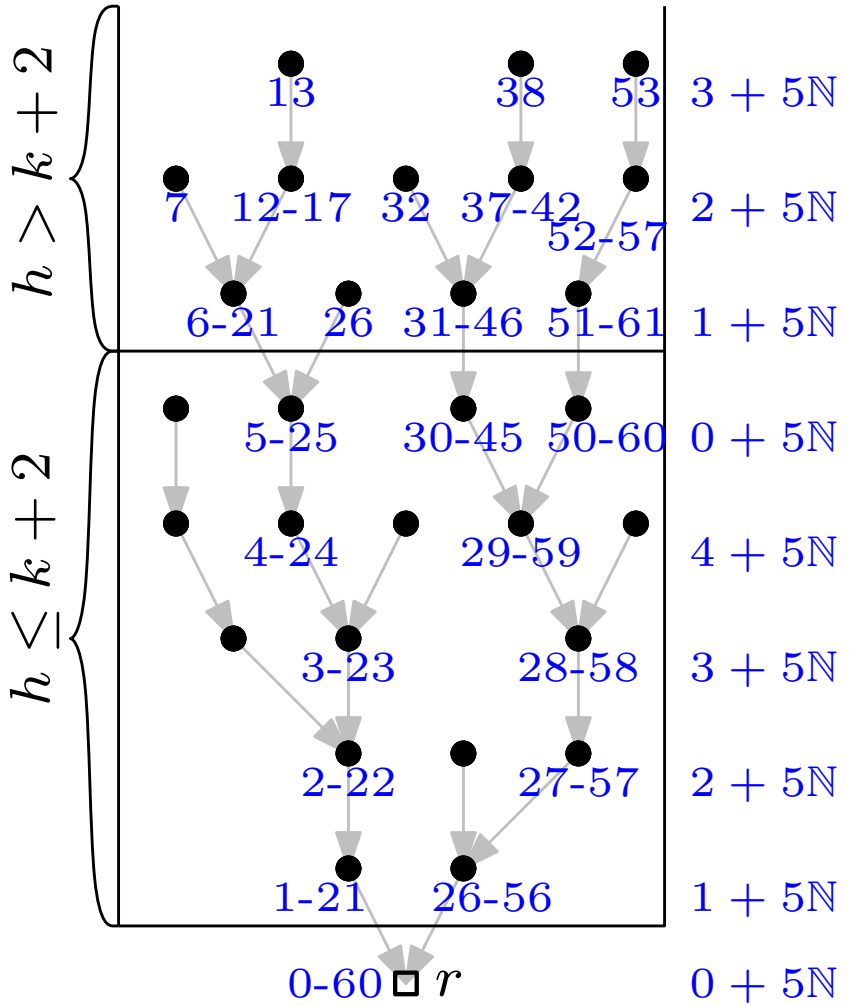
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



