INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 4, PP. 381–388
Manuscript received October 31, 2012; revised Decemer, 2012. DOI: 10.2478/v10177-012-0052-0

Overlay Multicast Optimization: *IBM ILOG CPLEX*

Michał Kucharzak, Dawid Zydek, and Iwona Poźniak-Koszałka

***Abstract*—*IBM ILOG CPLEX Optimization Studio* delivers advanced and complex optimization libraries that solve linear programming (LP) and related problems, e.g., mixed integer. Moreover, the optimization tool provides users with its *Academic Research Edition*, which is available for teaching and non-commercial research at no-charge. This paper describes the usage of *CPLEX* C++ API for solving linear problems and, as an exhaustive example, optimization of network flows in overlay multicast is taken into account. Applying continuous and integral variables and implementing various constraints, including equations and inequalities, as well as setting some global parameters of the solver are presented and widely explained.**

***Keywords*—Overlay multicast, maximum flows, linear programming, mixed integer programming.**

I. Introduction

LINEAR PROGRAMMING is one of the most important areas of optimization. It takes various linear equalities and inequalities related to specific situation and, in general, determines the optimal value obtainable under defined constraints. Recently, there is a huge number of practical applications employing linear programming dedicated for industry.

In order to model various problems and solve them with linear (or linear-based) programs, one could imagine it is necessary to implement some algorithms completely from scratch. Fortunately, there exist a lot of tools and libraries that might be incorporated to solve linear problems. Among variety of examples, it is worth mentioning GLPK [1] based on GNU licenses; GuRoBi [2], QSopt [3] library that provides a set of functions for creating, manipulating, and solving linear programming problems; GAMS/XPRESS linear and mixed-integer programming solver [4] (actually it is not actively supported); XA [5]; LiPS [6]; LP_Solve [7] and a lot of preliminary academic or experimental versions of solvers. Even Matlab or Microsoft Excel deliver some tools with methods of the linear optimization.

This paper shows an example usage of linear solver, namely *IBM ILOG CPLEX*, that is one of the most advanced optimization tools. Although, its commercial use requires appropriate, more-or-less paid licenses, IBM also provides academic licenses for non-commercial use at no-charge. The paper discusses taking advantage of *CPLEX* C++ interface and as an optimization problem, overlay multicast flow maximization is taken into consideration.

The paper is organized as follows. In Section II, a general illustration and explanation of multicast and overlay networks is given. Section III describes the problem of maximum flow assignment in overlay multicast system. It also includes a mixed integer program model of the Maximum Flow Trees (MFT) problem. *IBM ILOG CPLEX* C++ implementation is presented and discussed in Section IV. There are some listings with C++ code as well that show example usage of the CPLEX API. Next in Section V, an alternative and relaxed version of the MFT problem is defined. This section explains Fractional Spanning Trees (FST) packing problem that is used for maximum flow assignment in overlay multicast systems. Moreover, Section V presents a sample implementation of the FST linear program in C++ with CPLEX studio and illustrates a random heuristic that might be used for solving the maximum flow problem. Results of experimentation results are presented in Section VI, where using some CPLEX parameters and useful methods is also shown. Finally, the paper is summarized in Section VII.

The research leading to these results is co-financed by the European Union as part of the European Social Fund according to the Operational Programme Human Capital *National Cohesion Strategy*.

M. Kucharzak and I. Poźniak-Koszałka are with the Department of Systems and Computer Networks, Wrocław University of Technology, Poland (e-mails: michal.kucharzak@pwr.wroc.pl; iwona.pozniak-koszalka@pwr.wroc.pl).

D. Zydek is with the Department of Electrical Engineering, Idaho State University, USA (e-mail: zydedawi@isu.edu).

II. Overlay Networks and Multicast

In contrast to traditionally understood networks, where nodes and links usually represent physical resources (e.g. routers, switches, cables or wireless links), overlay networks are a kind of abstraction and describe virtual or logical networks that lay over the physical resources and over transportation layers. Such overlay networks actually employ underlying physical network technologies in order to provide end-system (end-host) related communication and over the years overlays have been getting more and more attention in research community as well as in business world. Since overlays tackle many drawbacks present in pure "link-router-network" engineering, they have become an excellent solution for multimedia-oriented applications. A good example comprises multicast communications, where the same content is being delivered to a group of users (e.g., video streaming, e-lectures, e-conferences, etc.). Figure 1 illustrates general concepts of multicast implementations, where network layer approach (e.g., IP Multicast [8]) uses physical infrastructure and overlays implement multicast in application layer of end-hosts. Readers who are interested in the research subject may find more details on the overlay multicast and its optimization in [9]–[19] and references therein.

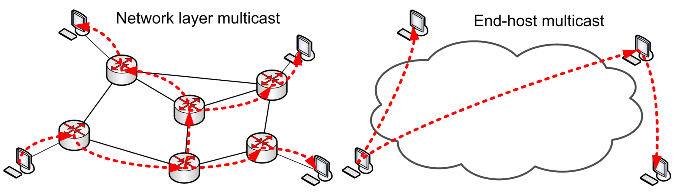

Fig. 1. Implementations of multicast communications: network layer multicast (left), end-host based multicast (right).

