# A Protocol Scheme for a Class of Minimum Delay Routing Algorithms

Petra J.M. van Haaften

RUU-CS-91-43
November 1991


Utrecht University

**Department of Computer Science**

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

# A Protocol Scheme for a Class of Minimum Delay Routing Algorithms

Petra J.M. van Haaften

Technical Report RUU-CS-91-43
November 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

# A Protocol Scheme for a Class of Minimum Delay Routing Algorithms *

Petra J.M. van Haaften

Department of Computer Science, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands

**Abstract**

In this report protocols and proofs of correctness are given for a class of minimum delay routing algorithms. The class of routing algorithms is described by the iteration

$$\phi_i^{k+1} = \phi_i^k + \Delta\phi_i^k, \qquad i = 1, ..., N-1$$

where for each i $\phi_i$ is the routing vector at node i and the vector $\Delta\phi_i^k$ is a solution to the following problem:

$$\text{minimise } \delta_i^T \Delta\phi_i + \frac{t_i}{2\alpha}\Delta\phi_i^T M_i \Delta\phi_i$$

$$\text{subject to } \phi_i^k + \Delta\phi_i \geq 0, \sum_l \Delta\phi_{il} = 0, \Delta\phi_{il}(t) = 0 \text{ for } l \in B_i^k(t).$$

The class was proposed by Bertsekas in [Bert78] and discussed in [Gaf79], [BGG82], and [BGG84].

In this report a general scheme is given for the protocols which provide a distributed implementation of the algorithms that belong to the described class. A proof scheme for these protocols is given. Example algorithms are given; for each algorithm, a protocol that implements the algorithm is shown and a correctness proof for the protocol is given. As far as we know, correct protocols and proofs for the example algorithms were not presented before.

*This work was partly supported by the ESPRIT Basic Research Actions of the EC under contract no. 3075 (project ALCOM).

# 1 Introduction

In this report a general protocol scheme is presented for a class of minimum delay routing algorithms for use in computer networks. The algorithms are meant for quasi static routing, which means that they can be used in an environment in which bursts of changes occur, followed by a relative long period in which no changes occur. From the protocol scheme a protocol skeleton, as defined in [Sch91], can be derived for each algorithm in the class. In the context of this report, an *algorithm* is an iteration for computing the optimal routing vectors at the nodes, together with the additional formulas for the computation of the variables that occur in the iteration. Such an algorithm does not describe the computation of an iteration value in a *distributed* environment. A *protocol* is needed that does describe this distributed computation: this protocol must control the communication between the processors in the network and make sure that the actual values are computed by each processor. Protocols that perform this task can be deduced easily from the general protocol scheme presented in this report.

The class of algorithms considered in this report was proposed by Bertsekas and discussed by Bertsekas, Gafni, and Gallager in their report [BGG82], which also appeared as a paper ([BGG84]). Whereas [BGG82] analyses the mathematical properties of convergence for the algorithms from the class, it did not actually present distributed algorithms for it. The *sequential* algorithms from the class presented in [BGG82] compute routing vectors $\phi_i(t)$ for each destination t. These vectors denote the fractions of flow that node i must send over each of its outgoing links. The algorithms derive from sequential quadratic programming methods and iteratively update the routing vectors towards a situation in which the total delay in the network is minimal. Each algorithm in the class can be described by the iteration

$$\phi_i^{k+1} = \phi_i^k + \Delta\phi_i^k, \qquad i = 1, ..., N-1$$

where the vector $\Delta\phi_i^k$ is a solution to the problem:

$$\text{minimise } \delta_i^T \Delta\phi_i + \frac{t_i}{2\alpha} \Delta\phi_i^T M_i \Delta\phi_i$$

$$\text{subject to } \phi_i^k + \Delta\phi_i \geq 0, \sum_l \Delta\phi_{il} = 0, \Delta\phi_{il}(t) = 0 \text{ for } l \in B_i^k(t).$$

Sequential quadratic programming (SQP) methods are regarded as the most effective to solve constrained optimisation problems of the form: $minimise_{f \in \mathbf{R}^n}\ D_T(f)$ *subject to* $c(f) \geq 0$. The function $D_T$, minimised by the algorithms described here, denotes the expected number of messages/sec being transmitted times the expected delay per message. $D_T$ depends on the current flow f. Minimising $D_T$ minimises the expected delay per message, as the message arrival rate is independent of the routing algorithm.

SQP methods iteratively update the vector $\phi_i$ with a 'search direction' $\Delta\phi_i^k$ that solves the subproblem: $\text{minimise}_{\Delta\phi \in \mathbf{R}^n}\ g_k^T \Delta\phi + \frac{1}{2}\Delta\phi^T H_k \Delta\phi$. This search direction is subject to some restrictions. In the formula defining the subproblem, the matrix $H_k$ is an approximation of the Hessian of the Lagrangian function. The Lagrangian function is a well-known function, used in many optimisation methods. The Hessian matrix for a certain function is the matrix containing the second order partial derivatives of this function. As these derivatives are often hard to compute, an approximation of the Hessian is used. The approximation is dependent of the algorithm under consideration. In the formula given before, which describes the class of algorithms, $H_k$ is replaced by a matrix M multiplied by a factor $t_i/\alpha$. Further, g denotes the marginal link delay: the increase in $D_T$ that results from an increase in the incoming traffic. The restrictions made are very straightforward:

the amount of flow over any link must never become negative and the total amount of outgoing flow of each node should remain constant. The restriction that $\Delta\phi_{il}(t) = 0$ for $l \in B_i^k(t)$ is included to prevent loops and will be discussed later.

The scaling factor $\alpha$ is necessary to prevent divergence when the starting value is far from optimal. Its value is constant during the execution of the algorithm, but may differ for each algorithm. Therefore the determination of its value must be considered. This is not an easy task: a scaling factor that is too small will result in slow convergence, but when a scaling factor is too large, the algorithm may not converge at all. For a moderately large scaling factor convergence can be proven, but large initial oscillations in the mean delay may occur. Instead of a 'general' scaling factor $\alpha$, it may be possible to use a different scaling factor $\alpha_{ik}$ for each link (i,k). A detailed analysis of the effect of the choice of $\alpha$ on the objective function can be found in [CAT90].

More mathematical background on SQP-methods in general can be found in [GMSW88] and more specifically on minimum delay routing methods in [BG87], [Sik86], [BGG84], [Gal77] and [Bert79].

The following list gives the variables used in the description of the class considered in this report. All variables are defined with respect to the routing vector $\phi$. The definitions equal the ones used in [BGG84] and [Gal77], but note that different definitions may occur in the literature on this subject as well.

- $L$ = the set of links in the network.
- $t_i(t)$ = the expected traffic in bits/sec at node i with destination t.
- $t_i$ = the expected traffic in bits/sec at node i.
- $\phi_{ik}(t)$ = the fraction of the flow $t_i(t)$ that is routed over link (i,k). This fraction is 0 if link (i,k) does not exist.
- $\phi_i(t)$ = the vector $[\ \phi_{i1}(t), ..., \phi_{iN-1}(t)]$.
- $\phi(t)$ = the vector $[\ \phi_1(t), ..., \phi_{N-1}(t)]$.
- $\Delta\phi_{ik}(t)$ = the update of $\phi_{ik}(t)$.
- $r_i(t)$ = the expected traffic in bits/sec entering the network at node i with destination t.
- $f_{ik}$ = the expected traffic ("flow") in bits/sec on link (i,k).
- $D_{ik}(f_{ik})$ = the expected number of messages/sec under transmission in link (i,k) times the expected delay per message.
- $D'_{ik}(f_{ik}) = \frac{dD_{ik}(f_{ik})}{df_{ik}}$
- $D_T = \sum_{(i,k)} D_{ik}(f_{ik})$, the total delay minimised by this algorithm.
- $\frac{\partial D_T}{\partial r_i(t)}$ = the marginal delay of messages from node i with destination t. This is also called the incremental delay, as it denotes the cost in terms of the increase in $D_T$ as a result of an increase in the incoming traffic.
- $\delta_{ik}(t) = D'_{ik} + \frac{\partial D_T}{\partial r_k(t)}$, the marginal link delay in link (i,k) with respect to destination t.
- $\delta_i(t)$ = the vector $[\delta_{i1}(t), ..., \delta_{iN-1}(t)]$.
- $B_i(t)$ = the set of nodes k that are blocked at i for destination t. The notion of blocked nodes will be explained later.
- $\alpha$ = a scaling parameter for the algorithm under consideration.

The value of the computed flow variable $\phi$ is defined to be *optimal* when it lies within acceptable bounds from the value that minimises the total delay $D_T$ in the network. The computation of $\phi_i^{k+1}$ as described above is iterated for $k \rightarrow \infty$ until the value of $\phi$ is

optimal. In every iteration a new value for the routing variable $\phi$ is computed.

In the distributed computation of the routing vectors, processor i computes $\phi_i$. For this computation i needs data from other processors. How the communication between the processors and the computation of the variables take place is described below.

The value of $\frac{\partial D_T}{\partial r_i(t)}$ is needed for two purposes in the general protocol scheme. $\frac{\partial D_T}{\partial r_i(t)}$ occurs in $M_i$ and so its value is needed in the computation of $\Delta\phi_i^k$. Its value is also needed in the computation of $B_i(t)$. In the general protocol scheme $\frac{\partial D_T}{\partial r_i(j)}$ is denoted by MD(i,j) and is computed according to the following scheme from [Gal77]:

(1) wait until MD(k,j) is received from each downstream neighbour $k \neq j$,
(2) compute $MD(i,j) = \sum_k \phi_{ik}(j)[D'_{ik}(f_{ik}) + MD(k,j)]$,
(3) broadcast MD(i,j).