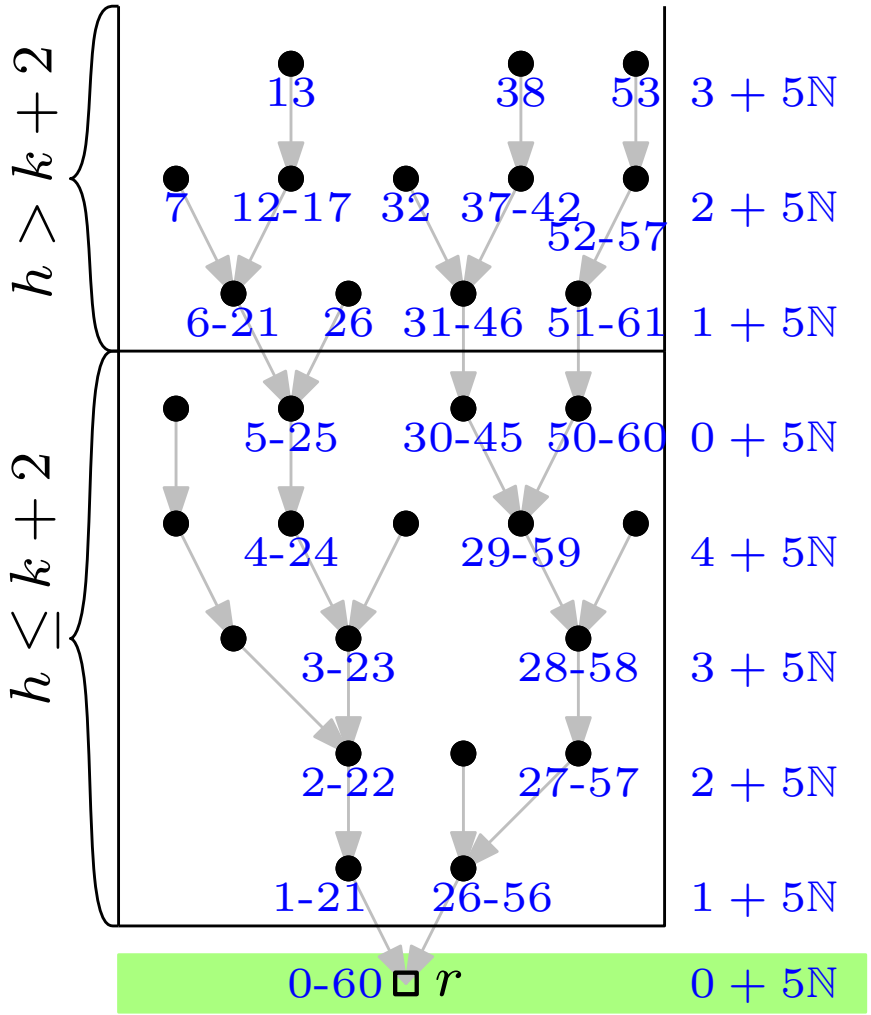
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



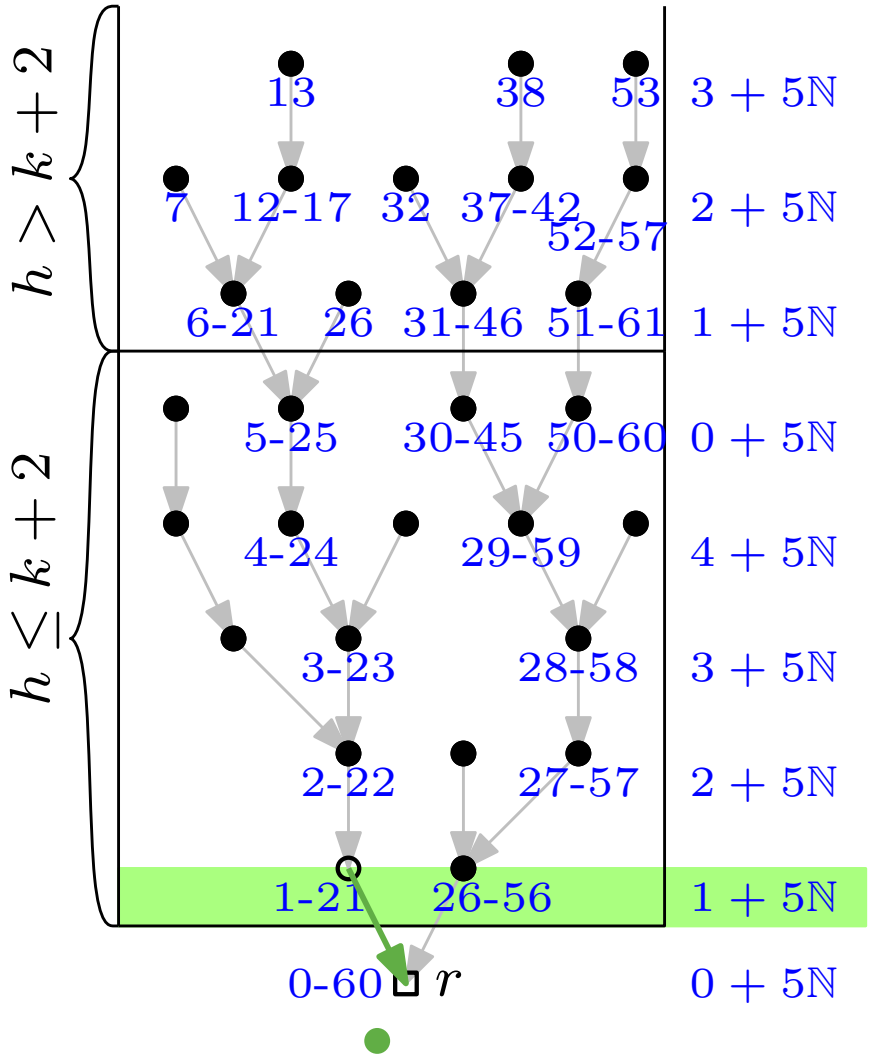
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