III. Maximum Flow Problem

A. Problem Description

Multicast system defined for overlay networks uses routing implemented by end hosts and might be optimized in centralized manner by employing linear programming techniques. We consider an overlay multicast system with a single server and multiple receivers, where the achievable streaming rate needs to be sustained for all receivers in the session. In addition, we address the problem for multicast systems that are relatively static, i.e. all participants are known in advance and no unexpected joins or disconnections of nodes are possible. Even though one could imagine overlay-based multicast systems as rather dynamic in their behavior, some static versions might be easily distinguished. Let us bring into discussion, e.g., VoD or IPTV systems based on set-top boxes, where users who paid for such services are intended to exist in a multicast group. Moreover, some systems applied for dissemination of critical information might comprise well known set of nodes, for instance, weather forecasting system with weather stations; software updates for static set of servers; stock exchange data with defined market players; traffic information systems or even e-lecture and teleconference systems with fixed topology and defined number of receiving nodes. Without loss of generality, such an assumption is also valid for quasi-static systems with relatively slow changes, where time required for routing convergence is shorter than topology changes and in case a new peer appears or disappears from the system, a new multicast structure is assigned. Additionally, multicast routing is assigned in a centralized way, i.e., a multicast server is responsible for calculating multicast flows and propagating them to other participants.

We assume the multicast stream can be split into several separate substreams and the substreams might be illustrated as separate spanning trees rooted in the same vertex. Such a concept assures the load in the network is balanced and user resources are utilized in more effective way. Moreover, multiple trees might satisfy some coding requirements or even provide reliability of the systems.

Eventually, the MFT problem defined for overlay systems describes multicast routing with maximum achievable streaming rate among all overlay nodes, where the nodes have limited access link capacities.

MFT Problem

indices

i, j	= $0, 1, ..., V-1$, end hosts (peers, overlay system nodes)
t	= $0, 1, ..., T-1$, trees, multicast substreams

constants

s	s, root node, server
u_i	available upload capacity of node i
d_j	available download capacity of node j

variables

r_t	flow assigned to tree t
x_{ijt}	=1 if tree t contains arc (i,j); 0 otherwise (binary)
f_{ijt}	conceptual flow on arc (i,j) in tree t (integral)
y_{ijt}	flow assigned to overlay arc (i,j) in tree t

objective

$$\max_{r,x,f,y} \sum_t r_t \tag{1}$$

constraints

$$\sum_{i \neq j} x_{ijt} = 1 \quad \forall j \neq s \quad \forall t \tag{2}$$

$$\sum_{i \neq j} f_{ijt} - \sum_{i \neq j,s} f_{jit} = 1 \quad \forall j \neq s \quad \forall t \tag{3}$$

$$f_{ijt} \leq (V-1)x_{ijt} \quad \forall i \quad \forall j \neq s, i \quad \forall t \tag{4}$$

$$y_{ijt} \leq u_i x_{ijt} \quad \forall i \quad \forall j \neq s, i \quad \forall t \tag{5}$$

$$\sum_{j \neq s} \sum_t y_{ijt} \leq u_i \quad \forall i \tag{6}$$

$$\sum_t r_t \leq \min\{d^{min}, u_s\} \tag{7}$$

$$\sum_{i \neq j} y_{ijt} = r_t \quad \forall j \neq s \quad \forall t \tag{8}$$

B. Modeling

Overlay multicast flow (stream) is allowed to be split into at most T substreams, which are realized on separate spanning trees. The main scope of the MFT formulation is twofold: it provides us with creating a set of overlay multicast trees (x_{ijt}) as well as streaming rate r_t assigned to each tree. Finally, maximized throughput in the system is expressed by the objective (1). Equation (2) refers to the completion constraint and assures each node except of the root has exactly one predecessor in each tree t. Constraint (3) is derived from the flow conservation concept and guarantees the structure of fractional flow t is a directed spanning tree rooted at node s. The equation assures that there is conserved only one logical inflow for every receiver j (j relays all flows but one directed from s to j). Formula (4) bounds logical flow f variables with tree variables x. The equations represent arc (i,j) cannot be traversed by more logical flows than $V-1$ (to all receivers) only if tree t contains this arc.

Having created T trees (feasible x), formulas (5) and (6) refer to as upload capacity constraints. The only non-zero flow y_{ijt} can be assigned to overlay link (i,j) in tree t if and only if the link is used in the tree ($x_{ijt} = 1$), in such a case, y_{ijt} cannot exceed upload capacity limit of node u_i. Additionally, total throughput transmitted from node i among all trees is also subject to u_i. Eventually, each tree t is supposed to route r_t traffic flow in such a way every link of tree t realizes the same volume of flow. Since every receiver is guaranteed by (2) to have single predecessor in each tree, constraints (8) ensure each link of tree t carries flow r_t. Therefore, the total system stream carried by all multicast substreams equals to $\sum_t r_t$. In order to provide feasible throughput, $\sum_t r_t$ cannot be greater than the minimum download capacity limit among all nodes and available upload capacity of the source node u_s (7).

IV. *IBM ILOG CPLEX* Usage

Full versions of the latest releases of the *IBM ILOG CPLEX Optimization Studio* are available at no-charge, along with professionally-developed courseware, to registered members of *IBM Academic Initiative* for teaching and non-commercial research. At the beginning, everyone who wants to use *CPLEX* should refer to exhaustive installation guide available at [20]. Note that the product cannot be used without installing and verifying a valid license key.

This work presents *IBM ILOG CPLEX* usage under Microsoft Visual Studio 2008 and Windows, however *CPLEX* also provides its versions for Linux and Mac OS.

