Online Information and ACK Aggregation

Master Thesis

Thibaut Britz
<britzt@student.ethz.ch>

Advisors:
Prof. Dr. Roger Wattenhofer
Stefan Schmid
Yvonne Anne Oswald

Distributed Computing Group
Computer Engineering and Networks Laboratory (TIK)
Department of Information Technology and Electrical Engineering

July 20, 2007

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Contents

1 Introduction

The aggregation of information is a fundamental operation in distributed computing. For example, in sensor networks, large numbers of nodes make measurements at their location and typically send this data back to a source. Very often, it is not necessary that the source knows the individual data of each node, but is only interested in an aggregated form thereof, e.g., the sum, the minimum, or the spikes in the data. By aggregating and delaying data cleverly on intermediary nodes, communication costs and energy can be saved, while keeping the source node up to date.
So far, information aggregation has mainly been studied in static environments.
This thesis investigates online information aggregation algorithms in dynamic distributed environments where node values change over time. We will propose three basic aggregation algorithms, prove their upper and lower bound on general trees and simulate our information aggregation algorithms on real world sensor data. Later on we will propose an alternative upper bound for the tree based distributed acknowledge problem, which is a special case of our information aggregation model.

2
Related Work

There has been some related work in the field of TCP acknowledgment which focuses mostly on the objective function of minimizing the number of acknowledgments sent and the sum of delay costs incurred for all the packets. Dooly, Goldman and Scott [1] propose a 2 competitive online algorithm for a single link (one leaf node and one root node connected through one edge) and show that the offline case can be optimally solved by using dynamic programming. Karlin, Kenyon, and Randall [2] propose a randomized online algorithm that achieves a competitive ratio of $\frac{e}{e-1}$ for the single link case as well. Sanjeev Khanna, Joseph Naor, and Dan Raz [3] study the problem of aggregating TCP acknowledge packets in a tree and propose a $O(h\ log(\alpha))$-competitive distributed algorithm, where α are the total communication costs in the tree. Brito, Koutsoupias and Vaya [4] propose general upper and lower bounds for the distributed asynchronous case in a tree.
Susanne Albers and Helge Bals [5] investigate a different objective function that minimizes the number of acknowledgments sent and the maximum delay incurred for any of these packets. They propose a $\frac{\pi}{6}$-competitive deterministic algorithm for the single link, which is also a lower bound for any deterministic online algorithm.
A different approach is taken by Frederiksen and Larsen [6]. They study the single link case and their objective function consists of measuring the total time elapsed while packets are waiting at the leaf node, but have not yet been delivered. They propose a 1.397-competitive randomized and a 1.618-competitive deterministic algorithm for this problem.
A similar model is introduced by Papadimitriou and Servan-Schreiber [7, 8]. Instead of leaves acknowledging packets, leaves aggregate information which has to be sent to the root as soon as possible. The arrival of messages is modeled by a poisson process and the messages are allowed to be aggregated as they are sent up the tree. They try to minimize the number of messages sent and the delay cost incured for all messages.
Kempe, Dobra and Gehrkey [9] analyze gossip-based[1] protocols for the computations of sums, averages, random samples, quantiles, and other aggregation functions in undirected graphs and analyze their speed of convergence. Mosk-Aoyama and Shah [10] propose a distributed randomized algorithm for computing separable functions based on properties of exponential random

[1] In gossip-based protocols, each node randomly contacts one or more nodes in each round, and exchanges information with these nodes.

variables in undirected graphs and analyze its speed of convergence as well.
Massouli and, Le Merrer, Kermarrec and Ganesh [11] analyze the complexity and the accuracy of aggregating data in undirected graphs by using a random walk technique. Patt-Shamir [12] proposes a deterministic protocol for computing the median in an undirected graph with polylog complexity and a randomized protocol that computes an approximate median with polyloglog communication complexity per node.
Our work focuses on the objective function of minimizing the number of messages sent and the delay costs incurred by not keeping the root up to date. In contrast to the TCP acknowledgment problem, we will analyze a more general case where each leaf node has a value which can increase or decrease. The root wants to know the aggregated value thereof (e.g. sum, minimum, maximum, over all nodes) and the difference between the aggregated value and the actual value at the root represent the delay costs that occur in each time step. Our goal is to design an online algorithm that minimizes these delay costs and the communication costs, thus the number of messages sent by the algorithm.
In the second part of this thesis, we show how the acknowledge problem can be mapped to our model and prove an alternative upper bound to the bound proposed by Khanna [3].

3 Algorithms for Information Aggregation

3.1 Model

We are given a rooted tree T. We view the tree T as being directed from the leaf nodes to the root, with each node n_i (except for the root) having exactly one link to its parent. This edge of node n_i is denoted as e_i and has cost $c(e_i)$, whereby the cost of sending over this edge is at least 1. Let N denote the set of nodes in the tree, L denote the set of leaf nodes in the tree and R denote the set of root nodes. Let n, r and l denote the respective number of nodes in each set ($r = 1$ as T is a rooted tree).
Each leaf node of the tree has a field with a value the root wants to know and which changes over time. Let the time be slotted (synchronous time model) and each value has to wait at least one time unit at a node before being sent upwards to its parent. Let v_i^t denote the actual value at the leaf and r_i^t its corresponding value at the root node at time t (which might not be up-to-date). Let ϵ ($\epsilon \leq 1$) be the minimal amount which can be added or subtracted from a value at a leaf node. Therefore all leaf node values have to be a multiple of ϵ. If $\epsilon = 1$, values at the leaf nodes are restricted to integers.
Our goal is to minimize the communication cost (thus minimize the number of messages) while trying to keep the root up to date with the values at the leaf nodes by sending messages. Multiple values can be aggregated to one single message and each message has to wait at least one time unit, before being sent to the next node.
Let Algorithm X denote an algorithm which takes an input sequence σ and decides when and what value to send. X_j^i will denote a specific sending event (sending event j) of Algorithm X at node i on the time axis, $t_{X_j^i}$ its corresponding time and $v_{X_j^i}$ will denote the value which was sent. If the index is omitted from X_j^i, it will denote the last sending event of X (e.g. v_{X^i} for the last sent value). Let C_X denote the total costs, CC_X the communication costs and DC_X the delay costs for an Algorithm X, i.e. $C_X = CC_X + DC_X$.
At the beginning, $v_X = r_i = actual\ value\ at\ the\ node$ for all nodes i. As the value at the root is equal to the value at the local node, no delay costs occur.

Let there be an adversary who can modify and choose a sequence of values at the leaf nodes such that they yield the worst case sequence for our algorithms. Let $\alpha = \sum_{i=0..n-1} c(e_i)$ define the total communication costs in a tree and let Δ define the maximal difference between two consecutive values, depending on all edge costs $c(e_i)$ ($\Delta = f(c(e_0), c(e_1), ..., c(e_n - 1))$). If Δ is not bounded, our adversary can achieve an infinite competitive ratio for any deterministic algorithm (this is shown in the lower bound section).
An online algorithm A is called $c - competitive$ if there exists a constant k such that $Cost_A(\sigma) \leq c\ Cost_{Opt}(\sigma) + k$ for all possible input sequences σ. Opt is the optimal offline algorithm that knows the entire input sequence σ in advance and A is our online algorithm which does not know the input sequence in advance. An online algorithm A is called $strictly\ c - competitive$ if $k = 0$.
An online algorithm can be either distributed or centralized. A distributed algorithm runs at each node locally and makes decision only based on the local information it has at that node. Thus all sending events of a node will depend on local information only. A centralized algorithm assumes a global information model, where a central entity determines which values are sent by which nodes. Sending events will depend on global information.
In both cases, future arrivals of values are not known to the algorithm and can be freely chosen by the adversary. In the case of two nodes (one leaf node and one root node), the distributed algorithm has the same information as the centralized algorithm. This thesis will address distributed algorithms only.

We choose the following global cost function which reflects our goal to minimize both the communication costs and the delay costs:

$$\begin{aligned} C &= Communication\ cost\ + Delay\ cost \\ &= Communication\ cost\ + \sum_{t\ \in\ time\ steps} |aggr(v_0^t, v_1^t, ..., v_{n-1}^t) - aggr(r_0^t, r_1^t, ..., r_{n-1}^t)| \end{aligned}$$

where $aggr$ represents one of the following aggregation functions: maximum, minimum, average or any other separable function.
E.g. if we are interested in the average of all leaf node values, the advantage of this cost function is that when a value at a node is increasing over time and another value at a different node is decreasing at the same rate, the delay costs will not increase without any value being sent towards the root, as the aggregated average stays unchanged.
If the delay costs would be defined as the sum of all non aggregated differences between the value at the root and the value at the leaf node, the delay costs would increase, as the changes would not yet have reached the root, while the result we are interested in (e.g. the average) would still stay the same. The changes would therefore generate delay costs even though the aggregated result at the root node might be accurate and does not change.
In the case of one leaf node n_i, one root node and one edge, the cost function can be reduced to:

$$\begin{aligned} C &= Communication\ cost\ + Delay\ cost \\ &= Communication\ cost\ + \sum_{t\ \in\ time\ steps} |v_i^t - r_i^t| \end{aligned}$$

3.2 Deterministic Lower Bound

Definition 3.2.1 (Offline Algorithm O)**.** *Let O be the optimal offline algorithm that can send any value it wants, anytime it wants.*

As the delay costs are calculated by an aggregation function and do not depend directly on the difference between the actual node value at the root and at the node, O could omit sending events by relying on global information and we therefore have to restrict O's knowledge to make sending decisions based on local values (past and future values) only.
If we would allow O to have global knowledge, an infinite competitive ratio can not be prevented. E.g., if we are interested in the maximum node value in a tree, the adversary could choose a very high value at one node (which will be propagated to the root by the online and the optimal offline algorithm) and then at a different node, switch between two values (both values must be less than the global maximum) back and forth. As the online algorithm is distributed, it has to propagate the change towards the root as each change at a node could influence the global maximum. O however would never send anything up, as it knows that the value at the root is the actual maximum of the entire tree. The same holds for other aggregation functions.
As for the centralized algorithm, it is unclear whether O can achieve an infinite competitive ratio as well or not. In the centralized case, our online algorithm also has the same global information as O. Neither our online algorithm, nor the optimal offline algorithm is allowed to change the value that is being propagated towards the root in the tree. If this would be allowed, both algorithms could simply omit sending events at lower levels, by changing values at a higher level in the tree, clearly a behavior which hardly reflects reality.

3.2.1 2-Node Networks

Let there be a deterministic arbitrary Algorithm A with no limitations on what value the algorithm sends or when the value is sent.
The adversary will choose the following sequence of values:
Each sending interval of A (all the values between two sending events of A) will consist of the same value x. As A is deterministic, the adversary knows which value A will send and chooses $x + \Delta$ or $x - \Delta$ (depending on which one is furthest away from the value that A will send) as the next x value for the next sending interval.
Also let us assume that O, the optimal algorithm, will always send the value of the next interval one time unit in advance, such that O only has $c(e_i)$ communication costs in each sending interval of A. Any further decrease of O's costs would increase the lower bound, we can therefore safely assume that O has at most $c(e_i)$ costs.
Thus the following costs will occur at most for O:

$$C_O \quad \leq \quad c(e_i) \quad (communication\ costs)$$

A will have at least the following costs:

$$C_A \quad \geq \quad k\,\Delta + c(e_i) \quad (delay + communication\ costs)$$

As the adversary chooses the value that is furthest away from the last sent value of A, A must have at least Δ delay costs in each time step. k denotes the number of time steps A will wait before having a sending event. As we do not know the sending criterion of A, we must assume that $k = 1$, because this minimizes A's delay costs.
A must also have a sending event at the end, thus $c(e_i)$ is added to A's costs. If A would not have a sending event, the lower bound would be ∞ as k would be infinite (no sending event ever arrives).

Please note that these Δ delay costs for A can be achieved independently of the aggregation function used in the cost function, because in the case of only one leaf node, the aggregation function can be reduced to the difference between v_i^t and r_i^t.
We thus obtain the following lower bound for A's competitive ratio:

$$\begin{aligned} \frac{C_A}{C_O} &\geq \frac{\Delta + c(e_i)}{c(e_i)} \\ &= \frac{\Delta}{c(e_i)} + 1 \end{aligned}$$

Therefore A's competitive ratio has a lower bound of $\frac{\Delta}{c(e_i)} + 1$. This lower bound also holds for the centralized case, as in the case of two nodes, a distributed algorithm has the same information as a centralized algorithm.

3.2.2 Chain Graphs

Let there be a chain of nodes $n_0, n_1, n_2, ..., n_h$, $h+1$ nodes in total. Let n_0 denote the leaf node and n_h denote the root node. Each node n_i has exactly one communication link $c(e_i)$ to its parent node n_{i+1}.

The adversary will apply the same strategy as in the case of two nodes, but instead of choosing $x + \Delta$ or $x - \Delta$ directly after A has a sending event at the leaf node, he will keep the original x until A has completed propagating the change towards the root and change the value at the leaf node to $x + \Delta$ or $x - \Delta$ thereafter.
Also let us assume that O will always send the value of the next interval $h - 1$ time units in advance, such that O only has $c(e_i)$ communication costs and no delay costs at each node in the chain for each sending interval of A at the leaf node.
Thus the following costs will occur at most for O (Let O be the optimal algorithm):

$$C_O \leq \sum_{i=0}^{h-1} c(e_i) \quad (communication\ costs)$$

A will have at least the following costs:

$$C_A \geq \sum_{i=0}^{h-1} (k\ \Delta + c(e_i)) \quad (delay + communication\ costs)$$

For the same reasons as in the case of two nodes, $k = 1$ and A must have $c(e_i)$ communication costs. As A is a distributed algorithm, A's behavior will be the same at each node.
We thus obtain the following lower bound for A's competitive ratio:

$$\begin{aligned} \frac{C_A}{C_O} &\geq \frac{\sum_{i=0}^{h-1} (\Delta + c(e_i))}{\sum_{i=0}^{h-1} c(e_i)} \\ &= \frac{h\ \Delta}{\sum_{i=0}^{h-1} c(e_i)} + 1 \end{aligned}$$

Therefore A has a lower bound of $\frac{h\ \Delta}{\sum_{i=0}^{h-1} c(e_i)} + 1$ for the competitive ratio.

3.2.3 Trees of Height h

Let there be a tree composed of the nodes $n_0, n_1, n_2, ..., n_{n-1}$, n nodes in total. Let $n_0, ..., n_{l-1}$ denote the leaves and n_{n-1} denote the root node. Each node n_i has exactly one edge with communication cost $c(e_i)$ to its parent node n_{i+1}.

The adversary will apply the same tactic (with the same values) at each leaf node in the tree as in the chain of nodes. As our local algorithm A and O both can merge packets on their way up the tree, we have to assume that A is able to merge those packets and has minimal communication costs only. The adversary will change the values at each leaf node at the right time, such that O can also merge all the packets on their way up the tree, and has at most minimal communication costs as well. Therefore O has at most the following costs in each round (a round is defined as the time needed for A to propagate the change at each leaf node towards the root until the adversary is able to start changing values at the leaf nodes again without any values from the previous round interfering with the next change):

$$C_O \leq \sum_{i=0}^{n-2} c(e_i) \quad (communication\ costs)$$

A will have at least the following costs in each "round":

$$C_A \geq \sum_{i=0}^{n-2} (k\ \Delta + c(e_i)) \quad (delay + communication\ costs)$$

For the same reasons as in the case of two nodes, $k = 1$, as this minimizes A's delay costs. As A is a distributed algorithm, A's behavior will be the same at each node.
We thus obtain the following lower bound for A:

$$\begin{aligned} \frac{C_A}{C_O} &\geq \frac{\sum_{i=0}^{n-2} (\Delta + c(e_i))}{\sum_{i=0}^{n-2} c(e_i)} \\ &= \frac{(n-1)\ \Delta}{\sum_{i=0}^{n-2} c(e_i)} + 1 \end{aligned}$$

Therefore A's competitive ratio is at least $\frac{(n-1)\ \Delta}{\sum_{i=0}^{n-2} c(e_i)} + 1$.

3.3 2-Node Networks

3.3.1 Online Algorithm A

The criterion of when to send and what to send has to be chosen wisely. E.g., if an algorithm decides to send periodically, the adversary could achieve an infinite competitive ratio by not

changing any values at all. Another possibility would be to send only after a specific threshold between the last sent value and the current value has been reached. The adversary can achieve an infinite competitive ratio here as well by choosing a value below the threshold, which will never be sent towards the root by the algorithm, but will be sent to the root by the optimal algorithm directly at the beginning.
A sending criterion, depending only on the last sent value and the current value at the root and having no memory about past values, will never be good, as the adversary can always choose a value such that the algorithm does not have a sending event and keep the same value for the next time steps to come.

Definition 3.3.1 (Online Algorithm A)**.** *Let A be our distributed local online algorithm. A sends its last value v_i at node i if*

$$\sum_{t=time\ where\ last\ value\ was\ sent}^{current\ t} |v_A - v_i^t| \geq c(e_i)$$

Whereby v_A is the last send value of the node and v_i^t the value of the node i at time t.

This implies that the delay cost for A never exceeds $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ between two consecutive sending events of A, where Δ is the maximal difference between two consecutive values at a node. $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ is reached when the adversary chooses a sequence of values such that the value at time $t_A + 1$ is at distance $c(e_i) - 1$ from v_A. Thus the value thereafter can at most increase the delay cost for A by $c(e_i) - 1 + \Delta$.

For the analysis of Algorithm A, we will analyze all possible kinds of intervals that exist for A between two consecutive sending events of A at one node, written as interval $]t_{A_x}, t_{A_{x+1}}]$. This interval contains all node values of each time step between t_{A_x} and (including) $t_{A_{x+1}}$. (The start event at the beginning of the entire sequence of values also counts as a sending event but without communication costs).
We can then divide these intervals in 4 types of intervals, depending on whether v_O matches v_A at the beginning and/or at the end of the interval and compute an upper bound for the competitive ratio in any of these intervals.
By fixing A's delay cost to $c(e_i)$ in each of these intervals, the number of different cases to analyze is reduced significantly. This is proven in the next two Lemmas, which imply that by having a higher delay cost than $c(e_i)$ for A in any of these intervals, the competitive ratio in these intervals can only increase by a constant factor.

Lemma 3.3.1. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. When the delay cost for A in this interval is equal to $2(c(e_i) - 1) + k$, where $k \geq 0$, then O has at least costs (delay or communication costs) of k in $[t_{A_1} - 1, t_{A_1}]$, whereby these costs for O can not exceed $c(e_i)$.*

Proof. Let v_l be the value at t_{A_1}, and v_{l-1} the value at $t_{A_1} - 1$. Since the delay costs for A are greater than $2(c(e_i)-1)$ in the entire interval, A must have at least a delay cost of $(c(e_i)-1)+k$ on v_l (otherwise A would have an earlier sending event) and the delay costs of the values before v_l in the interval are at most $c(e_i) - 1$. This implies that the difference between v_l and v_{l-1} is at least k and O will have at least delay costs of k in $[t_{A_1} - 1, t_{A_1}]$ (under the condition that O has no sending event), no matter what value v_O was.
If $k \geq c(e_i)$, then O can limit the costs to $c(e_i)$ by using a sending event on $t_{A_1} - 1$ and sending v_l. Thus the costs for O can not exceed $c(e_i)$ in $[t_{A_1} - 1, t_{A_1}]$ and will be equal to k as long as $k \leq c(e_i)$. □

Lemma 3.3.2. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. Let the delay costs for A be fixed to $c(e_i)$ in this interval and let ρ be A's competitive ratio.*
By letting the adversary choose another sequence of values in this interval, such that the delay costs for A are greater than $c(e_i)$ in this interval, the adversary can at most increase $\rho_{DC_A=c(e_i)}$ by $\frac{1}{2}\ \rho_{DC_A=c(e_i)} + \frac{\Delta}{c(e_i)}$, thus giving us $\rho_{DC_A \geq c(e_i)} \leq \frac{3}{2}\rho_{DC_A=c(e_i)} + \frac{\Delta}{c(e_i)}$.

