Paper

Distributed asynchronous algorithms in the Internet – new routing and traffic control methods

Andrzej Karbowski

Abstract— The paper presents several new algorithms concerning the third (network) and the fourth (transport) layer of ISO/OSI network model. For the third layer two classes of the shortest paths algorithms – label correcting and auction algorithms – are proposed. For the fourth layer an application of price decomposition to network optimization and Internet congestion control is suggested.

Keywords— computer networks, optimization, shortest path, traffic control, decomposition, distributed computations, asynchronous algorithms.

1. Introduction

In the last decade two new approaches were intensively studied to solve shortest path/routing problems in parallel and distributed environment, namely, label correcting [4, 8], and auction [18] algorithms.

The label correcting algorithms may be treated as a generalization of the Bellman-Ford algorithm. The name "label" refers to the distance from the origin node "1" to a node $i \neq 1$, the correction is its change in subsequent iterations of the algorithm leading towards the optimum (these algorithms may be similarly applied to problems with multiple origins and a single destination, by reversing the roles of origins and destinations and the direction of each arc). A notion of the candidate list of nodes is introduced. Different shortest path algorithms are distinguished by the method of selecting the node to exit the candidate list at each iteration. The simplest method from this family is the Bellman-Ford method – where the candidate list is simply a FIFO queue. More sophisticated label correcting methods maintain two queues and use more complex removal and insertion strategies. The objective is to reduce the number of node reentries to the candidate list. Some of these methods – namely SLF (small label first) and SLF-LLL (large label last) will be described in the paper.

An interesting alternative to these algorithms seem to be auction algorithms. In the basic version [2, 8] an auction algorithm maintains a path starting at the origin and a price for each node. The terminal node of the path "bids" for neighboring nodes basing on their prices and the lengths of the connecting arcs. At each iteration the path is either extended by adding a new node or contracted by deleting its terminal node. When the destination becomes the terminal node of the path, the algorithm terminates.

In recent years a considerable progress was also done in the area of distributed traffic control in the Internet. A special interest was paid to the asynchronous price method. In particular, the theory presented in [14] was thoroughly examined by the author of this paper. Unfortunately, an error in the proof of convergence of the asynchronous version of the algorithm was found. Later on this proof was corrected [12], so this interesting and important theory has been saved. At the same time Steven Low and his collaborates have shown how important for the Internet this theory is. In their works they used it not only as a tool to analyze the stability of different implementations of TCP congestion control protocols [16, 17], but also as a basis for the development of new, more efficient protocols [10]. At the end of the article some of these results will be presented.

2. Classical Bellman-Ford routing algorithm

We consider a directed graph consisting of n nodes (routers). Let us denote by N the set of all these nodes and by N_i the set of neighbours of the ith node (that is, the set of all nodes from the set N, to which arcs starting from i go). Let us assume, that every arc $(i, j) : j \in N_i$ is characterized by a positive scalar value a_{ij}, which we will treat as the cost of passage from i to j, that is the distance measure (metric).

Let us choose a node $m \in N$ and assume, that it may be reached from all other nodes. It can be easily proved (e.g., by the contradiction), that the paths of the minimum costs $\hat{d}_{im}$ (so-called shortest paths) can be obtained through the solution of the following set of equations:

$$\hat{d}_{im} = \min_{j \in N_i} \left(a_{ij} + \hat{d}_{jm} \right) \quad i \neq m \,. \tag{1}$$

Let us take now

$$h_{im}(d) = \begin{cases} \min_{j \in N_i} (a_{ij} + d_{jm}) & i \neq m \\ d_{ii} & i = m \end{cases} \tag{2}$$

and

$$d = [d_{11} d_{12}, \ldots, d_{1n}, \ldots, d_{n1}, d_{n2}, \ldots, d_{nn}] \,, \tag{3}$$

$$h_i(d) = [h_{i1}(d), \ldots, \ldots, h_{in}(d)] \,, \tag{4}$$

$$h = [h_1, h_2, \ldots, h_n] \,. \tag{5}$$

To find the solution we may apply the Bellman-Ford algorithm

$$d := h(d) \tag{6}$$

starting from $d_{ii} = 0$; $d_{ij} = \infty, i \neq j, \forall i, j \in N$. This algorithm is based on the order preserving (monotone) mapping h, which may be implemented in a distributed, totally asynchronous version [6].