A. Environment and Model Objects

C++ projects require including *CPLEX* libraries and the main *CPLEX* interface is to be included with the statement shown in Listing 1.

Listing 1. Include *CPLEX* interface.

```
1  #include <ilcplex/ilocplex.h>
```

Before creating modeling objects, it is required to construct an object of the class `IloEnv`. This object is known as the environment and it is constructed with the statement in line 1 of Listing 2. Next, to formulate a full optimization problem, the objects that are part of the problem need to be selected. This is done by adding them to an instance of `IloModel`, which is the class used to represent optimization problems (line 2).

Listing 2. Environment, model and variables declaration.

```
1  IloEnv env;
2  IloModel model(env);
3
4  IloNumVar * r;
5  IloNumVar *** x;
6  IloNumVar *** f;
7  IloNumVar *** y;
```

Finally, objects of `IloNumVar` represent specific variables defined for the MFT problem. Note that first we only declare pointers or pointers to pointers (to pointers) to `IloNumVar` in order to further dynamic memory allocation.

B. Defining Variables

Listing 3 presents how to define variables under previously declared pointers. In line 1, new `T` variables that represent flow assigned to every multicast tree are prepared, and from now, `r[t]` represents r_t.

Next, every object `r[t]` is attached to the environment `env` object and represents continuous variable (`ILOFLOAT`) with range from 0 to `UPPER_R`, where `UPPER_R` is a maximum achievable flow on tree t and might be easily taken from minimum values between u_s or $\min_j\{d_{j\neq s}\}$.

In lines 8-19, variables `x` are defined, where every `x[i][j][t]` represents binary x_{ijt} (`ILOBOOL`). By analogy, variables `f` and `y` are defined in lines 21-32 and 34-46, respectively. Auxiliary variables `f` refer to every f_{ijt} and are used in flow balance constraints, whereas variables `y` represent each continuous y_{ijt} and help to bound feasible r_t for all t's.

Listing 3. Variables definition.

```
1  r = new IloNumVar[T];
2
3  for(int t=0;t<T;t++)
4  {
5      r[t]=IloNumVar(env,0,UPPER_R,ILOFLOAT);
6  }
7
8  x = new IloNumVar**[V];
9  for(int i=0;i<V;i++)
10 {
11     x[i] = new IloNumVar*[V];
12     for(int j=0;j<V;j++)
13         if(i!=j && j!=s)
14         {
15             x[i][j] = new  IloNumVar[T];
16             for(int t=0;t<T;t++)
17                x[i][j][t] = IloNumVar(env,0,1,ILOBOOL);
18         }
19 }
20
21 f = new IloNumVar**[V];
22 for(int i=0;i<V;i++)
23 {
24     f[i] = new IloNumVar*[V];
25     for(int j=0;j<V;j++)
26         if(i!=j && j!=s)
27         {
28             f[i][j] = new  IloNumVar[T];
29             for(int t=0;t<T;t++)
30               f[i][j][t] = IloNumVar(env,0,V-1,ILOINT);
31         }
32 }
33
34 y = new IloNumVar**[V];
35 for(int i=0;i<V;i++)
36 {
37     y[i] = new IloNumVar*[V];
38     for(int j=0;j<V;j++)
39         if(i!=j && j!=s)
40         {
41             y[i][j] = new  IloNumVar[T];
42             for(int t=0;t<T;t++)
43                 y[i][j][t] =
44                     IloNumVar(env,0,UPPER_R,ILOFLOAT);
45         }
46 }
```

C. Defining Objective

In order to define objective function, it is worth using `IloExpr` object that defines mathematical expressions embracing defined variables. The MFT objective is expressed by formula (1) and total streaming rate is to be maximized. First step (Listing 4, lines 1-4) defines expression `obj` that sums all `r[t]` up. In line 6, an object `objective` of `IloObjective` class is created. The object is initialized with `obj` expression and the optimization goal is set to maximize (`IloObjective::Maximize`). In the next step, `objective` is added to the model

Listing 4. Objective definition.

```
1  IloExpr obj(env);
2
3  for(int t=0;t<T;t++)
4      obj+=r[t];
5
6  IloObjective objective(env,obj,IloObjective::Maximize);
7  model.add(objective);
8
9  obj.end();
```

Note that variables need not be added to the model explicitly, as they are implicitly considered if any of the other modeling objects in the model use them, i.e., the objective uses only r variables; however, there are also x, y and f applied to the model. When using an expression is finished (i.e., any constraint is created with it), it is desired to delete the expression by calling its method `end()` (line 9).

D. Defining Constraints

Listing 5 presents a source code that defines and applies all constraints (2)-(8). The easiest way to add the constraints to the model instance is to do it by employing aforementioned objects of `IloExpr`. In lines 1-15 completion constraints (2) are implemented. Loops `for` and conditions `if` guarantee appropriate indexing scheme. For example (2) defines constraints for $j = 1, ..., V : j \neq s$ and for $t = 1, ..., T$, where every left-hand side of such expressions equals to $\sum_{i \neq j} x_{ijt}$. Finally, `model.add(parent == 1)` applies constraints to the model. Similarly to the expression instance used in the objective, `end()` method deletes the expression after applying it to the optimization model object.