Proof. Let C_A and C_O be the total costs for A and O, under the condition that A has a delay cost of $c(e_i)$. By letting the adversary choose a different sequence of values in this interval, he might increase the delay costs for A to $2(c(e_i)-1)$ while leaving the delay costs for O unchanged, thus nearly doubling the delay costs for A. According to the preceding Lemma, any further attempt by the adversary to increase the delay cost for A by another k ($k \leq c(e_i)$), will also increase the costs for O by the same amount. If $k \geq c(e_i)$ ($k \leq \Delta$), A alone will have k costs while O's costs are limited to $c(e_i)$.
Thus, $\rho_{DC_A \geq c(e_i)} \leq \frac{3}{2}\rho_{DC_A=c(e_i)} + \frac{\Delta}{c(e_i)}$. □

We can now conclude our analysis by fixing A's delay cost in each interval $]t_{A_x}, t_{A_{x+1}}]$ to $c(e_i)$ and apply Lemma 3.3.2 at the end. The next Lemma determines how a worst case sequence for A will look like for the competitive ratio to be maximal in an interval.

Lemma 3.3.3. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. Let v_O denote the last send value of O before t_{A_1}. For the competitive ratio to be maximal in this interval, the worst case sequence consists of:*

(i) values less than or equal to v_O if $v_O < v_{A_0}$

(ii) values greater than or equal to v_O if $v_O > v_{A_0}$

Proof.

(i) $v_O < v_{A_0}$:
If the sequence consists of values greater than or equal to v_{A_0}, the adversary could always increase the competitive ratio by choosing values less than or equal to v_{A_0}, as these values yield smaller delay costs for O and higher delay costs for A. Thus the worst case sequence will never consist of values greater than or equal to v_{A_0}.
If the sequence consists of values less than or equal to v_{A_0}, but greater than v_O, the adversary could always increase the competitive ratio by choosing values less than or equal to v_O, as these values yield higher delay costs for A.
Thus the worst case sequence will consist of values less than or equal to v_O (depending on whether v_O should be equal to v_{A_1} at the end or not), with the closest values to v_O yielding the highest competitive ratio.

(ii) $v_O > v_{A_0}$:
By using the same argument as above, we proof that the worst case sequence consists of values greater than or equal to v_O.

□

We now have to analyze all the different types of intervals that exist. Depending on whether v_O matches v_A at the beginning and/or at the end of the interval, we have to analyze 4 different kinds of intervals.

Lemma 3.3.4. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. Let δ_0 denote the difference between v_O and v_{A_0} ($\delta_0 = |v_O - v_{A_0}|$) at time t_{A_0} and δ_1 denote the difference between v_O and v_{A_1} ($\delta_1 = |v_O - v_{A_1}|$) at time t_{A_1}. If:*

(i) $\delta_0 = 0$ *and* $\delta_1 = 0$*, then the competitive ratio in this interval will be at most* 2*. Let this interval be called* I_1*.*

(ii) $\delta_0 \neq 0$ *and* $\delta_1 = 0$*, then the competitive ratio in this interval will be* ∞*. Let this interval be called* I_2*.*

(iii) $\delta_0 = 0$ *and* $\delta_1 \neq 0$*, then the competitive ratio in this interval will be at most* 2*. Let this interval be called* I_3*.*

(iv) $\delta_0 \neq 0$ *and* $\delta_1 \neq 0$*, then the competitive ratio in this interval will be at most* $2\ \epsilon^{-1}\ c(e_i)$*. Let this interval be called* I_4*.*

Proof. In each of the following intervals, it is assumed that A has exactly $2c(e_i)$ costs, as we fixed the delay costs for A to be $c(e_i)$.

(i) $\delta_0 = 0$ and $\delta_1 = 0$:
Since $\delta_0 = 0$, $\delta_1 = 0$ and A chooses the last value in this interval as the new v_A, O must have at least one sending event in this interval. If O would not have one sending event in this interval, $\delta_1 = 0$ could never be reached. Thus the costs for O will be at least $c(e_i)$, while the costs for A are $2c(e_i)$, yielding a competitive ratio of at most 2 for A in this interval.

(ii) $\delta_0 \neq 0$ and $\delta_1 = 0$:
The adversary will choose a worst case sequence that consists of the value v_O during the interval $]t_{A_0}, t_{A_1}]$. As O has no costs at all, this will be the worst case sequence yielding a competitive ratio of ∞ for A in this interval.

(iii) $\delta_0 = 0$ and $\delta_1 \neq 0$:
If O has a sending event in this interval, O will have at least communication costs of $c(e_i)$. If O has no sending event in this interval, O must have delay costs equal to the delay costs of A, namely $c(e_i)$.
Thus the competitive ratio will be at most 2 in both cases for A in this interval.

(iv) $\delta_0 \neq 0$ and $\delta_1 \neq 0$:
If O has a sending event in this interval, O will have at least communication costs of $c(e_i)$. If O has no sending event in this interval, the last value in the interval must be different from v_O (as $\delta_1 \neq 0$). O must therefore have at least delay costs of ϵ at t_{A_1}, yielding a competitive ratio of at most $2\ \epsilon^{-1}\ c(e_i)$ for A in this interval.
This interval can be repeated indefinitely often. This implies that the competitive ratio of an arbitrary sequence can not be lower than $2\ \epsilon^{-1}\ c(e_i)$.

□

We therefore obtain the following table of competitive ratios:

	Interval	**maximal competitive ratio**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	2
I_2	$\delta_0 \neq 0$ and $\delta_1 = 0$	∞
I_3	$\delta_0 = 0$ and $\delta_1 \neq 0$	2
I_4	$\delta_0 \neq 0$ and $\delta_1 \neq 0$	$2\ \epsilon^{-1}\ c(e_i)$

From these 4 different types of intervals, the interval I_2 yields the highest competitive ratio, since O has no costs at all. The next Lemma proves that each interval I_2 is preceded by exactly one interval I_3. Thus, the number of intervals having I_2 will be less than or equal to the number of intervals I_3 and we can analyze the amortized competitive ratio[1] for both intervals together.

Lemma 3.3.5. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. Let δ_0 denote the difference between v_O and v_{A_0} ($\delta_0 = |v_O - v_{A_0}|$) at time t_{A_0} and δ_1 denote the difference between v_O and v_{A_1} ($\delta_1 = |v_O - v_{A_1}|$) at time t_{A_1}.*
Each interval having $\delta_0 \neq 0$ and $\delta_1 = 0$ is preceded by exactly one interval having $\delta_0 = 0$ and $\delta_1 \neq 0$.

Proof. As, by definition of the model, at the beginning of the entire sequence of values at the node (from start to end), $v_O = v_A$ and from the 4 cases from above, the only way to go from $\delta_1 = 0$ to $\delta_1 \neq 0$ is by an interval having $\delta_0 = 0$ and $\delta_1 \neq 0$, there must be exactly one interval having $\delta_0 = 0$ and $\delta_1 \neq 0$ before each interval having $\delta_0 \neq 0$ and $\delta_1 = 0$.

□

This is also covers the case where the interval I_3 is not directly followed by an interval I_2, but might have one or more intervals I_4 in between.

Lemma 3.3.6. *Let there be one interval having $\delta_0 \neq 0$ and $\delta_1 = 0$ and one interval having $\delta_0 = 0$ and $\delta_1 \neq 0$. By analyzing both intervals together and adding their costs, both intervals will have an amortized competitive ratio of at most 4.*

Proof. Adding A's and O's costs for both intervals as defined in 3.3.4 gives us $\frac{2c(e_i)+2c(e_i)}{c(e_i)+0} = 4$ as new upper bound for the competitive ratio of the combined interval.
This also bounds the case if there is no sending event for A at the end of the entire sequence of values (from start to end).

□

We therefore obtain the following table of amortized competitive ratios:

	Interval	**maximal c.r.**	**amortized maximal c.r.**	**amortizes**	**amortized by**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	2	2		
I_2	$\delta_0 \neq 0$ and $\delta_1 = 0$	∞	4		I_3
I_3	$\delta_0 = 0$ and $\delta_1 \neq 0$	2	4	I_2	
I_4	$\delta_0 \neq 0$ and $\delta_1 \neq 0$	$2\ \epsilon^{-1}\ c(e_i)$	$2\ \epsilon^{-1}\ c(e_i)$		

Thus the competitive ratio of ∞ is never reached, and Algorithm A has an upper bound of $2\epsilon^{-1}\ c(e_i)$, under the condition that A's delay costs are fixed to $c(e_i)$ for each interval.

Theorem 3.3.7. *Algorithm A is $O(\epsilon^{-1}\ c(e_i) + \frac{\Delta}{c(e_i)})$-competitive to O.*

Proof. According to Lemma 3.3.4 and Lemma 3.3.6, the amortized competitive ratio of an interval can never exceed $2\ \epsilon^{-1}\ c(e_i)$. Thus the final competitive ratio of an arbitrary sequence can never exceed $2\ \epsilon^{-1}\ c(e_i)$, under the condition that A's delay costs are fixed to $c(e_i)$ for each interval. According to Lemma 3.3.2, removing this limitation will increase the upper bound to $\frac{3}{2} 2\ \epsilon^{-1}\ c(e_i) + \frac{\Delta}{c(e_i)} = 3\ \epsilon^{-1}\ c(e_i) + \frac{\Delta}{c(e_i)}$.
Thus A has an upper bound of $O(\epsilon^{-1}\ c(e_i) + \frac{\Delta}{c(e_i)})$ for the competitive ratio. □

[1] Let the amortized competitive ratio be defined as the competitive ratio of one ore more intervals analyzed together and considered as one interval only.

Theorem 3.3.8. *Algorithm A's competitive ratio has a lower bound of $\Omega(\epsilon^{-1}\ c(e_i))$.*

Proof. The following sequence yields a competitive ratio of $\epsilon^{-1}\ c(e_i)$. a^b means that the value a is repeated b times.

$$0, \epsilon^{\frac{c(e_i)}{\epsilon}}, \quad 0^{\frac{c(e_i)}{\epsilon}-\epsilon}, -\epsilon, 0^{\frac{c(e_i)}{\epsilon}-\epsilon}, \epsilon, \quad 0^{\frac{c(e_i)}{\epsilon}-\epsilon}, -\epsilon, 0^{\frac{c(e_i)}{\epsilon}-\epsilon}, \epsilon, \quad \ldots \quad , 0^{\frac{c(e_i)}{\epsilon}}$$

O will never have a sending event and each sequence $0^{\frac{c(e_i)}{\epsilon}-\epsilon}, -\epsilon, 0^{\frac{c(e_i)}{\epsilon}-\epsilon}, \epsilon$, yields a competitive ratio of $2\ \epsilon^{-1}\ c(e_i)$, as O only has $2\ \epsilon$ delay costs and A has $4\ c(e_i)$ costs in each sequence. Thus the entire sequence will reach a competitive ratio of $2\ \epsilon^{-1}\ c(e_i)$ at the end, and by consequence, Algorithm A is thus $\theta(\epsilon^{-1}\ c(e_i))$-competitive. □

3.3.2 Online Algorithm B

Definition 3.3.2 (Online Algorithm B)**.** *Let B be our distributed local online algorithm with the same sending criterion as A. Opposed to A, which sends the last value in the interval, B will send the value which occurred most in the last interval. If there is more than one candidate, B will send the candidate which is furthest away from v_B. If there are two candidates at the same distance from v_B and furthest away from v_B, B will send the smallest candidate. If the candidate (the new v_B) is equal to the old v_B, B will omit the sending event and reset the local delay cost counter to 0.*

This implies that the delay cost for B never exceeds $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ between two consecutive sending events of B, where Δ is the maximal difference between two consecutive values at a node. $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ is reached when the adversary chooses a sequence of values such that the value at time $t_B + 1$ is at distance $c(e_i) - 1$ from v_B. Thus the value thereafter can at most increase the delay cost for B by $c(e_i) - 1 + \Delta$.

For the analysis of Algorithm B, we will analyze all possible kind of intervals that exist for B between two consecutive sending events of B at one node, written as interval $]t_{B_x}, t_{B_{x+1}}]$. This interval contains all node values of each time step between t_{B_x} and (including) $t_{B_{x+1}}$. (The start event at the beginning of the entire sequence of values also counts as a sending event but without communication costs).
There are 3 types of intervals for which we can compute an upper bound for the competitive ratio:

(i) $|v_O - v_{B_x}| = |v_O - v_{B_x+1}|$

(ii) $|v_O - v_{B_x}| > |v_O - v_{B_x+1}|$

(iii) $|v_O - v_{B_x}| < |v_O - v_{B_x+1}|$

Lemma 3.3.1 and Lemma 3.3.2 also hold for Algorithm B since both Lemmas are proven without making any assumptions on the values being sent by A. We can therefore continue our analysis by fixing B's delay cost to $c(e_i)$ and apply Lemma 3.3.2 at the end.
We now have to analyze the different types of intervals that exist.

Lemma 3.3.9. *Consider an arbitrary interval $]t_{B_0}, t_{B_1}]$ between two sending occurrences of B. Let δ_0 denote the difference between v_O and v_{B_0} ($\delta_0 = |v_O - v_{B_0}|$) at time t_{B_0} and δ_1 denote the difference between v_O and v_{B_1} ($\delta_1 = |v_O - v_{B_1}|$) at time t_{B_1}. If:*

(i) $\delta_0 = \delta_1$ *and* $\delta_0 = 0$*: impossible. Let this interval be called* I_1*.*

(ii) $\delta_0 = \delta_1$ *and* $\delta_0 \geq \epsilon$*, then the competitive ratio in this interval will be at most* ϵ^{-1} $(8 + 2\sqrt{\delta_0})$*. Let this interval be called* I_2*.*

(iii) $\delta_0 < \delta_1$ *and* $\delta_0 = 0$*, then the competitive ratio in this interval will be at most* 2*. Let this interval be called* I_3*.*

(iv) $\delta_0 < \delta_1$ *and* $\delta_0 \geq \epsilon$*, then the competitive ratio in this interval will be at most* ϵ^{-1} $(8 + 2\sqrt{\delta_0})$*. Let this interval be called* I_4*.*

(v) $\delta_0 > \delta_1$ *and* $\delta_1 \geq \epsilon$*, then the competitive ratio in this interval will be at most* $2\ \epsilon^{-1}\ c(e_i)$*. Let this interval be called* I_5*.*

(vi) $\delta_0 > \delta_1$ *and* $\delta_1 = 0$*, then the competitive ratio in this interval is* ∞*. Let this interval be called* I_6*.*

Proof. In each of the following intervals, it is assumed that B has exactly $2c(e_i)$ costs, as we fixed the delay costs for B to be $c(e_i)$.

(i) $\delta_0 = \delta_1$ and $\delta_0 = 0$:
As $\delta_0 = \delta_1 = 0$, this kind of interval will never occur because B will not send when the new v_B is equal to the old v_B.

(ii) $\delta_0 = \delta_1$ and $\delta_0 \geq \epsilon$:
If O has no sending event:
Let us assume that $v_{B_0} > v_O$ at time t_{B_0}, that the adversary can only choose values less than or equal to v_O and that $\epsilon = 1$ (All three limitations will be removed at the end of the proof).
For $\delta_0 = \delta_1$, the adversary will choose the following worst case sequence of values:
Without loss of generality, we can fix the order of the values in a sending interval as it has no influence on neither the competitive ratio, nor the choice of candidate by B.
The adversary will choose its first value to be at distance δ_0 from v_O (value $v_O - \delta_0$), as δ_0 has to occur at least once in the sequence or otherwise $\delta_0 \neq \delta_1$, as B only sends values which occurred during the interval.
The rest of the sequence consists of values at distance $0, 1, 2, ..., y-1$ from v_O (values $v_O, v_O - 1, v_O - 2, ..., v_O - (y-1)$), y values in total, $y \leq \delta_0$. The values closest to v_O yield the highest competitive ratio, thus those values will be chosen first. These values can occur at most once, since otherwise B will not choose the value at distance δ_0 from v_O as the new v_B and $\delta_0 \neq \delta_1$.
Repeating the same sequence again (values $\delta_0, 0, 1, 2, ..., y-1$) can not increase the competitive ratio, because the value δ_0 has the greatest $\frac{C_B}{C_O}$ ratio from all these values and must be repeated first, as otherwise a different value is chosen as sending candidate by B. We can therefore restrict our analysis to one sequence (values $\delta_0, 0, 1, 2, ..., y-1$) only.
We limited ourself to all the values less than or equal to v_O, but the worst case sequence could also consist of values greater than v_O. But as all values at a certain distance from v_O and greater than v_O yield a smaller competitive ratio as if those values are at the same distance, but less than v_O, we can limit our analysis by assuming that all values are less than or equal to v_O, but that all values in $[v_O, v_O - (y-1)]$ can occur twice instead of once.
The analogue reasoning holds for $v_{B_0} < v_O$.

We thus obtain the following costs (communication and delay) for B and O in this interval:

$$\begin{aligned}
C_B &= \frac{2\delta_0 + 2((\delta_0+0)+(\delta_0+1)+(\delta_0+2)+...+(\delta_0+y-1))}{2\delta_0 + 2((\delta_0+0)+(\delta_0+1)+(\delta_0+2)+...+(\delta_0+y-1))}c(e_i) + c(e_i) \\
&= \frac{2\delta_0 + 2(y\ \delta_0 + \frac{y}{2}(y-1))}{2\delta_0 + 2(y\ \delta_0 + \frac{y}{2}(y-1))}c(e_i) + c(e_i) \\
&= 2\ c(e_i) \\
C_O &\geq \frac{\delta_0 + 2(0+1+2+...+(y-1))}{2\delta_0 + 2(y\ \delta_0 + \frac{y}{2}(y-1))}c(e_i) \\
&\geq \frac{\delta_0 + 2(\frac{y}{2}(y-1))}{2\delta_0 + 2(y\ \delta_0 + \frac{y}{2}(y-1))}c(e_i) \\
&\geq \frac{\delta_0 + y(y-1)}{2\delta_0 + 2y\ \delta_0 + y(y-1)}c(e_i) \\
\frac{C_B}{C_O} &\leq \frac{2c(e_i)}{\frac{\delta_0+y(y-1)}{2\delta_0+2y\ \delta_0+y(y-1)}c(e_i)} \\
&= 2\frac{2\delta_0 + 2y\ \delta_0 + y(y-1)}{\delta_0 + y(y-1)}
\end{aligned}$$

In order to find the extrema of the function, we maximize $\frac{C_B}{C_O}$ by the following calculation:

$$\frac{d}{dy}\frac{C_B}{C_O} = 0 \iff y = -\frac{1}{2} + \frac{1}{2}\sqrt{3+4\delta_0}\ or\ y = -\frac{1}{2} - \frac{1}{2}\sqrt{3+4\delta_0}$$

$\frac{C_B}{C_O}$ is thus maximized by $y = -\frac{1}{2} + \frac{1}{2}\sqrt{3+4\delta_0}$. The second solution can be discarded because y has to be greater than or equal to 0.
The maximal competitive ratio will thus be:

$$\frac{C_B}{C_O} \leq 2\frac{4\delta_0 + 2\delta_0\sqrt{3+4\delta_0} + 3 - 2\sqrt{3+4\delta_0}}{4\delta_0 + 3 - 2\sqrt{3+4\delta_0}}$$

By analyzing the extrema and calculating the limit (if δ_0 goes to infinity) of $\frac{C_B}{C_O}$, we can deduce that $\frac{C_B}{C_O} \leq 8 + 2\sqrt{\delta_0}$, $\delta_0 \geq 1$.
In reality, infinity is never reached for δ_0 and $\delta_0 \leq c(e_i)$, as we fixed the delay costs for B to be at most $c(e_i)$ and by the definition of our model, $c(e_i)$ is at least 1. We can therefore bound the competitive ratio to $8 + 2\sqrt{c(e_i)}$.
As we limited ourself to integers ($\epsilon = 1$), removing this limitation will at most increase the competitive ratio for B to $\epsilon^{-1}\ \frac{C_{B_{\epsilon=1}}}{C_{C_{\epsilon=1}}}$, as the delay costs for O can be reduced to at most $\epsilon\ DC_{C_{\epsilon=1}}$. We can therefore increase the upper bound to $\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$.
If O has a sending event in this interval, the competitive ratio can be at most 2 in this interval. As $2 < \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$ ($\epsilon \leq 1$), O will not have a sending event for the competitive ratio to be maximal in this interval and B will be at most $\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$ competitive in this interval.