The optimization algorithm (1)–(6) may be applied respectively – to adapt routing to the current situation in a network. In that case the cost a_{ij} should be a measure of the quality of transmission, dependent on the current flow (transmission rate) f_{ij} between nodes i and j. A very popular flow cost function $a_{ij}(.)$ is:

$$a_{ij}(f_{ij}) = \frac{f_{ij}}{(c_{ij} - f_{ij}) + \varepsilon_{ij}} + d_{ij} \cdot f_{ij}, \tag{7}$$

where c_{ij} is the transmission capacity of arc (i, j) and d_{ij} is the processing and propagation delay (of course we assume $0 \leq f_{ij} \leq c_{ij}$; $\varepsilon_{ij} > 0$ is a small constant to avoid zero in the denominator). However, in this – adaptive – case one should remember, that the cost functions (7), dependent monotonically on flow, should be augmented with constant components δ_{ij} (so-called bias factors, interpreted as link costs/lengths at zero load), because otherwise oscillations (in subsequent optimal routings) may occur [5].

The presented adaptive routing approach is used in the Internet [7] in demons *routed*, *gated* and protocols RIP and Hello.

In the first protocol so-called "hop count metrics" is used, what means, that simply all elementary arcs are counted for; in the second "network delay metrics", that is the time of transmission, is taken into account. In the active state, all messages used to the optimization of the routing tables (i.e., the tables of the shortest path neighbours for different destinations) are sent by every computer to all direct neighbours every 30 seconds.

3. Generic shortest path algorithm

The algorithm presented in the previous section may be treated as a special case from a more general class of algorithms. We will present these algorithms to solve problems as formulated in the cited works.

Let us take now that for each node $i \in N$ we want to find a path of minimum length (cost) that starts at node 1 and ends at i (these algorithms may be similarly applied to problems with multiple origins and a single destination, by reversing the roles of origins and destinations and the direction of each arc). We assume, that all arc lengths are positive and that there exists at least one path from node 1 to each other node.

A general class of algorithms to which belongs, among others, Bellman-Ford algorithm are label correcting algorithms. The name "label" refers to the distance $d_i \triangleq d_{1i}$ from the origin node "1" to a node $i \neq 1$, the correction is its change in subsequent iterations of the algorithm leading towards the optimum. A notion of the candidate list of nodes is introduced. Let us denote it by V. In addition to this, let us denote by A the set of all arcs in the directed graph (that is, the set of all links in the network). Assuming that V is nonempty, a typical iteration of a shortest path algorithm (not necessarily of label correcting type) is as follows [3]:

Initialization:

$$d_1 = 0, \; d_i = \infty \;\; \text{for} \;\; i \neq 1,$$

$$V = \{1\}.$$

Typical iteration of the generic shortest path algorithm:

Remove a node i from the candidate list V.
For each outgoing arc $(i, j) \in A$, with $j \neq 1$,
if $d_j > d_i + a_{ij}$, set:

$$d_j := d_i + a_{ij} \tag{8}$$

and add j to V if it does not already belong to V.

Different shortest path algorithms are distinguished by the method of selecting the node to exit the candidate list V at each iteration. For example in Dijkstra method the node exiting V is the node whose label is minimum over all other nodes in V. This guarantees, that every node enters and exits V exactly once and its label is not changed in later iterations. Because of that, these methods are called label setting methods. Label correcting methods avoid the overhead associated with finding the minimum label node at the expense of multiple entrances of nodes into V. In these methods a queue Q is used to maintain the candidate list V. Bellman-Ford method is the simplest method from this family. In the terms of the above generic shortest path algorithm it maintains V in a FIFO queue Q; nodes are removed from the top of the queue and are added to the bottom of Q.

4. SLF and LLL strategies

More sophisticated label correcting methods maintain V in one or in two queues (eg., in so-called threshold algorithms the candidate list is partitioned into two separate queues on the basis of some threshold parameter) and use more complex removal and insertion strategies. The objective is to reduce the number of node reentries in V. Some of these methods are significantly faster than Bellman-Ford method. The most effective proved to be SLF and SLF-LLL methods [4].

In the SLF method the candidate list V is maintained as a double ended queue Q. At each iteration, the node removed is the top node of Q. The rule for inserting new nodes is as follows:

Let i be the top node of Q, and j be a node that enters Q,

if $d_j \leq d_i$, then enter j at the top of Q;
else, enter j at the bottom of Q.