Listing 5. Defining constraints.

```
1  for(int j=0;j<V;j++)
2      if(j!=s)
3      {
4          for(int t=0;t<T;t++)
5          {
6              IloExpr parent(env);
7
8              for(int i=0;i<V;i++)
9                  if(i!=j)
10                     parent+=x[i][j][t];
11
12             model.add(parent == 1);
13             parent.end();
14         }
15     }
16
17 for(int t=0;t<T;t++)
18     for(int j=0;j<V;j++)
19         if(j!=s)
20         {
21             IloExpr flow(env);
22
23             for(int i=0;i<V;i++)
24                 if(j!=i)
25                     flow+=f[i][j][t];
26
27             for(int i=0;i<V;i++)
28                 if(j!=i && i!=s)
29                     flow-=f[j][i][t];
30
31             model.add(flow==1);
32             flow.end();
33         }
34
35 for(int t=0;t<T;t++)
36     for(int i=0;i<V;i++)
37         for(int j=0;j<V;j++)
38             if(j!=s && i!=j)
39               model.add(f[i][j][t] <= (V-1)*x[i][j][t])
                     ;
40
41 for(int i=0;i<V;i++)
42     for(int j=0;j<V;j++)
43         if(i!=j && j!=s)
44             for(int t=0;t<T;t++)
45                 model.add(y[i][j][t]<=x[i][j][t]*u[i]);

46
47 for(int i=0;i<V;i++)
48 {
49     IloExpr upload(env);
50
51     for(int j=0;j<V;j++)
52         if(i!=j && j!=s)
53             for(int t=0;t<T;t++)
54                 upload+=y[i][j][t];
55
56     model.add(upload<=u[i]);
57     upload.end();
58 }
59
60 IloExpr download(env);
61 for(int t=0;t<T;t++)
62     download+=r[t];
63 model.add(download<=dmin);
64 download.end();
65
66 for(int t=0;t<T;t++)
67     for(int j=0;j<V;j++)
68         if(j!=s)
69         {
70             IloExpr boundYR(env);
71
72             for(int i=0;i<V;i++)
73                 if(i!=j)
74                     boundYR+=y[i][j][t];
75                 model.add(boundYR == r[t]);
76                 boundYR.end();
77         }
```

Flow conservation or flow balance constraints (3) are defined in lines 17-33, where `flow` is an instance of `IloExpr` and aggregates $\sum_{i \neq j} f_{ijt} - \sum_{i \neq j,s} f_{jit}$ for every t and $j \neq s$. The constraints are created by assuring `flow` equals to 1. In order to apply constraint given by formula (4), a snippet of the code in lines 35-39 is employed. Note that it is not obligatory to use expression instances of `IloExpr` to formulate model constraints and `f[i][j][t] <= (V-1)*x[i][j][t]` refers to $f_{ijt} \leq (V-1)x_{ijt}$. In the same manner constraints (5) are implemented in lines 41-45 in Listing 5. Without explicit using `IloExpr`, the constraints are added to `model` object by simple expression `model.add(y[i][j][t]<=x[i][j][t]*u[i]);`. Upload capacity constraints defined by (6) are coded in lines 47-58 and download capacity constraint defined by (7) is applied in lines 60-64, where `dmin` represents a minimum value among $\min_{j \neq s} d_j$ and u_s. The last set of constraints (8) coupling y and r variables is defined in lines 66-77, where `boundYR` is an expression comprising $\sum_{i \neq j} y_{ijt}$. The constrains yield `boundYR == r[t]` for $t = 1, ..., T$ and $j = 1, ..., V : j \neq s$.

E. Solving the Model

This subsection introduces the C++ class `IloCplex`. An instance of the class `IloCplex` is used to solve the model. Actually, `IloCplex` derives from `IloAlgorithm` and not every model might be solved by the `IloCplex` instances.

Listing 6 presents the simplest usage of *CPLEX* algorithms. In line 1, the `cplex` object is created by the constructor `IloCplex(model)`. This constructor extracts the data from the model into the appropriate efficient data structures (sparse matrices), which *CPLEX* uses for solving the problem. It is done by extracting each of the modeling objects previously added to the model and each of the objects referenced by them. For every extracted modeling object, corresponding data structures are created internally in the `cplex` object.

Listing 6. Solving the model

```
1 IloCplex cplex(model);
2 cplex.solve();
```

After the model is extracted to the `cplex` object, the problem might be solved by calling `solve()` method (line 2). For most problems, this is everything that is needed for solving the model. Nonetheless, *CPLEX* offers a variety of controls and parameters that provide specific adaptation for the solution process. For detailed descriptions please refer to the documentation, especially please see method `IloCplex::setParam`.

V. Problem Relaxation

A. FST

In contrast to the previous MFT formulation, FST includes a predefined set of trees $t \in \{1, 2, ..., T\}$ and a corresponding set of variables r_t describing the steaming rate allocated to tree t. Thereby, the MFT problem is relaxed to only find maximum flows and there is no need for simultaneous multicast trees