$D'_{ik}(f_{ik})$ and $D''_{ik}(f_{ik})$ can be computed from $D_{ik}(f_{ik})$ but, because it is difficult to compute $D_{ik}(f_{ik})$, it is preferable to estimate its derivatives directly. This estimation can be made on-line, for example by using the perturbation analysis method described in [CAT90].

The routing vectors that result from the algorithm should be updated in such a way that they remain loop free. The ordering imposed on the nodes by the marginal delays MD should be consistent with the downstream partial ordering: a downstream node must have a smaller marginal delay. Routing variable $\phi_{lm}(t)$ is *improper* if $\phi_{lm}(t) > 0$ and $MD(l,t) \leq MD(m,t)$. To prevent the occurrence of improper variables the notion of *blocked nodes* is introduced.

**Definition 1.1** A node i is defined to be *blocked* relative to destination t if i has a path to t containing some link (l,m) for which $\phi_{lm}(t)$ is *improper*.

This definition originates from [Gal77]. In Gallager's algorithm the set B(i,t) is used, which is defined as the set of nodes k that are blocked relative to destination t and have $\phi_{ik}(t) = 0$. The computation of this set, presented in [Gal77], is as follows:

Each node l determines for all neighbours m that are downstream for t whether $\phi_{lm}(t)$ is improper. If any of these neighbours satisfies this condition, l adds a tag to its broadcast message containing MD(l,t), that is sent in step 3 of the scheme for the computation of MD given above. Node l also tags its messages sent in step 3 if it receives a tagged message containing MD(m,t) from a downstream neighbour m. Now B(i,t) is the set of nodes l from which node i received a tagged message for destination t.

In Section 2 a general scheme is given for protocols which provide distributed implementations of all algorithms that belong to the considered class. The relation between this scheme and the class of algorithms is described in Section 2.2. In the protocol scheme a notation is used that differs from the notation used in this introduction. This notation is introduced to distinguish the variables used in the protocol scheme from the variables of the algorithms. A list of the variables used in the general protocol scheme is given in Section 2.1. A protocol for a particular algorithm can be obtained by refining the general protocol scheme. This refinement is described in Section 2.2. A proof scheme for proving the correctness of such refinements is given in Section 3.

In Sections 4 and 5 example algorithms are given. For each algorithm a protocol that implements the algorithm is shown and correctness proofs for these protocols are given. The relation between the algorithms and the protocols, and especially between the variables used in the algorithms and those used in the protocols, is made explicit

here. Section 4 describes a protocol for Gallager's minimum delay routing algorithm from [Gal77]; Section 5 describes a second derivative algorithm originating from [BGG84].

# 2 The General Protocol Scheme

In Section 2.2 a general protocol scheme is given from which a protocol skeleton can be derived for each algorithm in the class described in the introduction. For clarity an overview of the variables used in this scheme is given in Section 2.1.

## 2.1 A List of Variables

Below the meaning of every variable used in the protocol scheme of Section 2.2 is given.

| | |
|---|---|
| R(i) | = a bit that denotes whether node i is involved in a round of the computation. |
| traf(i,t) | = the amount of traffic in node i with destination t. |
| traf(i) | = the amount of traffic in node i. |
| $\phi$(i,b,t) | = a routing variable in node i, which denotes the fraction of traf(i,t) that must be sent over link (i,b). |
| F(i,b,t) | = the flow with destination t over link (i,b) in bits/sec. |
| F(i,b) | = the flow over link (i,b) in bits/sec. |
| input(i,t) | = the amount of input at node i with destination t. |
| input(i) | = the amount of input at node i. |
| D(i,b) | = the total delay of messages on (i,b) as a function of the flow. The flow parameter F(i,b) is left out in the parameter list. |
| $D_T$ | = the total delay of all messages in the network, $\sum_{(i,b)} D(i,b)$. |
| MD(i,t) | = $\frac{\partial D_T}{\partial input_i(t)}$, the marginal delay of messages from i with destination t. |
| B(i,t) | = a bit that denotes whether node i is blocked for destination t. |
| MDTAB(i,b,t) | = a table in node i containing values [MD(b,t),B(b,t)] for each neighbour b. |
| MDQUE | = a queue used to store messages sent by a node executing round k that node i receives while it is executing some round k' < k. The queued messages are handled when i executes round k himself. |
| DATA(i,t) | = the data necessary for the computation of $\Delta\phi$(i,t). |
| DD(i,t) | = the data necessary for the computation of DATA(i,t). |
| DF(DATA) | = a function used to compute the update $\Delta\phi$. DATA, DD, and DF are dependent on the algorithm under consideration. |
| $\delta$(i,b,t) | = $D'_{ib}$ + MD(b,t). |
| $\alpha$ | = the scaling factor for the iteration. |

Note the difference with the variables in the algorithm where, in particular, B(i,t) is a set. This set corresponds to the set of B(i,t)'s of the protocol with value 1. The bit notation is chosen for efficiency, as sending a bit is preferred to sending a set.

## 2.2 The Protocol Skeleton

The general protocol scheme is described by a protocol skeleton for a node i and a destination t. When the word *downstream* is used in the protocol skeleton, 'downstream

for destination t' is meant. Each node i uses the same protocol skeleton. The protocol skeleton consists of actions and operations. An *action* can be executed when its guard evaluates to true. A node can execute at most one action at a time. An *operation* is a computation of a value and is executed within an action. The use of operations makes the skeleton easier to understand and makes it easier to reason about the computations.

The general scheme was modeled after a protocol for Gallager's minimum delay routing algorithm (which can be found in [Gal77]), as presented by Siksma in [Sik86].

The protocol scheme works in rounds. Every time a round is finished and the value of the routing variable $\phi$ is not optimal, a new round is started. A round can be started by any node that is not involved in a round and that detects that the value of $\phi$ is not optimal. Rounds are initiated in Action Init. As an effect of Action Init, all nodes are activated and each node executes Action RecMD for each neighbour-destination pair (b,t). In each round one iteration of the computation of $\phi$ is executed and all nodes are involved in this computation. It will be proven in Section 3 that a node can be no more than 1 round ahead of any other node. So a node can receive messages from the current and from the next round. The messages originating from the next round are stored in a queue named MDQUE. Before a node starts a new round, it handles all messages stored in MDQUE without any interruption.

$\phi$ is computed in Operation CompFlow, which corresponds to the original algorithm. The subroutines necessary for the computation of the values of the used variables are Operations CompDD, CompData, CompMD, and CompB. In Operation CompData the value of DATA(i,t) is computed. This value is necessary in the computation of the new value for $\phi$. For the computation of DATA(i,t) DD(i,t) is needed. DD(i,t) can be computed from data received from downstream neighbours; this computation is executed in Operation CompDD. In Operation CompMD, MD(i,t) is computed according to the scheme given in the introduction. The set of blocked nodes is computed in Operation CompB. The protocol scheme is shown below and continued on the next two pages.

From the general protocol scheme, protocols that implement particular algorithms can be derived. Any such protocol is a refinement of the general protocol scheme in which Operations CompDD and CompData and function DF(DATA) are specified. DF(DATA) describes the iteration used to compute $\phi$ in a particular algorithm implemented by a refined protocol. DATA(i,t) is the set of data necessary in the iteration.

## General Protocol Scheme

%First the operations are specified, then the actions.%

**Operation CompDD**

%node i computes DD(i,t)%

(1) compute DD(i,t) from the DD(b,t) sets received from the downstream neighbours b of i

**Operation CompData**

%node i computes DATA(i,t)%

(1) compute DATA(i,t) with data received from the downstream neighbours of i

# General Protocol Scheme (Continued)

**Operation CompMD**
%node i computes MD(i,t)%
(1) MD(i,t) := 0;
(2) FOR all neighbours b of t with $\phi$(i,b,t) > 0
DO MD(i,t) := MD(i,t) + $\phi$(i,b,t)·[D'(i,b) + MD(b,t)]

**Operation CompB**
%node i computes B(i,t)%
(1) FOR all neighbours b of i
DO IF $\phi$(i,b,t) > 0 AND MD(i,t) $\leq$ MD(b,t)
THEN B(i,t) := 1;
(2) FOR all neighbours b with $\phi$(i,b,t) > 0
DO B(i,t) := B(i,t) OR B(b,t)

**Operation CompFlow**
%node i recomputes $\phi$(i,b,t) for all neighbours b of i and all destinations t%
(1) FOR all destinations t
DO (a) FOR all neighbours b of i with B(b,t) = 1
DO IF $\phi$(i,b,t) = 0
THEN $\Delta\phi$(i,b,t) := 0
ELSE $\Delta\phi$(i,b,t) := $-\phi$(i,b,t);
(b) FOR all neighbours b with B(b,t) $\neq$ 1
DO $\Delta\phi$(i,b,t) := traf(i,t)·DF(DATA)/$\alpha$;
(c) FOR all neighbours b of i
DO $\phi$(i,b,t) := $\phi$(i,b,t) + $\Delta\phi$(i,b,t)

**Action Init**
**Guard:** R(i)=0, $\phi$ is not optimal;
(1) R(i) := 1;
(2) send {DD(i,i), MD(i,i), B(i,i)} to all neighbours

## General Protocol Scheme (Continued)

**Action RecMD**
**Guard:** {DD(b,t), MD(b,t), B(b,t)} can be received ;
(1) IF R(i) = 0
THEN do Action Init;
(2) IF MDTAB(i,b,t) $\neq$ NIL
THEN MDQUE(i,b,t) := {DD(b,t), MD(b,t), B(b,t)}
ELSE (a) MDTAB(i,b,t) := {DD(b,t), MD(b,t), B(b,t)}
(b) IF b is downstream of i for t
AND messages {DD($b_j$,t), MD($b_j$,t), B($b_j$,t)} are received from all downstream neighbours $b_j$ of i
THEN do Operation CompDD;
do Operation CompData;
do Operation CompMD;
do Operation CompB;
send {DD(i,t), MD(i,t), B(i,t)} to all neighbours that are not downstream of i
(c) IF messages {DD($b_j$,t), MD($b_j$,t),B($b_j$,t)} are received from all neighbours $b_j$ of i
THEN send {DD(i,t), MD(i,t), B(i,t) = 0} to all downstream neighbours
(3) IF messages {DD($b_j$, $t_k$), MD($b_j$, $t_k$), B($b_j$, $t_k$)} are received for all $t_k \neq$ i from all neighbours $b_j$ of i, and t $\neq$ i
THEN (a) do Operation CompFlow;
(b) MDTAB(i,$b_j$, $t_k$) := NIL for all $b_j$ and $t_k$;
(c) R(i) := 0;
(d) IF MDQUE $\neq$ NIL
THEN handle each value of MDQUE in Action RecMD as if it was a message that arrives at this moment and set MDQUE to NIL (Step d is an atomic action.)