(iii) $\delta_0 < \delta_1$ and $\delta_0 = 0$:
If O has a sending event in this interval, O will have at least $c(e_i)$ communication costs.
If O has no sending event in this interval, O will have at least $c(e_i)$ delay costs as $\delta_0 = 0$

and $v_O = v_B$ at the beginning of the interval.
Hence the competitive ratio for B can not exceed 2 in this interval.

(iv) $\delta_0 < \delta_1$ and $\delta_0 \geq \epsilon$:
The analogue reasoning as in the case where $\delta_0 = \delta_1$ holds, but instead of the adversary choosing δ_0 as the first value in the interval, he chooses δ_1. As δ_1 is greater than δ_0 and B's delay costs are at least $c(e_i)$ in this interval, O must have at least the same delay costs as O where $\delta_0 = \delta_1$. (The number of values y might be reduced, but the delay costs that O loses on those values are compensated by the increase in delay costs due to $\delta_1 > \delta_0$.) Thus the competitive ratio can be at most the competitive ratio of the case where $\delta_0 = \delta_1$, namely $\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$.

(v) $\delta_0 > \delta_1$ and $\delta_1 \geq \epsilon$:
If O has a sending event in this interval, O will have at least $c(e_i)$ communication costs.
If O has no sending event in this interval, O will have at least delay costs of ϵ (because $1 \leq \delta_1 < \delta_0$). The competitive ratio will therefore be at most $2\epsilon^{-1}\ c(e_i)$ for B in this interval.

(vi) $\delta_0 > \delta_1$ and $\delta_1 = 0$:
The adversary will choose a worst case sequence that consists of the value v_O during the interval $]t_{B_0}, t_{B_1}]$, such that at the end, $\delta_1 = 0$. As O has no costs at all, this sequence yields a competitive ratio of ∞ for B in this interval.

□

We therefore obtain the following table of competitive ratios:

	Interval	**maximal competitive ratio**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	/
I_2	$\delta_0 = \delta_1$ and $\delta_0 \geq \epsilon$	$\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$
I_3	$\delta_0 < \delta_1$ and $\delta_0 = 0$	2
I_4	$\delta_0 < \delta_1$ and $\delta_0 \geq \epsilon$	$\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$
I_5	$\delta_0 > \delta_1$ and $\delta_1 \geq \epsilon$	$2\,\epsilon^{-1}\ c(e_i)$
I_6	$\delta_0 > \delta_1$ and $\delta_1 = 0$	∞

From these 6 different types of intervals, the interval I_6 yields the highest competitive ratio, since O has no costs at all. The interval I_5 yields a competitive ratio of $2\epsilon^{-1}\ c(e_i)$.
The next Lemma proves that each series of intervals having $\delta_0 > \delta_1$ is at least preceded by one interval having $\delta_0 < \delta_1$. *A* series of intervals having $\delta_0 > \delta_1$ is defined as all the intervals having $\delta_0 > \delta_1$ until an interval having $\delta_0 < \delta_1$ is reached. As a side note, if the values at the leaf nodes are restricted to integers, then there can be at most Δ intervals having $\delta_0 > \delta_1$ for each interval having $\delta_0 < \delta_1$, because the difference between δ_0 and δ_1 is at most Δ.
We can thus analyze the competitive ratio for these intervals together. A's new amortized competitive ratio for each of the two intervals is proved in the next three lemmas.

Lemma 3.3.10. *Consider an arbitrary interval $]t_{B_0}, t_{B_1}]$ between two sending occurrences of B. Let δ_0 denote the difference between v_O and v_{B_0} ($\delta_0 = |v_O - v_{B_0}|$) at time t_{B_0} and δ_1 denote the difference between v_O and v_{B_1} ($\delta_1 = |v_O - v_{B_1}|$) at time t_{B_1}.*
Each series of intervals having $\delta_0 > \delta_1$ is preceded by at least one interval having $\delta_0 < \delta_1$.

Proof. As, by definition of the model, at the beginning of the entire sequence of values at the node (from start to end), $v_O = v_B$ and an interval having $\delta_0 < \delta_1$ is the only way of increasing the difference between v_O and v_B at the end of the interval, there must be at least one interval having $\delta_0 < \delta_1$ before each series of intervals $\delta_0 > \delta_1$. □

Lemma 3.3.11. *Consider an arbitrary interval $]t_{B_0}, t_{B_1}]$ between two sending occurrences of B. Let δ_0 denote the difference between v_O and v_{B_0} ($\delta_0 = |v_O - v_{B_0}|$) at time t_{B_0} and δ_1 denote the difference between v_O and v_{B_1} ($\delta_1 = |v_O - v_{B_1}|$) at time t_{B_1}.*
A series of intervals having $\delta_0 > \delta_1$ without a different type of interval in between has a competitive ratio of at most $2\ \epsilon^{-1}\ c(e_i)$.

Proof. If O has no sending event:
If this type of interval is repeated more than once without a different type of interval in between, such that distance between v_O and v_B is decreased in multiple steps, then O's delay costs must be increased by at least ϵ in each step (starting with ϵ). Thus if the distance between v_O and v_B is decreased in x ($x \geq 1$) steps, O must have at least $\epsilon(1 + 2 + 3 + 4 + ... + x)$ delay costs, and the competitive ratio will be at most:

$$\frac{C_B}{C_O} \leq \frac{2\ x\ c(e_i)}{\epsilon(1 + 2 + 3 + 4 + ... + x)} = \epsilon^{-1}\ \frac{2\ x\ c(e_i)}{\frac{x}{2}(x + 1)} = \epsilon^{-1}\ \frac{4\ c(e_i)}{x + 1}$$

The competitive ratio $\frac{C_B}{C_O} \leq \epsilon^{-1}\ \frac{4\ c(e_i)}{x+1}$ is maximal if $x = 1$ and will never exceed $2\ \epsilon^{-1}\ c(e_i)$.

If O has one or more sending events, then the series is composed of the following two type of intervals:

(i) One or more series of intervals with O having a sending event. The competitive ratio of each of these series will be at most:

$$\frac{C_B}{C_O} \leq \frac{2\ x\ c(e_i)}{x\ c(e_i)} = 2 \leq 2\ \epsilon^{-1}\ c(e_i)$$

(ii) At most one series of intervals where O has no sending event. The competitive ratio will be at most (as proven above):

$$\frac{C_B}{C_O} \leq \frac{2\ x\ c(e_i)}{\epsilon(1 + 2 + 3 + 4 + ... + x)} = \epsilon^{-1}\ \frac{2\ x\ c(e_i)}{\frac{x}{2}(x + 1)} = \epsilon^{-1}\ \frac{4\ c(e_i)}{x + 1} \leq 2\ \epsilon^{-1}\ c(e_i)$$

Hence a series of intervals having $\delta_0 > \delta_1$ without a different type of interval in between has a competitive ratio of at most $2\ \epsilon^{-1}\ c(e_i)$. □

Lemma 3.3.12. *Let there be one interval having $\delta_0 < \delta_1$ and a series of intervals having $\delta_0 > \delta_1$. By analyzing those intervals together and adding their costs, the combined interval will have a competitive ratio of at most 3 times the competitive ratio of the interval having $\delta_0 = \delta_1$.*

Proof. Adding B's and O's costs for those intervals as defined in Lemma 3.3.9 gives us

$$\begin{aligned}
\frac{C_B}{C_O} &= \frac{C_{B_{\delta_0<\delta_1}} + C_{B_{\delta_0>\delta_1}}}{C_{O_{\delta_0<\delta_1}} + C_{O_{\delta_0>\delta_1}}} \\
&\leq \frac{C_{B_{I_4}} + C_{B_{I_5}} + C_{B_{I_6}}}{C_{O_{I_4}} + C_{O_{I_5}} + C_{O_{I_6}}} \\
&\leq \frac{C_{B_{I_4}} + 2c(e_i) + 2c(e_i)}{C_{O_{I_4}} + \epsilon + 0} \\
&\leq \frac{C_{B_{I_4}} + 2c(e_i) + 2c(e_i)}{C_{O_{I_4}}} \\
&\leq 3\frac{C_{B_{I_4}}}{C_{O_{I_4}}} \\
&\leq 3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})
\end{aligned}$$

as new upper bound for the competitive ratio of the combined intervals.
This also bounds the case if there is no sending event for B at the end of the entire sequence of values (from start to end).

□

We therefore obtain the following table of amortized competitive ratios:

	Interval	**maximal c.r.**	**amortized maximal c.r.**	**amortizes**	**amortized by**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	/	/		
I_2	$\delta_0 = \delta_1$ and $\delta_0 \geq \epsilon$	$\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$	$\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$		
I_3	$\delta_0 < \delta_1$ and $\delta_0 = 0$	2	2		
I_4	$\delta_0 < \delta_1$ and $\delta_0 \geq \epsilon$	$\epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$	$3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$	I_5 *and* I_6	
I_5	$\delta_0 > \delta_1$ and $\delta_1 \geq \epsilon$	$2\ \epsilon^{-1}\ c(e_i)$	$3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$		I_4
I_6	$\delta_0 > \delta_1$ and $\delta_1 = 0$	∞	$3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$		I_4

Thus the competitive ratio of ∞ or $2\ \epsilon^{-1}\ c(e_i)$ is never reached, and Algorithm B has an upper bound of $3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$ for the competitive ratio, under the condition that B's delay costs are fixed to $c(e_i)$ for each interval.

Theorem 3.3.13. *Algorithm B is $O(\epsilon^{-1}\ \sqrt{c(e_i)} + \frac{\Delta}{c(e_i)})$-competitive to O*

Proof. According to Lemma 3.3.9 and Lemma 3.3.12, the amortized competitive ratio of an interval can never exceed $3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$. Thus the final competitive ratio of an arbitrary sequence can never exceed $3\ \epsilon^{-1}\ (8 + 2\sqrt{c(e_i)})$, under the condition that B's delay costs are fixed to $c(e_i)$ for each interval. According to Lemma 3.3.2, removing this limitation yields $\frac{9}{2}\ \epsilon^{-1}\ (8+2\sqrt{c(e_i)}) + \frac{\Delta}{c(e_i)} = 9\ \epsilon^{-1}\ (4+\sqrt{c(e_i)}) + \frac{\Delta}{c(e_i)}$ as new upper bound for the competitive ratio.
Therefore B's competitive ratio has an upper bound of $O(\epsilon^{-1}\ \sqrt{c(e_i)} + \frac{\Delta}{c(e_i)})$. □

3.3.3 Online Algorithm C

Definition 3.3.3 (Online Algorithm C)**.** *Let C be our distributed local online algorithm with the same sending criterion as A. Opposed to A, which sends the last value in the interval, C will send the average of all values which occurred in the last interval.*

Again as before, the delay costs for C can not exceed $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ between two consecutive sending events of C, where Δ is the maximal difference between two consecutive values at a node. Also, let $]t_{A_x}, t_{A_{x+1}}]$ denote a sending interval of C as we defined it before.
Lemma 3.3.1 and Lemma 3.3.2 also hold for Algorithm C, since no assumption about which values are being sent by A was made in the proof. We can therefore continue our analysis by fixing C's delay cost to $c(e_i)$ and apply Lemma 3.3.2 at the end.
Let us analyze the different kind of intervals that exist:

Lemma 3.3.14. *Consider an arbitrary interval $]t_{C_0}, t_{C_1}]$ between two sending events of C. Let δ_0 denote the difference between v_O and v_{C_0} ($\delta_0 = |v_O - v_{C_0}|$) at time t_{C_0} and δ_1 denote the difference between v_O and v_{C_1} ($\delta_1 = |v_O - v_{C_1}|$) at time t_{C_1}. If:*

(i) $\delta_0 = 0$ and $\delta_1 = 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_1.

(ii) $\delta_0 \neq 0$ and $\delta_1 = 0$, then the competitive ratio in this interval will be ∞. Let this interval be called I_2.

(iii) $\delta_0 = 0$ and $\delta_1 \neq 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_3.

(iv) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 > \delta_1$, then the competitive ratio in this interval will be at most $2\ \epsilon^{-1}\ c(e_i)$. Let this interval be called I_4.

(v) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 < \delta_1$, then the competitive ratio in this interval will be at most 4. Let this interval be called I_5.

(vi) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 = \delta_1$, then the competitive ratio in this interval will be at most 4. Let this interval be called I_6.

Proof. In each of the following intervals, it is assumed that C has exactly $2c(e_i)$ costs, as we fixed the delay costs for C to be $c(e_i)$.

(i) $\delta_0 = 0$ and $\delta_1 = 0$:
If O has a sending event, O must have at least $c(e_i)$ communication costs.
If O has no sending event, O must have the same delay costs as C as $\delta_0 = 0$.
Therefore the competitive ratio for C will be at most 2.

(ii) $\delta_0 \neq 0$ and $\delta_1 = 0$:
The adversary will choose a worst case sequence that consists of the value v_O during the interval $]t_{C_0}, t_{C_1}]$. As O has no costs at all, this will be the worst case sequence yielding a competitive ratio of ∞ for C in this interval.

(iii) $\delta_0 = 0$ and $\delta_1 \neq 0$:
If O has a sending event, O will have at least communication costs of $c(e_i)$.
If O has no sending event in this interval, O must have at least delay costs equal to the delay costs of C as $\delta_0 = 0$.
Thus the competitive ratio will be at most 2 for C in this interval.

(iv) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 > \delta_1$:
If O has a sending event, O will have at least communication costs of $c(e_i)$.
If O has no sending event, O will have at least delay costs of ϵ (because $\delta_0 > \delta_1 \neq 0$).
Hence the competitive ratio will be at most $2\epsilon^{-1}\ c(e_i)$ for C in this interval.

(v) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 < \delta_1$:

(i) $v_{C_0} > v_O$ and $v_{C_1} < v_O$:
$v_{C_0} < v_O$ and $v_{C_1} > v_O$:
If O has a sending event, O will have at least communication costs of $c(e_i)$.
If O has no sending event, O must have on average δ_1 delay costs for each value in the interval, while C has $\delta_0 + \delta_1$ delay costs for each value in the interval on average. Thus for the competitive ratio to be maximal, δ_1 will be as small as possible, namely $\delta_0 + \epsilon$, and we obtain the following competitive ratio:

$$\begin{aligned} \frac{C_C}{C_O} &\leq 2\frac{\delta_0 + \delta_1}{\delta_1} \\ &\leq 2\frac{2\delta_0 + \epsilon}{\delta_0 + \epsilon} \\ &\leq 4 \end{aligned}$$

The factor 2 comes from the fact that C's communication costs are equal to the delay costs.
If O has a sending event, the competitive ratio can be at most 2. Hence the competitive ratio can not exceed 4 in this interval.

(ii) $v_{C_0} > v_O$ and $v_{C_1} > v_O$:
$v_{C_0} < v_O$ and $v_{C_1} < v_O$:
If O has a sending event, O will have at least communication costs of $c(e_i)$.
If O has no sending event, O has on each value more delay costs than C. Thus the competitive ratio can be at most 2.

The competitive ratio can therefore not exceed 4 in this interval.

(vi) $\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 = \delta_1$:
If O has a sending event, O will have at least $c(e_i)$ costs.
If O has no sending event, O must have on average at least half of the delay costs of C and we obtain the following competitive ratio:

$$\begin{aligned} \frac{C_C}{C_O} &\leq 2\,\frac{2}{1} \\ &= 4 \end{aligned}$$

The factor 2 comes from the fact that C's communication costs are equal to the delay costs. Thus the competitive ratio can be at most 4.

□

We therefore obtain the following table of competitive ratios:

	Interval	**maximal competitive ratio**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	2
I_2	$\delta_0 \neq 0$ and $\delta_1 = 0$	∞
I_3	$\delta_0 = 0$ and $\delta_1 \neq 0$	2
I_4	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 > \delta_1$	$2\,\epsilon^{-1}\,c(e_i)$
I_5	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 < \delta_1$	4
I_6	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 = \delta_1$	4

From these 6 different kind of intervals, the interval I_2 yields the highest competitive ratio, since O has no costs at all.

Lemma 3.3.15. *Consider an arbitrary interval $]t_{C_0}, t_{C_1}]$ between two sending events of C. Let δ_0 denote the difference between v_O and v_{C_0} ($\delta_0 = |v_O - v_{C_0}|$) at time t_{C_0} and δ_1 denote the difference between v_O and v_{C_1} ($\delta_1 = |v_O - v_{C_1}|$) at time t_{C_1}. A series of intervals having $\delta_0 \neq \delta_1 \neq 0$ and $\delta_0 > \delta_1$ without a different type of interval in between has a competitive ratio of at most $2\ \epsilon^{-1}\ c(e_i)$.*

Proof. If O has no sending event:
If this type of interval is repeated more than once without a different type of interval in between, such that distance between v_O and v_C is decreased in multiple steps, then O's delay costs must be increased by at least ϵ in each step (starting with ϵ). Thus if the distance between v_O and v_C is decreased in x ($x \geq 1$) steps, O must have at least $\epsilon(1 + 2 + 3 + 4 + ... + x)$ delay costs, and the competitive ratio will be at most:

$$\frac{C_C}{C_O} \leq \frac{2\ x\ c(e_i)}{\epsilon(1 + 2 + 3 + 4 + ... + x)} = \epsilon^{-1}\ \frac{2\ x\ c(e_i)}{\frac{x}{2}(x+1)} = \epsilon^{-1}\ \frac{4\ c(e_i)}{x+1}$$

The competitive ratio $\frac{C_C}{C_O} \leq \epsilon^{-1}\ \frac{4\ c(e_i)}{x+1}$ is maximal if $x = 1$ and will never exceed $2\ \epsilon^{-1}\ c(e_i)$.

If O has one or more sending events, then the series is composed of the following two type of intervals:

(i) One or more series of intervals with O having a sending event. The competitive ratio of each of these series will be at most:

$$\frac{C_C}{C_O} \leq \frac{2\ x\ c(e_i)}{x\ c(e_i)} = 2 \leq 2\ \epsilon^{-1}\ c(e_i)$$

(ii) At most one series of intervals where O has no sending event. The competitive ratio will be at most (as proven above):

$$\frac{C_C}{C_O} \leq \frac{2\ x\ c(e_i)}{\epsilon(1 + 2 + 3 + 4 + ... + x)} = \epsilon^{-1}\ \frac{2\ x\ c(e_i)}{\frac{x}{2}(x+1)} = \epsilon^{-1}\ \frac{4\ c(e_i)}{x+1} \leq 2\ \epsilon^{-1}\ c(e_i)$$

Hence a series of intervals having $\delta_0 > \delta_1$ without a different type of interval in between has a competitive ratio of at most $2\ \epsilon^{-1}\ c(e_i)$. □

As Lemma 3.3.5 has been made without any assumptions of which values are being sent by the algorithm, it also holds for this algorithm, and we can analyze each interval I_2 in combination with an interval I_3.
The same amortized cost analysis can be applied to a series of intervals I_4. In this case, the series stops if a different type of interval is reached. The amortized competitive ratio of both combined intervals is proven in the next lemma.