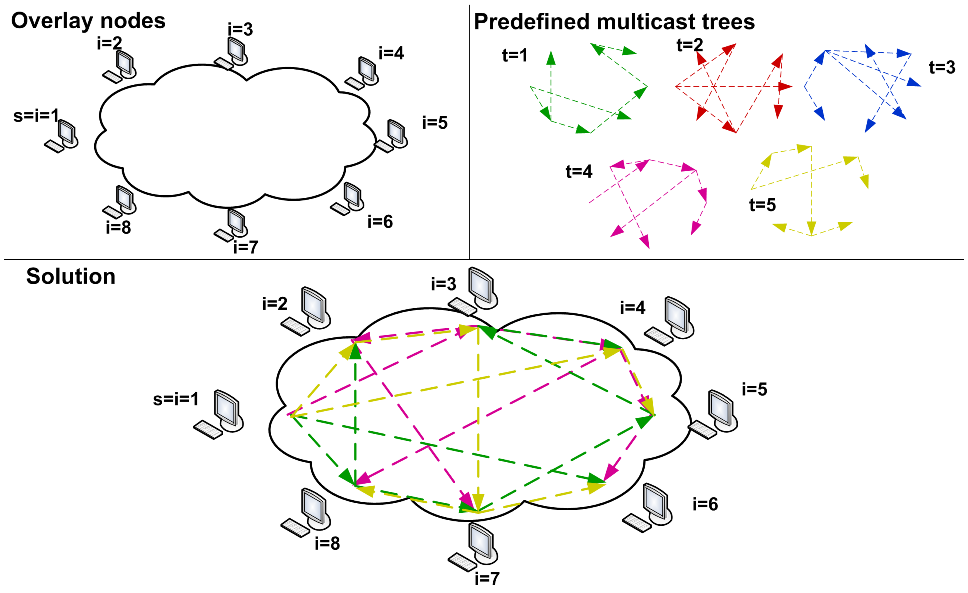

Fig. 2. An illustrative example of FST packing for overlay multicast.

construction. Such a concept is derived from the fractional Steiner packing trees described in [21].

The FST version of the problem comprises multicast trees represented by vector β, which contains a number of children of every node in each tree t. The problem is to find a maximal stream assignment to predefined trees regarding available capacity limits of participants' access links. Every tree t represents a fractional multicast stream. Note that the vector β does not define an exact topology of every tree (actual arcs in the tree cannot be resolved), but such a tree representation is sufficient in order to formulate node capacity constraints and overlay multicast flows.

Figure 2 illustrates an example overlay system with $V = 8$ nodes and a set of $T = 5$ predefined trees, where, i.e., for $t = 1$ $\beta_1 = [2,0,1,0,1,0,1,2]$, for $t = 2$ $\beta_2 = [3,0,1,1,0,0,2,0]$, for $t = 3$ $\beta_3 = [2,3,0,1,0,1,0,0]$, etc. Additionally, not every tree is required to carry non-zero substream, e.g., $r_2 = 0$ and $r_3 = 0$ in the presented example.

FST Packing Problem

indices

i, j	$= 0, 1, ..., V-1$ overlay nodes
t	$= 0, 1, ..., T-1$ predefined trees

constants (additional)

s	source, root node, server
u_i	available upload capacity limit of node i
d_j	available download capacity limit of node j
β_{ti}	number of i's children in tree t; 0 if i is a leaf

variable

r_t	flow assigned to tree t

objective

$$\max_{r} \sum_{t} r_t \tag{9}$$

constraints

$$\sum_{t} \beta_{ti} r_t \le u_i \quad \forall i \tag{10}$$

$$\sum_{t} r_t \le \min\{d^{min}, u_s\} \tag{11}$$

The objective function (9) maximizes summarized throughput of all fractional streams assigned to multicast spanning trees spread among V nodes with a single source of the content s. Next, node capacity constraints are introduced and each node i can neither contribute to the system exceeding its upload capacity nor download more than its download limit. Constraints (10) formulate upload limit based on available upload capacities u_i of every node in the system. Note that node i is a parent node in tree t, it uploads content of size r_t exactly β_{ti} times. Therefore i's total upload given by $\sum_t \beta_{ti} r_t$ cannot exceed its upload limit u_i. Upload capacity limit is defined for every node in the system, therefore there are exactly V inequalities. By analogy to upload bound (10), a set of constraints (11) is introduced in order to guarantee download limits of nodes are not surpassed. Basically, the download limit expresses that $\sum_t r_t$ cannot be greater than d_i, for every i. However, taking into account the overlay system, where all overlay nodes are connected to all trees, the actual throughput $\sum_t r_t$ is restricted not to exceed the minimal download limit among all overlay nodes ($min_{i \neq s}\{d_i\}$). Since we consider the system where source node s only uploads the content, the download capacity limit of s is not taken into consideration in this multicast flow allocation problem. Moreover, a source's upload limit affects the total stream $\sum_t r_t$ to be less or equal than u_s, i.e., all flows originate from root node s and the root cannot produce more throughput than its available upload limit u_s. Finally, a minimum value among d^{min} or u_s limits overall throughput packed into the multicast system. In such a way, a number of capacity constraints that corresponds to the overall multicast streaming rate in the system is decreased to exactly one.

It is worth referring to some previous works on overlay multicast, which employ FST linear formulation, e.g., see [22]–[24].

The important point is to note in the abovementioned formulations that in the MFT and FST models, all variables r_t may take both, continuous or integer values. However, whereas continuity of variables is not directly applicable in most of real networks (e.g., number of packets or number of bytes may be rather integral values), the proposed linear-based models might serve as continuous relaxations for actual models, often used in (meta-)heuristic algorithms or protocols development and as approximations.

B. CPLEX API

Listing 7 below illustrates an example implementation of the FST model using CPLEX C++ API.