# 3 Structure of a Correctness Proof for the Protocol Scheme

In this section a proof scheme is given for the general protocol scheme presented in the previous section. From the proof scheme proofs can be derived for protocols which are refinements of the general protocol scheme. All parts of the proof scheme that can be proven for the general protocol scheme are given explicitly, parts dependent of the refinement are given without proof and are indicated by an asterisk. The proof scheme is given for an arbitrary destination t. Many theorems are proven for an arbitrary node i. As the protocol is completely symmetric, this is sufficient to prove correctness for each node and destination. Before a proof can be given, a definition of optimality is necessary.

**Definition 3.1** A *delay constraint* is a constraint on the allowed deviation of the minimum delay caused by $\phi$.

The following example of a delay constraint for node i is given in [Sik86]:

> There is no neighbour b of i with $\phi$(i,b,t) = 0 and D'(i,b) + MD(b,t) < (D'(i,b') + MD(b',t)), for any neighbour b' of i with $\phi$(i,b',t) > 0 and there are no neighbours b and b' of i with D'(i,b) + MD(b,t) − D'(i,b') + MD(b',t) ≤ B, where B is some previously determined boundary-value.

A delay constraint can impose requirements on the delay to several destinations.

**Definition 3.2** A flow $\phi$ is *optimal* when $\phi$ satisfies the chosen delay constraint for each node i.

The start criterion of the protocol is that the delay constraint evaluates to false for some node i. (Think of the guard '$\phi$ is not optimal' for Action Init, as 'the start criterion evaluates to false'.)

## 3.1 Assumptions

The first step of the correctness proof states the general assumptions, which are given below.

**Assumption 3.1** Low-level functions work correctly.

Assumption 3.1 implies that no messages are lost.

**Assumption 3.2** Channels are FIFO.

**Assumption 3.3** The network is connected.

**Assumption 3.4** No changes occur during the execution of the protocol.

Assumption 3.4 is realistic, because the protocols are meant to be used for quasi-static routing. The assumption will be used implicitly in the proof. This assumption is important, because messages generated in one run of a protocol, should not be received in another run of the protocol.

## 3.2 Correctness within One Round

In the second step of the correctness proof we concentrate on the events in one iterative computation of $\phi$. In every round a new value for $\phi$ is computed once. For now we assume that the data that are used in the round under consideration are the same data that ought to be used in this round and are not used before. This fact is represented in the proof by the phrase 'if the data that node i knows are recent'.

In the proof scheme properties, lemmas, theorems and corollaries are given. Properties are facts that follow immediately from the program text. Lemmas are used to prove theorems. In a theorem the correct computation of a variable is claimed. Corollaries follow directly from a lemma or theorem.

For each variable of importance it is proven that its value is computed only when all the necessary data are available, that the computation corresponds to the original algorithm and that this computation is executed correctly. The first variable for which these facts are proven is MD(i,t).

**Property 3.1** A message containing MD(i,t) is sent by node i to nodes that are not downstream of i only when a message containing MD(b,t) is received from each neighbour b of i which is downstream for t.

**Property 3.2** A message containing MD(i,i) is sent to each neighbour b of i once every round.

**Property 3.3** A message containing MD(i,t) is sent downstream by node i only when a message containing MD(b,t) is received from each neighbour b of i.

These properties follow directly from the text of Action RecMD of the protocol.

**Lemma 3.1** Node i sends {DD(i,t), MD(i,t), B(i,t)} to each neighbour b once every round.

**Proof:** {DD(i,t), MD(i,t), B(i,t)} is sent by i in Action RecMD. Node i sends this message to his upstream neighbours when it has received messages {DD(b,t), MD(b,t), B(b,t)} from all its downstream neighbours b (Property 3.1) and he sends it to its downstream neighbours when messages {DD(b,t), MD(b,t), B(b,t)} are received from all its neighbours b (Property 3.2). Suppose node i has sent messages {DD(i,t), MD(i,t), B(i,t)} to its upstream neighbours, but not to its downstream neighbours and a message {DD(b,t), MD(b,t), B(b,t)} arrives. If this message was sent by a downstream neighbour b, MDTAB(i,b,t) $\neq$ NIL and no messages {DD(i,t), MD(i,t), B(i,t)} are sent by node i to its upstream neighbours. Suppose node i has sent messages {DD(i,t), MD(i,t), B(i,t)} to all its neighbours and a message {DD(b,t), MD(b,t), B(b,t)} arrives. As messages are received from all neighbours already, the MDTAB-entry is filled and no messages are sent. When a new round is started, MDTAB is emptied. □

**Corollary 3.1** In each round of the protocol Action RecMD is executed exactly once for each neighbour-destination pair (b,t).

**Property 3.4** MD(i,t) is computed by node i if and only if i has received the value of MD(b,t) from all neighbours b which are downstream for t.

This property follows directly from the program text of Action RecMD.

**Lemma 3.2** When node i starts computing MD(i,t), i has knowledge of all the necessary data.

**Proof:** For the computation of MD(i,t) node i needs to know the values of D'(i,b) and $\phi$(i,b,t) for each neighbour b and for each neighbour b with $\phi$(i,b,t) > 0 it needs MD(b,t). Node i knows the current value of $\phi$(i,b,t) and can compute D'(i,b) for each neighbour b. When i starts computing MD(i,t), it has received MD(b,t) from each downstream neighbour b, according to Property 3.4. So i has knowledge of all the data it needs. □

**Theorem 3.1** If the data that node i knows are recent, then the computation of MD(i,t) in Operation CompMD is done in a way that corresponds to the computation scheme for $\frac{\partial D_T}{\partial r_i(t)}$ presented in [Gal77], and this computation is executed correctly.

**Proof:** For initiator t MD(t,t) = $\frac{\partial D_T}{\partial r_i(t)} = 0$. Now it suffices to prove that MD(i,t) is set to $\frac{\partial D_T}{\partial r_i(t)}$ in Operation CompMD in Action RecMD when MD(b,t) = $\frac{\partial D_T}{\partial r_b(t)}$ for each b downstream of i for t. $\frac{\partial D_T}{\partial r_i(t)}$ satisfies the following relation:

$$\frac{\partial D_T}{\partial r_i(t)} = \sum_{\text{neighbours b of i}} \phi(i,b,t) \cdot [D'_{ib} + \frac{\partial D_T}{\partial r_b(t)}].$$

At the end of Operation CompMD

$$\begin{aligned} MD(i,t) &= \sum_{\text{neighbours b with } \phi(i,b,t)>0} \phi(i,b,t) \cdot [D'(i,b) + MD(b,t)] \\ &= \sum_{\text{neighbours b of i}} \phi(i,b,t) \cdot [D'(i,b) + MD(b,t)] \\ &= \sum_{\text{neighbours b of i}} \phi(i,b,t) \cdot [D'_{ib} + \frac{\partial D_T}{\partial r_b(t)}] \\ &\quad \text{if the value of MD(b, t) is correct for each downstream} \\ &\quad \text{neighbour b of i for t} \\ &= \frac{\partial D_T}{\partial r_i(t)}. \end{aligned}$$

Because of this result and Lemma 3.2, MD(i,t) is computed correctly and corresponds to $\frac{\partial D_T}{\partial r_i(t)}$. □

The second variable of importance is B(i,t). Facts about the computation of B(i,t) are given below.

**Property 3.5** Each message containing MD(i,t) also contains B(i,t) and DD(i,t), and vice versa.

**Property 3.6** B(i,t) is computed by node i if and only if messages containing B(b,t) are received by i from all neighbours b of i which are downstream for t.

This property follows directly from the text of Action RecMD.

**Lemma 3.3** When node i starts computing B(i,t), i has knowledge of all the data it needs for the computation of B(i,t).

**Proof:** To compute B(i,t) i needs the following data: MD(i,t), $\phi$(i,b,t) for each neighbour b of i and D'(i,b), B(b,t), and MD(b,t) for each neighbour b of i with $\phi$(i,b,t) > 0. It follows from the text of Action RecMD that i starts computing B(i,t) directly after computing MD(i,t). D'(i,b) can be computed by i. The current value of $\phi$(i,b,t) is known to i. Property 3.6 states that i received B(b,t) from each downstream neighbour b before it started the computation. This implies that i also received MD(b,t) from each downstream b, because of Property 3.5. So i has knowledge of all the data it needs when it starts computing B(i,t). □

The next fact to prove is that the B(i,t)'s are computed correctly. In the following lemma $B_{prot}$(i,t) is the bit B(i,t) as used in the general protocol scheme and $B_{alg}$(i,t) is the set B(i,t) as used in the algorithm from [Gal77], which was described in the introduction.

**Lemma 3.4** { b | $B_{prot}(b,t) = 1$ and b is a neighbour of i} = { b | $b \in B_{alg}(i,t)$ and b is a neighbour of i}

**Proof:** In Operation CompB B(i,t) is set to 1 if there exists a neighbour b of i with $\phi(i,b,t) > 0$, and $MD(i,t) \leq MD(b,t)$ or $B(b,t) = 1$.