Lemma 3.3.16. *Consider an arbitrary interval $]t_{C_0}, t_{C_1}]$ between two sending events of C. Let δ_0 denote the difference between v_O and v_{C_0} ($\delta_0 = |v_O - v_{C_0}|$) at time t_{C_0} and δ_1 denote the difference between v_O and v_{C_1} ($\delta_1 = |v_O - v_{C_1}|$) at time t_{C_1}.*
Then:

(i) The amortized competitive ratio of the interval I_4 is at most $4\ \epsilon^{-1}\ + 4$.

(ii) The amortized competitive ratio of the interval I_2 will be at most 8.

Proof. It is clear that each series of intervals I_4 must be preceded by at least one of the intervals I_3,I_5 or I_6, and according to Lemma 3.3.5, each interval I_2 is preceded by exactly one interval I_3.
According to Lemma 3.3.6, the competitive ratio of interval I_2 can be amortized with the help of interval I_3. As I_3 can be used to amortize both intervals I_2 and I_4, we have to double I_3's competitive ratio in each of the amortized competitive ratio analysis for both intervals.

(i) We thus obtain a new upper bound for the amortized interval I_2, namely $\frac{2c(e_i)+2c(e_i)}{\frac{1}{2}c(e_i)} = 8$.

(ii) From these intervals (I_3,I_5 or I_6), 4 is the worst competitive ratio, so the amortized competitive ratio of the combined interval I_4 will be at most:

$$\begin{aligned}
\frac{C_C}{C_O} &= \frac{max(\ C_{C_{I_3}}\ ,\ C_{C_{I_5}}\ ,\ C_{C_{I_6}}\ ) + C_{C_{I_4}}}{max(\ C_{O_{I_3}}\ ,\ C_{O_{I_5}}\ ,\ C_{O_{I_6}}\ ) + C_{O_{I_4}}} \\
&\leq \frac{2c(e_i) + C_{C_{I_4}}}{\frac{1}{2}c(e_i) + C_{O_{I_4}}} \\
&\leq \frac{2c(e_i) + 2\ \epsilon^{-1}\ c(e_i)}{\frac{1}{2}c(e_i) + 0} \\
&\leq 4\ \epsilon^{-1}\ + 4
\end{aligned}$$

□

We therefore obtain the following table of amortized competitive ratios:

	Interval	**maximal c.r.**	**amortized maximal c.r.**	**amortizes**	**amortized by**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	2	2		
I_2	$\delta_0 \neq 0$ and $\delta_1 = 0$	∞	8		I_3
I_3	$\delta_0 = 0$ and $\delta_1 \neq 0$	2	$4\ \epsilon^{-1} + 8$	I_2 and I_4	
I_4	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 > \delta_1$	$2\ \epsilon^{-1}\ c(e_i)$	$4\ \epsilon^{-1} + 4$		I_3, I_5 or I_6
I_5	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 < \delta_1$	4	$4\ \epsilon^{-1} + 4$	I_4	
I_6	$\delta_0 \neq 0$, $\delta_1 \neq 0$ and $\delta_0 = \delta_1$	4	$4\ \epsilon^{-1} + 4$	I_4	

Thus the competitive ratio of ∞ or $2\ \epsilon^{-1}\ c(e_i)$ is never reached, and Algorithm C has an upper bound of $4\ \epsilon^{-1} + 8$ for the competitive ratio, under the condition that C's delay costs are fixed to $c(e_i)$ for each interval.

Theorem 3.3.17. *Algorithm C is $O(\epsilon^{-1} + \frac{\Delta}{c(e_i)})$-competitive to O.*

Proof. As the amortized competitive ratio of an interval can never exceed $4\epsilon^{-1} + 8$, the final competitive ratio of an arbitrary sequence can never exceed $4\epsilon^{-1} + 8$, under the condition that C's delay costs are fixed to $c(e_i)$ for each interval. According to Lemma 3.3.2, removing this limitation will increase the upper bound to $\frac{3}{2}(4\epsilon^{-1} + 8) + \frac{\Delta}{c(e_i)} = 6\ \epsilon^{-1} + 12 + \frac{\Delta}{c(e_i)}$.
Therefore C will have an upper bound of $O(\epsilon^{-1} + \frac{\Delta}{c(e_i)})$. □

3.4 General Trees

3.4.1 Sending Criterion

If there are more than two nodes in a tree, the sending criterion from A, B and C has to be adjusted accordingly. This is shown in Algorithm 1 which is executed at each node n_i. Let each message sent be composed of the value, the weight (the number of data nodes this message represents) and the link it is sent through (e.g the node's nodeid who sent the message).
As all the values will be aggregated, each message will have a fixed size and each node will have at most one sending event in each time step. Each node can reconstruct the aggregated value of its children on the basis of the aggregated values it received, the number of nodes those aggregated values represent (the weight), and who was sending the aggregated value (the link), as we assumed that the tree is fixed and no node can leave or join the tree.

Figure 3.1 shows an example of a small tree with two leaf nodes 1 and 2, both having node 3 as parent, which in turn has the root node as parent. Assume we are interested in the average over all values of the tree, we therefore use the average function to aggregate values at the nodes which are chosen by algorithm A. The default last sent value (l.s.v.) is set to 0 and each node keeps track of the l.s.v. for each incoming edge. Only the values at the leaves change over time and the edge costs (e.c) are chosen as shown in the figure below. Table 3.1 shows the execution of the algorithm on the tree.

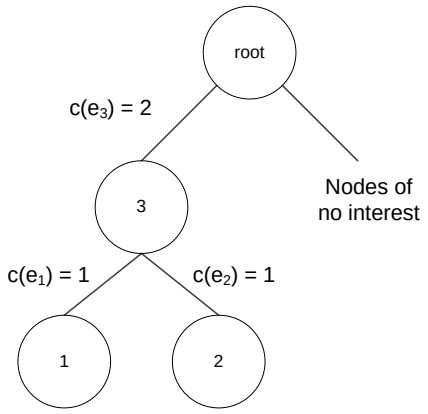

Figure 3.1: Tree T

3.4.2 Lower Bound for A,B and C

Lemma 3.4.1. *All three algorithms (A, B and C) have a lower bound of* $\Omega(\frac{\sum_{i=0..h-1} c(e_i)}{c(e_0)})$ *for the competitive ratio for the chain of nodes.*

Proof. Let the adversary choose a fixed value (e.g. the value 1) such that the leaf node (A, B and C have the same sending criterion and will in this case send the same value) sends the value 1 to its parent. Just after the sending event of our online algorithm, the adversary will change the value back to its original value such that no delay costs will occur for O anymore. O will therefore have at most the following costs:

$$C_O \quad \leq \quad c(e_0)$$

Algorithm 1 Algorithm A, B or C executed at node n_i

Require: $Sold_{i,0} = \varnothing$, $S_{i,0} = \varnothing$, $dc = 0$

for $t = 0$ to ∞ **do**
  $S_{i,t}\{i\} = (thisnodevalue, 1)$ // if the node is a data node
  //Update $S_{i,t}$ with newly received messages
  for $(value, weight, link) \in$ *incoming message at time* t **do**
    $S_{i,t}\{link\} \leftarrow (value, weight)$ // insert/overwrite old tuple sent from same link
  end for
  // calculate costs
  for $link \in S_{i,t}$ **do**
    $(value, weight) \leftarrow S_{i,t}\{link\}$
    if $link \in Sold_{i,t}$ **then**
      $(lastsentvalue, lastsentweight) \leftarrow Sold_{i,t}\{link\}$
    else
      $(lastsentvalue, lastsentweight) \leftarrow (0, 0)$
    end if
    $dc \leftarrow dc + weight \cdot |value - lastsentvalue|$
  end for
  // send messages
  if $dc \geq c(e_i)$ **then**
    for $link \in S_{i,t}$ **do**
      $(value, weight) \leftarrow S_{i,t}\{link\}$
      $value \leftarrow update(value)$ // depending on A, B, C
      $S_{i,t}\{link\} \leftarrow (value, weight)$
    end for
    // agregation function over all values in set
    // sends aggregated value and sum over all weights
    $send(aggr(S_{i,t}))$
    $Sold_{i,t+1} \leftarrow Sold_{i,t}$
    for $link \in S_{i,t}$ **do**
      $Sold_{i,t+1}\{link\} \leftarrow S_{i,t}\{link\}$
    end for
    $S_{i,t+1} \leftarrow \varnothing$
    $dc \leftarrow 0$
  else
    $S_{i,t+1} \leftarrow S_{i,t}$
    $Sold_{i,t+1} \leftarrow Sold_{i,t}$
  end if
end for

t	node id	values from	value	l.s.v.	e.c.	d.c	$\sum d.c.$	remarks
0	1	1	0	0	1	0	0	No change.
	2	2	0	0	1	0	0	No change.
	3	/	/	/	2	0	0	
	r	/	/	/	/	/	/	
1	1	1	2	0	1	2	2	Sends $(2, 1)$ to parent.
	2	2	0	0	1	0	0	No change.
	3	/	/	/	2	0	0	
	r	/	/	/	/	/	/	
2	1	1	2	2	1	0	0	l.s.v set. d.c. reset.
	2	2	1	0	1	1	1	Sends $(1, 1)$ to parent.
	3	1	2	0	2	2	2	Sends $(2, 1)$ to root.
	r	/	/	/	/	/	/	
3	1	1	2	2	1	0	0	
	2	2	1	1	1	0	0	l.s.v set. d.c. reset.
	3	1	2	2	2	0	0	l.s.v set. d.c. reset.
		2	1	0	2	1	1	$\sum d.c. < e.c.$
	r	3	2	/	/	/	/	Aggr. root value changed to $\frac{2 \cdot 1}{1} = 2$
4	1	1	2	2	1	0	0	
	2	2	1	1	1	0	0	
	3	1	2	2	2	0	0	
		2	1	0	2	1	2	Sends $(1.5, 2)$ to root.
	r	3	2	/	/	/	/	
5	1	1	2	2	1	0	0	
	2	2	1	1	1	0	0	
	3	1	2	2	2	2	2	
		2	1	1	2	0	0	l.s.v set. d.c. reset.
	r	3	1.5	/	/	/	/	Aggr. root value changed to $\frac{1.5 \cdot 2}{2} = 1.5$

Table 3.1: Execution of Algorithm A on T.

While our online algorithm has at least the following costs (the value 1 will be forwarded until it reaches the root):

$$C_A \geq \sum_{i=0..h-1} 2\, c(e_i)$$

The competitive ratio will therefore be at least $\Omega(\frac{\sum_{i=0..h-1} c(e_i)}{c(e_0)})$. □

The same lower bound can naturally also be achieved in the tree.

3.4.3 Upper Bound for C in General Trees

From here on, we will restrict ourself to the analysis of Algorithm C only, because C yields the lowest upper bound for the case of 2 nodes. The analysis of algorithm A and B is similar.

Let there be a chain of nodes $n_0, n_1, n_2, ..., n_h$, $h + 1$ nodes in total. Let n_0 denote the leaf node and n_h denote the root node. Each node n_i has exactly one edge with communication costs $c(e_i)$ to its parent node n_{i+1}.

Lemma 3.4.2. *Algorithm C is $O(\epsilon^{-1} \frac{\sum_{i=0}^{h-1} c(e_i)}{c(e_0)} + h \frac{\Delta}{c(e_0)})$-competitive.*

Proof. Again, as in the case of two nodes, we look at all the different types of intervals that exist for C at the leaf node n_0.
For a conservative upper bound, we assume that for each sending event of C at the leaf node n_0, C waits and is not able to aggregate the value on its way up with the value of another sending event, such that there will be exactly one sending event at the nodes $n_1, ...n_{h-1}$ for each sending event of C at the leaf node n_0. The costs for C can not be greater if we would allow C to aggregate the values (which it does in reality). We can therefore conclude our analysis with the above assumption.
We use the sum of local delay costs to calculate the global delay costs of the cost function of our online algorithm C. The local delay costs at time t are defined as the difference between the last sent value and the current value at time t. It is clearly seen that:

$$global\ delay\ costs\ at\ time\ t \quad \leq \quad \sum_{i=0}^{h-1} local\ delay\ cost\ at\ node\ n_i\ at\ time\ t$$

E.g., let's assume that the adversary chooses one value and switches directly after C's sending event back to the original value. At that point, the local delay costs are still increased until both changes reach the root, while the global delay costs stay unchanged.
As for O, we assume that O has no communication costs and no delay costs for each of the consecutive nodes in the chain (As shown in Lemma 3.4.1, this is not far fetched), such that only costs at the leaf node occur for O.
Even though O's sending events take longer than in the case of two nodes, O must have at least the costs from the case of two nodes, as otherwise this would be a contradiction to our proof made before (we could use that specific case for the case of two nodes by delaying O's sending events by $h-1$ time steps).
We therefore obtain the following costs for the leaf node ($\Delta \geq 0$): (This is taken from the case of two nodes)

$$\begin{aligned} C_C &\leq (6\ \epsilon^{-1} + 12)\ c(e_0) + \Delta \\ C_O &\geq c(e_0) \end{aligned}$$

And the following costs at the nodes $n_1, ..., n_{h-1}$ for each sending event of C at the leaf node n_0:

$$\begin{aligned} C_C &= communication\ cost + delay\ cost \\ &\leq c(e_i) + \mathbf{max}(\ 2\ c(e_i), \Delta\ ) \\ &\leq 3\ c(e_i) + \Delta \\ C_O &\geq 0 \end{aligned}$$

Thus for each sending interval at the leaf node n_0, C and O will have the following costs over

the entire chain from leaf to root:

$$\begin{aligned} C_C &\leq (6\,\epsilon^{-1}+12)\,c(e_0)+\Delta+\sum_{i=1}^{h-1}(3\,c(e_i)+\Delta) \\ &\leq (6\,\epsilon^{-1}+12)\sum_{i=0}^{h-1}c(e_i)+h\,\Delta \\ C_O &\geq c(e_0) \\ \frac{C_C}{C_O} &\leq \frac{(6\,\epsilon^{-1}+12)\sum_{i=0}^{h-1}c(e_i)+h\,\Delta}{c(e_0)} \\ &= (6\,\epsilon^{-1}+12)\,\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}+h\,\frac{\Delta}{c(e_0)} \end{aligned}$$

Please note that $\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}$ is equal to the lower bound for Algorithm C. Therefore C is at most $O(\epsilon^{-1}\,\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}+h\,\frac{\Delta}{c(e_0)})$-competitive for each sending event of C at the leaf node n_0.
C is thus $O(\epsilon^{-1}\,\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}+h\,\frac{\Delta}{c(e_0)})$-competitive. □

3.4.4 Upper Bound for C in Trees of Height h

Theorem 3.4.3. *Algorithm C is* $O(\boldsymbol{max}_{\,l\in\,L}(\,\epsilon^{-1}\,\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}+h\,\frac{\Delta}{c(e_0)})\,)\,)$*-competitive.*

Proof. As we assumed that O has no communication and delay costs at the nodes $n_1,....n_{h-1}$ and that C can not merge any packets at the nodes $n_1,....n_{h-1}$, the proof from Lemma 3.4.2 is also valid for each leaf-root chain in the tree.
As there is more than one leaf-root chain in the tree, we have to take the one with the highest competitive ratio.
C is thus $O(\mathbf{max}_{\,l\in\,L}(\,\epsilon^{-1}\,\frac{\sum_{i=0}^{h-1}c(e_i)}{c(e_0)}+h\,\frac{\Delta}{c(e_0)})\,)\,)$-competitive. □

3.5 Optimal Offline Algorithm

3.5.1 2-Node Networks

The optimal algorithm can send any value any time it wants. Hence at each sending event, the optimal algorithm can choose the sending candidate from an infinite pool of numbers between the maximal and minimal value of the entire input sequence σ. We can therefore only calculate an approximation of the optimal algorithm, as there are infinite candidates for each sending event of the optimal algorithm.
One possibility is to reduce the optimal algorithm, such that it can only send values from the input sequence σ. Even then, a brute force approach would take $|\sigma|^{(|\sigma|+1)}$ steps to compute, as in each time step, the optimal algorithm can decide whether to send (if so choose one value from the pool of at most $|\sigma|$ values) or not to send.
We can however compute the optimal solution by using a modified variant of Dooly's algorithm [1]. Dooly's algorithm computes the optimal solution based on optimal subsolutions, which it combines by using dynamic programming.
Let $LeafValues[]$ denote an array containing the value of the leaf node in each time step, let $edgecosts$ denote the costs of sending and let $psv[]$ denote the array of distinct values in σ (each

value occurs at most once) and psv denote the number of values in $psv[]$. Algorithm 2 computes the optimal solution in $O(t^3\ psv)$. In the second loop, j is increased up to $i+1$ to also handle the case when there is no sending event at time $t=0$ (all fields in the sending events array for $i+1$ are always set to $false$). The cost function needs at most $t\ psv$ steps to complete, as at most psv different values have to be checked for each optimal subsolution's last sending event.
Table 3.2 shows the execution of the algorithm for a small example. Each row i takes at most $(t+1)\ psv$ steps to compute, as at most psv different sending values have to be checked for at most $t+1$ subsolutions in each row.
It is clear that when the node value at the leaf node stays the same for x time steps until time t, the optimal algorithm will not have a sending event in any of the $x-1$ time steps before t. Furthermore, instead of recalculating the costs of the subsolutions again and again, we can further improve the algorithm by caching the costs of the optimal subsolutions. Let $LeafValues[]$ denote an array of consecutive distinct values in σ (each value differs from its previous value) and let $at[]$ denote their arrival time. Algorithm 3 computes an optimal solution in $O(n^3\ psv)$, where n denotes the number of consecutive distinct values in σ and psv denotes the number of distinct values in σ ($n \geq psv$).
Please note that Algorithm 3 also assumes that it can freely choose the $lastsentvalue$ (instead of setting it to the default value 0) at the begining of the sequence σ. This can easily be fixed by removing the loop on line 9 and setting $lastsentvalue$ to 0 on line 1, if $j == i+1$ on line 7.

v_t	0	1	2	3	3	0	0	$\sum costs$
$i=0$	$send$							4
$i=1$		$send$						5
$i=2$			$send$					7
$i=3$	$send(2)$			$send$				10
or		$send(2)$		$send$				10
or		$send(3)$		$send$				10
or				$send$				10
$i=4$		$send(3)$			$send$			10
$i=5$	$send(2)$					$send$		13
or		$send(2)$				$send$		13
or						$send$		13
$i=6$							$send$	13

Table 3.2: Execution of Algorithm 2 with $c(e_0)=4$, default last sent value is set to 0.