Listing 7. Example implementation of FST.

```
1   IloEnv env;
2   IloModel model(env);
3
4   IloNumVar * r;
5   r=new IloNumVar[T];
6
7   for(int t=0;t<T;t++)
8       r[t]=IloNumVar(env,0,UPPER_R,ILOFLOAT);
9
10  IloExpr obj(env);
11
12  for(int t=0;t<T;t++)
13      obj+=r[t];
14
15  model.add(IloMaximize(env, obj));
16  obj.end();
17
18  for(int i=0;i<V;i++)
19  {
20      IloExpr upload(env);
21      for(int t=0;t<T;t++)
22          upload+=r[t]*beta[t][i];
23
24      model.add(upload <= u[v]);
25      upload.end();
26  }
27
28  IloExpr download(env);
29  for(int t=0;t<T;t++)
30      download+=r[t];
31
32  model.add(download <= dmin);
33  download.end();
34
35  IloCplex cplex(model);
36
37  cplex.solve();
```

In lines 1 and 2, environment and model objects are created, respectively. Line 5 dynamically allocates T pointers to `IloNumVar` that represent variables r. Next, loop `for` in line 7 defines variables in the model. In lines 10-16 the objective is defined. Note that the objective is added to the model with function `IloMaximize` and it is alternate to the approach shown in Listing 4. Constraints (10) are implemented within loop `for` (lines 18-26) and (11) is defined between lines 28-33. Line 35 instantiate the CPLEX solver and solves the problem by calling `solve` method of `IloCplex` (line 37).

C. Random Algorithm with FST

On the one hand, exact algorithms such as these provided by CPLEX might provide optimal benchmarks for the defined problem. However, on the other hand, overlay multicast systems are rather random-like in their behavior. Therefore random algorithm is implemented in order to provide some benchmarks representing random nature of overlays.

A pseudocode Algorithm 1 illustrates general random approach to solving the problem in a random based way and it might be described in two steps. Loop in lines 2-4 initializes vector β with randomly created T multicast trees and in line 5 optimal flow assignment is provided by solving the FST model.

Algorithm 1 Random Algorithm

```
1: procedure RandomAlgorithm($V, T, s, d, u$)
2:     for all $t \in T$ do
3:         $\beta_t \leftarrow$ RandomTreeGenerator($V, s$)
4:     end for
5:     Solve FST program (9)-(11)
6: end procedure
```

Taking into account the fact that the FST model requires overlay nodes with their capacity limits as well as a predefined set of spanning trees as an input, Algorithm 2 shows a procedure that generates a random spanning tree among V nodes with a single source and returns β describing a number of children nodes served by every node in the tree.

Algorithm 2 Random Tree Generator

```
1: procedure RandomTreeGenerator($V, s$)
2:     $A \leftarrow s$
3:     $B \leftarrow \emptyset$
4:     for all $i$ do
5:         $\beta_i \leftarrow 0$
6:         if $i \neq s$ then
7:             $B \leftarrow B \cup i$
8:         end if
9:     end for
10:    while $B \neq \emptyset$ do
11:        $j \leftarrow$ SelectRandomNode($B$)
12:        $i \leftarrow$ SelectRandomNode($A$)
13:        $\beta_i \leftarrow \beta_i + 1$
14:        $A \leftarrow A \cup \{j\}$
15:        $B \leftarrow B \backslash \{j\}$
16:    end while
17:    return $\beta$
18: end procedure
```

At the beginning, two auxiliary sets are initialized in lines 2 and 3. Set A contains all nodes that are feasible to be parents and before any new nodes join the tree, it includes only the source node s. Set B will comprise nodes to be connected. In loop `for` in lines 4-9, every node i but the source s is included to the set B; and β_i with h_i are set to 0, where β_i is a number of children of node i and h_i refers to a number of hops from the source to node i.

Next, until all nodes are not connected to the tree and B is an empty set, actual random and hop-constrained tree is constructed in loop `while` (lines 10-16). Procedure `SelectRandomNode` randomly chooses next child node j from set B and its parent i from set A, in lines 11 and 12, respectively. Next, a number of i's children increases by 1 (line 13) and j extends the A set as well as might be selected as a parent node in the further loop execution. After j is connected to the tree (line 14), it is removed from set B in line 15. Final step in line 17 returns a vector β that is unambiguously applied to the FST linear program.

VI. Experimentation Results

A. *Environment and Scenario Configuration*

This section illustrates comparison of maximum flow overlay multicast problem represented by MFT and random heuristic with the FST linear program. Two overlay systems are taken into consideration. The first comprises 10 nodes and the second 20 nodes. All nodes are connected to the network using ADSLs (Asymmetric Digital Subscriber Lines). Nodes are proportionally distributed among top 11 Polish Internet Service Providers (ISPs); and available upload and download capacities of the nodes are taken from recent SpeedTest-based reports that are provided by Net Index [25]. Available upload capacity of nodes are distributed among values from 2.20 to 12.58 Mb/s, where the source s belongs to an ISP that provides 12.58 Mb/s. Download capacities are, depending on the ISPs, from 5.67 to 28.79 Mb/s. Since the total flow of the stream remains the same for all nodes, the upper bound of maximum multicast flow equals to 5.67 Mb/s.

Computational experiments were carried out on an Intel Core 2 Duo CPU with 2.13 GHz clock and 4GB RAM, with x64 Windows 7 Professional. An experimentation system was implemented in C++ under Microsoft Visual Studio 2008 and CPLEX 12.3 is employed for solving linear and mixed integer programs (see [26] for former version). Measurements of execution time are obtained with `GetTickCount()` function from *Windows.h* library.