In the scheme to determine the $B_i(t)$'s described in [Gal77] the following happens: when node i is blocked for t, i tags the messages containing MD(i,t). Node i decides to tag its messages if for some downstream neighbour m $\phi(i,m,t)$ is improper and $\phi(i,m,t) \geq \alpha \cdot [D'(i,m)(F(i,m)) + MD(m,t) - MD(i,t)\ ] / \ traf(i,t)$ or if i received a tagged message containing MD(m,t) from some downstream neighbour m. A variable $\phi(i,k,t)$ is improper if $\phi(i,k,t) > 0$ and $MD(i,t) \leq MD(k,t)$.

'$B(i,t) = 1$' is equivalent to '$(B_i(t) - \{k \mid$ there is no link $(i,k)\ \}) \neq \emptyset$'. '$\phi(i,b,t) > 0$' is equivalent to 'b is downstream from i with respect to t'. $B(b,t) = 1$ in Operation CompB means that b is blocked for t. In this case b will tag its messages when b executes the scheme described in [Gal77] and i will receive a tagged message from one of its downstream neighbours. So Operation CompB is equivalent to the scheme for the determination of the $B_i(t)$'s, described in [Gal77]. □

**Theorem 3.2** If the data that node i knows are recent, then the set of b's with $B_{prot}(b,t) = 1$ is computed in Operation CompFlow corresponding to the scheme from [Gal77] for the computation of $B_{alg}(i,t)$ and this computation is executed correctly.

**Proof:** This follows from Lemmas 3.3 and 3.4. □

Now DD(i,t) is considered.

**Lemma 3.5** * When node i starts computing the new value of DD(i,t) in Operation CompDD, i has knowledge of all the necessary data.

**Theorem 3.3** * If the data that i knows are recent, then DD is computed as in the computation scheme for the variables corresponding to DD in the original algorithm.

The next variable to be considered is DATA(i,t).

**Lemma 3.6** * When node i starts computing the new value of DATA(i,t) in Operation CompData, i has knowledge of all the necessary data.

**Theorem 3.4** * If the data that i knows are recent, then DATA(i,t) is computed as in the computation scheme for the variables corresponding to DATA(i,t) in the original algorithm.

Now proofs can be given for the availability of all the necessary data for the computation of $\phi$, the fact that the computation of $\phi$ in the protocol corresponds to the computation of $\phi$ in the original algorithm and the fact that this computation is performed correctly.

**Property 3.7** If and only if node i has received $\{DD(b,t_k), MD(b,t_k), B(b,t_k)\}$ for all neighbours b and for all destinations $t_k$, node i computes new values for $\phi(i,b,t)$ for all neighbours b of i and for all destinations t.

This property follows from Action RecMD of the protocol.

**Lemma 3.7** * When node i starts computing the new value of $\phi$ in Operation CompFlow, i has knowledge of all the necessary data.

Use Property 3.7 to prove this lemma.

**Corollary 3.2** In each round of the protocol a new value for $\phi$ is computed exactly once.

**Theorem 3.5** * If the data that each node knows are recent, then $\phi$ is computed in Operation CompFlow corresponding to the original algorithm and this computation is executed correctly.

Lemma 3.7 shows that each node has knowledge of all the necessary data at the moment it starts computing $\phi$. These data are computed correctly (by Theorems 3.1, 3.2, and 3.4). $\phi$ is computed once every round (Corollary 3.2). Now it has to be shown that the updating of $\phi$ in Operation CompFlow is equivalent to the updating of $\phi$ in the original algorithm. This part of the proof is dependent of the computation scheme for $\phi$ used in the original algorithm.

## 3.3 Multiple Rounds

Many rounds of the protocol may be necessary to compute an optimal value for $\phi$. Therefore in the next step of the correctness proof the succession of the rounds is considered. It is proven that the data used in each round indeed are the data that belong to that round.

**Lemma 3.8** If node i initiates round k, then all nodes will participate in round k.

**Proof:** If node i initiates round k, it sends a message containing MD(i,i) to all its neighbours. On receipt of this message each neighbour that was not participating in this round yet, starts participating in round k and sends a message to all its neighbours. As the network is connected, each node will receive at least one message. In this way each node starts participating in round k. □

**Lemma 3.9** If node i finishes a computation of $\phi$, i disposes of the stored values of MD(b,t), B(b,t), and DD(b,t) for all neighbours b.

**Proof:** Node i stores the values of MD(b,t), B(b,t), and DD(b,t) in MDTAB(i,b,t). After i finishes the computation of $\phi$ in step 3a of Action RecMD, i sets MDTAB(i,b,t) to NIL in step 3b of Action RecMD for each neighbour b of i. □

**Lemma 3.10** Node i never uses values of MD(b,t), B(b,t), and DD(b,t) for a neighbour b, which it stored in MDTAB during round k', in its computation of $\phi$ in any round k with $k > k'$.

**Proof:** Node i disposes of the stored values used in a computation of $\phi$ as soon as the computation is finished (Lemma 3.9), so i can not use these values in a new computation of $\phi$. □

**Lemma 3.11** A message containing MD(b,t), sent to node i in round k, is received by i before a message containing MD(b,t), sent to i in round k+1.

**Proof:** Node b starts participating in round k+1, only after it finished round k. So all messages that b sent in round k were sent before b sends any message in round k+1. As channels are FIFO (Assumption 3.2), a message containing MD(b,t) sent in round k is received before a message containing MD(b,t) sent in round k+1. □

**Lemma 3.12** If there are nodes in the network which are involved in different rounds, then these nodes are involved in round k' or in round k'+1, for some number k'.

**Proof:** A node i can only start a round k+1 if it has finished round k. It can only finish round k, if all nodes have participated in round k. Suppose node i is in round k+1. There are two possible cases: (a) all nodes, other than i, are still participating in round k or are participating in round k+1 or (b) all nodes have finished round k and are participating in round k+1 or k+2. For suppose some node l is still participating in round k and some node m is participating in round k+2. Node m can only execute round k+2 when m finished round k+1, because no node participates in two rounds at the same time. This implies that all nodes participated in round k+1. So node l can not be in round k, because it must have participated in round k+1. This is a contradiction. □

**Lemma 3.13** If a message containing MD(b,t) is stored in MDQUE during round k, then this message originates from round k+1.

**Proof:** Suppose node i is in round k. This implies that every node in the network is in round k or k−1, or every node is in round k or k+1.(Lemma 3.12). Suppose every node is in round k or k−1. A node can not start round k before he has finished round k−1. Round k−1 is finished only when messages containing MD(b,t) from this round are received from all neighbours b. Each node b sends a message containing MD(b,t) once every round (Lemma 3.1). So when node i is in round k, he does not receive any messages from round k−1; it can only receive messages originating from round k or k+1. If i receives a message containing MD(b,t) sent in round k, MDTAB(i,b,t) is empty(Lemmas 3.1 and 3.10) and DD(b,t), MD(b,t), and B(b,t) are stored in MDTAB. If i receives a message containing MD(b,t), originating from round k+1, i already received a message containing MD(b,t) originating from round k (Lemmas 3.1 and 3.11). MDTAB is not emptied before Action RecMD is entirely executed, so MDTAB $\neq$ NIL. Now the message is stored in MDQUE. □

**Lemma 3.14** All messages stored in MDTAB during round k, were sent during round k.

**Proof:** A message containing MD(b,t) is stored in MDTAB during round k if this message was received during round k and originated from round k (this can be seen in the proof of Lemma 3.13) or when this message was the first message containing MD(b,t) in MDQUE at the start of round k. Because each round a message containing MD(b,t) is sent, this message was sent when i was in round k−1. Lemma 3.13 implies that this message originates from round k. □

**Lemma 3.15** In round k a message from MDQUE is stored in MDTAB if and only if this message originates from round k.

**Proof:** The following assertion is proven by induction: If a message in MDQUE originates from round k, then this message is stored in MDTAB during round k.

Base.
It is easily seen that a message in MDQUE, originating from round 1, is stored in MDTAB during round 1: the proof as given for the induction step can be given for k=1 without use of the induction hypothesis, because no messages are stored in MDQUE during round 0.

Induction step.
Assume that the assertion holds for all rounds j with $j < k$. Consider the assertion for round k. When node i starts participating in round k, a message containing MD(b,t) and originating from round k is the first message containing MD(b,t) in MDQUE. (During execution of round k−1 by i, only messages from round k were stored in MDQUE (Lemma 3.13). All messages originating from round $j < k$ were removed from MDQUE during this round (induction hypothesis). MDQUE is a FIFO-queue, channels are FIFO (Assumption 3.2), and each message sent in round k is received before any message sent in round k+1 (Lemma 3.11).) At the end of round k−1 MDTAB is emptied and no messages are received between this step and the handling of the messages in MDQUE, so the first message containing MD(b,t) in MDQUE is stored in MDTAB during round k. This is the message originating from round k, as proven above. This proves the assertion for round k.

The only-if part of the lemma follows from Lemma 3.14. □

**Lemma 3.16** All messages originating from round k are used for the computation of $\phi$ and stored in MDTAB in round k.

**Proof:** No messages can be sent by a node in round k if some node is still in round $k' < k-1$. If a message originating from round k is received by a node i involved in round k, it is stored immediately in MDTAB as was proven before. If such a message is received by a node i involved in round k−1, i stores it in MDQUE. This message is stored in MDTAB when i executes round k, as proven in Lemma 3.15. The values stored in MDTAB are used for the computation of $\phi$. □

**Theorem 3.6** In each round of the protocol the new value of $\phi$ is computed as in the original algorithm and this computation is executed correctly.

**Proof:** This follows from Theorem 3.5 and Lemma 3.16. □

**Theorem 3.7** When $\phi$ is computed according to the original algorithm, the value of $\phi$ converges to the value that causes the minimum total delay in the network.