3.6 Simulation

In order to see how our algorithms perform in real life scenarios (the "average" case), we built a simulation framework to compare our algorithms. Moreover we are interested in finding out if the algorithm having the lowest upper bound (Algorithm C) outperforms Algorithm A, whose lower bound is greater than Algorithm C's upper bound for two nodes. We are also interested in seeing how the algorithms perform with more nodes, as the proven lower bound for C in a general tree is highly artificial and probably rarely happens in a real world scenario.
We simulated our information aggregation algorithms with freely available sensor data measurements from the Sensorscope project [13]. The Sensorscope project consists of about 100 nodes

Algorithm 2 Optimal offline information aggregation algorithm for two nodes, $O(t^3\ psv)$

Initialize: $edgecosts \leftarrow c(e_i)$, $LeafValues[]$, $psv[]$
$sendingevents[LeafValues.size+1][LeafValues.size]$
$sentvalues[values.size+1][LeafValues.size]$
$sendingevents[.][.] \leftarrow false$, $sentvalues[.][.] \leftarrow 0$
// compute solution
for $i = 0$ to $LeafValues.size - 1$ **do**
  $min \leftarrow \infty$
  for $j = 0$ to $i + 1$ **do**
    $(costs, chosenvalue) \leftarrow getcosts(0, i, j) + edgecosts$
    if $costs < min$ **then**
      $min \leftarrow costs$
      $sendingevents[i][.] \leftarrow sendingevents[j][.]$
      $sendingevents[i][i] \leftarrow true$
      $sentvalues[i][.] \leftarrow sentvalues[j][.]$
      if $j \leq i$ **then**
        $sentvalues[i][j] \leftarrow chosenvalue$
      end if
    end if
  end for
end for
// Optimal sending events for leaf node in sendingevents[LeafValues.size-1]
// Optimal sending candidates in sentvalues[LeafValues.size-1]

Procedure getcosts(start,stop,j):
// Calculates costs from start to stop
$min \leftarrow infty$
for $k = 0$ to $psv.size - 1$ **do**
  $costs \leftarrow 0$, $chosenvalue \leftarrow 0$, $lastsentvalue \leftarrow 0$
  for $m = start$ to $stop$ **do**
    $costs \leftarrow costs + |lastsentvalue - LeafValues[m]|$
    if $sendingevents[j][m] == true$ **then**
      $costs \leftarrow costs + edgecosts$
      if $m == j$ **then**
        $lastsentvalue \leftarrow psv[k]$
      else
        $lastsentvalue \leftarrow sentvalues[j][m]$
      end if
    end if
  end for
  if $costs < min$ **then**
    $chosenvalue \leftarrow lastsentvalue$
    $min \leftarrow costs$
  end if
end for
return $(costs, chosenvalue)$
End Procedure

Algorithm 3 Optimal offline information aggregation algorithm for two nodes, $O(n^3\ psv)$

Initialize: $edgecosts \leftarrow c(e_i)$, $LeafValues[]$, $psv[]$, $at[]$
$costcache[LeafValues.size+1]$, $sentvalues[values.size+1][LeafValues.size]$
$sendingevents[LeafValues.size+1][LeafValues.size]$
$sendingevents[.][.] \leftarrow false$, $sentvalues[.][.] \leftarrow 0$
$at[LeafValues.size] \leftarrow t$ // otherwise last value is skipped
for $i = 0$ to $LeafValues.size - 1$ **do**
  $min \leftarrow \infty$
  for $j = 0$ to $i + 1$ **do**
    $costs \leftarrow \infty$
    $chosenvalue \leftarrow 0$
    if $i == 0$ **or** $j == i + 1$ **then**
      // costcache not set
      for $k = 0$ to $psv.size - 1$ **do**
        $lcosts \leftarrow 0$
        $lastsentvalue \leftarrow psv[k]$
        for $m = 0$ to i **do**
          $lcosts \leftarrow lcosts + (at[m+1] - at[m]) \cdot |lastsentvalue - values[m]|$
        end for
        if $lcosts < costs$ **then**
          $chosenvalue \leftarrow lastsentvalue$
          $costs \leftarrow lcosts$
        end if
      end for
      $costs \leftarrow costs + edgecosts$
    else
      for $k = 0$ to $psv.size - 1$ **do**
        $lcosts \leftarrow 0$
        $lastsentvalue \leftarrow psv[k]$
        for $m = j + 1$ to i **do**
          $lcosts \leftarrow lcosts + (at[m+1] - at[m]) \cdot |lastsentvalue - values[m]|$
        end for
        if $lcosts < costs$ **then**
          $chosenvalue \leftarrow lastsentvalue$
          $costs \leftarrow lcosts$
        end if
      end for
      $costs \leftarrow costs + costcache[j] + edgecosts$
    end if
    if $costs < min$ **then**
      $min \leftarrow costs$
      $sendingevents[i][.] \leftarrow sendingevents[j][.]$
      $sendingevents[i][i] \leftarrow true$
      $sentvalues[i][.] \leftarrow sentvalues[j][.]$
      $costcache[i] \leftarrow costs$
      if $j \leq i$ **then**
        $sentvalues[i][j] \leftarrow chosenvalue$
      end if
    end if
  end for
end for

dispersed over a relatively small area (the EPFL campus) for which sensor data measurements are available at the project website.

3.6.1 Topology

As GPS positions are available for each node, we connected all the nodes by setting each link's weight to $distance^2$ and constructed a directed rooted shortest path tree for each node (The idea is that a node's value probably does not differ much from it's neighbour's value and fluctuations probably affect both of them, such that it makes sense to connect them). We then finally choose the shortest path tree with the minimal total edge weight for our simulations. Figure 4.1 shows the resulting tree.
The remaining edges' weights were then set to the same constant as we assumed that the cost of sending a message is independent of the distance between the two nodes (Referring to the energy requirements for sending and receiving in the Sensor Network Museum [14]).

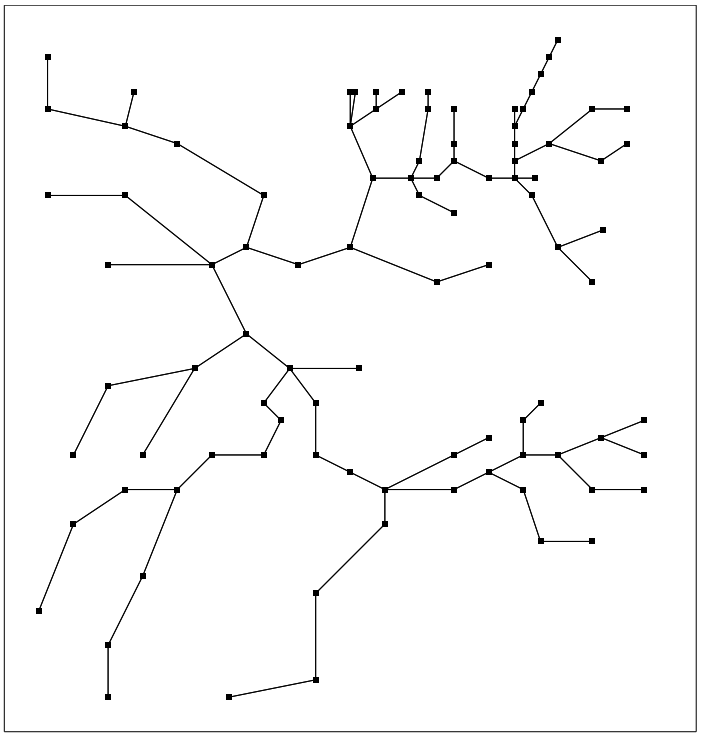

Figure 3.2: Sensorscope tree as of March 2007.

3.6.2 Data

All nodes in the sensorscope project have at least a temperature sensor and some even have additional sensors (e.g., for wind, rain, solar...). We decided to use the ambient temperature, as this sensor measurement was present at all of the sensor scope nodes. Unfortunately, the time between two consecutive measurements can take up to 30 seconds depending on the node, and some data even contains incorrect measurements, such as spikes or completely wrong measurements due to improper sensor behavior or transmission errors. Therefore we decided to ignore all sensor measurements spikes over 5 °C and use the last validated value instead.
Our online algorithms are all synchronized algorithms and each algorithm is executed once in

each time step, which is equal to 1 second in the simulation. As it can take up to 30 seconds for the next measurement to be read, we decided not to interpolate the data and always use the last measured value when no new sensor data is available.
The following results are based on one day of measurements as a larger data set would take too long to simulate. An initial 1000 rounds were not assessed, such that the start up phase does not interfere with the maximal variance (Experimental results showed that the first values reach the root after at most 500 rounds).

3.6.3 Results

As an optimal solution, which minimizes both the communication costs and the delay costs, is hard to compute, we compare the results to a pseudo optimal algorithm which has no communication costs and immediately sends the results to the root. Every change will therefore take exactly the number of hops seconds to reach the root.
We also evaluated a time based online algorithm ($10s$), which whenever a new value or a measurement arrives, waits 10 seconds and then sends the aggregated value to its parent.
Last but not least, we added a synchronized periodic online algorithm ($10s$ $sync.$), which forwards all values every 10 seconds towards the root. Every change will therefore take at most the number of hops plus 10 seconds to reach the root.

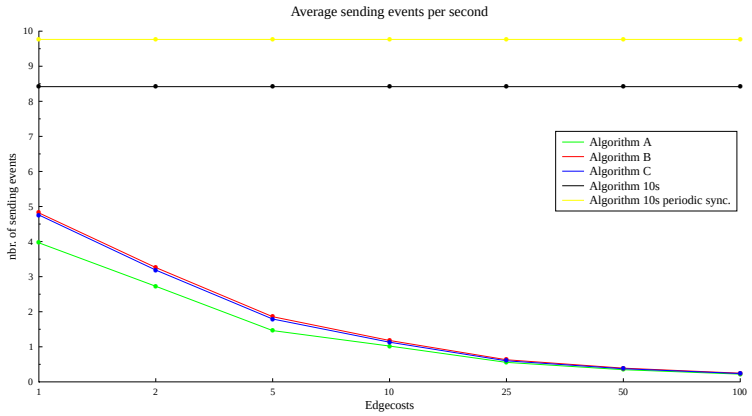

Figure 3.3: Average number of sending events per second

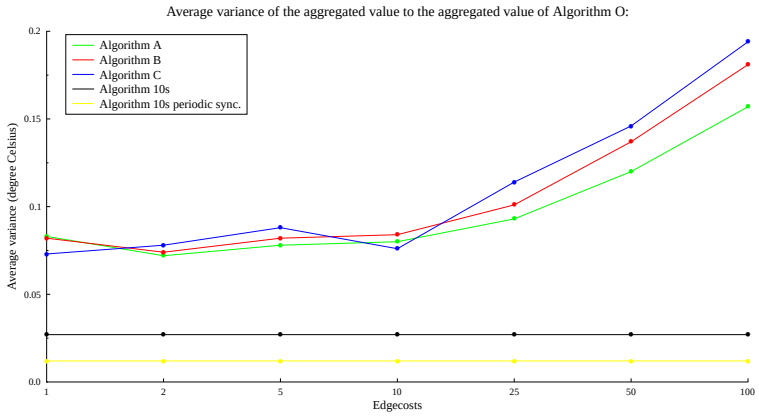

Figure 3.4: Average variance of the aggregated value to the aggregated value of Algorithm O

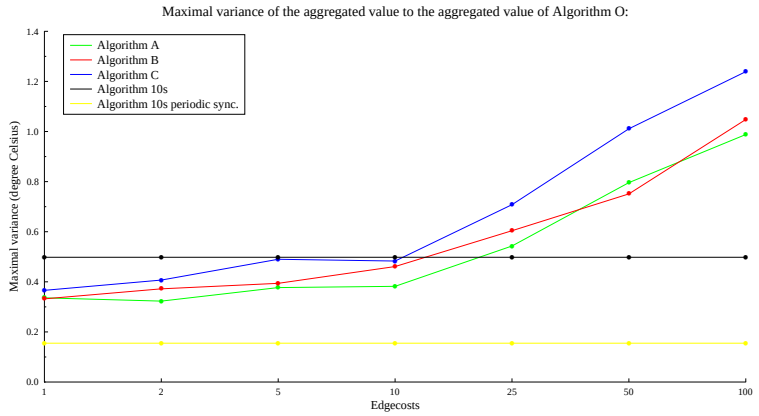

Figure 3.5: Maximal variance of the aggregated value to the aggregated value of Algorithm O

Figure 3.3 shows the average number of sending events per second. As A, B and C's wait before sending a value all depend on the edge costs, it is clear that the average number of sending events is decreasing with increasing edge costs. $10s$'s and $10s$ $sync$'s average number of sending events stays constant, as its sending criterion is independent of the edge costs. With small edge costs, A has a significant advantage over B and C. This comes from the fact that A sends the last value in the interval opposed to the value that occurred most or the average value. As most temperature measurements at the nodes increase or decrease over a longer period of time, it is clear that sending the last value is better than sending any other value, as it is closer to the future measurements and generates less local delay costs and therefore less sending events. As the edge costs raise, this advantage gets smaller and smaller.
Figure 3.4 and 3.5 show the average/maximal aggregated value variance to the aggregated value of Algorithm O. Algorithm A outperforms both B and C and its variance is only in rare occasions greater than these of B and C, while A's number of sending events is always smaller than B or C's. $10s$'s and $10s$ $sync$'s variances are much smaller than A's, but $10s$ and $10s$ $sync$ also

have more sending events than A.
E.g. for edge costs set to 25, A has 0.55596 sending events in average, while $10s$ has 8.418 and $10s$ $sync$ has 9.765. A's average root value variance is 0.093 °C while $10s$'s is 0.027 °C and $10s$ $sync$'s average root value variance is 0.012 °C. A therefore only has one sending event for every 17 sending events $10s$ $sync$ has, while its average variance is only 7.75 times higher than $10s$ $sync$'s.
Hence, even though A's lower bound is higher than the upper bound of B and C for the one link case, its performance is generally better than the performance of B or C, since A's lower bound example is practically inexistent with real sensor data. Also if it is acceptable to have a slightly greater variance, A, B and C all have fewer sending events than any of the time based online algorithms.

4 The Acknowledgement Problem

4.1 Model

Instead of a leaf node having values which change over time and which have to be sent to the root, each leaf node has to acknowledge each packet it receives. These ACK packets have to be sent to the root and multiple ACK packets can be merged to one ACK packet. Again, the goal is to minimize the delay costs for the ACK packets while keeping the communication costs low. This model can be mapped to our information aggregation model:

- Leaf node values are increased by 1 for each ACK packet which has to be sent towards the root.
- The sum function is used as aggregation function at all the nodes.
- On a sending event, both algorithms (our algorithm and the optimal one) have to send the current value at the node. This value minus the last sent value represents the number of ACK packets which are being sent to the parent.

In consequence, each ACK packet not having reached the root, will therefore increase the global delay costs by 1 in each time step. There are a few observations to consider:

- Values at the nodes can only increase. This comes from the fact that the values at the leaf nodes are increased by 1 for each ACK packet received.
- As the optimal algorithm is bound to send the current value at the node, the optimal algorithm can not acknowledge packets in advance. (In our information aggregation model, the optimal algorithm could send a higher value in advance such that it has less costs in the future)

Let O have global knowledge over all packets waiting in the tree. As each packet represents a unique token which has to be send towards the root, none of the algorithms is able to ”skip” or acknowledge those packets in advance.
Again as in the preceding model, we use Algorithm A at each node to send back the ACK packets, in our case, the last value in the sending interval. Algorithm A has a sending event if

the sum of delay costs in the sending interval is greater than or equal to the outgoing edge costs. In the context of ACK packets, this means that A will acknowledge the number of ACK packets at the node if the delay costs for the ACK packets at the node are greater than or equal to the outgoing edge costs. This algorithm is equal to the distributed algorithm proposed by Khanna [3][1].
Let us recall that the delay costs for A can never exceed $c(e_i) - 1 + (c(e_i) - 1 + \Delta)$ between two consecutive sending events of A, where Δ is the maximal difference between two consecutive values at a leaf node n_i. In this case, Δ will denote the maximum number of ACK packets which can arrive in one time unit a leaf node n_i.
Also node values at the leaf nodes are restricted to integers (as it does not make sense to send half ACK packets), and Δ can be indefinitely large. The edge costs $c(e_i)$ can be an arbitrary function as well.
We assume that the optimal algorithm has global knowledge.

4.2 2-Node Networks

With the help of the two observations from above, we are able to reduce the competitive ratio of A from $O(c(e_i) + \frac{\Delta}{c(e_i)})$ to 3. This is shown in the following lemmas.
Again as before, we will analyze all possible kinds of intervals that exist for A between two consecutive sending events of A at one node, written as interval $]t_{A_x}, t_{A_{x+1}}]$. This interval contains all node values of each time step between t_{A_x} and (including) $t_{A_{x+1}}$. (The start event at the beginning of the entire sequence of values also counts as a sending event but without communication costs).
With the help of the two observations from above, we can enhance Lemma 3.3.1, eliminate O's upper bound for the costs and imply that O must have at least k delay costs in this interval. Hence there must be at least k ACK packet arrivals for O in this interval.

Lemma 4.2.1. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. When the delay costs for A in this interval are equal to $2(c(e_i) - 1) + k$, where $k \geq 0$, then O has at least delay costs of k in $[t_{A_1} - 1, t_{A_1}]$. In addition there must be at least k ACK packet arrivals in this interval.*

Proof. Let v_l be the value at t_{A_1}, and v_{l-1} the value at $t_{A_1} - 1$. Since the delay costs for A are greater than $2(c(e_i)-1)$ in the entire interval, A must have at least a delay cost of $(c(e_i)-1)+k$ on v_l (otherwise A would have an earlier sending event) and the delay costs of the values before v_l in the interval are at most $c(e_i) - 1$. This implies that the difference between v_l and v_{l-1} is at least k and O will have at least delay costs of k in $[t_{A_1} - 1, t_{A_1}]$ (under the condition that O has no sending event), no matter what value v_O was.
Even if O has a sending event, O would still have delay costs of at least k in $[t_{A_1} - 1, t_{A_1}]$, as O can not acknowledge packets in advance.
This implies that there must be at least k ACK packet arrivals for O in this interval as the delay costs for O are at least k in this interval. □

Due to the removed limitations in Lemma 3.3.1, we can also improve Lemma 3.3.2 and remove the limitation on Δ.

[1] Khanna's paper [3] is unclear about whether the remaining delay costs of the child (delaycosts - edge costs) are added to the parent's delay costs in the first time step or not. We assume that the remaining delay costs are not added. Our proof also holds for the other case, as long as there are only costs added and not deducted from the delay costs.

Lemma 4.2.2. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A. Let the delay costs for A be fixed to $c(e_i)$ in this interval and let ρ be the competitive ratio of A in this interval.*
By letting the adversary choose another sequence of values in this interval, such that the delay costs for A are greater than $c(e_i)$ in this interval, the adversary can at most increase $\rho_{DC_A=c(e_i)}$ by $\frac{1}{2}\rho_{DC_A=c(e_i)}$, thus giving us $\rho_{DC_A\geq c(e_i)} \leq \frac{3}{2}\rho_{DC_A=c(e_i)}$.

Proof. Let DC_A denote the delay costs of A. By letting the adversary choose a different sequence of values in this interval, he might increase the delay costs for A from $c(e_i)$ to $2(c(e_i)-1)$ while leaving the delay costs for O unchanged, thus nearly doubling the delay costs for A. According to the preceding lemma, any further attempt by the adversary to increase the delay cost for A by another k, will also increase the costs for O by the same amount.
Thus, $\rho_{DC_A\geq c(e_i)} \leq \frac{3}{2}\rho_{DC_A=c(e_i)}$. □

We can now conclude our analysis by fixing A's delay cost in each interval $]t_{A_x}, t_{A_{x+1}}]$ to $c(e_i)$ and apply Lemma 4.2.2 afterwards.

Lemma 4.2.3. *Consider an arbitrary interval $]t_{A_0}, t_{A_1}]$ between two sending events of A and let A's delay costs be fixed to $c(e_i)$. Let δ_0 denote the difference between v_O and v_{A_0} ($|v_O - v_{A_0}|$) at time t_{A_0} and δ_1 denote the difference between v_O and v_{A_1} ($|v_O - v_{A_1}|$) at time t_{A_1}. If:*

(i) $\delta_0 = 0$ and $\delta_1 = 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_1.

(ii) $\delta_0 \neq 0$ and $\delta_1 \neq 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_2.

(iii) $\delta_0 = 0$ and $\delta_1 \neq 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_3.

(iv) $\delta_0 \neq 0$ and $\delta_1 = 0$, then the competitive ratio in this interval will be at most 2. Let this interval be called I_4.

Proof. In each of the following intervals, it is assumed that A has exactly $2c(e_i)$ costs, as we fixed the delay costs for A to be $c(e_i)$. It is clear that if O has at least one sending event in any of these intervals, the competitive ratio can be at most 2 for that interval.
Also, as all the values at the nodes can only rise and both O and A will send the current value at the node at a sending event, $v_O \leq v_A$ on t_{A_0} and t_{A_1}. In between t_{A_0} and t_{A_1}, v_O might also be greater than v_A as O might have a sending event in between.