B. *Useful CPLEX Parameters and Methods*

Listing 8 shows how to apply some basic parameters to the CPLEX solver.

Listing 8. CPLEX parameters and output.
```
1  cplex.setParam(IloCplex::TiLim, 3600);
2  cplex.setParam(IloCplex::EpAGap , 0);
3  cplex.setParam(IloCplex::MIPDisplay, 0);
4  cplex.setOut(env.getNullStream());
```

Parameter `IloCplex::TiLim` sets the maximum time, for computations before termination. The time limit includes preprocessing time and is given in seconds. Another way to terminate the mixed integer optimization is to define `IloCplex::EpAGap` that refers to an absolute tolerance on the gap between the proved upper bound for maximization problems and the objective of the best node remaining. When this difference falls below the defined value in percents, the optimization process is stopped. In line 3 `IloCplex::MIPDisplay` determines what CPLEX reports to the screen during mixed integer optimization, here it is set to 0 and causes no node log to be displayed until the optimization is finished. Since displaying information might affect time efficiency, setting `IloCplex::MIPDisplay` might be very useful while one would decrease total computational time. However, the parameter might be applied to mixed integer optimization only and it does not remove all messages appearing on the screen, it might be even easiest to use method `getNullStream()` of `IloCplex::IloEnv` as it is shown in line 4. The method forks all CPLEX output to a null stream and no information from the solver appears, e.g., on screen.

Since `cplex.solve()` function finishes its operation and the optimization is brought to the end or terminated, listing 9 presents six basic get-based methods that are useful while getting some results. Method `IloCplex::getStatus()` in line 1 provides a status of the result, i.e., optimal, feasible, infeasible, unbounded, infeasible or unbounded, unknown, or if any error appeared while solving problems. Method `IloCplex::getObjValue()` in line 2 returns an `IloNum` instance with the best objective value if the solution status is optimal or feasible. This value might be cast to, e.g., `double`. In a similar way, `IloCplex::getBestObjValue()` in line 3 yields an upper bound of the MFT problem.

Listing 9. Getting CPLEX results.
```
1  cplex.getStatus();
2  cplex.getObjValue();
3  cplex.getBestObjValue();
4  cplex.getMIPRelativeGap();
5  cplex.getTime();
6  cplex.getValue(r[t]);
```

In line 4 `getMIPRelativeGap()` returns the gap between the proved upper bound and the objective value, but note that its applicable only to mixed integer optimization. Computational time might be reported with `getTime()` in line 5. In order to get actual values of variables, it is necessary to employ `getValue` method that takes `IloNumVar` as its argument, what is shown in line 6.

C. *Results Summary*

Table I reports optimization results of CPLEX MFT implementation and random algorithm with CPLEX FST implementation for overlay multicast networks with $V = 10$ and $V = 20$ nodes and T from 1 to 7. In order to limit more-or-less unpredictable computational time of exact methods implemented by CPLEX, `IloCplex::TiLim` is set to 3600 seconds (see an example in Listing 8, line 1). Feasible results are provided in bold, and for only $V = 10$ nodes and T=1 multicast tree, MFT problem was solved in an optimal way. Besides the fact that the CPLEX MFT implementation cannot guarantee optimality within one hour computation for other instances, it is easily noticeable that the total multicast flow increases if more multicast trees might be utilized in the routing structure. It ranges from 3.350 or 3.145 Mb/s if only single spanning tree realizes the multicast stream to at least 5.296 or 4.996 Mb/s with 7 separate trees, for 10 and 20 node systems, respectively. While the optimality is not guaranteed, it is worth showing upper bounds in order to indicate a general quality of feasible results.

Second part of Tab. I focuses on time and result efficiency of the random multicast tree construction applied along with CPLEX FST flow assignment problem. Such a heuristic, on the one hand, provides feasible results extremely faster in comparison to CPLEX MFT, especially for larger instances, where optimal flow trees with maximum stream are even not found. On the other hand, random-based results fall below some distance to the results by CPLEX MFT and they might be 6.5-34% worse for 10 node overlay networks and even about 30-40% for bigger instances with 20 nodes. In order to present an overall results quality of random approach, some statistics should be provided. Here, 100 independent repetitions of the random algorithm was performed to gather best (in bold), average (avg.), median and standard deviation (std.). Note that time reported for random algorithm refers to solve all 100 repetitions together; and solving a single random instance actually requires fractions of seconds.

TABLE I
RESULTS FOR OVERLAY NETWORKS WITH 10 AND 20 NODES

V	10							20						
T	1	2	3	4	5	6	7	1	2	3	4	5	6	7
CPLEX MFT [Mb/s]	**3.350***	**4.926**	**5.217**	**5.267**	**5.293**	**5.294**	**5.296**	**3.145**	**4.592**	**4.715**	**4.818**	**4.924**	**4.958**	**4.996**
upper bound [Mb/s]	3.350	5.297	5.297	5.297	5.297	5.297	5.297	3.561	5.018	5.018	5.018	5.018	5.018	5.018
time [s]	5.0	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600
Random [Mb/s]	**2.720**	**3.251**	**4.258**	**4.478**	**4.641**	**4.775**	**4.952**	**1.923**	**2.664**	**2.768**	**3.075**	**3.316**	**3.426**	**3.507**
gap to MFT [%]	-18.8	-34.0	-18.4	-15.0	-12.3	-9.8	-6.5	-38.9	-42.0	-41.3	-36.2	-32.7	-30.9	-29.8
avg. [Mb/s]	1.377	2.126	2.753	3.142	3.414	3.688	3.844	0.959	1.460	1.847	2.130	2.322	2.570	2.698
median [Mb/s]	1.303	2.208	2.775	3.227	3.439	3.743	3.875	0.906	1.418	1.841	2.130	2.337	2.588	2.678
std. [Mb/s]	0.513	0.584	0.623	0.626	0.620	0.534	0.542	0.281	0.367	0.346	0.384	0.374	0.381	0.353
time [s]	6.3	11.4	15.4	16.5	17.9	18.7	20.4	7.1	11.6	18.9	20.7	21.9	23.2	25.9