The LLL method defines a more complicated strategy of removal a node from Q, which aims to remove from Q nodes with small labels. At each iteration, when the node at the top of Q has a larger label than the average node label in Q (defined as the sum of the labels of the nodes in Q divided by the cardinality $\overline{\overline{Q}}$ of Q), this node is not removed from Q, but rather it is repositioned to the bottom of Q. It may be described as follows:

Let i be the top node of Q, and let $s = \sum_{j \in Q} d_j / \overline{\overline{Q}}$,
if $d_i > s$, then move i to the bottom of Q.
Repeat until a node i such that $d_i \leq s$ is found and is removed from Q.

It is possible to combine the SLF queue insertion and the LLL node selection strategies. The resulting method is denoted by SLF-LLL. The proof of convergence of these two algorithms closely resembles the proof of the convergence of Bellman-Ford algorithm and is based on the monotonicity property of the label modification mapping [4, 8].

In the parallel implementation, each processor removes the top node from Q (perhaps after some shifts of the queue in the case of LLL strategy), updates the labels of its adjacent nodes and adds these nodes (if necessary) into Q according to SLF insertion strategy. This means, that several nodes can be simultaneously removed from the candidate list and the labels of the adjacent nodes can be updated in parallel. In the distributed version an additional processor responsible for maintaining the candidate queue Q is useful. When the algorithm is implemented in an asynchronous version, a new node may be removed from the candidate list by some processor while other processors are still updating the labels of other nodes. Of course, only one processor at a time can modify a given label. Hence, it is very easy to implement this method on a parallel shared-memory or ccNUMA machines using locks to assure the consistency of data.

A multiple queues version of this algorithm showed also very good features [4]. In this version each processor uses a separate queue (a node can reside in at most one queue). It extracts nodes from the top of its queue, updates the labels for adjacent nodes and uses a heuristic procedure for choosing the queue to insert a node that enters V. For example, it may be the one with the minimum current value of the sum of the outgoing arcs in that list. This heuristic is easy to implement and ensures good load balancing among the processors. For checking whether a node is present in the candidate list (that is, in some queue) it is suggested [4, 8] to associate with every node a Boolean variable, which is updated each time a node enters or exits a candidate list. This algorithm is easily generalized to the problem of finding several distinct shortest paths [8] showing in many problems very high efficiency.

Other label correcting methods (however, not so efficient in tests), e.g., with threshold dividing queues, are presented in [4].

5. Auction algorithm

A more efficient alternative to the presented algorithms seem to be auction algorithms. In the basic version [2, 18] such an algorithm maintains a path starting at the origin s and a price for each node. The terminal node of the path "bids" for neighboring nodes basing on their prices and the lengths of the connecting arcs. At each iteration the path is either extended by adding a new node or contracted by deleting its terminal node. When the destination becomes the terminal node of the path, the algorithm terminates.

To present the algorithm in a formal way, let us denote by P a path starting at the origin, that is: $P = (s, i_1, i_2, \ldots, i_k)$, where $(i_m, i_{m+1}) \in A$, $m = 1, \ldots, k-1$. We assume that $i_{j_1} \neq i_{j_2}$, $j_1 \neq j_2$, that is a path does not contain any cycle. The node i_k is called the terminal node of P. The degenerate path $P = (s)$ may be also obtained in the course of the algorithm.

If i_{k+1} is a node that does not belong to a path $P = (s, i_1, i_2, \ldots, i_k)$ and (i_k, i_{k+1}) is an arc, extending P by i_{k+1} means replacing P by the path $(s, i_1, i_2, \ldots, i_k, i_{k+1})$. If P does not consist of just the origin node s, contracting P means replacing P with the path $(s, i_1, i_2, \ldots, i_{k-1})$.

In addition to the path, the algorithm maintains a price p_i for each node $i \in N$ in the network. Let us denote by p the vector of all prices p_i. We say, that a path-price pair (P, p) satisfies complementary slackness (CS) if the following relations hold:

$$p_i \leq a_{ij} + p_j, \ \forall (i,j) \in A, \tag{9}$$

$$p_i = a_{ij} + p_j \tag{10}$$

for all pairs of successive nodes i and j of P.