(i) $\delta_0 = 0$ and $\delta_1 = 0$:
As $\delta_0 = 0$ and $\delta_1 = 0$, O must have at least one sending event in this interval and therefore the competitive ratio can be at most 2.

(ii) $\delta_0 \neq 0$ and $\delta_1 \neq 0$:
If O has no sending event in this interval, O must have at least the delay costs of A, as v_O is less than v_{A_0} at time t_{A_0} and the value at the node must have changed, as otherwise A would not send.
Therefore the competitive ratio can be at most 2.

(iii) $\delta_0 = 0$ and $\delta_1 \neq 0$:
If O has no sending event in this interval, O must have at least the delay costs of A as $\delta_0 = 0$.
Therefore the competitive ratio can be at most 2.

(iv) $\delta_0 \neq 0$ and $\delta_1 = 0$:
If O has no sending event in this interval, O must have at least the delay costs of A, as v_O is less than v_{A_0} at time t_{A_0}.
Therefore the competitive ratio can be at most 2.

□

We therefore obtain the following table of competitive ratios:

	Interval	**maximal competitive ratio**
I_1	$\delta_0 = 0$ and $\delta_1 = 0$	2
I_2	$\delta_0 \neq 0$ and $\delta_1 \neq 0$	2
I_3	$\delta_0 = 0$ and $\delta_1 \neq 0$	2
I_4	$\delta_0 \neq 0$ and $\delta_1 = 0$	2

Theorem 4.2.4. *Algorithm A is 3-competitive.*

Proof. According to Lemma 4.2.3, the competitive ratio of an interval can never exceed 2, under the condition that A's delay costs are fixed to $c(e_i)$. Thus the final competitive ratio of an arbitrary sequence can never exceed 2, under the condition that A's delay costs are fixed to $c(e_i)$ for each interval. According to Lemma 4.2.2, removing this limitation yields $\frac{3}{2}\ 2 = 3$ as new upper bound for the competitive ratio.
A is therefore 3-competitive. □

4.3 Chain Graphs

Let there be a chain of nodes $n_0, n_1, n_2, ..., n_h$, $h+1$ nodes in total. Let n_0 denote the leaf node and n_h denote the root node. Each node n_i has exactly one edge with communication cost $c(e_i)$ to its parent node n_{i+1}.

Lemma 4.3.1. *Algorithm A is $O(h\ \boldsymbol{max}_{i=0..h-1}(\ \frac{c(e_i)}{c(e_0)}\ ))$ competitive.*

Proof. Again, as in the case of two nodes, we look at all the different types of intervals that exist for A at the leaf node n_0.
For a conservative upper bound, we assume that for each sending event of A at the leaf node n_0, A waits and is not able to merge those packets on their way up with those of another sending event, such that there will be exactly one sending event at the nodes $n_1, ...n_{h-1}$ for each sending event of A at the leaf node n_0. The costs for A can not be greater if we allow A to merge packets (which it does in reality). We can therefore conclude our analysis with the above assumption.
Let us recall that A has a maximal delay cost of $2(c(e_i) - 1) + k$ ($k \geq 0$) in one sending interval at the leaf node n_0 and that O must have at least k delay costs as there are at least k ACK packet arrivals in this interval. O will have at least k delay costs at each of the nodes $n_1, ...n_{h-1}$ which follow, as each packet has to wait at least one time unit before being forwarded. As it is unclear on how O merges different packets, we assume that O has no communication costs at the nodes $n_1, ...n_{h-1}$.
We therefore obtain the following costs for the leaf node ($k \geq 1$, as A has a sending event at the leaf node and there must therefore exist at least one packet): (this is taken from the case of two

nodes)

$$\begin{aligned} C_A &\leq c(e_0) + 2\ c(e_0) + k = 3\ c(e_0) + k \\ C_O &\geq c(e_0) + k) \\ \frac{C_A}{C_O} &\leq \frac{3\ c(e_0) + k}{c(e_0) + k} \\ &\leq 3 \end{aligned}$$

And the following costs at the nodes $n_1, ...n_{h-1}$ for each sending event of A at the leaf node n_0:

- $c(e_i) \leq c(e_0)$:
  As the link costs at node n_i are less than or equal to the link costs at the leaf node n_0, the delay costs for A can not exceed $2(c(e_i) - 1) + k$. We obtain the following costs:

  $$\begin{aligned} C_A &\leq c(e_i) + 2c(e_0) + k \\ C_O &\geq k \end{aligned}$$

  O has k delay cost as each ACK packet must wait at least one time unit at each of the nodes $n_1, ...n_{h-1}$. This cost for O might occur at a different time unit than for A, but this cost will occur, as each packet will reach the root at the end.

- $c(e_i) > c(e_0)$:
  A must have at least $2\ c(e_0) + k$ delay costs, or in the case that the delay costs of the packets are less than $c(e_i)$, the delay costs for A must be less than $2\ c(e_i)$. (As the delay costs for A can at most double in each time step and A sends if the delay costs are greater than or equal to $c(e_i)$).
  We therefore obtain the following costs:

  $$\begin{aligned} C_A &\leq c(e_i) + \mathbf{max}(2\ c(e_0) + k\ , 2\ c(e_i)\ ) \\ &= \mathbf{max}(\ c(e_i) + 2\ c(e_0) + k\ ,\ 3\ c(e_i)\ ) \\ C_O &\geq k \end{aligned}$$

Thus for each sending interval at the leaf node n_0, A and O will have the following costs over the entire chain from leaf to root:

$$
\begin{aligned}
C_A &\leq 3\,c(e_0) + k + \sum_{i=1}^{h-1} \mathbf{max}(\, c(e_i) + 2\,c(e_0) + k\,,\; 3\,c(e_i)\,)\,) \\
&\leq 3\,h\,k + 3\,c(e_0) + 3\,(h-1)\,\mathbf{max}_{i=1..h-1}(\, c(e_i)\,) \\
&\leq 3\,h\,k + 3\,h\,\mathbf{max}_{i=0..h-1}(\, c(e_i)\,) \\
C_O &\geq c(e_0) + k + \sum_{i=1}^{h-1} k \\
&= c(e_0) + h\,k \\
\frac{C_A}{C_O} &\leq 3\,\frac{h\,k + h\,\mathbf{max}_{i=0..h-1}(\, c(e_i)\,)}{c(e_0) + h\,k} \\
&\leq 3\,\frac{h\,\mathbf{max}_{i=0..h-1}(\, c(e_i)\,)}{c(e_0)} \\
&= 3\,h\,\mathbf{max}_{i=0..h-1}(\,\frac{c(e_i)}{c(e_0)}\,)
\end{aligned}
$$

Therefore, A is at most $O(h\,\mathbf{max}_{i=0..h-1}(\,\frac{c(e_i)}{c(e_0)}\,))$-competitive for each sending event of A at the leaf node n_0.
Thus the final competitive ratio of an arbitrary sequence can never exceed $O(h\,\mathbf{max}_{i=0..h-1}(\,\frac{c(e_i)}{c(e_0)}\,))$, and A is $O(h\,\mathbf{max}_{i=0..h-1}(\,\frac{c(e_i)}{c(e_0)}\,))$-competitive. □

We can further improve this upper bound, as the chain of nodes only consists of one data node.

Lemma 4.3.2. *Let there be a chain of nodes $n_0, n_1, ..., n_h$ and let O have global information. For each sending event of O at the leaf node n_0, O has exactly one sending event at the nodes $n_1, n_2, ..., n_{h-1}$.*

Proof by contradiction. Let us assume this is not the case and let p_1 and p_2 denote the packets which will be merged on their way up in the chain. O could save communication costs by waiting and sending p_1 together with p_2 at the leaf node n_0, while the global delay costs rest unchanged. Hence O is not optimal and there will be exactly one sending event at the nodes $n_1, n_2, ..., n_{h-1}$ for each sending even at the leaf node. □

The next lemma limits the number of sending events of A between two sending events of O.

Lemma 4.3.3. *Let there be a chain of nodes $n_0, n_1, ..., n_h$. If:*

(i) O has local knowledge only, there are at most $2\,\sqrt{c(e_0)}$ sending events of A between two sending events of O at the leaf node n_0.

(ii) O has global knowledge, there are at most $2\,\sqrt{\sum_{i=0..h-1} c(e_i)}$ sending events of A between two sending events of O at the leaf node n_0.

Proof.

(i) O has local knowledge only:
Let p_1 denote the number of packets before A has its first sending event, p_2 denote the number of packets between the first and the second sending event of A, and so forth. It is clear that O must have a sending event if the number of packets reaches $c(e_0)$, as the delay

costs in the next step alone match the communication costs of sending the packets. The following inequations have to hold:

$$\begin{aligned} p_1 \frac{c(e_0)}{p_2} &< c(e_0) \\ (p_1 + p_2) \frac{c(e_0)}{p_3} &< c(e_0) \\ (p_1 + p_2 + p_3) \frac{c(e_0)}{p_4} &< c(e_0) \\ &\ldots \\ (p_1 + p_2 + p_3 + ... + p_n) \frac{c(e_0)}{p_{n+1}} &< c(e_0) \end{aligned}$$

$\frac{c(e_0)}{p_2}$ represents the minimum amount of time steps between the first sending event of A and the second sending event. If $p_1 \frac{c(e_0)}{p_2} \geq c(e_0)$, O has a sending event on the first sending event of A, as the costs of not acknowledging the packets p_1 are greater than or equal to the costs of sending them. The same argument can be applied to the the other inequations as well.
The inequations imply that the number of packets in p_1, p_2,...,p_n have to be increased by at least 1 in each step, the last inequation can therefore be reduced to:

$$(1 + 2 + 3 + 4 + 5 + ... + n) \frac{c(e_0)}{1+2+3+4+5+...+n+1} < c(e_0)$$

$1 + 2 + 3 + 4 + 5 + ... + n + 1 = \frac{n(n+1)}{2} + 1$ has to be less than $c(e_0)$, as otherwise it would be favorable for O to have a sending event as the delay costs in the next step alone match the communication costs of sending these packets. We therefore have to solve the following inequation and express n (the number of sending events of A) as a function of $c(e_0)$:

$$\begin{aligned} & \frac{n(n+1)}{2} + 1 \geq c(e_0) \\ \Rightarrow \quad & n \geq 2\sqrt{c(e_0)} \end{aligned}$$

This implies that n, the number of the sending events of A, can never exceed $2\sqrt{c(e_0)}$.

(ii) O has global knowledge:
It is clear that O must have a sending event if the number of packets reaches $\sum_{i=0..h-1} c(e_i)$, as the delay costs in the next step alone match the communication costs of sending the packets up to the root. Hence the same arguments as for the case where O has local knowledge only can be applied and the number of sending events of A between two sending events of O can not exceed $2\sqrt{\sum_{i=0..h-1} c(e_i)}$.

□

Lemma 4.3.4. *Algorithm A is $O(h \, \textbf{\textit{max}}_{i=0..h-1}(\, \textbf{\textit{min}} \, (\sqrt{c(e_i)} \, , \, \frac{c(e_i)}{c(e_0)} \,) \,) \,)$-competitive.*

Proof. In Lemma 4.3.1 we assumed that O had no communication costs at all. Lemma 4.3.3 proves that O has at least one sending event for each $2\sqrt{\sum_{n=0..h-1} c(e_i)}$ sending events of A. Instead of assuming that O has no communication costs (as we did in Lemma 4.3.1), we can imply with the help of Lemma 4.3.2 that O must have at least $\frac{c(e_i)}{2\sqrt{\sum_{n=0..h-1} c(e_i)}}$ communication

costs at each node n_i in the chain of nodes.
Thus for each sending interval at the leaf node n_0, A and O will have the following costs over the entire chain from leaf to root (the costs for A stay unchanged):

$$
\begin{aligned}
C_A &\leq 3\ h\ k + 3\ h\ \mathbf{max}_{i=0..h-1}(\ c(e_i)\ ) \\
C_O &\geq c(e_0) + k + \sum_{i=1}^{h-1} (k + \frac{c(e_i)}{2\ \sqrt{\sum_{i=0..h-1} c(e_i)}}) \\
&= c(e_0) + h\ k + \sum_{i=1}^{h-1} \frac{c(e_i)}{2\ \sqrt{\sum_{i=0..h-1} c(e_i)}} \\
&\geq h\ k + \frac{1}{2} \frac{\sum_{i=0..h-1} c(e_i)}{\sqrt{\sum_{i=0..h-1} c(e_i)}} \\
&= h\ k + \frac{1}{2} \sqrt{\sum_{i=0..h-1} c(e_i)} \\
\frac{C_A}{C_O} &\leq 3\ \frac{h\ k + h\ \mathbf{max}_{i=0..h-1}(\ c(e_i)\ )}{h\ k + \frac{1}{2}\sqrt{\sum_{i=0..h-1} c(e_i)}} \\
&\leq 3\ \frac{h\ k + h\ \mathbf{max}_{i=0..h-1}(\ c(e_i)\ )}{h\ k + \frac{1}{2}\mathbf{max}_{i=0..h-1}(\ \sqrt{c(e_i)}\ )} \qquad (c(e_i) \geq 1) \\
&\leq 6\ h\ \frac{\mathbf{max}_{i=0..h-1}(\ c(e_i)\ )}{\mathbf{max}_{i=0..h-1}(\ \sqrt{c(e_i)}\ )} \\
&\leq 6\ h\ \mathbf{max}_{i=0..h-1}(\ \sqrt{c(e_i)}\ )
\end{aligned}
$$

Therefore, A is at most $O(h\ \mathbf{max}_{i=0..h-1}(\ \sqrt{c(e_i)}\ ))$-competitive for each sending event of A at the leaf node n_0.
Thus the final competitive ratio of an arbitrary sequence can never exceed neither $O(h\ \mathbf{max}_{i=0..h-1}(\ \sqrt{c(e_i)}\ ))$ nor $O(h\ \mathbf{max}_{i=0..h-1}(\ \frac{c(e_i)}{c(e_0)}\ ))$ (according to Lemma 4.3.1), and A is thus $O(h\ \mathbf{max}_{i=0..h-1}(\ \mathbf{min}\ (\ \sqrt{c(e_i)}\ ,\ \frac{c(e_i)}{c(e_0)}\ )\ )\ )$-competitive. □

4.4 Trees of Height h

Theorem 4.4.1. *Algorithm A is $O(h\ \mathbf{max}_{\ l\in\ L}(\ \mathbf{max}_{i=0..h-1}(\ \frac{c(e_i)}{c(e_0)}\ )\ )\ )$-competitive.*

Proof. As we assumed that O has no communication costs at the nodes $n_1, ...n_{h-1}$, that only the minimal delay costs (when each packet is sent directly to the root) occur for O, and that A can not merge any packets at the nodes $n_1, ...n_{h-1}$, the proof from Lemma 4.3.1 is also valid for each leaf-root chain in the tree.
As there is more than one leaf-root chain in the tree, we have to take the one with the highest competitive ratio.
A is thus $O(h\ \mathbf{max}_{\ l\in\ L}(\ \mathbf{max}_{i=0..h-1}(\ \frac{c(e_i)}{c(e_0)}\ )\ )\ )$-competitive. □

Khanna [3] proved that algorithm A has an upper bound of $O(h\ log(\alpha))$, where α denotes the entire communication costs in the tree ($\alpha = \sum_{i=0}^{n-2} c(e_i)$). Our proof shows that the same

algorithm has an upper bound of $O(h \ \mathbf{max}_{l \in L}(\ \mathbf{max}_{i=0..h-1}(\ \frac{c(e_i)}{c(e_0)} \ ) \ ) \ )$. Depending on the type of tree, the one upper bound outranks the other. E.g., in a tree where all edges have equal weight, our upper bound can be reduced to $O(h)$, while Khanna's upper bound still depends on $h \ log(\alpha)$.

4.5 Optimal Offline Algorithm

4.5.1 2-Node Networks

Dooly's algorithm [1] can compute the optimal solution on a single link in $O(n^2)$ (whereby n denotes the number of packet arrivals), but only handles arrivals of single packets at the node and assumes that no delay costs occur for packets if they are sent in the same time step.
We therefore modified Dooly's algorithm to take into account that more than one packet can arrive at a time and that a packet causes delay costs even if sent in the same time step.
As the problem of choosing the right sending events possesses the optimal substructure property, we can use a dynamic programming algorithm to compute the optimal solution. Our version is shown in Algorithm 4. Let $nbrAcks[]$ denote an array containing the number of packets arriving in each time step and $edgecosts$ denote the costs of sending. Algorithm 4 computes the optimal solution in $O(t^3)$ time. In the inner loop, j is increased up to $i+1$ to also handle the case when there is no sending event at time $t = 0$ (all fields in the sending events array for $i+1$ are always set to $false$).
It is clear that the optimal algorithm will not have a sending event if no new packets arrive. Furthermore, if a packet arrives at time t, O must have a sending event in the next $c(e_i)$ time steps. In this case, the inner loop therefore only has to check the last $c(e_i)$ subsolutions at time $t + c(e_i)$, as there must be an optimal subsolution in the last $c(e_i)$ time steps. Also, we can further improve the algorithm by caching the costs of the optimal subsolutions and skip the empty packet arrivals. Let $nbrAcks[]$ denote an array containing the number of non empty packet arrivals ($\#packets > 0$!) and let $arrivaltimes[]$ denote their arrival time. Algorithm 5 computes an optimal solution in $O(n(n + c(e_i)^2))$, where n denotes the number of packet arrivals at the leaf node.

4.5.2 Chain Graphs

Lemma 4.3.2 proved that the optimal algorithm does not aggregate packets on their way up in the chain of nodes, which implies that each packet will take h time steps to reach the root after being sent at the leaf node, whereby h denotes the height of the chain of nodes ($h + 1 = \#nodes$). Hence the problem of choosing the right sending events only depends on the packets at the leaf node, as all other nodes will directly send towards their parent and we can use our modified variant of Dooly's algorithm to compute an optimal solution for the sending events at the leaf node.
This is shown in Algorithm 6. Instead of taking the edge costs of a single link, we use the sum of all edge costs, as the optimal algorithm will directly send the packets to the root if once forwarded from the leaf node. The so occurred additional delay costs (as the packets do not reach the root until $h - 1$ time steps later) are added to the costs of sending at the leaf node. Hence, all our subsolutions are optimal subsolutions and we can use dynamic programming to compute the optimal solution of the entire sequence. Algorithm 6 computes an optimal solution in $O(n(n + (\sum c(e_i))^2)$, where n denotes the number of packet arrivals at the leaf node.