Note that this theorem concerns the original algorithm, not the protocol. The convergence of the algorithm will usually be proven before a protocol for the algorithm is considered.

**Corollary 3.3** Iteration of Operation CompFlow results in convergence of $\phi$ to the value that minimises the total delay in the network.

In practice, Operation CompFlow is not iterated infinitely many times, but until a value is reached that implies an acceptable deviation of the minimum delay, i.e., a value that satisfies the delay constraint for all i.

**Theorem 3.8** Actions of the protocol are executed until $\phi$ is optimal.

**Proof:** Whenever a node reaches the end of a round with a value for $\phi$ which does not satisfy the delay constraint, this node starts a new round of the protocol. The value of $\phi$ converges to the value that minimises the delay (Corollary 3.3), hence an optimal value for $\phi$ is reached. □

**Lemma 3.17** If no actions are executed by node i after round k, then no node will execute an action after round k.

**Proof:** Suppose node i does not execute any action after round k and suppose there is a node j which does execute an action after round k. Thus node j participates or has participated in round k+1. There must be a node that initiated round k+1. But if a node initiates round k+1, then all nodes will participate in round k+1 (Lemma 3.8). This contradicts the assumption that node i does not execute any action after round k. □

**Theorem 3.9** If a flow $\phi$ is optimal at the end of a round k and no further changes occur, then $\phi$ remains optimal and no more actions will be executed.

**Proof:** Suppose at moment $\tau$ $\phi$ is optimal and no round is executed. Evaluation of the start criterion produces the value false at each node. So no node will start a new round and therefore $\phi$ will not be updated after $\tau$ and remains optimal.

Suppose at moment $\tau$ $\phi$ is optimal and round k is being executed and suppose $\phi$ became optimal at moment $\tau' < \tau$, during which node i updated $\phi$ in Operation CompFlow. Suppose all nodes, other than i, updated $\phi$ while executing Operation CompFlow in this round before $\tau'$. If MDQUE is empty, i will not be involved in round k+1 started by another node. No node will start a new round after $\tau$, because $\phi$ is optimal. If MDQUE is not empty, then there is a node $l \neq i$ that started round k+1. A node starts a new round only when its start criterion evaluates to true and round k+1 was started before $\tau'$, which implies that $\phi$ was not optimal at $\tau'$. This is in contradiction with the fact that $\phi$ is optimal at $\tau'$. So MDQUE is empty, no new round will start after $\tau'$ and $\phi$ will remain optimal.

Suppose at $\tau'$ there are some nodes, which did not update $\phi$ yet in round k and l is the last node that updates $\phi$ in round k . l's update can have two effects on the optimality of $\phi$. $\phi$ can be updated to a value that causes a delay smaller than the delay caused by the previous value or to a value that causes a greater delay. In the first case $\phi$ remains optimal, in the second case $\phi$ is not optimal at the end of the round and a new round is started. □

The correctness of the protocol is expressed by the following theorem:

**Theorem 3.10** Continuous execution of the protocol yields an optimal value for $\phi$.

**Proof:** This follows from Theorems 3.6, 3.7, 3.8 and 3.9. □

The general proof structure is a tool for proving the correctness of a protocol for any algorithm in the class described before. As each protocol, implementing an algorithm from the considered class, is a refinement of the general protocol scheme, all proofs given in this section are valid for such a protocol. Only the proofs of Lemmas 3.5, 3.6, and 3.7 and of Theorems 3.3, 3.4, 3.5, and 3.7 must be given for a particular refinement of the general protocol scheme. These proofs are sufficient to prove correctness of the protocol. In the following sections some important algorithms belonging to the class described in the introduction are given, together with protocols that implement these algorithms and with correctness proofs for the protocols.

# 4 Gallager's Algorithm

In this section Gallager's minimum delay routing algorithm is considered. In Section 4.1 the original algorithm is given, as published in [Gal77]. A protocol for this algorithm is given in Section 4.2. The correctness of this protocol is proven in Section 4.3.

## 4.1 The Algorithm

Gallager's minimum delay routing algorithm ([Gal77]) is given below.

**Algorithm A**
FOR $k \in B_i(j)$
DO $\phi_{ik}(j) := 0$ and $\Delta_{ik}(j) := 0$;
FOR $k \notin B_i(j)$
DO IF $k \neq k_{min}(i,j)$
THEN $\phi_{ik}(j) := \phi_{ik}(j) - \Delta_{ik}(j)$
ELSE $\phi_{ik}(j) := \phi_{ik}(j) + \sum_{k \neq k_{min}(i,j)} \Delta_{ik}(j)$

WHERE $\Delta_{ik}(j) = \min[\phi_{ik}(j), \eta a_{ik}(j)/t_i(j)]$,
$\eta$ = a scaling factor,
$a_{ik}(j) = D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} - \min_{m \notin B_i(j)}[D'_{im}(f_{im}) + \frac{\partial D_T}{\partial r_m(j)}]$,
and $k_{min}$ is the node m that achieves the minimum in the expression that defines $a_{ik}(j)$.

In this algorithm every link determines the gradient of the average delay with respect to its own traffic flow. First each node i determines the neighbour $k_{min}$ corresponding to the link $(i,k_{min})$ with the least marginal delay. Then i computes the amount of flow with destination j, $\Delta_{ik}(j)$, which is to be transferred from link (i,k) to link $(i,k_{min})$. The minimum is taken of the computed quantity and the current flow variable to prevent routing updates from giving rise to negative flows. The amount of reduction of the objective function depends on the difference between the marginal delay on links (i,k) and $(i,k_{min})$, and on the scaling factor $\eta$.

This algorithm uses a stronger definition of blocked nodes than the one used in the general protocol scheme. This definition is necessary to prove convergence of the algorithm.

**Definition 4.1** A node i is *blocked relative to j* if i has a path to j containing some link (l,m) with $\phi_{lm}(j)$ improper and the additional requirement that

$$\phi_{lm}(j) \geq \alpha[D'_{lm}(f_{lm}) + \frac{\partial D_T}{\partial r_m(j)} - \frac{\partial D_T}{\partial r_l(j)}]/t_l(j),$$

where $\alpha$ is a scaling factor equal to the $\eta$ used before.

In the algorithm the set $B_i(j)$ is used, which is defined as the set of nodes k that are blocked relative to j and have $\phi_{ik}(j) = 0$, together with the set of nodes k with $(i,k) \notin L$.

Gallager's algorithm is now specified one more time, using the variable names that were used in the general protocol scheme. Variables of the algorithm that have no corresponding variables in the protocol are left unchanged.

**Algorithm A**
FOR k with B(k,j) = 1 or $(i,k) \notin L$
DO $\phi(i,k,j) := 0$ and $\Delta\phi(i,k,j) := 0$
FOR k with B(k,j) = 0 and $(i,k) \in L$
DO IF $k \neq m$
THEN $\phi(i,k,j) := \phi(i,k,j) - \Delta\phi(i,k,j)$
ELSE $\phi(i,k,j) := \phi(i,k,j) + \sum_{k \neq m} \Delta\phi(i,k,j)$

WHERE $\Delta\phi(i,k,j) = \min[\phi(i,k,j), \alpha a_{ik}(j)/traf(i,j)]$,
$a_{ik}(j) = D'(i,k)(F(i,k)) + MD(k,j) -$
$\min_{m:B(m,j)=0 \text{ and } (i,m) \in L}[D'(i,m)(F(i,m)) + MD(m,j)]$,
and m is the node that achieves the minimum in the expression defining $a_{ik}(j)$.

## 4.2 A Protocol for Gallager's Algorithm

A protocol for this algorithm was first presented by Cees Siksma in [Sik86]. It is presented here with some minor changes. Most of these changes are denotational. As mentioned before, the definition of B(i,j) is different from the definition used in the general protocol scheme and instead B(i,j) is computed according to the following scheme: A node l determines for all downstream neighbours m whether $\phi_{lm}(j)$ is improper and whether $\phi_{lm}(j) \geq \alpha[D'(l,m)(F(l,m)) + \frac{\partial D_T}{\partial r_m(j)} - \frac{\partial D_T}{\partial r_l(j)}]/t_l(j)$. If any of these neighbours satisfies both conditions, l adds a tag to its broadcast message containing $\frac{\partial D_T}{\partial r_l(j)}$. If l receives a tagged message containing $\frac{\partial D_T}{\partial r_m(j)}$ of a downstream neighbour m, l also tags its own messages. B(i,j) now represents the set of nodes k for which $(i,k) \notin L$ or from which a tagged message containing MD(k,j) was received by i. The protocol skeleton for Gallager's algorithm for node i is shown below.