An important property is that if a path-price pair (P, p) satisfies CS, then portion of P between node s and any node $i \in P$ is the shortest path from s to i and $d_{si} = p_s - p_i$ is the corresponding shortest distance.

The algorithm proceeds in iterations, transforming a pair (P, p) satisfying CS into another pair satisfying CS. At each iteration the path P is either extended by a new node or contracted by deleting its terminal node. In the latter case the price of the terminal node is increased strictly. In may be described in the following way:

Typical iteration:

Let i be a terminal node of P.

- *Step 0:* (Scanning of successor nodes)

  $$\text{If } p_i < \min_{\{j | (i,j) \in A\}} \{a_{ij} + p_j\} \tag{11}$$

  go to Step 1; else go to Step 2.

- *Step 1:* (Contract path)

  $$\text{Set } p_i := \min_{\{j|(i,j)\in A\}} \{a_{ij} + p_j\} \quad (12)$$

  and if $i \neq s$ contract P.

- *Step 2:* (Extend path)

  Extend P by node j_i, where

  $$j_i = \arg \min_{\{j|(i,j)\in A\}} \{a_{ij} + p_j\}. \quad (13)$$

  If j_i is the destination d, stop; P is the desired shortest path.

The algorithm starts with the default pair:

$$P = (s), \quad p_i = 0, \forall i$$

and stops when the destination node d becomes the terminal node of the path. This iteration is called the "forward algorithm". It is possible to apply also the "reverse algorithm", where not the source s, but the destination node d is fixed (and forms the initial path) and the path extends by inserting and contracts by deleting the starting node, and to use a combined algorithm, where there are two paths: P_f – starting at the origin s and P_r – ending at the destination d. In this algorithm alternately several forward and reverse iterations are performed at least one of which leads to an increase of, respectively, origin price p_s or the destination price p_d. This two-sided algorithm terminates when the two paths have a common node. In many tests [2] this hybrid approach proved to be the most effective, much faster than the sided Dijkstra algorithm.

When there are several origin nodes, the shortest path auction algorithm may be implemented in a parallel way in a distributed environment [2]. In the basic (i.e., forward) version for each origin i there is a separate processor that executes the forward algorithm and keeps in local memory a price vector p^i (build of the snapshots of prices p_l^j sent by other nodes) and a corresponding path P^i satisfying CS together with p^i. The price vectors are communicated at various times to other processors, perhaps irregularly. A processor operating on P^i upon reception of a price vector p^j from another processor j, adopts as the price of each node l the maximum of the prices of l according to the existing and the received (i.e., the snapshots) price vectors, that is:

$$p_l^i := \max(p_l^i, p_l^j), \ \forall l \in N. \quad (14)$$

This guarantees keeping the CS property, monotonicity of the mapping:

$$p_i := \min_{\{j|(i,j)\in A\}} \{a_{ij} + p_j\} \quad (15)$$

and the asynchronous convergence of the algorithm.

The parallel, synchronous and asynchronous implementations of the two-sided different shortest path algorithms for different problems (including many origin – many destinations routing problem) are more complicated due to the possibility of losing CS and, in the consequence, the oscillation of prices. To avoid it, all nodes are equipped with counters for forward and reverse extensions without an intervening contractions. However, in most tests (on a shared-memory machine) much simpler, one sided, forward scheme showed superiority. The details are described in [18].

6. The application of the price method to network flow optimization

In this section we will return to a problem presented in [11], because since that time there were some progress both in a better justification of this approach and in the understanding of its importance for the Internet. There were also successful implementations of this method to improve Internet congestion control protocols at the TCP level.

We will consider the situation, where the capacities of links are too small to carry all traffic and it is necessary to reduce the users' transmission rates. So, we will deal with the decision variables – flexible transmission rates x_w, where $w \in W$ is the connection and W is the set of all active connections (i.e., source-destination pairs).

The transmission rates should belong to some intervals:

$$\underline{x}_w \leq x_w \leq \overline{x}_w. \quad (16)$$

Every customer, that is the user of the network, assesses the satisfaction from the use of the network through his utility function $U_w(x_w)$, defined on the interval $[\underline{x}_w, \overline{x}_w]$. In this problem, instead of minimization of the total cost of transmission, which is not so important for the operator of the network (because most links are fully used), we will strive to maximize the satisfaction of the customers, that is the sum of their utility functions.