Algorithm 4 Optimal algorithm for two nodes, $O(t^3)$

Initialize: $edgecosts \leftarrow c(e_i)$, $nbrAcks[]$
$sendingevents[nbrAcks.size + 1][nbrAcks.size]$
$sendingevents[.][.] \leftarrow false$
// compute solution
for $i = 0$ to $nbrAcks.size - 1$ **do**
  $min \leftarrow \infty$
  for $j = 0$ to $i + 1$ **do**
    $costs \leftarrow getcosts(0, i, j) + edgecosts$
    if $costs < min$ **then**
      $min \leftarrow costs$
      $sendingevents[i][.] \leftarrow sendingevents[j][.]$
      $sendingevents[i][i] \leftarrow true$
    end if
  end for
end for
// Optimal sending events for leaf node in sendingevents[nbrAcks.size-1]

Procedure getcosts(start,stop,j):
// Calculates costs from start to stop
$costs \leftarrow 0$, $packets \leftarrow 0$
for $k = start$ to $stop$ **do**
  $packets \leftarrow packets + nbrAcks[k]$
  $costs \leftarrow costs + packets$
  if $sendingevents[j][k] == true$ **then**
    $costs \leftarrow costs + edgecosts$
    $packets \leftarrow 0$
  end if
end for
return $costs$
End Procedure

Algorithm 5 Optimal algorithm for two nodes, $O(n(n + c(e_i)^2))$

Initialize: $edgecosts \leftarrow c(e_i)$, $nbrAcks[]$, $arrivaltimes[]$
$sendingevents[nbrAcks.size + 1][nbrAcks.size]$
$costcache[nbrAcks.size + 1]$
$sendingevents[.][.] \leftarrow false$, $costcache[.] \leftarrow 0$, $jpos \leftarrow 0$
// compute solution
for $i = 0$ to $nbrAcks.size - 1$ **do**
  $min \leftarrow \infty$
  for $j = jpos$ to $i + 1$ **do**
    $costs \leftarrow edgecosts$
    if $i == 0$ **or** $j == i + 1$ **then**
      // costcache not set
      for $k = 0$ to i **do**
        $costs \leftarrow costs + (arrivaltimes[i] - arrivaltimes[k] + 1) \cdot nbrAcks[k]$
        // +1 such that packets will also have delay costs at the sending event.
      end for
    else
      for $k = j + 1$ to i **do**
        $costs \leftarrow costs + (arrivaltimes[i] - arrivaltimes[k] + 1) \cdot nbrAcks[k]$
        // +1 such that packets will also have delay costs at the sending event.
      end for
      $costs \leftarrow costs + costcache[j]$
    end if
    if $costs < min$ **then**
      $min \leftarrow costs$
      $sendingevents[i][.] \leftarrow sendingevents[j][.]$
      $sendingevents[i][i] \leftarrow true$
      $costcache[i] \leftarrow costs$
    end if
  end for
  // update jpos
  if $i - edgecosts > 0$ **then**
    $jpos \leftarrow i - edgecosts$
  end if
end for
// Optimal sending events for leaf node in sendingevents[nbrAcks.size-1]

Algorithm 6 Optimal algorithm for chain of nodes nodes, $O(n(n + (\sum c(e_i))^2)$

Initialize: $edgecosts \leftarrow \sum_{i=0..h-1} c(e_i)$, $nbrAcks[]$, $arrivaltimes[]$, h
$sendingevents[nbrAcks.size + 1][nbrAcks.size]$
$costcache[nbrAcks.size + 1]$
$sendingevents[.][.] \leftarrow false$, $costcache[.] \leftarrow 0$, $jpos \leftarrow 0$
// compute solution
for $i = 0$ to $nbrAcks.size - 1$ **do**
  $min \leftarrow \infty$
  for $j = jpos$ to $i + 1$ **do**
    $costs \leftarrow edgecosts$
    if $i == 0$ **or** $j == i + 1$ **then**
      // costcache not set
      for $k = 0$ to i **do**
        $costs \leftarrow costs + (arrivaltimes[i] - arrivaltimes[k] + h) \cdot nbrAcks[k]$
        // +h such that packets will also have delay costs until they reach the root.
      end for
    else
      for $k = j + 1$ to i **do**
        $costs \leftarrow costs + (arrivaltimes[i] - arrivaltimes[k] + h) \cdot nbrAcks[k]$
        // +h such that packets will also have delay costs until they reach the root.
      end for
      $costs \leftarrow costs + costcache[j]$
    end if
    if $costs < min$ **then**
      $min \leftarrow costs$
      $sendingevents[i][.] \leftarrow sendingevents[j][.]$
      $sendingevents[i][i] \leftarrow true$
      $costcache[i] \leftarrow costs$
    end if
  end for
  // update jpos
  if $i - edgecosts > 0$ **then**
    $jpos \leftarrow i - edgecosts$
  end if
end for
// Optimal sending events for leaf node in sendingevents[nbrAcks.size-1]

4.5.3 Trees

As the local subsolutions are not necessarily optimal ones (the sending events also depend on the arrival of packets at other leaf nodes), we can not apply the same approach as above. E.g., if a lot of packets arrive at another leaf node such that the optimal algorithm will directly send them to the root, it could be favorable for another leaf node to send its packets as well such that the packets can be aggregated with those of the other leaf node on their way up the tree (even though sending might not be the best local solution). It is still an open question whether the problem of finding an optimal offline solution for the tree is NP-hard or not.

4.6 Simulation

In order to simulate the different algorithms, we developed a simulation framework based on Sinalgo [15]. In order to get as accurate results as possible, all the simulations were run multiple times with different seeds. The sum of the communication costs and the delay costs (the smaller the better) were used to measure the performance of the different algorithms:

$$\begin{aligned} C &= Communication\ cost\ + Delay\ cost \\ &= Communication\ cost\ + \sum_{t\ \in\ time\ steps} |aggr(v_0^t, v_1^t, ..., v_{n-1}^t) - aggr(r_0^t, r_1^t, ..., r_{n-1}^t)| \end{aligned}$$

The random trees in our simulation are created as follows:
The nodes are uniformly placed at random on a plane and connected to each other by setting each link's weight to $distance^2$. We then constructed a directed rooted shortest path tree for each node and chose the shortest path tree with the minimal total edge weight for our simulations.
The arrival rate was modeled as a poisson process as this is widely used as a model for simulation of network traffic.
All our simulation results have to be considered with precaution. Even though we will see that some algorithms outperform others, we can not imply that the one algorithm is better than the other algorithm, as our simulations only represent a very tiny subset of all possible configurations and provide no theoretical proof for the performance of each algorithm.

4.6.1 Algorithms

Periodical Algorithms:

Definition 4.6.1 ($Periodic_1$)**.** *If there are no packets at the node and a new packet arrives, the node waits x seconds and then sends all its packets towards its parent.*

Definition 4.6.2 ($Periodic_2$)**.** *The node sends its packets every x seconds towards its parent. The nodes are synchronized such that each child has a sending event one time step before its parent has a sending event.*

Both algorithms have the deficiency of sending independently of the number of packets waiting at the node and independently of the edge costs. E.g., if there are very few packets waiting at the node, the node will send the packets even though the costs of sending can be very high. Or if there are many packets waiting at the node such that the costs of keeping them for one more time step outweigh the costs of sending them, the node still waits until the timer is reached.

Khanna Algorithm:
We tested three different implementations:

Definition 4.6.3 ($Khanna_1$)**.** *For each packet at a node, the local delay costs are increased by* 1 *in each time step. If the local total delay costs are greater than or equal to the edge costs, all the packets at the node are sent to the parent and the total delay costs are reset to* 0*.*
In addition, when one or more packets arrive at a parent node from a child, the remaining delay costs of the child (total local delay costs of the child minus costs of sending the packets) are added to the delay costs of the parent in the first time step.

Definition 4.6.4 ($Khanna_2$)**.** *As* $Khanna_1$*, but no delay costs from the child are added to the parent in the first time step.*

Definition 4.6.5 ($Khanna_3$)**.** *As* $Khanna_1$*, but instead of adding the remaining delay costs of the child, the cost of sending the packets is not deducted and all the delay costs of the child are added.*

As written previously, Khanna's paper [3] is unclear about whether the remaining delay costs of the child (delay costs - edge costs) are added to the parent's delay costs in the first time step or not. Our proof holds for all three interpretations, as long as the delay costs are only added and not deducted.
The performance of $Khanna_2$ and $Khanna_1$ is about equal (as we will see in the Results section), but far better than the performance of $Khanna_3$. This is due to the fact that the delay costs only increase and never decrease in $Khanna_3$. Hence the more hops a packet has passed, the more delay costs it has accumulated and the decision of sending will depend less and less on the number of packets waiting at the node, but more and more on the delay costs the packets have accumulated so far. E.g., in a tree where all edges have equal weight, $Khanna_3$ will only wait at the leaf nodes, and immediately send at each inner node of the tree.
This also shows that the upper bound for the competitive ratio for $Khanna_1$ and $Khanna_2$ is not tight, as the proof holds for all three interpretations.

Randomized Algorithms:

Definition 4.6.6 ($Random_1$)**.** *At each time step, the node chooses uniformly at random a value* x *between* 0 *and* 1*. The node will then send its packets if* x *is smaller than* $\chi \frac{\#packets}{c(e_i)}$*.*

Definition 4.6.7 ($Random_2$)**.** *At each time step, the node chooses uniformly at random a value* x *between* 0 *and* 1*. The node will then send its packets if* x *is smaller than* $\chi \frac{delay\ costs}{c(e_i)}$*. The delay costs are calculated as in* $Khanna_1$*.*

For an arrival rate modeled as poisson process with $\lambda = 10$ and 2 nodes, experimental simulation results showed that it is favorable to choose $\chi = 0.75$ for $Random_1$.
If $\chi = 0.5$, $Random_1$ sends on average whenever $\#packets = c(e_i)$, but it is better to increase χ and send the packets earlier, as the delay costs not only depend on the number of packets at the node, but also on the time the packets spend at the node.
For $Random_2$, experimental results showed that it is favorable to choose $\chi < 0.1$, because the delay costs increase much faster than the number of packets, as they also depend on time and not only on the number of packets at the node.
The disadvantage of these two randomized algorithms is that the performance of the algorithm heavily depends on the chosen χ.

Hierarchical Algorithms:
Another interesting approach is to make use of the hierarchical information the nodes might have. Let us assume that each node has information about s nodes above itself in the tree, and

the goal is to find a correlation between s and the performance of the executed algorithm.
The next algorithm assumes that each node is able to tell the nodes s hops further down in the tree when they are not allowed to send packets to their parents.

Definition 4.6.8 ($Hierarchical_1$)**.** *If a node receives new packets in a round, it adds them to an array of fixed size and calculates the $average$ of all values in this array (If the array is full, the first added entry in the array is dropped before adding the new value and calculating the average).*
The node then calculates the number of rounds it would take for $average$ packets to reach the sending criterion, namely send if the delay costs are greater or equal to the edge costs.
This number of rounds, which represents the next number of rounds the nodes are not allowed to send, is sent to all nodes s hops further down in the tree (It is assumed that the root can transmit this information in instant time).
Each node executes $Khanna_1$ and only sends if the delay costs are greater or equal to the edge costs and if the root did not explicitly tell them not to send.

Experimental simulation results showed that the performance of this algorithms decreases when the array size is reduced or when the average is approximated by only keeping track of the average and the number of values, and not of the entire array.
In some cases, this algorithm's performance is much worse than that of $Khanna_1$. E.g., let us assume the tree is composed of two nodes, the leaf node and the root node, and that $s = 1$ such that the root will tell the leaf node when it is not allowed to send. The adversary now delivers one packet to the leaf node, which will eventually be send to the root by both $Khanna_1$ and $Hierarchical_1$. In $Hierarchical_1$, the root will then calculate the next number of time steps the leaf node is not allowed to send, in this case $\frac{c(e_i)}{1} = c(e_i)$ time steps, and broadcast it to the leaf node. It is easily seen that the adversary can now achieve a very large competitive ratio by delivering a very high number of packets to the leaf node, which will not be delivered for the next $c(e_i)$ time steps by $Hierarchical_1$, while $Khanna_1$ will deliver them immediately. The performance thus heavily depends on the input distribution.
The next algorithm assumes that each node knows the maximal $\frac{edge\ cost}{node\ degree}$ ratio of the path leading to the node s hops further above and only sends if the delay costs of the packets waiting at the node are greater than the maximal $\frac{edgecost}{node\ degree}$ ratio of that path. The leaves' node degree is set to 1. The idea behind is that when a node has a sending event, the packets reach the node s hops above without having to wait.

Definition 4.6.9 ($Hierarchical_2$)**.** *Each node is allowed to periodically send packets to its parent every s time steps (A node is allowed to send one time step before its parent is allowed to send). Each node calculates the delay costs as in $Khanna_1$ and sends all its packets if it is allowed to send and if the delay costs are greater than or equal to the maximal $\frac{edge\ cost}{node\ degree}$ of the path leading to the node s hops further above.*

This algorithm also relies heavily on the input distribution, as it assumes that the other nodes send at the same rate then the node itself. As $Hierarchical_1$, this algorithm's performance can be also much worse than that of $Khanna_1$. E.g., let us assume the tree consists of many nodes as shown in Figure 4.1, and that $s = 2$, such that $\frac{edge\ cost}{node\ degree} = 1$ for both node n_0 and node n_1. The adversary now delivers one packet to the leaf node n_0, waits two time steps, delivers another one, etc. All of them will be send directly to the root, while $Khanna_1$ will merge them at node n_1, and not send them until $c(e_1)$ delay costs are reached (This also holds if $Hierarchical_2$ is not synchronized).
Another bad scenario is the case when there are many packets arriving at the leaf nodes, such

that it is favorable to send them immediately to the root. $Hierarchical_2$ only sends the packets every s time steps, while $Khanna_1$ sends immediately.
The next algorithm assumes that each node knows the total sum of all edge costs of the path to

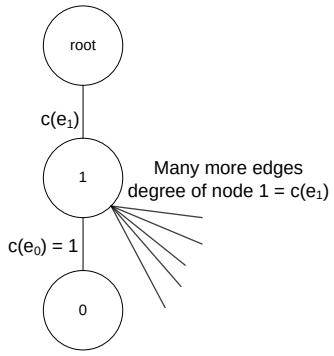

Figure 4.1: An example for $Hierarchical_2$'s bad performance

the node s hops further above.

Definition 4.6.10 ($Hierarchical_3$)**.** *Each node calculates the delay costs as in $Khanna_1$ and sends if the delay costs are greater or equal to $\sum_{i..i+s} c(e_i)$.*

This algorithms guarantees that when a node has sending event, the packets will reach the node s hops above without having to wait in between. If s is chosen too large, the performance of this algorithm will decrease, as the algorithm then only tries to aggregate packets at very few nodes. Figure 4.2 shows the influence of s on the different hierarchical algorithms' performances on a random tree with 256 nodes, $\lambda = 10$ and $c(e_i) = 100$.
It can be observed that $Hierarchical_2$'s performance increases with increasing s and that all hierarchical algorithms outperform $Khanna_1$. One possible reason is the following: if one node with high degree is connected to another node and there are many packets arriving, $Khanna_1$ will send very often, while all three hierarchical algorithms will aggregate the packets and only send once every few time steps (depending on s, the average number of packets received or the sum of edge costs). It's also favorable to choose s not too large, as otherwise the performance of $Hierarchical_1$ and $Hierarchical_3$ gets worse.
Figure 4.3 shows the influence of s in a random tree where the edges have not equal weight. $Hierarchical_2$ outranks $Khanna_1$ as well, for the same reason as mentioned above. $Hierarchical_1$'s performance however is worse than that of $Khanna_1$. $Hierarchical_3$'s performance heavily depends on the chosen s. As the total edge costs in this graph are greater than in the previous graph, $Hierarchical_3$'s performance is better with smaller s. This however makes the algorithm inappropriate for dynamic graphs where the topology is unknown, as s must best be chosen depending on the arrival rate and the weights of the edges.

Optimal Algorithms:

Definition 4.6.11 ($LocalOptimum$)**.** *Let $LocalOptimum$ be the optimal offline algorithm for a single link (one root node, one leaf node, and one link connecting both nodes) executed first at the leaf nodes, then at the leaf nodes' parents, and so forth, until the root of the tree is reached.*

As $LocalOptimum$ makes sending decisions only based on local information the node holds, $LocalOptimum$'s performance is with very high probability worse than the performance of an optimal algorithm with global knowledge, as it can not optimize the arrival of packets at the nodes higher up the tree.

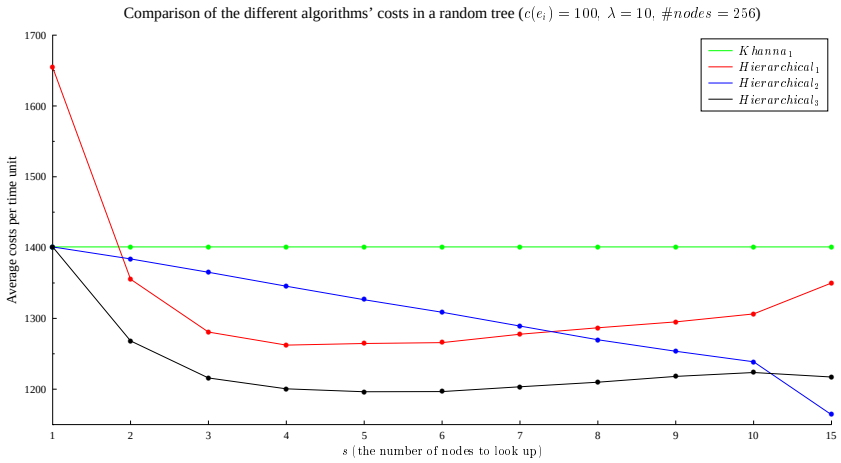

Figure 4.2: Comparison of the different algorithms' costs in a random tree ($c(e_i) = 100$, $\lambda = 10$)

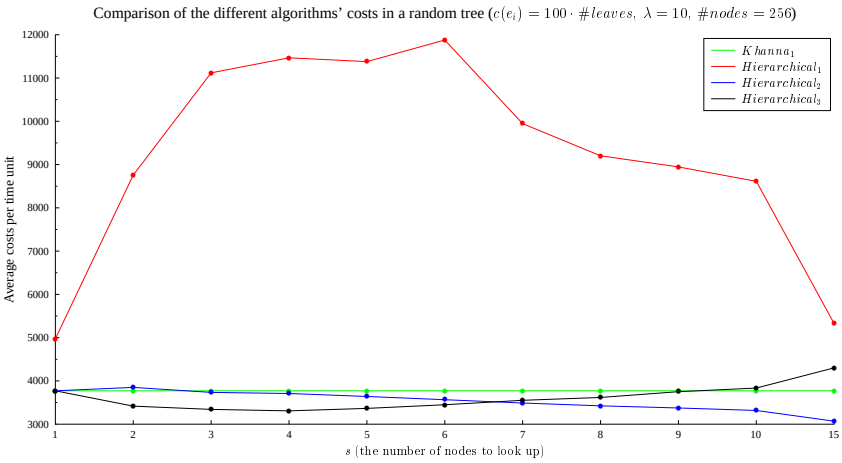

Figure 4.3: Comparison of the different algorithms' costs in a random tree ($c(e_i) = 100 * \#leaves$, $\lambda = 10$)

Figure 4.4 shows $Khanna_1$ in comparison to $LocalOptimum$. As the number of nodes increases, the performance of $Khanna_1$ outranks that of $LocalOptimum$. This is due to the fact that $Khanna_1$ aggregates more packets on their way up to the root, opposed to $LocalOptimum$, which only tries to minimize the costs on each link.

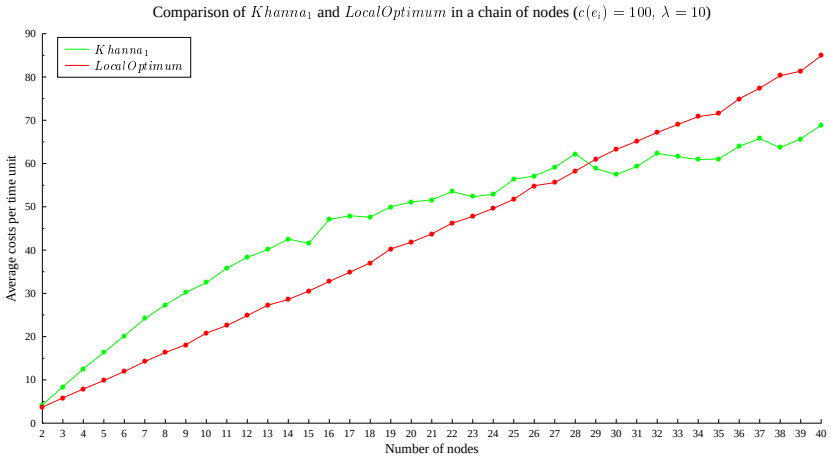

Figure 4.4: Comparison of $Khanna_1$ and $LocalOptimum$ in a chain of nodes ($c(e_i) = 100$, $\lambda = 10$)

4.6.2 Results

2 Nodes:
Figure 4.5 shows the costs of different algorithms in the case of two nodes. The arrival rate was modeled as a poisson process. χ and x were chosen such that the algorithms' performances are best for $\lambda = 10$.
Not surprisingly, $LocalOptimum$'s performance outranks all others. $Periodic_2$'s performance gets worse with smaller λ, as $Periodic_2$ waits too long before sending the packets. The same holds for $\lambda > 100$ (not shown in the graph), as $Periodic_2$ will send too often. Surprisingly, none of the other algorithms' overhead (compared to $LocalOptimum$) increases or decreases much with varying λ. This is probably due to the low arrival rate of packets (as at most one packet arrives at a time).