## Protocol G

**Operation CompMD**
%node i computes MD(i,t)%
(1) MD(i,t) := 0;
(2) FOR all neighbours b of t with $\phi$(i,b,t) > 0
DO MD(i,t) := MD(i,t) + $\phi$(i,b,t)·[D'(i,b) + MD(b,t)]

**Operation CompB**
%node i computes B(i,t)%
(1) FOR all neighbours b of i
DO IF $\phi$(i,b,t) > 0 AND MD(i,t) ≤ MD(b,t)
AND $\phi$(i,b,t) ≥ $\alpha$·[D'(i,b) + MD(b,t) − MD(i,t)]/traf(i,t)
THEN B(i,t) := 1;
(2) FOR all neighbours b with $\phi$(i,b,t) > 0
DO B(i,t) := B(i,t) OR B(b,t)

**Operation CompFlow**
%node i recomputes $\phi$(i,b,t) for all neighbours b of i%
(1) m := the neighbour b of i with B(b,t) = 0 which minimises D'(i,b) + MD(i,b,t);
(2) FOR all neighbours b with B(b,t) = 1
DO IF $\phi$(i,b,t) = 0
THEN $\Delta\phi$(i,b,t) := 0
ELSE $\Delta\phi$(i,b,t) := $-\phi$(i,b,t);
(3) FOR all neighbours b ≠ m with B(b,t) ≠ 1
DO (a) $\delta\phi$(i,b,t) := [D'(i,b) + MD(b,t) − D'(i,m) − MD(m,t)]/traf(i,t);
(b) $\Delta\phi$(i,b,t) := − min[$\phi$(i,b,t), $\alpha \cdot \delta\phi$(i,b,t)];
(4) $\Delta\phi$(i,m,t) := $-\sum_{b \neq m} \Delta\phi$(i,b,t);
(5) FOR all neighbours b of i
DO $\phi$(i,b,t) := $\phi$(i,b,t) + $\Delta\phi$(i,b,t)

**Action Init**
**Guard:** R(i)=0, $\phi$ is not optimal;
(1) R(i) := 1;
(2) send {MD(i,i), B(i,i)} to all neighbours

**Action RecMD**
**Guard:** a message {MD(b,t), B(b,t)} can be received ;
(1) IF R(i) = 0
THEN do Action Init;
(2) IF MDTAB(i,b,t) $\neq$ NIL
THEN MDQUE(i,b,t) := {MD(b,t), B(b,t)}
ELSE (a) MDTAB(i,b,t) := MD(b,t);
(b) IF b is downstream of i for t
AND messages {MD($b_j$,t), B($b_j$,t)} are received from all
downstream neighbours $b_j$ of i
THEN do Operation CompMD;
do Operation CompB;
send {MD(i,t), B(i,t)} to all neighbours that are not
downstream of i
(c) IF messages {MD($b_j$,t), B($b_j$,t)} are received from all
neighbours $b_j$ of i
THEN send {MD(i,t), B(i,t) = 0} to all downstream neighbours
(3) IF messages {MD($b_j$, $t_k$), B($b_j$, $t_k$)} are received from all neighbours $b_j$
of i, for all $t_k \neq i$ and $t \neq i$
THEN (a) do Operation CompFlow;
(b) MDTAB(i,$b_j$, $t_k$) := NIL for all $b_j$ and $t_k$;
(c) R(i) := 0;
(d) IF MDQUE $\neq$ NIL
THEN handle each value of MDQUE in Action RecMD as if it was
a message that arrives at this moment and set MDQUE to NIL
(Step d is an atomic action.)

If B(b,t) is set to 1 in Action RecMD, $\phi$(i,b,t) can still have a value > 0. This value must be set to 0 in Operation CompFlow. This is done in step 5. The routing variables for blocked nodes with value 0 do not need to be changed. The corresponding change variable is set to 0 in step 2. The presentation in [Sik86] contained some minor flaws, that were corrected in the given scheme. These flaws are discussed below. The remainder of this section can be skipped by readers unfamiliar with [Sik86].

In [Sik86] Siksma states that the following equation is valid:

$$\frac{\partial D_T}{\partial r_i(t)} = \Sigma\phi(i,b,t)\cdot[MD(i,b)(F(i,b)) + \frac{\partial D_T}{\partial r_b(t)}]$$

As MD(i,b) = $\frac{\partial D_T}{\partial r_i(b)}$, D'(i,b) = $\frac{dD_{ib}(f_{ib})}{df_{ib}}$, and $\frac{\partial D_T}{\partial r_i(b)} \neq \frac{dD_{ib}(f_{ib})}{df_{ib}}$ (node i does not necessarily route its messages with destination b over link (i,b)), this equation is incorrect. Siksma's computation of MD(i,t) is based on this equation; therefore this computation must be corrected. Operation CompMD was presented as follows:

**Operation CompMD**
(1) MD(i,t) := 0;
(2) FOR all neighbours b with $\phi$(i,b,t) > 0
DO MD(i,t) := MD(i,t) + $\phi$(i,b,t)·MD(b,t)

Operation CompMD (2) should be changed into:

(2) FOR all neighbours b with $\phi$(i,b,t) > 0
DO MD(i,t) := MD(i,t) + $\phi$(i,b,t)·[D'(i,b) + MD(b,t)]

The first two steps of Operation CompFlow were

(1) m := the neighbour b of i which minimises D'(i,b) + MD(i,b,t);
(2) FOR all neighbours b $\neq$ m with B(b,t) = 1 AND $\phi$(i,b,t) = 0
DO $\Delta\phi$(i,b,t) := 0;

According to these steps a node k could be chosen for m with B(k,t) = 1. This is in contradiction with the definition of m given by Gallager in [Gal77]. This flaw is corrected in the following version of Operation CompFlow.

**Operation CompFlow**
(1) m := the neighbour b of i with B(b,t) = 0 which minimises D'(i,b) + MD(i,b,t);
(2) FOR all neighbours b with B(b,t) = 1 AND $\phi$(i,b,t) = 0
DO $\Delta\phi$(i,b,t) := 0;
(3) FOR all neighbours b $\neq$ m with B(b,t) $\neq$ 1 AND $\phi$(i,b,t) $\neq$ 0
DO (a) $\delta\phi$(i,b,t) := [D'(i,b) + MD(b,t) − D'(i,m) − MD(m,t)]/traf(i,t);
(b) $\Delta\phi$(i,b,t) := − min[$\phi$(i,b,t), $\alpha \cdot \delta\phi$(i,b,t)];
(4) $\Delta\phi$(i,m,t) := − $\sum_{b \neq m} \Delta\phi$(i,b,t);
(5) FOR all neighbours b of i
DO $\phi$(i,b,t) := $\phi$(i,b,t) + $\Delta\phi$(i,b,t)

This operation still contains a flaw. If it is detected in the current run of the protocol that node b is blocked relative to t, then a variable $\phi$(i,b,t) > 0 should be set to 0. This is supposed to happen in step 5 of Operation CompFlow. Therefore $\Delta\phi$(i,b,t) should be equal to $-\phi$(i,b,t) for these i, b and t. This is the case when $\phi$(i,b,t) $\leq \alpha \cdot \phi$(i,b,t). Because of the definition of a blocked node $\phi$(i,b,t) $\geq \alpha \cdot \phi$(i,b,t). So when $\phi$(i,b,t) > $\alpha \cdot \phi$(i,b,t), $\phi$(i,b,t) is not set to 0. This flaw can be easily corrected, as is shown in the version of Operation CompFlow, included in Protocol G.

## 4.3 A Correctness Proof for Protocol G

In this section it is proven that Protocol G yields an optimal value for the routing variable $\phi$. To prove correctness, only Lemma 3.7 and Theorems 3.5 and 3.7 need to be proven correct. The correctness of the protocol then follows from the proof skeleton described in Section 3. As mentioned before, in this algorithm all nodes k without a link (i,k) are added to $B_i(t)$. We do not consider these nodes here; they are only added to simplify some tests in Algorithm A. Lemma 3.4 remains valid.

**Lemma 3.7** When node i starts computing the new value of $\phi$ in Operation CompFlow, i has knowledge of all the necessary data.

**Proof:** Node i needs the following variables to compute $\phi$: D'(i,b), MD(b,t), B(b,t), and $\phi$(i,b,t) for all neighbours b of i and $\alpha$. $\alpha$ is a known value. D'(i,b) can be computed by i. The old value of $\phi$(i,b,t) is known to i. Node i does not start computing a new value of $\phi$ before it has received MD(b,t) and B(b,t) from all neighbours b (Property 3.7). □

**Theorem 3.5** If the data that each node knows are recent, then $\phi$ is computed in Operation CompFlow corresponding to Algorithm A presented in [Gal77] and this computation is executed correctly.

**Proof:** Lemma 3.7 shows that each node has knowledge of all the necessary data at any moment it starts computing. The following shows that the update of $\phi$ in Operation CompFlow is equivalent to the update of $\phi$ by the rewritten Algorithm A as described in Section 4.1.
In Operation CompFlow the following updates of $\phi$ are executed:

$$\begin{aligned}
\phi(i,m,t) &:= \phi(i,m,t) + \Delta\phi(i,m,t) \\
&= \phi(i,m,t) - \sum_{\text{neighbours } b\neq m} \Delta\phi(i,b,t) \\
&= \phi(i,m,t) - \sum_{\text{neighbours } b\neq m} (-\min[\phi(i,b,t), \alpha\delta\phi(i,b,t)]) \\
&= \phi(i,m,t) + \\
&\quad \sum_{\text{neighbours } b\neq m} \min[\phi(i,b,t), \frac{\alpha[D'(i,b) + MD(b,t) - D'(i,m) - MD(m,t)]}{traf(i,t)}]
\end{aligned}$$

For $b \neq m$, $b \notin B(i,t)$:

$$\begin{aligned}
\phi(i,b,t) &:= \phi(i,b,t) + \Delta\phi(i,b,t) \\
&= \phi(i,b,t) - \min[\phi(i,b,t), \alpha\delta\phi(i,b,t)]
\end{aligned}$$

For $b \neq m$, $b \in B(i,t)$:

$$\begin{aligned}
\phi(i,b,t) &:= \phi(i,b,t) + \Delta\phi(i,b,t) \\
&= \phi(i,b,t) - \phi(i,b,t) \\
&= 0
\end{aligned}$$

This is exactly the result of the assignments of Algorithm A written out in full. □

In [Gal77] Gallager proves the convergence of Algorithm A to the value of $\phi$ that implies the minimum delay in the following theorem:

**Theorem 3.7** Assume that for all links (i,k) $D_{ik}(f_{ik})$ has a positive first derivative and nonnegative second derivative for $0 \leq f_{ik} < C_{ik}$, where $C_{ik}$ is the capacity of link (i,k), and that

$$\lim_{f_{ik}\to C_{ik}} D_{ik}(f_{ik}) = \infty.$$

For every positive number $D_0$ there exists a scaling factor $\alpha$ for algorithm A such that if $\phi^0$ satisfies $D_T(\phi^0) \leq D_0$, then

$$\lim_{m\to\infty} D_T(\phi^m) = \min_{\phi} D_T(\phi),$$

where $\phi^m = A(\phi^{m-1})$ for all $m \geq 1$.