Hence, our problem will be as follows:

$$\max_x \sum_{w\in W} U_w(x_w), \quad (17)$$

$$\underline{x}_w \leq x_w \leq \overline{x}_w, \ \ w \in W, \quad (18)$$

$$f_{ij} = \sum_{w\in W_{ij}} x_w \leq c_{ij}, \ \ \forall (i,j) \in A. \quad (19)$$

The last inequality expresses capacity constraints of the links; W_{ij} denotes the set of connections (virtual paths) traversing arc (i,j), that is:

$$W_{ij} = \{w|(i,j) \in A_w\}, \quad (20)$$

where A_w is the set of arcs (links) used by connection w.

The Lagrange function for problem (17)–(19) will be as follows:

$$\mathrm{L}(x,p) = \sum_{w\in W} U_w(x_w) - \sum_{(i,j)\in A} p_{ij}\left(\sum_{w\in W_{ij}} x_w - c_{ij}\right), \quad (21)$$

where

$$x = \left[x_w, w \in W\right]. \quad (22)$$

Let us notice now that:

$$\sum_{(i,j)\in A} p_{ij}\left(\sum_{w\in W_{ij}} x_w - c_{ij}\right)$$
$$= \sum_{(i,j)\in A} p_{ij} \sum_{w\in W_{ij}} x_w - \sum_{(i,j)\in A} p_{ij}c_{ij}$$
$$= \sum_{w\in W}\sum_{(i,j)\in A_w} p_{ij}x_w - \sum_{(i,j)\in A} p_{ij}c_{ij}$$
$$= \sum_{w\in W} x_w \sum_{(i,j)\in A_w} p_{ij} - \sum_{(i,j)\in A} p_{ij}c_{ij}$$
$$= \sum_{w\in W} x_w p_w - \sum_{(i,j)\in A} p_{ij}c_{ij}, \qquad (23)$$

where

$$p_w = \sum_{(i,j)\in A_w} p_{ij} \qquad (24)$$

is the price for the connection w along its path formed of arcs $(i,j)\in A_w$.
Applying (23) to (21) we get:

$$\mathrm{L}(x,p) = \sum_{w\in W}\left(U_w(x_w) - x_w p_w\right) + \sum_{(i,j)\in A} p_{ij}c_{ij}. \qquad (25)$$

According to the duality theory, the optimal solutions, both the optimal distribution of flows $\left(\hat{x}_w, w\in W\right)$ and the vector of optimal prices $[\hat{p}_{ij}, (i,j)\in A]$ may be obtained via the two-phase procedure:

$$\min_p\left[L_D(p) = \sum_{w\in W}\max_{\underline{x}_w\le x_w\le \overline{x}_w}\left(U_w(x_w) - x_w p_w\right) + \sum_{(i,j)\in A} p_{ij}c_{ij}\right]. \qquad (26)$$

In this way we obtained $\overline{\overline{W}}$ problems of optimization of connection transmission rates and an $\overline{\overline{A}} = n$-dimensional problem of optimization of prices of unit bandwidth ($\overline{\overline{V}}$ denotes the number of elements of the set V). It can be proved [12, 14], that they may be solved in a distributed, partially asynchronous way[1].
More precisely, the wth user solves the local optimization problem:

$$\max_{\underline{x}_w\le x_w\le \overline{x}_w}\left(U_w(x_w) - x_w\tilde{p}_w(t)\right), \qquad (27)$$

where $\tilde{p}_w(t)$ is the current estimate of the price of transmission w, that is:

$$\tilde{p}_w(t) = \sum_{(i,j)\in A_w}\sum_{\tau=t-B}^{t}\eta_{ij}^w(t,\tau)p_{ij}(\tau). \qquad (28)$$

In the last equation $\eta_{ij}^w(t,\tau)$ are (usually unknown) nonnegative coefficients such that:

$$\sum_{\tau=t-B}^{t}\eta_{ij}^w(t,\tau) = 1 \qquad (29)$$

and B is the length of the time window (i.e., the measure of asynchronizm). The optimal solution of the local problem (27) may be determined analytically [14] from the expression:

$$\hat{x}_w = \left[U_w^{'-1}(\tilde{p}_w)\right]_{\underline{x}_w}^{\overline{x}_w}, \qquad (30)$$