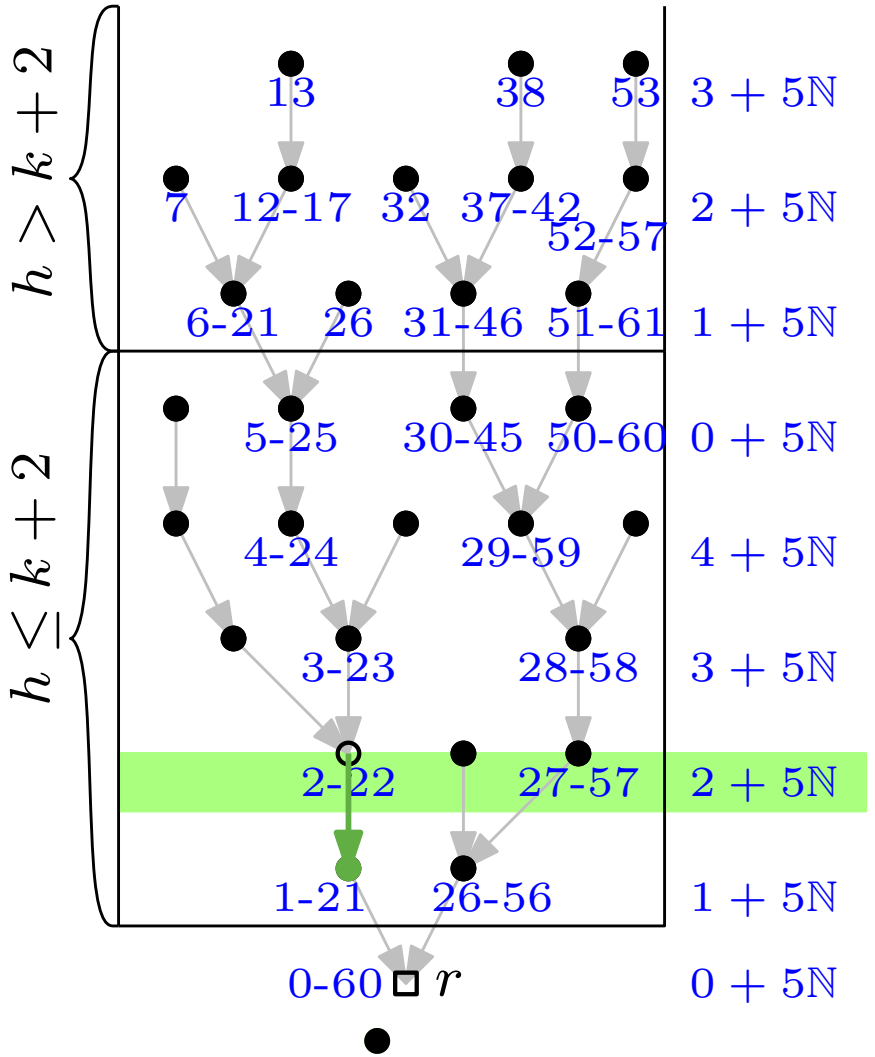
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