This theorem completes the correctness proof of the protocol for Gallager's algorithm.

# 5 A Second Derivative Algorithm

In this section an algorithm based on second derivatives of the delay function is given, together with a protocol that implements this algorithm. The algorithm was presented in [BGG84] and is given in Section 5.1. A protocol for the algorithm is presented in Section 5.2 and a correctness proof for this protocol is given in Section 5.3.

## 5.1 The Algorithm

The Second Derivative Algorithm consists of a particular computation of a new value for $\phi_{il}$ from the old value of $\phi_{il}$ and the computation of the values of all the data needed in the computation of $\phi$. A new value for $\phi$ is computed from the formula

$$\phi_{il}^{k+1} = \max[0, \phi_{il}^{k} - \frac{\alpha(\delta_{il} - \mu_i)}{traf(i,t)(D''_{il} + \overline{R}_l)}],$$

where $\alpha$ is a scaling factor, $\mu_i$ is a variable originating from optimisation techniques, and $\overline{R}_l = \sum_m \phi_{lm}^2 D''_{lm} + (\sum_m \phi_{lm}\sqrt{\overline{R}_m})^2$. The derivation of this algorithm from mathematical theories can be found in [BGG84].

## 5.2 A Protocol for the Second Derivative Algorithm

A protocol for the algorithm described above can be derived by substituting $\overline{R}$ for DD, $\overline{\Phi}$ and $\delta$ for DATA, and $-\frac{\alpha^2(\delta(i,b,t)-\mu_i)}{\overline{\Phi}(i,b,t)}$ for DF(DATA) in the general protocol scheme. $\overline{\Phi}$ is a variable introduced to separate the computation and use of DATA. The parts of the protocol skeleton that are specific for the algorithm are given below. Operations CompMD and CompB remain as in the general protocol scheme and are not mentioned below.

# Protocol SD

**Operation CompDD**
%node i computes $\overline{R}$(i,t)%
(1) $\overline{R}(i,t) := 0$;
(2) FOR all neighbours b of i with $\phi(i,b,t) > 0$
DO $\overline{R}(i,t) := \overline{R}(i,t) + \phi(i,b,t)^2 D''(i,b)$
(3) $\overline{R}(i,t) := \overline{R}(i,t) + (\sum_{b \text{ with} \phi(i,b,t)>0} \phi(i,b,t)\sqrt{\overline{R}(b,t)})^2$.

**Operation CompData**
%node i computes $\overline{\Phi}(i,b,t)$ and $\delta(i,b,t)$ for all downstream neighbours b of i %
(1) FOR all downstream neighbours b of i
DO (a) $\overline{\Phi}(i,b,t) := traf(i,t)^2(D''(i,b) + \overline{R}(b,t))$;
(b) $\delta(i,b,t) := D'(i,b) + MD(b,t)$

**Operation CompFlow**

%node i recomputes $\phi$(i,b,t) for all neighbours b of i %

(1) FOR all neighbours b of i with B(b,t) = 1
DO IF $\phi$(i,b,t) = 0
THEN $\Delta\phi$(i,b,t) := 0
ELSE $\Delta\phi$(i,b,t) := $-\phi$(i,b,t);

(2) (a) compute $\mu_i$;
(b) FOR all neighbours b with B(b,t) $\neq$ 1
DO $\Delta\phi(i,b,t) := -\frac{\alpha traf(i,t)(\delta(i,b,t)-\mu_i)}{\Phi(i,b,t)}$;

(3) FOR all neighbours b of i
DO $\phi$(i,b,t) := $\phi$(i,b,t) + $\Delta\phi$(i,b,t)

**Action Init**

**Guard:** R(i)=0, $\phi$ is not optimal;

(1) R(i) := 1;

(2) send {$\overline{R}$(i,i), MD(i,i), B(i,i)} to all neighbours

**Action RecMD**

**Guard:** {$\overline{R}$(b,t), MD(b,t), B(b,t)} can be received;

(1) IF R(i) = 0
THEN do Action Init;

(2) IF MDTAB(i,b,t) $\neq$ NIL
THEN MDQUE(i,b,t) := {$\overline{R}$(b,t), MD(b,t), B(b,t)}
ELSE (a) MDTAB(i,b,t) := {$\overline{R}$(b,t), MD(b,t), B(b,t)};
(b) IF b is downstream of i for t
AND messages {$\overline{R}(b_j,t)$, MD($b_j$,t), B($b_j$,t)} are received from all downstream neighbours $b_j$ of i
THEN do operation CompDD;
do operation CompMD;
do operation CompB;
send {$\overline{R}$(i,t), MD(i,t), B(i,t)} to all neighbours that are not downstream of i;
(c) IF messages {$\overline{R}(b_j,t)$, MD($b_j$,t),B($b_j$,t)} are received from all neighbours $b_j$ of i
THEN send {$\overline{R}$(i,t), MD(i,t), B(i,t) = 0} to all downstream neighbours;

(3) IF messages {$\overline{R}(b_j,t_k)$, MD($b_j$,$t_k$), B($b_j$,$t_k$)} are received from all neighbours $b_j$ of i, for all $t_k \neq i$ and $t \neq i$
THEN (a) do Operation CompFlow;
(b) MDTAB(i,$b_j$,$t_k$) := NIL for all $b_j$ and $t_k$;
(c) R(i) := 0;
(d) IF MDQUE $\neq$ NIL
THEN handle each value of MDQUE in Action RecMD as if it was a message that arrives at this moment and set MDQUE to NIL (Step d is an atomic action.)

## 5.3 Correctness Proof for the Protocol

To prove the correctness of the protocol, it is sufficient to prove the correctness of Lemmas 3.5, 3.6, and 3.7 and Theorems 3.3, 3.4, 3.5, and 3.7. Then the correctness of the protocol follows from the general proof scheme.

**Lemma 3.5** When node i starts computing the new value of $\overline{R}$(i,t) in Operation CompDD, i has knowledge of all the necessary data.

**Proof:** For the computation of $\overline{R}$(i,t) i needs to know $\phi$(i,b,t) for each neighbour b and needs D''(i,b) and $\overline{R}$(b,t) for each downstream neighbour b. Node i can compute D''(i,b) and has received $\overline{R}$(b,t) from all downstream neighbours before he starts CompDD. □

**Theorem 3.3** If the data that i knows are recent, then $\overline{R}$(i,t) is computed as in the formula given in the previous section.

**Proof:** It is easily seen from the program text of Operation CompDD, that $\overline{R}$(i,t) is equal to $\sum_{b \text{ with} \phi(i,b,t)>0} \phi(i,b,t)^2 D''(i,b) + (\sum_b \phi(i,b,t)\sqrt{\overline{R}(b,t)})^2$ after Operation CompDD. $\sum_{b \text{ with} \phi(i,b,t)>0} \phi(i,b,t)^2 D''(i,b) = \sum_b \phi(i,b,t)^2 D''(i,b)$. From the correspondence between $\overline{R}_i(t)$ and $\overline{R}$(i,t), $\phi_{ib}(t)$ and $\phi$(i,b,t), and between $D''_{ib}$ and D''(i,b), it now follows that $\overline{R}$(i,t) is computed in correspondence to the formula shown in the previous section. □

**Lemma 3.6** When node i starts computing the new values of $\overline{\Phi}$(i,b,t) and $\delta$(i,b,t) in Operation CompData, i has knowledge of all the necessary data.

**Proof:** For the computation of $\overline{\Phi}$(i,b,t) i needs the following data: traf(i,t), and D''(i,b), and $\overline{R}$(b,t) for each neighbour b. For the computation of $\delta$(i,b,t) i needs to know D'(i,b) and MD(b,t). So for each downstream neighbour b, i needs to know D''(i,b), $\overline{R}$(b,t), D'(i,b), and MD(b,t) and i needs to know traf(i,t). traf(i,t), D''(i,b), and D'(i,b) can be computed by i. It follows directly from the program text that node i starts executing Operation CompData only after it received messages containing $\overline{R}$(b,t) and MD(b,t) from all downstream neighbours b. So i has knowledge of all the necessary data. □

**Theorem 3.4** If the data that i knows are recent, then $\overline{\Phi}$(i,b,t) and $\delta$(i,b,t) are computed as in the computation scheme for $\overline{\Phi}_{ib}(t)$ and $\delta_{ib}(t)$ presented in [BGG84].

**Proof:** $\overline{\Phi}_{ib}$ is defined in [BGG84] to be $t_i^2(D''_{ib} + \overline{R}_b)$. In Operation CompData, $\overline{\Phi}$(i,b,t) is set to traf(i,t)$^2$(D''(i,b) + $\overline{R}$(b,t)). As traf(i) corresponds to $t_i$, D''(i,b) to $D''_{ib}$ and $\overline{R}$(b,t) to $\overline{R}_b$, it follows that the computation of $\overline{\Phi}$(i,b,t) in Operation CompData is equivalent to the computation of $\overline{\Phi}_{ib}$ as presented in [BGG84].

$\delta_{ib}$ is defined in [BGG84] to be $D'_{ib} + \frac{\partial D_T}{\partial r_b}$. In Operation CompData $\delta$(i,b,t) is set to D'(i,b) + MD(b,t). D'(i,b) corresponds to $D'_{ib}$ and MD(b,t) is defined as $\frac{\partial D_T}{\partial r_b(t)}$. So the computation of $\delta$(i,b,t) in Operation CompData is equivalent to the computation of $\delta_{ib}$ presented in [BGG84]. □

**Lemma 3.7** When node i starts computing the new value of $\phi$ in Operation CompFlow, i has knowledge of all the necessary data.