where $[z]_a^b = \min\{\max\{z,a\},b\}$ and $U_w^{'-1}$ is the inverse of $= U_w^{'}$.
The optimal link prices $\hat{p}_{ij}$ in problem (26) may be calculated in different ways. In the simplest case the steepest descent method is applied. According to Eq. (21) this is realized by the iteration:

$$p_{ij}(t+1) = \left[p_{ij}(t) + \gamma\left(\tilde{f}_{ij}(t) - c_{ij}\right)\right]^+, \qquad (31)$$

where

$$\tilde{f}_{ij}(t) = \sum_{w\in W_{ij}}\sum_{\tau=t-B}^{t}\eta_{ij}(t,\tau)x_w(\tau) \qquad (32)$$

is the estimate of the total flow through the link (i,j). In the last equation $\eta_{ij}(t,\tau)$ are (usually unknown) nonnegative coefficients such that:

$$\sum_{\tau=t-B}^{t}\eta_{ij}(t,\tau) = 1 \qquad (33)$$

and B is the length of the time window. Due to the theory presented in [12, 14], for sufficiently small values of the stepsize γ and bounded time intervals between consecutive updates of the optimal link prices p_{ij} and transmission rates x_w, the algorithm converges partially asynchronously. Since the algorithm was devoted to flow control in the Internet, where sending information on current internal prices of links (from the operator to users) and the calculation of current average transmission rates in all virtual paths would mean some additional equipment and the communication overheads, the following estimation mechanisms were proposed [1, 15]:

1. Instead of calculation of the aggregated flow rate $\tilde{f}_{ij}(t)$ in the link (i,j), the link operator measures the link buffer occupancy $v_{ij}(t)$. This occupancy evolves according to the equation:

$$v_{ij}(t+1) = \left[v_{ij}(t) + \sum_{w\in W_{ij}} x_w(t) - c_{ij}\right]^+, \qquad (34)$$

where $x_w(t)$ is the current flow rate of the transmission w. Then, the new link price $p_{ij}(t+1)$ is set as:

$$p_{ij}(t+1) = \gamma v_{ij}(t). \qquad (35)$$

2. Instead of passing the users directly the information on the current price of the unit of the bandwidth, the link operator applies random exponential marking (REM) algorithm. It allows for encoding this information in only one bit[2] of the stream of packets.

[1]The proof of the asynchronous convergence theorem in [14] had a serious error pointed out and corrected in [12].

[2]We mean explicit congestion notification (ECN) bit in the IP header.

Namely, it is assumed, that the link (i, j) marks each packet with a probability $m_{ij}(t)$ which is exponentially increasing in the price $p_{ij}(t)$ (or in the buffer occupancy $v_{ij}(t)$ – see (35)):

$$m_{ij}(t) = 1 - \phi^{-p_{ij}(t)}, \tag{36}$$

where $\phi > 0$ is a constant. Once a packet is marked, its mark is carried to the destination and then conveyed back to the source via acknowledgement (due to the TCP/IP protocol). The end-to-end probability that a packet of the connection w is marked after traversing the whole its way made of arcs form the set A_w is then:

$$m_w(t) = 1 - \prod_{(i,j)\in A_w} (1 - m_{ij}(t)) = 1 - \phi^{-p_w(t)}, \tag{37}$$

where $p_w(t) = \sum_{(i,j)\in A_w} p_{ij}(t)$ is the price for the transmission of the unit of bandwidth along the virtual path w. Then, the customer using this connection estimates the price of it $\tilde{p}_w(t)$ by the fraction $\tilde{m}^p(t)$ of his packets marked in some window before time t and inverting (37), that is:

$$\tilde{p}_w(t) = -\log_\phi (1 - \tilde{m}_w(t)). \tag{38}$$

Owing to this two improvements, there is no need for administrative communication between the operators of links and the end users of the network.

It should be noted, that the prices in this model are regarded rather as a control signal to guide sources' decisions, than a part of the charge a user pays [14]. In particular, these prices (i.e., Lagrange multipliers) equal zero when the traffic is below the capacity of the network. In other words: the user pays nothing for the highest desirable quality! On the contrary, he pays more and more for connections of lower quality, when there is a congestion in the network. Such pricing mechanism would be hardly acceptable for a human. So it is rather a tool for software agents.