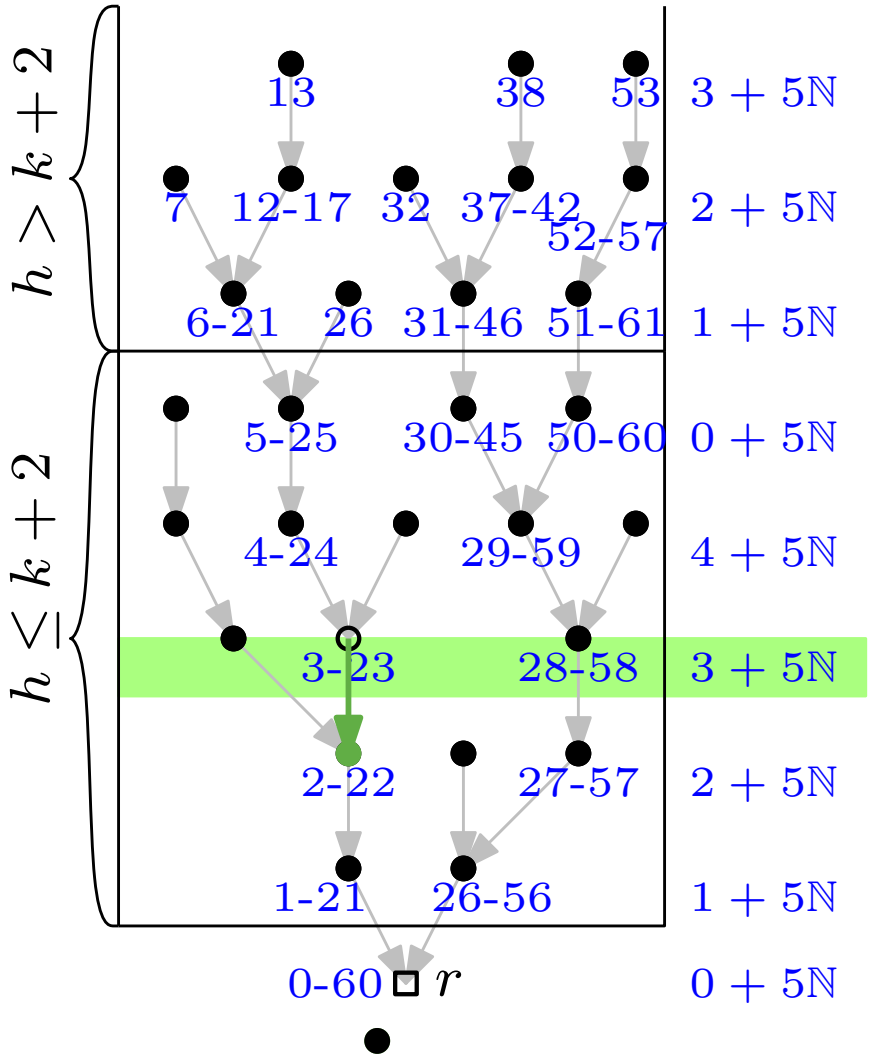
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



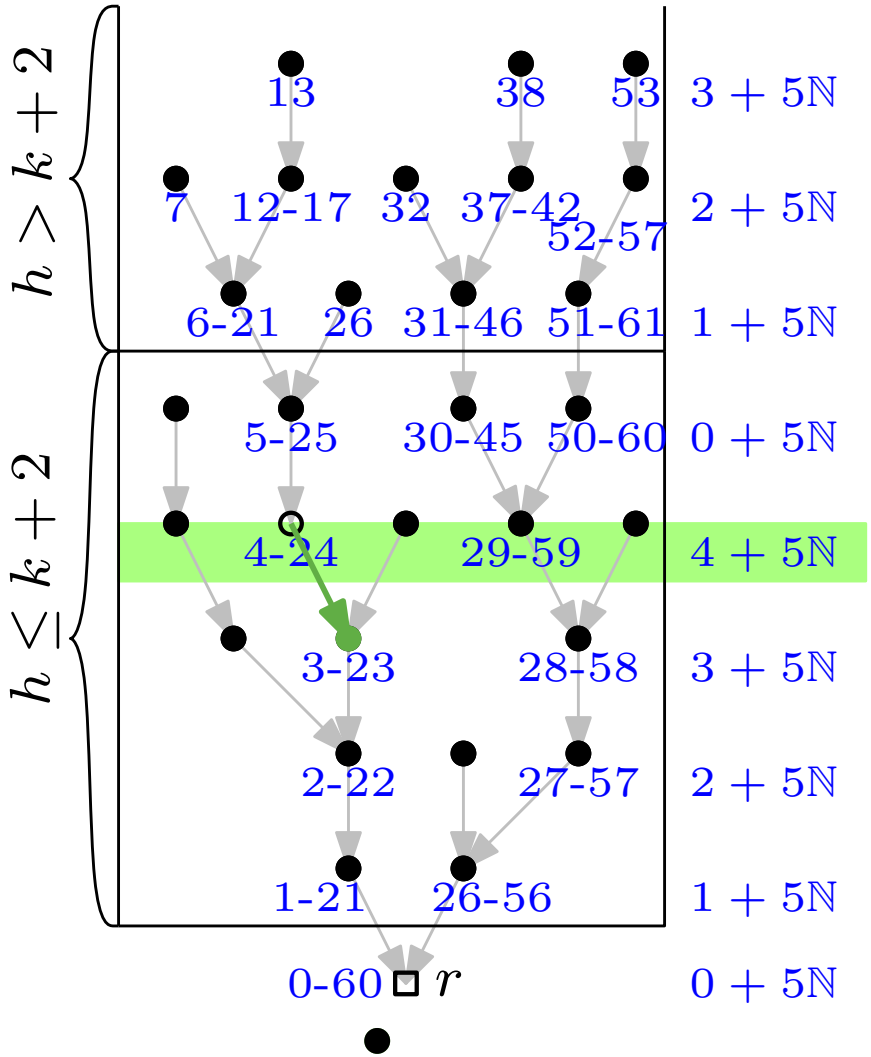
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