**Proof:** Node i needs to know $\alpha$, it needs to know B(b,t) and $\phi$(i,b,t) for all neighbours b and $\delta$(i,b,t) and $\overline{\Phi}$(i,b,t) for all downstream neighbours b. $\alpha$ is a known value. The old value of $\phi$(i,b,t) is known to i. Node i does not start computing a new value of $\phi$ before it has received B(b,t) from all neighbours b (Property 3.7). $\overline{\Phi}$(i,b,t) and $\delta$(i,b,t) are computed for all downstream neighbours b in Operation CompData. It follows from the program text that Operation CompData is executed before Operation CompFlow, so i has knowledge of all the necessary data. □

**Theorem 3.5** If the data that each node knows are recent, then $\phi$ is computed in Operation CompFlow corresponding to the Second Derivative Algorithm and this computation is executed correctly.

**Proof:** Lemma 3.7 shows that each node has knowledge of all the necessary data at the moment it starts computing. In the following it is shown that the updating of $\phi$ in Operation CompFlow is equivalent to the updating of $\phi$ as described in the Second Derivative Algorithm. In Operation CompFlow the following updates of $\phi$ are executed. For all neighbours b of i with B(b,t) = 1 and $\phi$(i,b,t) = 0:

$$\begin{aligned}\phi(i,b,t) &:= \phi(i,b,t) + \Delta\phi(i,b,t)\\ &= \phi(i,b,t)\\ &= 0\end{aligned}$$

For all neighbours b of i with B(b,t) = 1 and $\phi$(i,b,t) > 0:

$$\begin{aligned}\phi(i,b,t) &:= \phi(i,b,t) + \Delta\phi(i,b,t)\\ &= \phi(i,b,t) - \phi(i,b,t)\\ &= 0\end{aligned}$$

For all neighbours b of i with B(b,t) = 0:

$$\begin{aligned}\phi(i,b,t) &:= \phi(i,b,t) + \Delta\phi(i,b,t)\\ &= \phi(i,b,t) - \frac{\alpha\,traf(i,t)(\delta(i,b,t) - \mu_i)}{\overline{\Phi}(i,b,t)}\\ &= \phi(i,b,t) - \frac{\alpha\,traf(i,t)(\delta(i,b,t) - \mu_i)}{traf(i,t)^2(D''(i,b) + \overline{R}(b,t))}\\ &= \phi(i,b,t) - \frac{\alpha(\delta(i,b,t) - \mu_i)}{traf(i,t)(D''(i,b) + \overline{R}(b,t))}\end{aligned}$$

This is exactly the result in the assignment of the Second Derivative Algorithm. □

**Theorem 3.7** When $\phi$ is computed according to the Second Derivative Algorithm, the value of $\phi$ converges to the value that causes the minimum delay.

**Proof:** In [BGG84] the following theorem is stated. Its proof can be found in [Gaf79].

> **Theorem C** Let the initial routing $\phi^0$ be loop free and satisfy $D_T(\phi^0, r) \leq D_0$, where $D_0$ is some scalar. Assume also that there exist two positive scalars $\lambda$ and $\Lambda$ such that the sequences of matrices $\{M_i^k\}$ satisfy the following two

conditions:
a) The absolute value of each element of $M_i^k$ is bounded above by $\Lambda$.
b) There holds

$$\lambda |\, v_i \,|^2 \leq v_i^T M_i^k v_i \text{ for all } v_i \text{ in the subspace } \{v_i \mid \sum_{l \notin B(i,\phi^k)} v_{il} = 0\}$$

Then there exists a positive scalar $\overline{\alpha}$ (depending on $D_0$, $\lambda$ and $\Lambda$), such that for all $\alpha \in (0, \overline{\alpha}]$ and k = 0,1, ... the sequence $\{\phi^k\}$ generated by the general algorithm satisfies

$$D_T(\phi^{k+1}, r) \leq D_T(\phi^k, r) \text{ and}$$
$$\lim_{k \to \infty} D_T(\phi^k, r) = \min_{\phi \in \Phi} D_T(\phi, r)$$

Furthermore, every limit point of $\{\phi^k\}$ is an optimal solution of the minimisation problem:

$$\text{minimise } D_T(\phi, r) = \sum_{(i,l) \in L} D_{il}[f_{il}(\phi, r)]$$

subject to $\phi \in \Phi$.

To prove Theorem 3.7 it is sufficient to prove the assumptions made in Theorem C.
$M_i$ is a matrix with $\overline{\Phi}_{il}/\text{traf}(i,t)^2$ at the diagonal and 0 at every other position. $\overline{\Phi}_{il}/\text{traf}^2(i,t)$ $= D''_{il} + \overline{R}_l$. $0 < D''_{il}$ by assumption. $D(\phi^0, r) \leq D_0$ implies that the second derivatives $D''_{il}(\phi^0, r)$ are bounded. In [Gaf79] it is proven that $D(\phi, r)$ is a nonincreasing function, so $D(\phi^k, r) \leq D(\phi^0, r) \leq D_0$ for all k. It follows that the second derivatives $D''_{il}(\phi^k, r)$ are bounded from above for all k. Name the upper bound $D_{max}$. Now $0 < D''_{il}(\phi^k, r) < D_{max}$.

$0 < \overline{R}_l$ by definition. $\overline{R}_l \leq \max_l \overline{R}_l < \infty$. The finiteness of $\overline{R}_l$ follows from the definition of $\overline{R}_l$ and from the boundedness of the second derivatives $D''_{il}$. Choose $\Lambda = D_{max} + \max_l \overline{R}_l$ and choose $\lambda$ to be some scalar less than or equal to $D_{max} + \max_l \overline{R}_l$. The absolute value of each element of $M_i^k$ is bounded from above by $\Lambda$ and this bound is strictly positive. In the case of this algorithm $\lambda |\, v_i \,|^2 \leq v_i^T M_i^k v_i$ is equivalent to

$$\begin{aligned}
\lambda \sum_{(i,j) \in L} \Delta\phi_{ij}^k &\leq \sum_{(i,j) \in L} \Delta\phi_{ij}^k \overline{\Phi}_{ij}/\text{traf}^2(i,t) \\
&= \sum_{(i,j) \in L} \Delta\phi_{ij}^k \text{traf}^2(i,t)(D_{ij}^{k\prime\prime} + \overline{R}_j)/\text{traf}^2(i,t) \\
&= \sum_{(i,j) \in L} \Delta\phi_{ij}^k \cdot (D_{ij}^{k\prime\prime} + \overline{R}_j) \\
&\leq (D_{max} + \max_l \overline{R}_l) \cdot \sum_{(i,j) \in L} \Delta\phi_{ij}^k
\end{aligned}$$

This equation is equivalent to $\lambda \leq D_{max} + \max_l \overline{R}_l$.
As $D_{max} + \max_l \overline{R}_l > 0$, there exists a positive scalar $\lambda \leq D_{max} + \max_l \overline{R}_l$, and this $\lambda$ will be such that

$$\lambda |\, v_i \,|^2 \leq v_i^T M_i^k v_i$$

Now Theorem C can be applied and convergence of $\phi$ follows. □

# 6 Conclusion

This report presented a general protocol scheme for a class of minimum delay routing algorithms. From this scheme protocols can be derived which provide a distributed implementation of the algorithms. This derivation is simple and quick. The protocols can be proven correct by use of a general proof scheme which was also presented in this report. Using the scheme, proving the correctness of a protocol becomes very easy and takes little effort. For two example algorithms original protocols and correctness proofs were derived.

The general protocol scheme can be modified to fit a class of flow control algorithms. This modification will be discussed in another report that will appear soon.


# Acknowledgement

I thank Jan van Leeuwen for his helpful comments and suggestions. I also thank Prof.D.P. Bertsekas for explaining some details in [BGG84].


# References


[Bert78] Bertsekas, D.P. *Algorithms for Optimal Routing of Flow in Networks.* Coordinated Science Laboratory Working Paper, University of Illinois at Champaign-Urbana, June 1978.

[Bert79] Bertsekas, D.P. *Algorithms for Nonlinear Multicommodity Network Flow Problems.* in Bensoussan, A. and Lions, J.L. (Eds.), Proc. Int. Symp. Syst. Optimization and Analysis, Springer-Verlag, pp. 210-22, 1979.

[BG87] Bertsekas, D.P., Gallager, R.G. *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

[BGG82] Bertsekas, D.P., Gafni, E.M., Gallager, R.G. *Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks.* Lab. Inform. and Decision Syst., MIT, Cambridge, Rep. LIDS-P-1082-A, August 1982.

[BGG84] Bertsekas, D.P., Gafni, E.M., Gallager, R.G. *Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks.* IEEE Transactions on Communications, vol. COM-32, no. 8, August 1984, pp. 911-919.

[CAT90] Cassandras, C.G., Abidi, M.V., Towsley, D. *Distributed Routing with On-Line Marginal Delay Estimation.* IEEE Transactions on Communications, vol. COM-38, no. 3, March 1990, pp. 348-359.

[Gaf79] Gafni, E.M. *Convergence of a Routing Algorithm.* Lab. Inform. and Decision Syst., Mass. Inst. Technol., Cambridge, Rep. LIDS-TH-907, May 1979.

[Gal77] Gallager, R.G. *A Minimum Delay Routing Algorithm Using Distributed Computing.* IEEE Transactions on Communications, vol. COM-25, no. 1, January 1977, pp. 73-85.

[GMSW88] Gill, P.E., Murray, W., Saunders,M.A., Wright, M.H. *Recent developments in constrained optimization.* Journal of Computational and Applied Mathematics 22, 1988, pp. 257-269.

[Sch91] Schoone, A.A. *Assertional Verification in Distributed Computing.* Ph.D.Thesis, Department of Computer Science, Utrecht University, May 1991.

[Sik86] Siksma, C. *Routering in Computernetwerken.* M.Sc.Thesis (in Dutch), Department of Computer Science, Utrecht University, INF/SCR-86-03, 1986.