In the latest articles Low and collaborates [16, 17] present few such agents and interpret their utility functions. It is shown, that in the Internet these agents are simply different implementations of TCP congestion control protocols. The basic idea is to regard the process of congestion control as carrying out a distributed computation by sources and links over a network in real time to solve an optimization problem. The objective is to maximize aggregate source utility subject to capacity constraints. The source rates are interpreted as primal variables, congestion measures as dual variables, and TCP/AQM (active queue management) protocols as distributed primal-dual algorithms to solve this optimization problem and its associated dual problem. Different protocols, such as Reno, Vegas, RED, and REM, all solve the same prototypical problem with different utility functions, Moreover, all these protocols generate congestion measures (Lagrange multipliers) that solve the dual problem in equilibrium. It is described in the next section.

7. TCP window flow control through the price method and the consequences

The classical congestion control method, currently used in the Internet at the TCP level, is based on Jacobson algorithm [9] called TCP-Reno. If we denote by: $z(t)$ – the length of the source window at time t, that is the maximum number of unacknowledged packets that the source can inject into the network at the time t; RTT – round trip time, that is the time between sending the packet and receiving its acknowledgement; ACK – the acknowledgement packet; TOUT – timeout for waiting for ACK, this algorithm may be described in the following way:

1. Slow-Start phase
   - z(0) = 1,
   - after every ACK received $z(t+1) = z(t) + 1$ until attaining SSTRESH (slow-start threshold).
2. Congestion avoidance phase:

$$z(t+1) = \begin{cases} z(t) + \frac{1}{z(t)} & \text{OK} \equiv \text{ACK received before TOUT} \\ \frac{1}{2} z(t) & \text{the loss of packet} \equiv \text{ACK has not arrived before TOUT or 3 previous have been received} \end{cases} \tag{39}$$

In the recent works [16, 17]:

- z are treated as primal variables (actually it is taken $x = c \cdot z$, $c = const.$),
- the link congestion measures are treated as dual variables p,
- the dynamic (state) equations describing the modification of z (or rather x) and p are treated as a distributed primal-dual algorithm.

One may transform the most important congestion avoidance phase of TCP-Reno to the problem (17)–(19) taking:

$$\dot{z}_w(t) = \kappa_w(t) \cdot \left(1 - \frac{p_w(t)}{v_w(t)}\right), \tag{40}$$

where

$$\kappa_w(t) = \frac{1}{T_w(t)};\ v_w(t) = \frac{3}{2 z_w^2(t)} \tag{41}$$

and $T_w(t)$ is an estimate of RTT at time t for the wth connection.

Let us denote the transmission rate by $x_w(t)$:

$$x_w(t) = z_w(t)/T_w(t) \tag{42}$$

and define an utility function:

$$U(x_w) = \frac{\sqrt{3/2}}{T_w} \tan^{-1}\left(\sqrt{\frac{2}{3}} x_w \cdot T_w\right). \tag{43}$$

The dual variable p_w may be interpreted as the probability of the loss of a packet for the wth connection.
The TCP-Reno has some drawbacks:

- at the beginning the window grows too slowly,
- when the packets are lost the window is shortened too abruptly,
- often oscillations,
- the big packets are privileged (small packets are punished although the congestion is almost always caused by big packets).

Because of these drawbacks, some proposals appeared how to improve the effectiveness of using the link capacities and to make the network more just. The most successful proved to be the following model:

$$\kappa_w(t) = \gamma \cdot \alpha_w;\ \nu_w(t) = \alpha_w / x_w(t), \tag{44}$$

$$U(x_w) = \alpha_w \log x_w, \tag{45}$$

where κ_w, ν_w are from the state equation (40) and $U(x_w)$ is from the source optimization problem (17)–(19). The utility function is derived from the notion of weighted proportional fairness introduced by Kelly [13].
A vector of rates $\hat{x} = (\hat{x}_w, w \in W)$ is *weighted proportionally fair* if it is feasible and if for any other feasible vector x, the aggregate of proportional changes is zero or negative:

$$\sum_{w \in W} \alpha_w \frac{x_w - \hat{x}_w}{\hat{x}_w} \leq 0. \tag{46}$$

It easy to check, that the performance index (45) as a concave function satisfies weighted proportional fairness condition (46) and, if we treat all connections as a game between users, it is a Nash-equilibrium point [19].
The dual variable p_w in this model is interpreted as a delay caused by queues in routers for the wth connection.
This very approach became a basis for the development in California Institute of Technology (Caltech) by prof. S. Low group of the new TCP control protocol called FAST (Fast Active queue management Scalable Transmission control protocol). It was designed for high speed data tranfers over large distances, e.g., tens of gigabyte files across the Atlantic [10]. At the time of writing this paper the world record of the Internet transmission (8.6 Gbps from Los Angeles to Geneva (CERN) via Chicago) belonged to this very group (as the previous one).