Chain of Nodes:
Figure 4.6 shows the percentage of variance between the best algorithm's average costs and the other algorithms' costs in function of the number of nodes in a chain of nodes. The arrival rate was modeled as a poisson process with $\lambda = 10$. χ was chosen such that the algorithms' performances are best for $\lambda = 10$ with $\#\ nodes = 2$. s was set to the maximal number of hops in the graph (as we have seen this to be a good choice for $Hierarchical_2$) and all edges have equal weight.
$Khanna_1$'s and $Khanna_2$'s performance is about equal and always worse than that of $Hierarchical_2$. Interestingly, $Random_1$ and $Random_2$ (who are not surprisingly close to each other) outperform some of the other algorithms if there are between 4 and 32 nodes in the chain

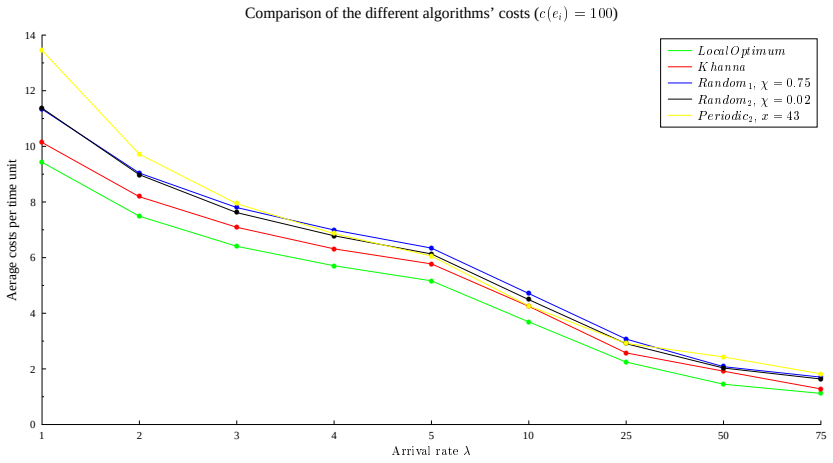

Figure 4.5: Comparison of different algorithms' costs for two nodes. ($c(e_0) = 100$)

of nodes. $Hierarchical_3$ outranks them all, as $Hierarchical_3$ only sends if the packet reach the root without any intermediate waiting, which is what the optimal algorithm would do as well. The greater the number of nodes, the smaller the gap between all the other algorithms and $Hierarchical_3$ is. This is due to the fact that $Hierarchical_3$ does not take into consideration the additional delay costs that occur until the packets reach the root ($Hierarchical_3$ could send the packets much earlier and they would even then reach the root without having to wait at intermediate nodes).
A similar result can be observed in Figure 4.7 where the edges have not equal weight and $c(e_i) = 100(maxhops - hops_{n_i})$.

Random Tree:
Figure ref 4.8 and 4.9 show the percentage of variance between the best algorithm's average costs and the other algorithms' costs in function of the number of nodes in a random tree. In all cases, $Hierarchical_2$ has the best performance of them all. The gap between $Hierarchical_2$ and the other algorithms increases with increasing number of nodes independently of the edge weights chosen. Also both $Random$ algorithms perform better than both versions of $Khanna$ with increasing number of nodes (although the number of nodes has to be much higher when the edges have not equal weight).

Fanout Tree:
Figure ref 4.10 and 4.11 show the percentage of variance between the best algorithm's average costs and the other algorithms' costs in function of the number of leaf nodes in tree with fanout set to 2. Again as in the case of a random tree, $Hierarchical_2$ has the best performance of them all. However, the gap between $Hierarchical_2$ and $Khanna$ is much smaller than before. When the edges do not have equal weight, it even gets smaller with greater number of leaf nodes.

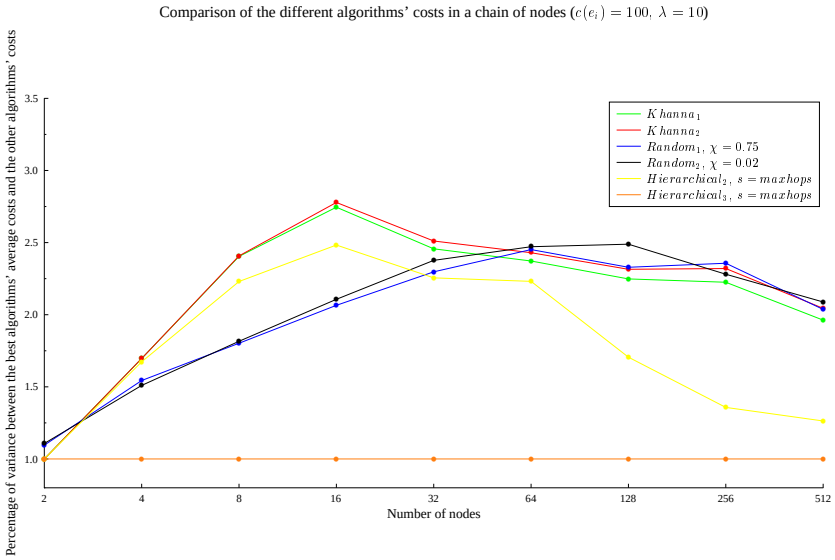

Figure 4.6: Comparison of different algorithms' costs for the chain of nodes. ($c(e_i) = 100$, $\lambda = 10$)

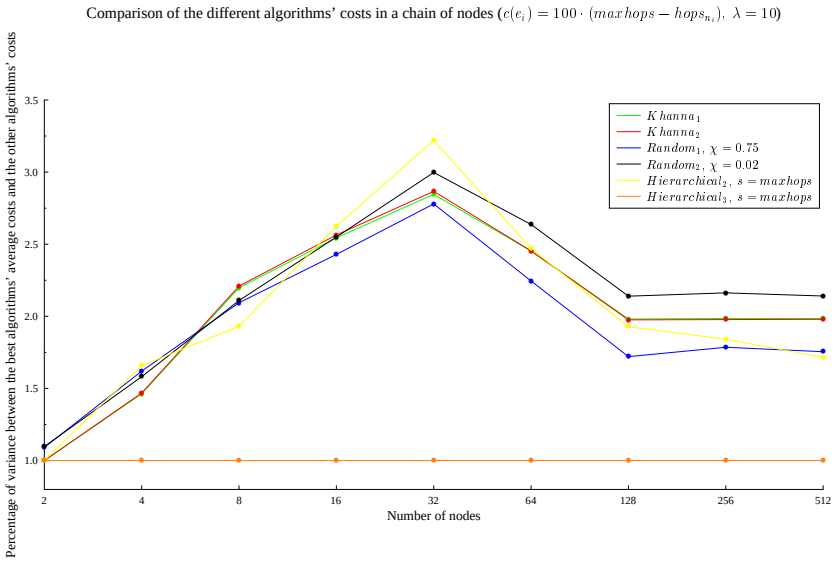

Figure 4.7: Comparison of different algorithms' costs for the chain of nodes. ($c(e_i) = 100(maxhops - hops_{n_i})$, $\lambda = 10$)

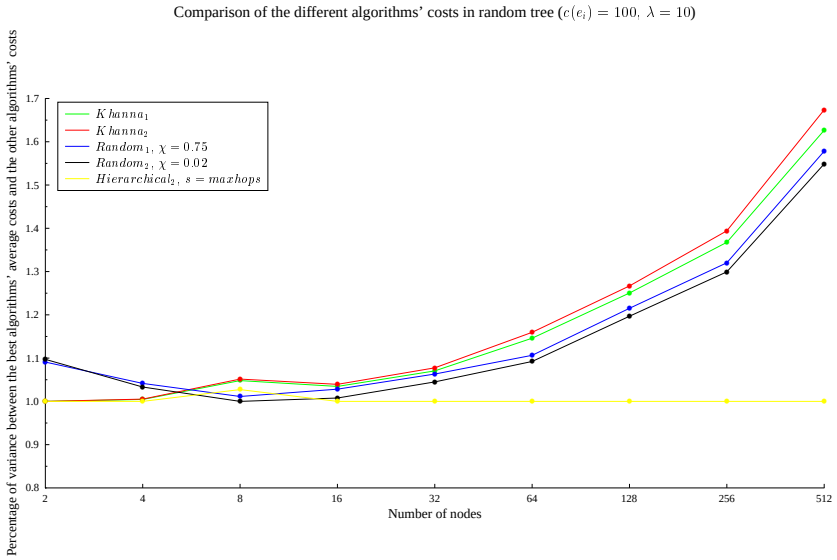

Figure 4.8: Comparison of different algorithms' costs in a random tree. ($c(e_i) = 100$, $\lambda = 10$)

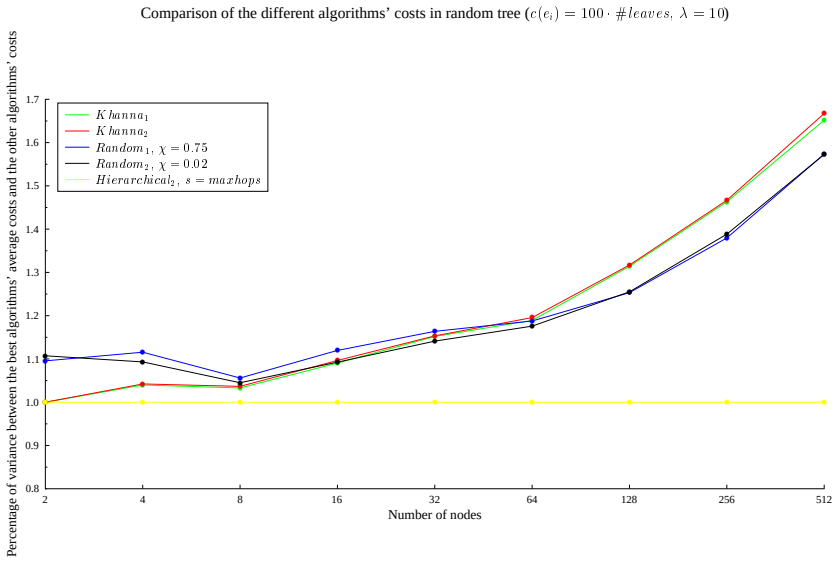

Figure 4.9: Comparison of different algorithms' costs in a random tree. ($c(e_i) = 100 * \#leaves$, $\lambda = 10$)

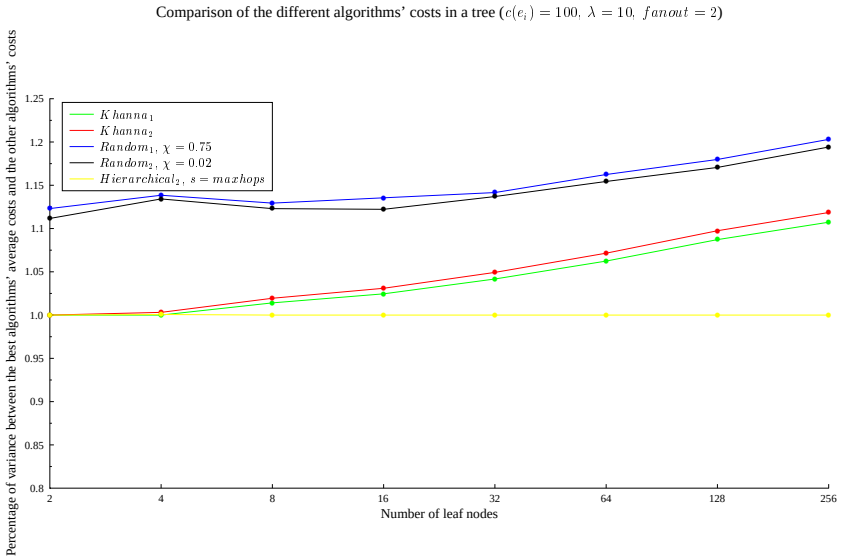

Figure 4.10: Comparison of different algorithms' costs in a tree. ($c(e_i) = 100$, $\lambda = 10$, $fanout = 2$)

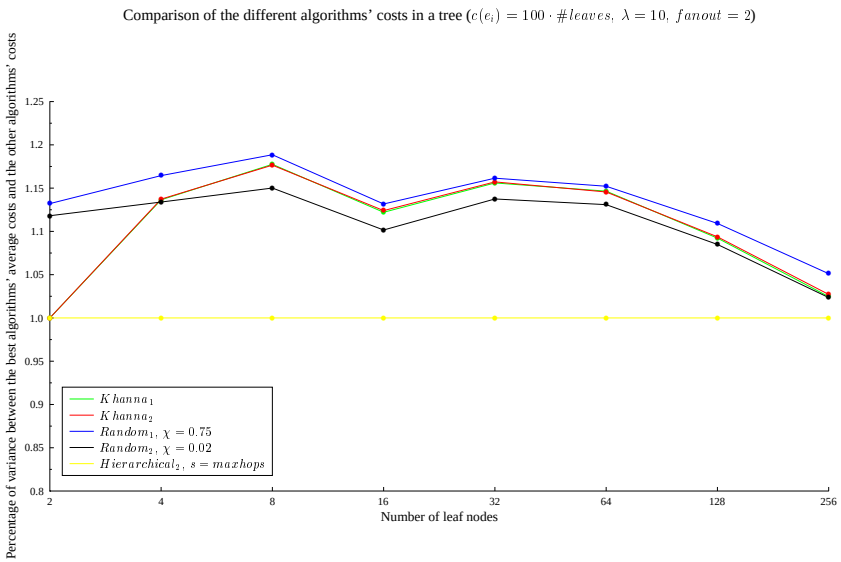

Figure 4.11: Comparison of different algorithms' costs in a tree. ($c(e_i) = 100 * \#leaves$, $\lambda = 10$, $fanout = 2$)

5 Conclusion

So far, information aggregation has mainly been studied in static environments. Chapter 3 analyzes three deterministic online information aggregation algorithms to gather dynamic data in a static tree. In addition, we introduce an optimal offline algorithm which computes the optimal solution for two nodes in $O(n^4)$. Even though we were unable to compute an optimal solution for the tree, simulations on real world sensor data showed that the algorithm's performance outranks the performance of timer based algorithms in terms of communication costs, but with slightly higher variance. In practice however, our online algorithm probably is even better, as fewer messages are sent and the arrival of children's messages at a node are not synchronized, such that less collisions will occur, which implies that even less messages will be sent.
One of many possible applications of our online algorithms could be on buoys implementing a tsunami warning system. The operator is interested in the maximal sea level and wants to be noticed as fast as possible if a tsunami arrives, while all other changes don't have to reach the operator immediately. If a tsunami arrives, our online algorithm (the one which sends the last value in the interval) forwards the new sea level directly to the root (the threshold has to be set accordingly), while small fluctuations take longer to reach the root, therefore saving energy.
While the focus on Chapter 3 was on information aggregation algorithms, Chapter 4 examines the acknowledge aggregation. We propose an alternative upper bound for Khanna's algorithm and present an optimal offline algorithm which computes the optimal solution for a chain of nodes in $O(n(n + (\sum c(e_i))^2)$ time steps. In addition, our simulations show that Khanna's performance is not always best, and many different algorithms can outrank Khanna depending on the topology and input distribution chosen.
Still, besides the already mentioned open questions, many open problems remain and can be addressed in future research projects. For example:

- For the ACK aggregation problem:
  - So far we extended Dooly's optimal offline algorithm to handle the single link, and the chain of nodes. Is it possible to use the same approach to calculate an optimal solution in a tree as well? If the problem of finding an optimal solution is NP-hard, is it possible to calculate a $k - approximation$ of the optimal solution?
  - What is the competitive ratio of randomized algorithms in trees?

- For the information aggregation problem:
  - can the optimal solution also be computed in the chain of nodes or even in a tree, where $\sum local\ delay\ costs \geq \sum global\ delay\ costs$?
  - Our analysis only covers deterministic algorithms. What would be the competitive ratio of randomized algorithms? What are their lower and upper bounds?
  - What influence on the complexity of our algorithms does it have if we drop the aggregation function in the delay cost function and use the sum of all $|v_i^t - r_i^t|$ instead? Will our algorithms improve? What happens if we limit the adversary to be only able to choose values such that the input sequence can only increase or decrease over a certain period of time (e.g two sending events of our online algorithm)? What happens if we introduce deadlines where values must reach the root after a certain time?
  - How do we best weight communication costs and delay costs? Are there better forms of measuring the performance of an information aggregation algorithm instead of adding delay costs and communication costs together?
  - In sensor networks, nodes often sleep, because being awake consumes too much energy. Can we design an efficient and clever sleeping algorithm, such that our information aggregation algorithms are still competitive?
  - How robust are our algorithms? How do we update our online algorithms to handle changing topologies (e.g. nodes joining/leaving, parent in rooted tree changes)? What happens if messages get lost or get modified? What if an adversary can choose which messages get lost or can modify single messages? What about collision of messages?
- For the simulations:
  - How do the different aggregation algorithms perform in comparison to the optimal offline algorithm?

Bibliography

[1] Daniel R. Dooly and Sally A. Goldman and Stephen D. Scott. TCP Dynamic Acknowledgment Delay: Theory and Practice. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 389–398, New York, 23–26 1998. ACM Press.

[2] Anna R. Karlin and Claire Kenyon and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In *STOC '01: Proceedings of the 33rd annual ACM symposium on Theory of computing*, pages 502–509, New York, NY, USA, 2001. ACM Press.

[3] Sanjeev Khanna and Joseph Naor and Danny Raz. Control Message Aggregation in Group Communication Protocols. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 135–146, London, UK, 2002. Springer-Verlag.

[4] Carlos Brito and Elias Koutsoupias and Shailesh Vaya. Competitive Analysis of Organization Networks or Multicast Acknowledgement: How Much to Wait? In *SODA '04: Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms*, pages 627–635, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[5] Susanne Albers and Helge Bals. Dynamic TCP acknowledgement: penalizing long delays. In *SODA '03: Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 47–55, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[6] Jens S. Frederiksen and Kim S. Larsen. Packet Bundling. In *SWAT '02: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, pages 328–337, London, UK, 2002. Springer-Verlag.

[7] Christos H. Papadimitriou and et al. Computational Aspects of Organization Theory (Extended Abstract). In *Proceedings of the 1996 European Symposium on Algorithms. Springer LNCS, 1996*.

[8] Christos H. Papadimitriou and Edouard Servan-Schreiber. The origins of the deadline: Optimizing communication in organizations. Complexity in Economics, 1999.

[9] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[10] Damon Mosk-Aoyama and Devavrat Shah. Computing Separable Functions via Gossip. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 113–122, 2006.

[11] Laurent Massoulié and Erwan Le Merrer and Anne-Marie Kermarrec and Ayalvadi Ganesh. Peer Counting and Sampling in Overlay Networks: Random Walk Methods. In *PODC '06: Proceedings of the 25th annual ACM symposium on Principles of distributed computing*, pages 123–132, New York, NY, USA, 2006. ACM Press.

[12] Boaz Patt-Shamir. A Note on Efficient Aggregate Queries in Sensor Networks. *Theor. Comput. Sci.*, 370(1-3):254–264, 2007.

[13] `http://sensorscope.epfl.ch/`. SensorScope Wireless Distributed Sensing System for Environmental Monitoring.

[14] `http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum`. The Sensor Network Museum.

[15] `http://dcg.ethz.ch/projects/sinalgo/`. Sinalgo: Simulator for Network Algorithms.