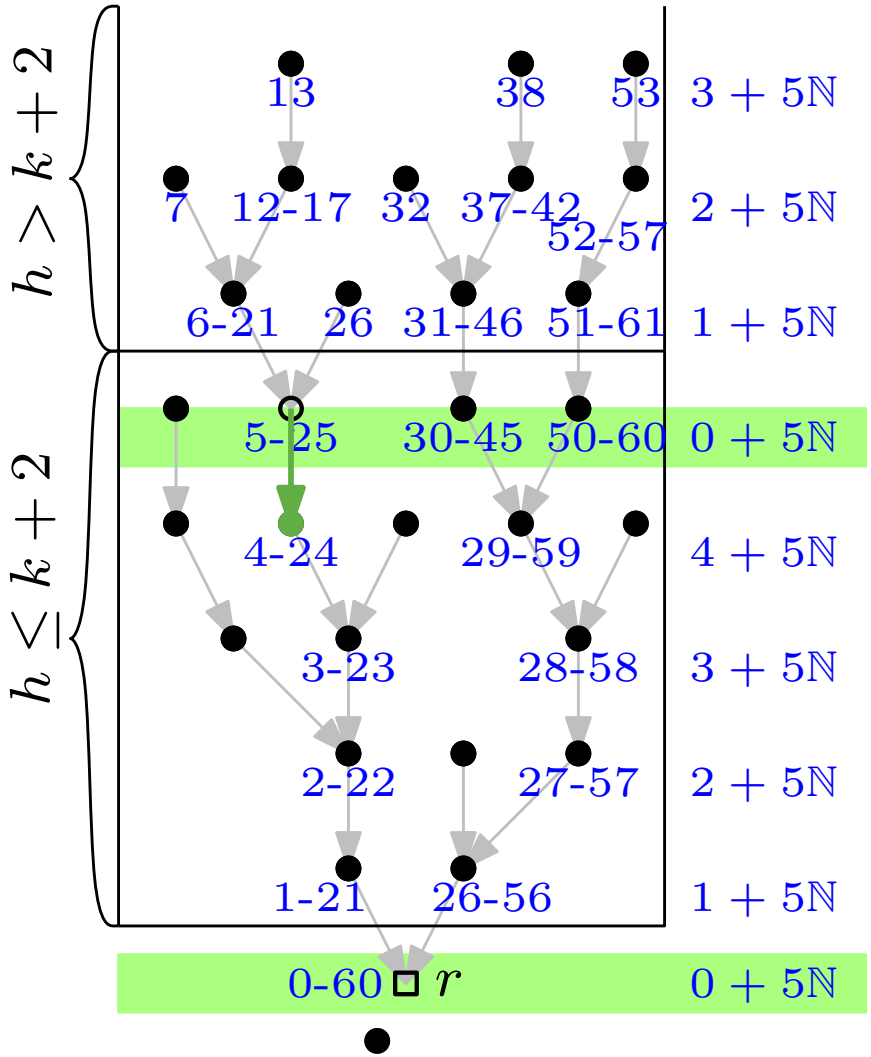
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



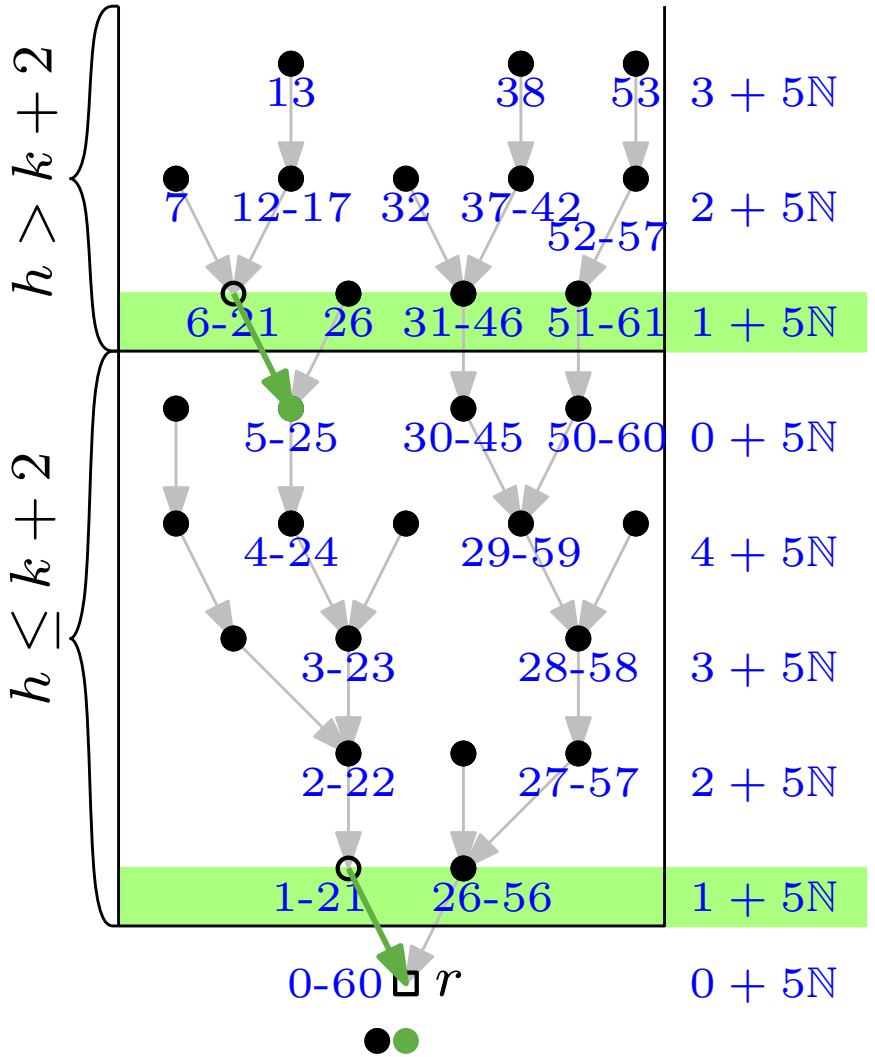
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



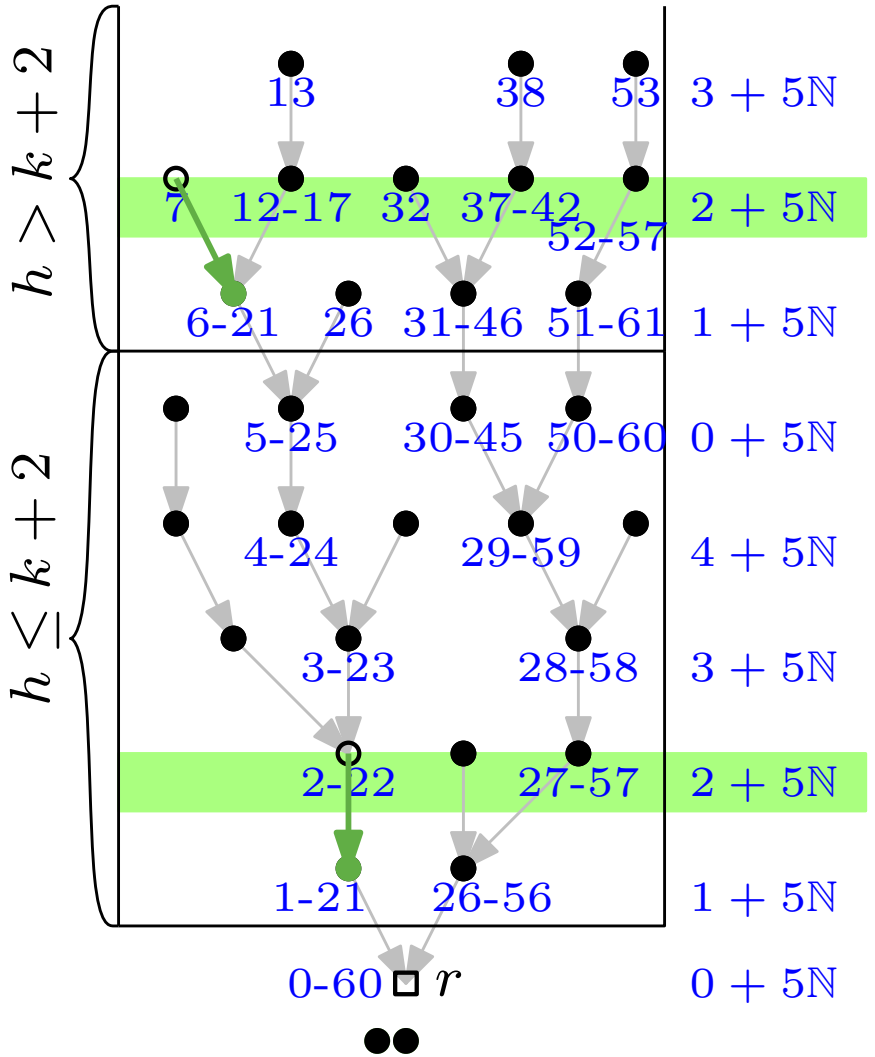
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot



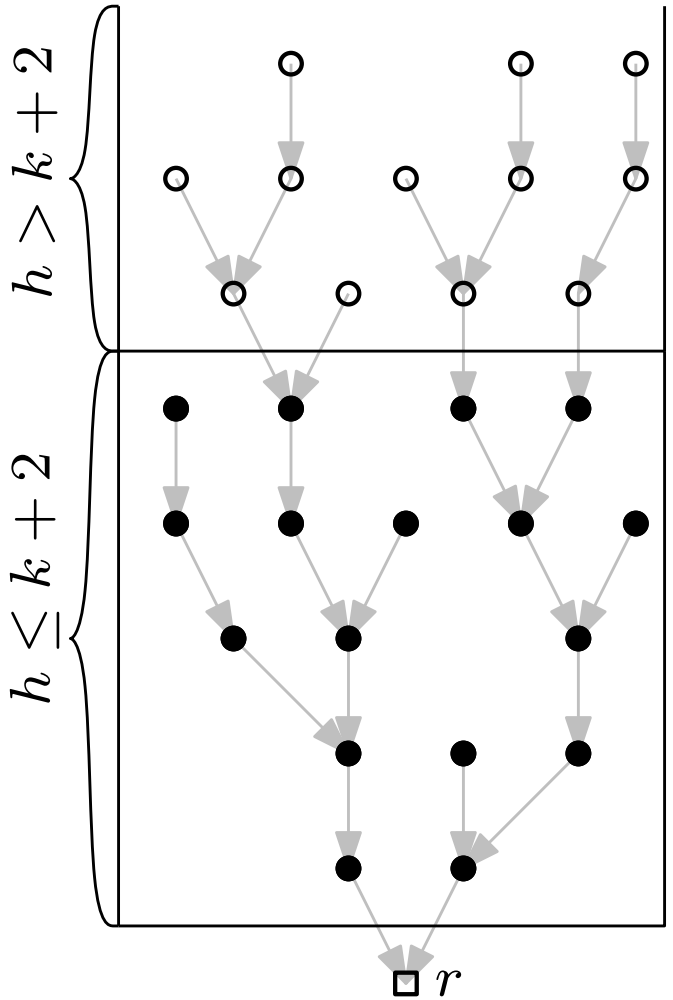
Phase I - Pipelining entfernter Pakete

- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot
- Für jedes Paket werden $k+2$ Slots benötigt



Phase I - Pipelining entfernter Pakete

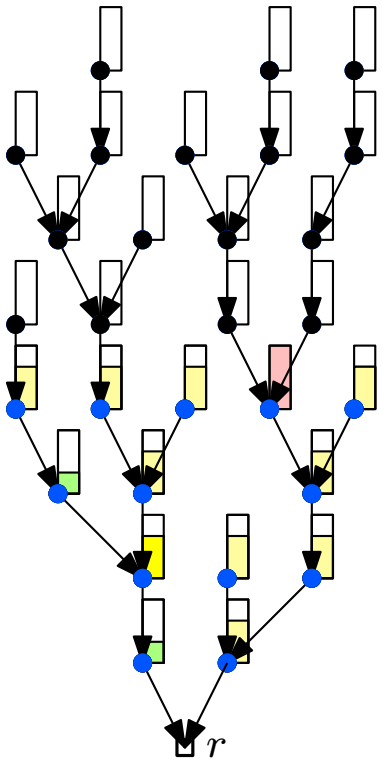
- Jeder Knoten lernt, wie viele Pakete mit Höhe über $k+2$ er sendet/weiterleitet
- Knoten senden in einem Intervall alle $k+2$ Slots
  - die Senke sendet "virtuell" in den Slots $0, k+2, \ldots$
  - ein Knoten, der sendet, empfängt im nächsten Slot
- Für jedes Paket werden $k+2$ Slots benötigt



Phase II - Einsammeln naher Pakete

Lemma (ohne Beweis)

Für Bäume mit konstant beschränkter Höhe kann ein Schedule verabredet werden, in dem in jedem Zeitschritt genau ein Paket übertragen wird!



- zentral ist das ohnehin einfach
  - jeweils linkester dichtester Knoten mit Paketen schickt Paket auf den Weg..
- das läßt sich auch verteilt durchnumerieren (ohne Beweis)

Analyse

Satz

Der 2-Phasen-Algorithmus benötigt höchstens 4 mal soviele Zeitschritte wie ein optimales Schedule!

- $\gg$ Phase I: Jedes Paket mit $h_u > k+2$ schlägt mit $k+2$ Zeitschritten zu Buche (Pipelining!)
- $\gg$ Phase II: Jedes Paket mit $h_u \leq k+2$ schlägt mit h_u Zeitschritten zu Buche (exklusives Senden)
- $\Rightarrow$ Gesamtzeit:

$$\sum_{u \in V} (p_u \cdot \min\{h_u, k+2\})$$

- $\gg$ $\min\{h_u, k+2\} \leq \frac{k+2}{\lfloor (k+1)/2 \rfloor} \min\{h_u, \lfloor (k+1)/2 \rfloor\}$
- $\gg$ $\frac{k+2}{\lfloor (k+1)/2 \rfloor} \leq 4$

Zum Mitnehmen

- Data Gathering ist ein Hauptzweck vieler Sensornetze
  - leider auch eine vielschichtige Herausforderung
- Datenaggregation
  - geschickte Verarbeitung im Netz kann Anfragen beschleunigen
  - bei Primitiven einigermaßen gut verstanden
  - bei komplexeren (Datenbank-)Operationen noch viele offene Fragen!
- Koordination/Scheduling
  - Klare Schedules sparen Mithören und unnötige Aktivzeiten

Literatur

1. Fabian Kuhn, Thomas Locher, Roger Wattenhofer: *Tight Bounds for Distributed Selection*. In: 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2007
2. Bastian Katz, Steffen Mecke, Dorothea Wagner: *Efficient Scheduling of Data Harvesting Trees*. In: Proceedings of the 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS), 2008