* optimal result

VII. SUMMARY

This paper described the usage of *IBM ILOG CPLEX* C++ API for solving linear problems in the area of computer networks. Among different optimization packages, *CPLEX* is one of the most advanced and provides ease and variety of applications. Moreover, it is available at no-charge for non-commercial use and academic societies. Streaming rate maximization problem in forms of MFT and FST in multicast overlay system was taken into consideration as a mixed integer and pure linear program example. The paper showed model implementation that uses continuous, binary and integral variables; various constraints including equations and inequalities as well as solving the model and setting some global parameters of the *CPLEX* solver.

REFERENCES

[1] *GNU Linear Programming Kit*, GNU Project, 2008, http://www.gnu.org/software/glpk/glpk.html.

[2] *Gurobi Optimizer Reference Manual*, Gurobi Optimization Inc., 2012, http://www.gurobi.com.

[3] *QSopt Version 1.0*, QSopt, 2003, http://www2.isye.gatech.edu/ wcook/qsopt/downloads/users.pdf.

[4] *XPRESS Solver*, FICO (Fair Isaac Corporation), http://www.gams.com/dd/docs/solvers/xpress.pdf.

[5] *XA Linear Optimizer System*, Sunset Software Technology, Inc., 2012, http://www.sunsetsoft.com/.

[6] *Linear Program Solver LiPS*, MIT, 2011, http://sourceforge.net/projects/lipside/.

[7] *lpsolve 5.5.2.0*, LP_SOLVE, http://lpsolve.sourceforge.net/5.5/.

[8] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, 1990.

[9] B. Akbari, H. R. Rabiee, and M. Ghanbari, "An optimal discrete rate allocation for overlay video multicasting," *Computer Communications*, vol. 31, no. 3, pp. 551–562, 2008, DOI: 10.1016/j.comcom.2007.08.025.

[10] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *in Proceedings of ACM Sigmetrics*, 2000, pp. 1–12.

[11] Y. Cui, Y. Xue, and K. Nahrstedt, "Optimal resource allocation in overlay multicast," in *in Proc. of 11th International Conference on Network Protocols, ICNP*, 2003.

[12] M. Kucharzak and K. Walkowiak, "A mixed integer formulation for multicast flow assignment in multilayer networks," in *Proc. Fifth Int Broadband and Biomedical Communications (IB2Com) Conf*, 2010, pp. 1–4.

[13] ——, "Maximum flow trees in overlay multicast: Modeling and optimization," in *Proc. 2nd Baltic Congress Future Internet Communications (BCFIC)*, 2012, pp. 260–267.

[14] ——, "Modelling of Minimum Cost Overlay Multicast Tree in Two Layer Networks," *International Journal of Electronics and Telecommunications*, vol. 57, no. 3, pp. 317–322, 2011.

[15] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari, "Will iptv ride the peer-to-peer stream? peer-to-peer multimedia streaming," *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 86–92, 2007.

[16] C. Wu and B. Li, "Optimal rate allocation in overlay content distribution," in *Networking*, 2007, pp. 678–690.

[17] ——, "On meeting p2p streaming bandwidth demand with limited supplies," in *In Proc. of the Fifteenth Annual SPIE/ACM International Conference on Multimedia Computing and Networking*, 2008.

[18] G. Wu and T. Chiueh, "Peer to peer file download and streaming. rpe report, tr-185," 2005.

[19] Y. Zhu and B. Li, "Overlay networks with linear capacity constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 2, pp. 159–173, 2008.

[20] IBM, "Quick start to IBM ILOG optimization products," Online, available at http://ibm.com.

[21] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing steiner trees," in *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, pp. 266–274.

[22] M. Kucharzak and K. Walkowiak, "Fractional spanning tree packing problem with survivability constraints for throughput maximization in overlay multicast networks," in *Proc. 3rd Int Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) Congress*, 2011, pp. 1–7.

[23] ——, "An improved annealing algorithm for throughput maximization in static overlay-based multicast systems." in *HAIS (1)*, ser. Lecture Notes in Computer Science, E. Corchado, M. Kurzynski, and M. Wozniak, Eds., vol. 6678. Springer, 2011, pp. 364–371, DOI: 10.1007/978-3-642-21219-2_46.

[24] ——, "On modelling of fair throughput allocation in overlay multicast networks," in *NEW2AN*, ser. Lecture Notes in Computer Science, S. Balandin, Y. Koucheryavy, and H. Hu, Eds., vol. 6869. Springer, 2011, pp. 529–540, DOI: 10.1007/978-3-642-22875-9_48.

[25] Ookla. (2012, October) Net index. [Online]. Available: www.netindex.com

[26] IBM ILOG CPLEX 12.1, *User's Manual for* CPLEX, 2009.