References

[1] S. Athuraliya and S. H. Low, "Optimization flow control". II. Implementation, 2000, http://netlab.caltech.edu

[2] D. P. Bertsekas, "An auction algorithm for shortest paths", *SIAM J. Opt.*, vol. 1, pp. 425–447, 1991.

[3] D. P. Bertsekas, "A simple and fast label correcting algorithm for shortest paths", *Networks*, vol. 23, pp. 703–709, 1993.

[4] D. P. Bertsekas, F. Guerriero, and R. Musmanno, "Parallel asynchronous label-correcting methods for shortest paths", *J. Opt. Theory Appl.*, vol. 88, pp. 297–321, 1996.

[5] D. P. Bertsekas and R. Gallager, *Data Networks*. Englewood-Cliffs: Prentice-Hall, 1992.

[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont: Athena Scientific, 1997.

[7] D. E. Comer, *Internetworking with TCP/IP*. Vol. I: *Principles, Protocols, and Architecture*, Englewood Cliffs: Prentice-Hall, 1991.

[8] F. Guerriero and R. Musmanno, "Parallel asynchronous algorithms for the K shortest paths problem", *J. Opt. Theory Appl.*, vol. 104, no. 1, pp. 91–108, 2000.

[9] V. Jacobson, "Congestion avoidance and control", *ACM Comput. Commun. Rev.*, vol. 18, pp. 314–329, 1988.

[10] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance", in *Proc. IEEE Infocom*, Hong Kong, 2004.

[11] A. Karbowski, "Distributed asynchronous routing in Internet – a survey of optimization algorithms", in *Proc. I Int. Conf. Decis. Sup. Telecommun. Inform. Soc. (DSTIS)*, Warsaw, Poland, pp. 89–97.

[12] A. Karbowski, "Correction to Low and Lapsley's article "Optimization flow control. I. Basic algorithm and convergence", *IEEE/ACM Trans. Netw.*, vol. 11, issue 2, pp. 338–339, 2003.

[13] F. P. Kelly, "Charging and rate control for elastic traffic", *Eur. Trans. Telecommun.*, vol. 8, no. 1, pp. 33–37, 1997.

[14] S. Low and D. E. Lapsley, "Optimization flow control. I. Basic algorithm and convergence", *IEEE/ACM Trans. Netw.*, vol. 7, issue 6, pp. 861–874, 1999.

[15] S. Low, "Optimization flow control with on-line measurement or multiple paths", in *Proc. 16th Int. Teletraf. Congr.*, Edinburgh, UK, 1999.

[16] S. H. Low, "A duality model of TCP and queue management algorithms", *IEEE/ACM Trans. Netw.*, vol. 11, issue 4, pp. 525–536, 2003.

[17] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control", *IEEE Contr. Syst. Mag.*, vol. 22, no. 1, pp. 28–43, 2002.

[18] L. Polymenakos and D. P. Bertsekas, "Parallel shortest path auction algorithms", *Paral. Comput.*, vol. 20, pp. 1221–1247, 1994.

[19] H. Yaiche, R. R. Mazumdar, and C. Rosenberg, "A game-theoretic framework for bandwidth allocation and pricing in broadband networks", *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 667–678, 2000.

Andrzej Karbowski received his M.Sc. degree in electronic engineering (specialization automatic control) from Warsaw University of Technology (Faculty of Electronics) in 1983. He received the Ph.D. in 1990 in automatic control and robotics. He works as adjunct both at Research and Academic Computer Network (NASK) and at the Faculty of Electronics and Information Technology (at the Institute of Control and Computation Engineering) of Warsaw University of Technology. His research interests concentrate on data networks management, optimal control in risk conditions, decomposition and parallel implementation of numerical algorithms.

e-mail: A.Karbowski@ia.pw.edu.pl
Research and Academic Computer Network (NASK)
Wąwozowa st 18
02-796 Warsaw, Poland

Institute of Control
and Computation Engineering
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland